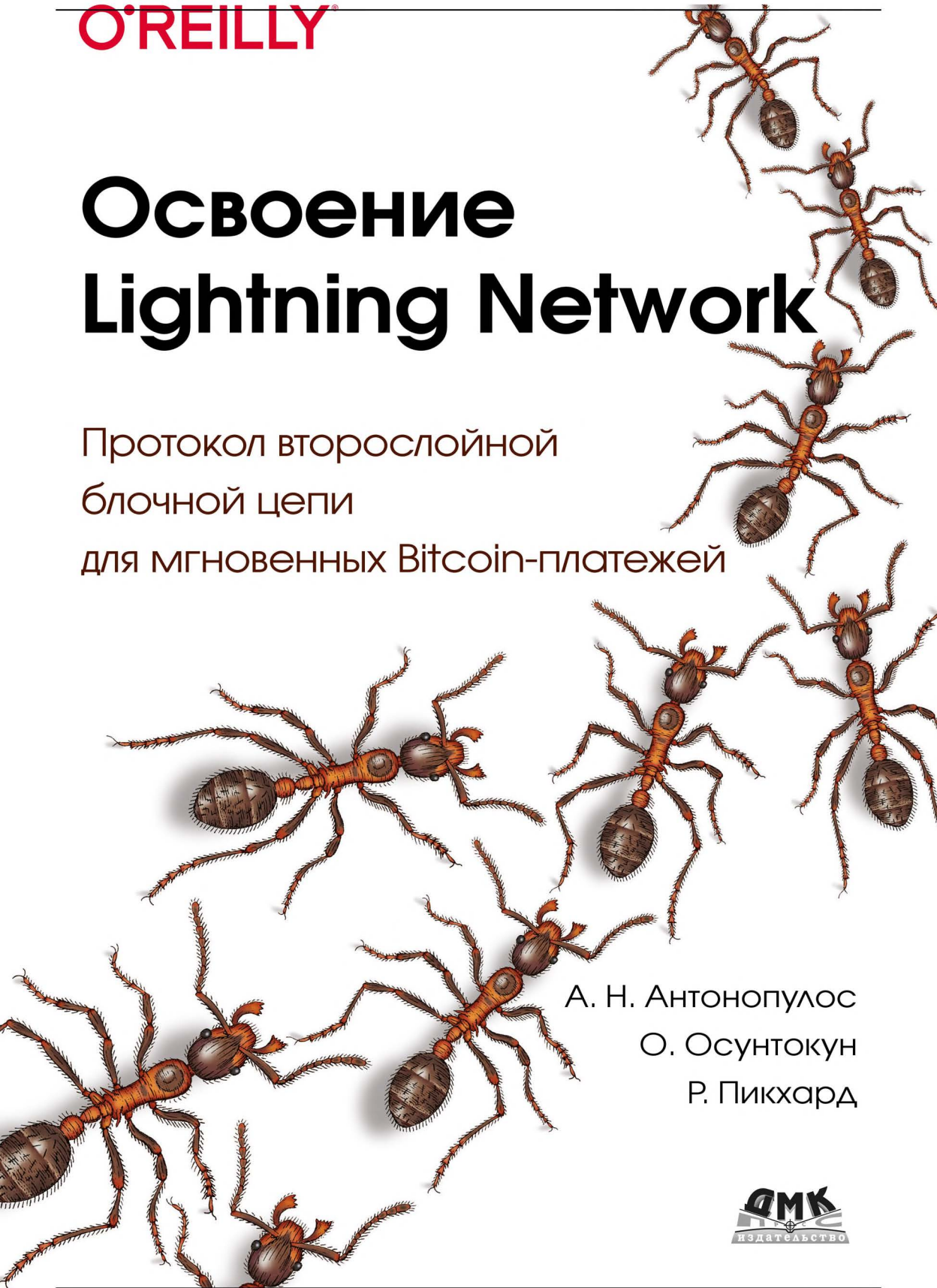
O'REILLY

Освоение Lightning Network

Протокол второслойной блочной цепи для мгновенных Bitcoin-платежей



А. Н. Антонопулос
О. Осунтокун
Р. Пикхард

ДМК ПРЕСС
издательство

Андреас Н. Антонопулос,
Олаулува Осунтокун
и Рене Пикхард

Освоение Lightning Network

Протокол второслойной блочной цепи для мгновенных Bitcoin-платежей

Mastering the Lightning Network

A Second Layer Blockchain Protocol for Instant Bitcoin Payments

Andreas M. Antonopoulos,
Olaoluwa Osuntokun
& René Pickhardt

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Освоение Lightning Network

Протокол второслойной блочной цепи для мгновенных Bitcoin-платежей

Андреас Н. Антонопулос,
Олаулува Осунтокун
и Рене Пикхард

ДМК Пресс
издательство

Москва, 2023

УДК 004.04, 336.74
ББК 32.372
А72

Андреас Н. Антонопулос, Олаулува Осунтокун и Рене Пикхардт.
А72 Освоение Lightning Network: Протокол второслойной блочной цепи для мгновенных Bitcoin-платежей / пер. с англ. А. В. Логунова. – М.: ДМК Пресс, 2022. – 450 с.: ил.

ISBN 978-5-93700-144-3

Lightning – маршрутизируемая сеть платежных каналов, которая предоставляет безопасные, дешевые, быстрые платежи Bitcoin с высокой степенью приватности, даже когда дело касается малых сумм. В этой книге приводится обзор сети Lightning, базовых концепций, которые легли в ее основу, и принципов ее работы. Примеры проиллюстрированы на языках Go, C++, Python и с использованием командной строки Unix-подобной операционной системы.

Книга адресована программистам, имеющим представление об основах системы Bitcoin, однако ряд глав доступен широкому кругу читателей, интересующихся блочными цепями.

УДК 004.04, 336.74
ББК 32.372

ISBN (анг.) 978-1-49205-486-3
ISBN (рус.) 978-5-93700-144-3

Оглавление

Предисловие **18**
Целевая аудитория 18
Условные обозначения в книге 18
Примеры исходного кода 19
Использование примеров исходного кода 20
Ссылки на компании и продукты 20
Адреса и транзакции в этой книге 20
Как с нами связаться 20
Связь с Андреасом 20
Связь с Рене 21
Связь с Олаолувой Осунтокун 21
Признательности от Андреаса 21
Признательности от Рене 21
Признательности от Олаолувы Осунтокун 22
Участники проекта 22
Источники 23

Об авторах **24**

Об иллюстрации на обложке (Колофон) **25**

ЧАСТЬ I. ПОНИМАНИЕ СЕТИ LIGHTNING **27**

Глава 1. Введение **28**
Базовые понятия сети Lightning 28
Доверие в децентрализованных сетях 30
Справедливость без центральной власти 31
Доверительные протоколы без посредников 32
Протокол справедливости в действии 33
Примитивы безопасности как строительные блоки 34
Пример протокола справедливости 35
Мотивация для сети Lightning 36
Масштабирование блочных цепей 36
Определяющие признаки сети Lightning 38
Примеры использования сети Lightning, пользователи и их истории 39
Вывод 40

Глава 2. Приступаем к работе 41
Первый кошелек Lightning Алисы 41
Узлы Lightning 42
Проводники Lightning 42
Кошельки Lightning 43
Тестовая сеть Bitcoin 46
Уравновешивание сложности и контроля 47
Скачивание и инсталляция кошелька Lightning 48
Создание нового кошелька 49
Ответственность за хранение ключей 49
Мнемонические слова 49
Безопасное хранение мнемоники 50
Загрузка биткойна в кошелек 51
Приобретение биткойна 51
Получение биткойна 52
Из системы Bitcoin в сеть Lightning 56
Каналы сети Lightning 56
Открытие канала Lightning 58
Покупка чашки кофе с помощью сети Lightning 61
Кофейня Боба 61
Счет Lightning 62
Вывод 64

Глава 3. Как работает сеть Lightning 65
Что такое платежный канал? 66
Основы платежного канала 66
Маршрутизирование платежей по каналам 67
Платежные каналы 68
Мультиподписной адрес 69
Финансовая транзакция 69
Пример плохой процедуры открытия канала 70
Фиксационная транзакция 70
Обман с предыдущим состоянием 72
Объявление канала 75
Закрытие канала 75
Взаимное закрытие (хороший путь) 76
Принудительное закрытие (плохой путь) 77
Нарушение протокола (уродливый путь) 78
Счета 79
Платежный хеш и прообраз 80
Дополнительные метаданные 81
Доставка платежа 82
Эпидемический протокол обмена сообщениями между одноранговыми узлами 82
Отыскание пути и маршрутизация 83
Отыскание пути на основе источника 84
Луковичная маршрутизация 85
Алгоритм пересылки платежей 87

Шифрование однорангового обмена сообщениями ... 88
Мысли о доверии ... 89
Сравнение с системой Bitcoin ... 89
Адреса против счетов, транзакции против платежей ... 89
Выбор выходов против отыскания пути ... 90
Выходы со сдачей в Bitcoin против отсутствия сдачи в Lightning ... 91
Майнинговые комиссионные против маршрутизационных комиссионных ... 91
Комиссионные, варьирующиеся в зависимости от трафика, против объявленных комиссионных ... 91
Публичные Bitcoin-транзакции против приватных платежей Lightning ... 92
Ожидание подтверждений против денежного расчета Lightning ... 93
Отправка произвольных сумм против ограничений по емкости ... 93
Стимулы для крупных платежей против малых платежей ... 94
Использование блочной цепи в качестве реестра против судебной системы ... 94
Офлайн против онлайн, асинхронность против синхронности ... 94
Сатоши против миллисатоши ... 95
Общие черты сетей Bitcoin и Lightning ... 95
Денежная единица ... 95
Необратимость и окончательность платежей ... 96
Доверие и риск контрагента ... 96
Безразрешительная работа ... 96
Открытый исходный код и открытая система ... 96
Вывод ... 96

Глава 4. Программное обеспечение узла Lightning ... 97

Среда разработки Lightning ... 98
Использование командной строки ... 98
Скачивание репозитория книги ... 99
Docker-контейнеры ... 100
Bitcoin Core и regtest ... 102
Сборка контейнера Bitcoin Core ... 102
Взаимодействие с контейнером bitcoin core ... 103
Проект c-lightning узла Lightning ... 105
Сборка c-lightning в качестве Docker-контейнера ... 105
Настройка сети Docker ... 106
Оперирование контейнерами bitcoind и c-lightning ... 107
Инсталлирование c-lightning из исходного кода ... 108
Инсталлирование необходимых библиотек и пакетов ... 108
Копирование исходного кода c-lightning ... 109
Компилирование исходного кода c-lightning ... 109
Проект демона узла сети Lightning ... 111
Docker-контейнер LND ... 111
Оперирование контейнерами bitcoind и LND ... 112
Инсталлирование LND из исходного кода ... 114
Копирование исходного кода LND ... 115
Компилирование исходного кода LND ... 115

Проект узла Lightning Eclair ... 116
Docker-контейнер Eclair ... 116
Оперирование контейнерами bitcoind и Eclair ... 117
Инсталлирование Eclair из исходного кода ... 118
Копирование исходного кода Eclair ... 119
Компилирование исходного кода Eclair ... 119
Сборка полной сети из разнообразных узлов Lightning ... 120
Использование docker-compose для оркестрирования Docker-контейнеров ... 120
Конфигурация docker-compose ... 121
Запуск образца сети Lightning ... 121
Открытие каналов и маршрутизирование платежа ... 122
Вывод ... 124

Глава 5. Оперирование узлом сети Lightning ... 125

Выбор своей платформы ... 126
Почему для оперирования узлом Lightning важна надежность? ... 126
Типы аппаратных узлов Lightning ... 127
Оперирование в «облаке» ... 127
Оперирование узлом дома ... 128
Какое оборудование требуется для работы узла Lightning? ... 129
Переключение серверной конфигурации в облаке ... 130
Постоянное хранилище данных (накопитель) ... 131
Использование инсталлятора или помощника ... 131
RaspiBlitz ... 131
myNode ... 133
Umbrel ... 133
BTCPay Server ... 134
Узел Bitcoin или облегченный узел Lightning ... 135
Выбор операционной системы ... 136
Выбор имплементации узла Lightning ... 136
Инсталлирование узла Bitcoin или Lightning ... 137
Фоновые службы ... 138
Изоляция процесса ... 138
Запуск узла ... 139
Конфигурирование узла ... 140
Конфигурирование сети ... 141
Это просто работает! ... 142
Автоматическая переадресация портов с использованием UPnP ... 143
Использование Tor для входящих соединений ... 144
Ручная переадресация портов ... 145
Безопасность вашего узла ... 146
Безопасность операционной системы ... 146
Доступ к узлу ... 147
Резервное копирование узла и каналов ... 148
Риск со стороны горячего кошелька ... 150

Зачистка средств ... 150
Внутрицепная зачистка ... 151
Внецепная зачистка ... 151
Зачистка на основе подводного свопа ... 151
Подводные свопы с помощью петли ... 152
Время безотказной работы и доступность узла Lightning ... 153
Допускайте неисправности и автоматизируйте ... 154
Мониторинг доступности узла ... 154
Сторожевые вышки ... 155
Управление каналами ... 156
Открытие исходящих каналов ... 157
Автопилот ... 157
Получение входящей ликвидности ... 160
Закрытие каналов ... 161
Перебалансировка каналов ... 161
Комиссионные за маршрутизацию ... 162
Управление узлом ... 164
Ride The Lightning ... 164
lndmon ... 164
ThunderHub ... 165
Вывод ... 165

ЧАСТЬ II. СЕТЬ LIGHTNING В ДЕТАЛЯХ ... 167

Глава 6. Архитектура сети Lightning ... 168

Комплект протоколов сети Lightning ... 168
Lightning в деталях ... 169

Глава 7. Платежные каналы ... 171

Другой способ использования системы Bitcoin ... 172
Владение биткойном и контроль над ним ... 173
Разнообразие форм (независимого) владения и мультиподпись ... 174
Совместное владение без независимого контроля ... 174
Предотвращение «привязанности» и нерасходуемости биткойна ... 174
Строительство платежного канала ... 175
Приватный и публичный ключи узла ... 175
Сетевой адрес узла ... 175
Идентификаторы узлов ... 176
Соединение узлов в качестве прямых одноранговых участников сети ... 176
Строительство канала ... 177
Одноранговый протокол для управления каналами ... 177
Поток сообщений об установлении канала ... 177
Сообщение open_channel ... 179
Сообщение accept_channel ... 180
Финансовая транзакция ... 181
Генерирование мультиподписного адреса ... 181
Сборка финансовой транзакции ... 181

Удерживание подписанных транзакций без широковещательной передачи ... 182
Возврат средств до финансирования ... 182
Сборка предварительно подписанной возвратной транзакции ... 183
Выстраивание транзакций в цепь без широковещательной передачи ... 183
Решение проблемы деформируемости (сегрегированный свидетель) ... 184
Сообщение funding_created ... 185
Сообщение funding_signed ... 186
Широковещательная передача финансовой транзакции ... 186
Сообщение funding_locked ... 187
Отправка платежей по каналу ... 187
Разделение остатка ... 187
Конкурирующие фиксации ... 188
Обман со старыми фиксационными транзакциями ... 189
Отзыв старых фиксационных транзакций ... 189
Асимметричные фиксационные транзакции ... 190
Задержанное (привязанное ко времени) расходование выхода to_self ... 191
Отзывные ключи ... 192
Фиксационная транзакция ... 193
Продвижение состояния канала вперед ... 195
Сообщение committement_signed ... 196
Сообщение об отзыве и возврате ... 196
Отзыв и рефиксация ... 197
Обман и наказание на практике ... 197
Резерв канала: обеспечение личной заинтересованности ... 200
Закрытие канала (кооперативное закрытие) ... 200
Сообщение shutdown ... 201
Сообщение closing_signed ... 202
Транзакция кооперативного закрытия ... 202
Вывод ... 203

Глава 8. Маршрутизация в сети платежных каналов ... 205

Маршрутизирование платежа ... 205
Маршрутизация против отыскания пути ... 207
Создание сети платежных каналов ... 207
Физический пример «маршрутизирования» ... 208
Протокол справедливости ... 214
Имплементирование атомарных бездоверительных многопереходных платежей ... 214
Возвращаясь к примеру с донатами ... 215
Внутрицепное и внецепное улаживание HTLC-контрактов ... 216
Контракты с привязкой к хешу и времени ... 216
HTLC-контракты на Bitcoin Script ... 217
Платежный прообраз и верификация хеша ... 218
Распространение HTLC-контрактов от Алисы к Дины ... 219
Обратное распространение секрета ... 220
Привязка подписи: предотвращение кражи HTLC-контрактов ... 222

Оптимизация хеша ... 223
Кооперативный отказ и отказ тайм-аута по HTLC-контракту ... 225
Декрементирование привязок ко времени ... 226
Вывод ... 227

Глава 9. Работа канала и пересылка платежей ... 228

Локальный (один) канал против маршрутизируемых (многочисленных) каналов ... 229
Пересылка платежей и обновление фиксаций с помощью HTLC-контрактов ... 229
HTLC-контракт и поток фиксационных сообщений ... 230
Пересылка платежей с помощью HTLC-контрактов ... 230
Добавление HTLC-контракта ... 231
Сообщение update_add_HTLC ... 231
HTLC-контракт в фиксационных транзакциях ... 232
Новая фиксация с выходом из HTLC-контракта ... 233
Алиса фиксирует ... 234
Боб признает новую фиксацию и отзывает старую ... 235
Боб фиксирует ... 238
Несколько HTLC-контрактов ... 239
Исполнение HTLC-контракта ... 240
Распространение HTLC-контракта ... 240
Дина исполняет HTLC-контракт с Чаном ... 240
Боб улаживает HTLC-контракт с Алисой ... 241
Удаление HTLC-контракта из-за ошибки или истечения срока ... 244
Осуществление локального платежа ... 245
Вывод ... 245

Глава 10. Луковичная маршрутизация ... 246

Физический пример, иллюстрирующий луковичную маршрутизацию ... 247
Выбор пути ... 247
Сборка слоев ... 248
Отслаивание слоев ... 250
Введение в луковичную маршрутизацию на основе HTLC-контрактов ... 251
Алиса выбирает путь ... 251
Алиса конструирует полезные грузы ... 253
Полезный груз для Дины в заключительном узле ... 253
Переходный полезный груз для Чана ... 254
Переходный полезный груз для Боба ... 255
Окончательные полезные грузы переходов ... 256
Генерация ключей ... 256
Сеансовый ключ Алисы ... 257
Детали генерации ключей ... 258
Генерация совместных секретов ... 258
Обертывание луковичных слоев ... 260
Луковицы фиксированной длины ... 260
Обертывание луковицы (в общих чертах) ... 261

Обертывание переходного полезного груза Дины 262
Луковично-маршрутизационная защита от повторного воспроизведения и его обнаружение 265
Обертывание переходного полезного груза Чана 266
Обертывание переходного полезного груза Боба 267
Заключительный луковичный пакет 268
Отправка луковицы 269
Сообщение update_add_htlc 269
Алиса отправляет луковицу Бобу 269
Боб проверяет луковицу 270
Боб генерирует заполнитель 270
Боб распутывает свой переходный полезный груз 271
Боб извлекает внешний HMAC для следующего перехода 272
Боб удаляет свой полезный груз и сдвигает луковицу влево 272
Боб конструирует новый луковичный пакет 273
Боб верифицирует детали HTLC-контракта 273
Боб отправляет update_add_htlc Чану 274
Чан пересылает луковицу 274
Дина получает заключительный полезный груз 275
Возвращение ошибок 275
Сообщения о сбоях 276
Застрявшие платежи 278
Спонтанные платежи keysend 279
Конкретно-прикладные луковичные TLV-записи 279
Отправка и получение платежей keysend 280
Платеж keysend и конкретно-прикладные записи в приложениях Lightning 280
Вывод 280

Глава 11. Сплетни и канальный граф 281

Обнаружение одноранговых узлов 283
Самозагрузка P2P-узлов 284
Самозагрузка адресов DNS-серверов 284
Рабочий поток самозагрузки нового одноранговой узла 285
Опции SRV-запроса 288
Канальный граф 289
Ориентированный граф 289
Сообщения эпидемического протокола 290
Сообщение node_announcement 291
Структура сообщения node_announcement 291
Валидация объявлений узла 292
Сообщение channel_announcement 292
Необъявленные (приватные) каналы 293
Локализация канала в блочной цепи Bitcoin 293
Короткий ИД канала 294
Структура сообщения channel_announcement 294
Валидация объявления канала 296
Сообщение channel_update 296

Текущее сопровождение канального графа 297
Вывод 298

Глава 12. Отыскание пути и доставка платежа 299

Отыскание пути в рамках комплекта протоколов Lightning 299
Где же BOLT? 300
Отыскание пути: какую задачу мы решаем? 300
Выбор наилучшего пути 301
Отыскание путей в математике и информатике 302
Емкость, остаток, ликвидность 302
Неопределенность остатков 303
Сложность отыскания пути 304
Без лишних сложностей 304
Отыскание пути и процесс доставки платежа 305
Построение канального графа 305
Неопределенность в канальном графе 308
Неопределенность ликвидности и вероятность 309
Комиссионные и другие метрики канала 310
Отыскание кандидатных путей 312
Доставка платежа (цикл проб и ошибок) 312
Первая попытка (путь №1) 313
Учеба на ошибках 313
Вторая попытка (путь № 4) 313
Учеба на успехах 314
Застоявшиеся знания? 314
Многокомпонентные платежи 314
Использование MPP 315
Разбивка платежей 315
Метод проб и ошибок в течение нескольких «раундов» 316
Вывод 318

Глава 13. Проводной протокол: фреймирование и расширяемость 319

Слой обмена сообщениями в рамках комплекта протоколов Lightning 319
Проводное фреймирование 320
Высокоуровневое фреймирование 320
Кодировка типа 321
Расширения «Тип–длина–значение для сообщений» 322
Протокол буферизует формат сообщения 322
Прямая и обратная совместимости 323
Формат «Тип–длина–значение» 323
Целочисленная кодировка BigSize 324
Ограничения TLV-кодирования 325
Каноническое TLV-кодирование 325
Биты функциональностей и расширяемость протокола 325
Биты функциональностей как механизм обеспечения обнаруживаемости модернизаций 326

TLV для прямой и обратной совместимостей 327
Таксономия механизмов модернизации 328
Модернизации внутренней сети 328
Сквозные модернизации 328
Модернизации уровня строительства канала 329
Вывод 329

Глава 14. Шифрованный транспорт сообщений Lightning 330

Шифрованный транспорт в рамках комплекта протоколов Lightning 330
Введение 330
Канальный граф как децентрализованная инфраструктура публичных ключей 331
Почему не TLS? 332
Каркас криптосвязи на основе протокола Noise 333
Шифрованный транспорт Lightning в деталях 333
Noise_XK: рукопожатие Noise в сети Lightning 333
Нотация рукопожатия и поток протокола 334
Высокоуровневый обзор 334
Рукопожатие в трех действиях 335
Инициализация состояния сеанса рукопожатия 337
Акты рукопожатия 337
Акт первый 338
Акт второй 339
Акт третий 340
Шифрование транспортных сообщений 342
Ротация ключей сообщений Lightning 343
Вывод 343

Глава 15. Платежные запросы Lightning 345

Счета в комплекте протоколов Lightning 345
Введение 345
Платежные запросы Lightning против Bitcoin-адресов 346
BOLT #11: сериализация и интерпретация платежных запросов Lightning 347
Кодирование платежного запроса на практике 347
Человекочитаемый префикс 347
bech32 и сегмент данных 348
Тегированные поля счета 349
Вывод 350

Глава 16. Безопасность и конфиденциальность сети Lightning 351

Почему важна конфиденциальность? 351
Определения конфиденциальности 351
Процесс оценивания конфиденциальности 352
Анонимностное множество 353

Различия между сетями Lightning и Bitcoin с точки зрения конфиденциальности ... 354
Атаки на Lightning ... 356
Наблюдение за суммами платежей ... 356
Связывание отправителей и получателей ... 356
Раскрытие остатков каналов (прощупывание) ... 358
Отказ в обслуживании ... 360
DoS в Bitcoin ... 360
DoS в Lightning ... 361
Известные DoS-атаки ... 361
Заклинивание фиксаций ... 362
Запирание ликвидности канала ... 362
Межслоевая деанонимизация ... 362
Внутрицепная кластеризация Bitcoin-сущностей ... 363
Контрмеры ... 364
Внецепная кластеризация узлов Lightning ... 364
Контрмеры ... 364
Межслоевое связывание: узлы Lightning и Bitcoin-сущности ... 365
Граф Lightning ... 365
Как выглядит граф Lightning в реальности? ... 365
Граф Lightning сегодня ... 366
Атаки на основе топологии ... 366
Темпоральность сети Lightning ... 367
Централизация в сети Lightning ... 368
Экономические стимулы и графовая структура ... 368
Практические советы пользователям по защите их конфиденциальности ... 369
Необъявленные каналы ... 369
Соображения по маршрутизации ... 370
Принятие каналов ... 371
Вывод ... 372
Справочные материалы и дальнейшее чтение ... 372
Конфиденциальность и атаки прощупыванием ... 372
Атаки переполнением ... 372
Соображения по маршрутизации ... 372

Глава 17. Заключение ... 373

Децентрализованные и асинхронные инновации ... 373
Инновации в Bitcoin-протоколе и в Bitcoin Script ... 374
Инновация в протоколе Lightning ... 374
Расширяемость TLV ... 375
Строительство платежного канала ... 375
Сквозные функциональности в порядке выбора ... 375
Lightning-приложения (LApps) ... 376
На старт, внимание, марш! ... 377

Приложение A. Обзор основных принципов системы Bitcoin 378

Ключи и цифровые подписи 378
Приватные и публичные ключи 379
Хеши 380
Цифровые подписи 382
Типы подписей 383
Транзакции Bitcoin 383
Входы и выходы 383
Транзакционные цепочки 385
TxID: идентификаторы транзакций 386
Выходные точки: выходные идентификаторы 387
Bitcoin Script 388
Работа языка Bitcoin Script 388
Привязывающие и отвязывающие скрипты 390
Привязывание к публичному ключу (подписи) 390
Привязывание к хешу (секрету) 391
Мультиподписные скрипты 392
Скрипты привязки ко времени 393
Скрипты с несколькими условиями 394
Использование управления потоком в скриптах 395

Приложение B. Базовая инсталляция и использование Docker 397

Инсталляция Docker 397
Базовые команды Docker 398
Сборка контейнера 398
Оперирование контейнером 398
Исполнение команды в контейнере 398
Остановка и запуск контейнера 398
Удаление контейнера по имени 399
Выведение списка оперируемых контейнеров 399
Выведение списка Docker-образов 399
Вывод 399

Приложение C. Сообщения проводного протокола 400

Типы сообщений 400
Структура сообщения 402
Сообщения об установлении соединения 402
Сообщение init 402
Сообщения об ошибке 403
Сообщение error 403
Оживленность соединения 404
Сообщение ping 404
Сообщение pong 404

Финансирование канала 405
Сообщение open_channel 405
Сообщение accept_channel 406
Сообщение funding_created 406
Сообщение funding_signed 407
Сообщение funding_locked 407
Закрытие канала 407
Сообщение shutdown 408
Сообщение closing_signed 408
Операция канала 408
Сообщение update_add_htlc 408
Сообщение update_fulfill_hltc 409
Сообщение update_fail_htlc 409
Сообщение committement_signed 409
Сообщение revoke_and_ack 410
Сообщение update_fee 410
Сообщение update_fail_malformed_htlc 410
Объявление канала 411
Сообщение channel_announcement 411
Сообщение node_announcement 411
Сообщение channel_update 412
Сообщение announce_signatures 412
Синхронизация канального графа 413
Сообщение query_short_chan_ids 413
Сообщение reply_short_chan_ids_end 413
Сообщение query_channel_range 413
Сообщение reply_channel_range 414
Сообщение gossip_timestamp_range 414

Приложение D. Источники и уведомления о лицензиях 415

Источники 415
Сервер BTCPay Server 416
Lamassu Industries AG 416

Глоссарий 417

Предметный указатель 436

Предисловие

Сеть Lightning (Lightning Network, аббр. LN, или сеть-молния) – это второслойная одноранговая сеть, которая позволяет совершать платежи Bitcoin «вне цепи», то есть без фиксации их в качестве транзакций в блочной цепи (блокчейне) системы Bitcoin.

Сеть Lightning предоставляет безопасные, дешевые, быстрые и гораздо более приватные платежи Bitcoin, даже для очень малых платежей.

Основываясь на идее платежных каналов, впервые предложенной изобретателем системы Bitcoin Сатоши Накамото, сеть Lightning представляет собой маршрутизируемую сеть платежных каналов, в которой платежи делают «прыжки» вдоль пути платежных каналов от отправителя к получателю.

Первоначальная идея сети Lightning была предложена в 2015 году в новаторской статье Джозефа Пуна (Joseph Poon) и Тадеуша Дриджа (Thaddeus Dryja) «Сеть Lightning в рамках системы Bitcoin: масштабируемые мгновенные платежи вне цепи» (The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments). К 2017 году в интернете была запущена «тестовая» сеть Lightning, по мере того разные группы строили совместимые имплементации и координировали работу, чтобы установить какие-нибудь стандарты совместимости. В 2018 году сеть Lightning заработала, и потекли платежи.

В 2019 году Андреас М. Антонопулос, Олаолува Осунтокун и Рене Пикхардт согласились сотрудничать в написании этой книги. И похоже, мы добились успеха!

Целевая аудитория

Эта книга в основном предназначена для технически подкованных читателей, имеющих представление об основах системы Bitcoin и других открытых блочных цепей.

Условные обозначения в книге

В книге используются следующие типографические условные обозначения:

курсивный шрифт
: обозначает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов;

моноширинный шрифт
: используется для листингов программ, а также внутри абзацев для ссылки на элементы программ, такие как переменные или имена функций, базы данных, типы данных, переменные среды, инструкции и ключевые слова;

`жирный моноширинный шрифт`
: показывает команды либо другой текст, который должен быть набран пользователем;

`моноширинный шрифт курсивом`
: показывает текст, который должен быть заменен значениями, передаваемыми пользователем, либо значениями, определяемыми по контексту.

Данный элемент обозначает подсказку или совет.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

Примеры исходного кода

Примеры проиллюстрированы на языках Go, C++, Python и с использованием командной строки Unix-подобной операционной системы. Все фрагменты исходного кода доступны в репозитории на GitHub в подкаталоге *code*. Сделайте ответвление исходного кода книги, попробуйте примеры кода или отправьте исправления через GitHub[1].

Все фрагменты исходного кода могут быть воспроизведены в большинстве операционных систем с минимальной инсталляцией компиляторов, интерпретаторов и библиотек для соответствующих языков.

Там, где это необходимо, мы предоставляем базовые инструкции по инсталляции и пошаговые примеры результата работы этих инструкций.

Некоторые фрагменты исходного кода и результаты его работы были переформатированы для печати. Во всех таких случаях строки были разделены символом обратной косой черты (\), за которым следует символ новой строки. При транскрибировании примеров удалите эти два символа и снова соедините строки, и вы должны увидеть результаты, идентичные тем, которые показаны в примере.

Во всех фрагментах исходного кода, там, где это возможно, используются реально существующие значения и вычисления, так что вы можете переходить

[1] См. https://github.com/lnbook/lnbook.

от примера к примеру и видеть одни и те же результаты в любом программном коде, который вы пишете для вычисления одних и тех же значений. Например, все приватные ключи и соответствующие им публичные ключи и адреса реально существуют.

Использование примеров исходного кода

Если у вас возникли технические вопросы или проблемы с использованием примеров исходного кода, то, пожалуйста, отправьте электронное письмо по адресу bookquestions@oreilly.com.

Ссылки на компании и продукты

Все ссылки на компании и продукты предназначены для образовательных, демонстрационных и справочных целей. Авторы не поддерживают ни одну из упомянутых компаний или продуктов. Мы не тестировали работу или безопасность ни одного из продуктов, проектов или сегментов исходного кода, показанных в этой книге. Используйте их на свой страх и риск!

Адреса и транзакции в этой книге

Адреса, транзакции, ключи, QR-коды и данные блочной цепи Bitcoin, используемые в этой книге, по большей части реальны. Это означает, что вы можете просматривать блочную цепь, просматривать предлагаемые в качестве примеров транзакции, извлекать их с помощью ваших собственных скриптов или программ и т. д.

Однако обратите внимание, что приватные ключи, использованные для создания адресов, напечатанных в этой книге, были «сожжены». Это означает, что если вы отправите деньги на любой из этих адресов, то деньги будут либо потеряны навсегда, либо (что более вероятно) присвоены, поскольку любой, кто читает книгу, может забрать их, используя напечатанные здесь приватные ключи.

НЕ ОТПРАВЛЯЙТЕ ДЕНЬГИ НИ НА ОДИН ИЗ АДРЕСОВ, УКАЗАННЫХ В ЭТОЙ КНИГЕ. Ваши деньги будут взяты другим читателем или потеряны навсегда.

Как с нами связаться

Информация о книге «Освоение сети Lightning», а также открытое издание и переводы доступны по адресу https://lnbook.info.

Связь с Андреасом

С Андреасом М. Антонопулосом можно связаться на его личном веб-сайте: https://aantonop.com.

Подпишитесь на канал Андреаса на YouTube: https://www.youtube.com/aantonop.

Оставьте свой лайк на странице Андреаса в Facebook: https://www.facebook.com/AndreasMAntonopoulos.

Читайте твиты Андреаса в Твиттере: https://twitter.com/aantonop.

Свяжитес с Андреасом в LinkedIn: https://linkedin.com/company/aantonop.

Андреас также хотел бы поблагодарить патронов, которые поддерживают его работу ежемесячными пожертвованиями. Вы можете поддержать Андреаса на Patreon по адресу https://patreon.com/aantonop.

Связь с Рене

С Рене Пикхардтом можно связаться на его личном веб-сайте: https://ln.rene-pickhardt.de.

Подпишитесь на канал Рене на YouTube: https://www.youtube.com/user/RenePickhardt.

Читайте твиты Рене в Твиттере: https://twitter.com/renepickhardt.

Свяжитесь с Рене в LinkedIn: https://www.linkedin.com/in/rene-pickhardt-80313744.

Рене также хотел бы поблагодарить всех патронов, которые поддерживают его работу ежемесячными пожертвованиями. Вы можете поддержать Рене на Patreon по адресу https://patreon.com/renepickhardt.

Или же вы можете поддержать его работу напрямую посредством системы Bitcoin (также через сеть Lightning) по адресу https://donate.ln.rene-pickhardt.de, за что Рене так же благодарит, как и своих патронов.

Связь с Олаолувой Осунтокун

С Олаолувой Осунтокуном можно связаться по его профессиональному электронному адресу: laolu@lightning.engineering.

Читайте твиты Олаолувы в Twitter: https://twitter.com/roasbeef.

Признательности от Андреаса

Своей любовью к словам и книгам я обязан своей матери Терезе, которая вырастила меня в доме, где книги стояли вдоль каждой стены. Моя мать также купила мне мой первый компьютер в 1982 году, несмотря на то что я называл себя технофобом. Мой отец Менелаос, инженер-строитель, опубликовавший свою первую книгу в 80 лет, был именно тем, кто научил меня логическому и аналитическому мышлению, а также любви к науке и технике.

Спасибо вам всем за то, что поддерживали меня на протяжении всего этого путешествия.

Признательности от Рене

Я хочу поблагодарить немецкую систему образования, благодаря которой я приобрел знания, на которых строится моя работа. Это один из величайших

даров, которые мне были даны. Точно так же я хочу поблагодарить немецкую общественную систему здравоохранения и каждого человека, посвящающего свое время работе в этой отрасли. Ваши усилия и выносливость делают вас моими личными героями, и я никогда не забуду помощь, терпение и поддержку, которые я получал, когда нуждался. Выражаю благодарность всем студентам, которым мне разрешили преподавать и которые участвовали в интересных дискуссиях и задавали вопросы. У вас я научился большему. Я очень благодарен сообществу системы Bitcoin и сети Lightning, которое тепло приветствовало меня, а также энтузиастам и частным лицам, которые финансово поддерживали и продолжают поддерживать мою работу. В частности, я благодарен всем разработчикам открытого исходного кода (не только сети Bitcoin и сети Lightning) и людям, которые их финансируют, чтобы сделать эту технологию возможной. Особая благодарность моим соавторам за то, что они прошли со мной через шторм. И последнее, но не менее важное: я благодарен своим близким.

Признательности от Олаолувы Осунтокун

Хотел бы поблагодарить удивительную команду компании Lightning Labs, так как без вас всех не было бы LND. Я также хотел бы поблагодарить авторов оригинальной спецификации BOLT: Расти Рассела, Фабриса Друина, Коннера Фромнкета, Пьера Мари Падиу, Лайзу Нейгут и Кристиана Декера. И последнее, но не менее важное: хотел бы поблагодарить Джозефа Пуна и Тадж Дриджа, авторов оригинальной статьи о сети Lightning, поскольку без них не было бы сети Lightning, о которой можно было бы написать книгу.

Участники проекта

Многие авторы предлагали комментарии, исправления и дополнения к книге, по мере того как она писалась совместно на GitHub.

Ниже приведен отсортированный в английском алфавитном порядке список всех участников GitHub, включая их идентификаторы GitHub в круглых скобках:

- 8go (@8go);
- Акиль Азиз (@batmanscode);
- Александр Гнип (@quantumcthulhu);
- Альфа К. Смит (@alpha_github_id);
- Бен Ски (@benskee);
- Брайан Л. Макмайкл (@brianmcmichael);
- CandleHater (@CandleHater);
- Дэниел Гокель (@dancodery);
- Дапенг Ли (@luislee818);
- Дариус Э. Парвин (@DariusParvin);
- Дору Мунтян (@критон);
- Эдуардо Лима III (@elima-iii);
- Эмилио Норрманн (@enorrmann);
- Франсиско Кальдерон (@grunch);
- Франсиско Рекена (@FrankyFFV);

- Франсуа Дегрос (@fdegros);
- Джованни Зотта (@GiovanniZotta);
- Густаво Сильва (@GustavoRSSilva);
- Гай Таякорн (@saguywalker);
- Хаоюй Линь (@HAOYUatHZ);
- Хатим Буфничель (@boufni95);
- Имран Лоргат (@ImranLorgat);
- Джеффри Макларти (@jnmclarty);
- Джон Дэвис (@tigeryant);
- Жюльен Вендлинг (@trigger67);
- Юсси Тиира (@juhi24);
- Кори Ньютон (@korynewton);
- Лоуренс Уэббер (@lwebbz);
- Луиджи (@джин);
- Максимилиан Караш (@mknoszlig);
- Omega X. Last (@omega_github_id);
- Оуэн Ганден (@ogunden);
- Патрик Лемке (@PatrickLemke);
- Пол Вакероу (@wackerow);
- Рэнди Макмиллан (@RandyMcMillan);
- Рене Кенке (@rene78);
- Рикардо Маркес (@RicardoM17);
- Себастьян Фальбесонер (@Thestack);
- Сергей Тихомиров (@s-tikhomirov);
- Северин Александр Бюлер (@SeverinAlexB);
- Симона Бови (@SimoneBovi);
- Шриджан Бхушан (@srijanb);
- Тейлор Мастерсон (@tjmasterson);
- Умар Болатов (@bolatovumar);
- Уоррен Ван (@wlwanpan);
- Ибинь Чжан (@z4y1b2);
- Закари Хадденхэм (@senf42).

Без предложенной всеми перечисленными выше людьми помощи эта книга была бы невозможна. Ваш вклад демонстрирует силу открытого исходного кода и открытой культуры, и мы бесконечно благодарны вам за помощь.

Спасибо.

Источники

Некоторые материалы в этой книге взяты из различных общедоступных источников, источников с открытой лицензией или с разрешения автора. Подробности об источнике, лицензии и атрибуции см. в приложении D.

Об авторах

Андреас М. Антонопулос – автор бестселлеров, оратор, преподаватель и очень востребованный эксперт по системе Bitcoin и открытых технологий на основе блочных цепей. Он известен тем, что облегчает понимание сложных тем и подчеркивает как положительное, так и отрицательное воздействие, которое эти технологии могут оказывать на наши глобальные общества.

Андреас написал еще два технических бестселлера для программистов с O'Reilly Media, «Освоение системы Bitcoin» (Mastering Bitcoin) и «Освоение системы Ethereum» (Mastering Ethereum). Он также опубликовал серию книг «Интернет денег», в которых основное внимание уделяется социальному, политическому и экономическому значению и последствиям этих технологий. Андреас еженедельно выпускает бесплатный образовательный контент на своем канале YouTube и проводит виртуальные семинары на своем веб-сайте. Узнайте больше по адресу aantonop.com.

Олаолува Осунтокун является соучредителем и техническим директором компании Lightning Labs, а также ведущим разработчиком lnd, одной из главных имплементаций сети Lightning. Он получил степень бакалавра и магистра в области информатики в UCSB и в 2019 году был членом класса Forbes 30 для молодых специалистов до 30 лет. Во время учебы в аспирантуре он сосредоточился на прикладной криптографии, в частности на шифрованном поиске. Более пяти лет он активно участвовал в разработке системы Bitcoin и является автором нескольких предложений по совершенствованию системы Bitcoin (BIP-157 и 158). В наши дни его основное внимание сосредоточено на строительстве, конструировании и разработке частных масштабируемых автономных протоколов блочной цепи, таких как Lightning.

Рене Пикхардт – опытный математик и консультант по науке о данных, который использует свои статистические знания для проведения исследований с NTNU по отысканию путей, конфиденциальности, надежности платежей и соглашениям об уровне обслуживания в сети Lightning. Рене ведет технический и ориентированный на разработчиков канал YouTube[2] о сети Lightning и уже ответил примерно на половину вопросов о работе сети Lightning на бирже Bitcoin, что делает его одним из лучших консультантов для всех новых разработчиков, которые хотят присоединиться к этому пространству. Рене провел множество публичных и частных семинаров о сети Lightning, в том числе обучал студентов резидентуры Chaincode Labs 2019 года вместе с другими ключевыми разработчиками сети Lightning.

[2] См. https://www.youtube.com/renepickhardt.

Об иллюстрации на обложке (Колофон)

Животные на обложке книги «Освоение сети Lightning» – это древесные муравьи (Formica rufa). Термин «древесные муравьи» обычно используется для описания широкой группы муравьев, т. е. тех, которые либо строят муравейники в лесных районах, либо заражают древесину в доме. Однако Formica rufa конкретно относится к муравьям, строящим насыпи из красного дерева, которые в основном встречаются на юге Великобритании, в северной и средней Европе, на Пиренейском хребте и Сибири. Иногда они также встречаются в Северной Америке как в хвойных, так и в широколиственных редколесьях и парках.

Подвид лесных муравьев, также известный как южный лесной муравей, агрессивен, активен и имеет большой размер. Самки древесных муравьев обычно имеют размер 12–15 мм и могут жить до 15 лет. Рабочие муравьи, с другой стороны, чуть меньше, на 8–10 мм, и имеют продолжительность жизни от нескольких недель до семи лет, в зависимости от того, являются ли они самцами или самками (самцы умирают вскоре после спаривания).

Рыжие древесные муравьи способны вырабатывать муравьиную кислоту в своих брюшках и выбрасывать ее на расстояние до нескольких футов, когда им угрожают хищники. Их муравейники обычно представляют собой заметные холмики травы, веток или хвойных игл, часто построенные на трухлявом пне дерева в месте, куда легко проникает солнечный свет. Древесные муравьи живут большими колониями, в которых может быть от 100 000 до 400 000 рабочих и 100 самок. Красные древесные муравьи очень территориальны и, как известно, нападают на другие виды муравьев и удаляют их из этого района.

Рабочие рыжие древесные муравьи добывают корм на расстоянии до 50 метров от своего муравейника, чтобы собирать натуральную смолу, которая капает с сосен. В поведении, уникальном для древесных муравьев, отдельные муравьи ходят по смоле, чтобы обеззараживать себя от бактерий и грибков. Кроме того, они также питаются тлей, медвяной росой, мелкими насекомыми и паукообразными. Рыжие древесные муравьи обычно используются в лесном хозяйстве и часто вводятся в местность в качестве формы борьбы с вредителями.

Рыжие древесные муравьи в настоящее время являются охраняемым видом и классифицируются Международным союзом по охране природы как «находящиеся под угрозой исчезновения». Многие животные на обложках издательства O'Reilly находятся под угрозой исчезновения, и все они важны для мира.

Иллюстрация на обложке выполнена Карен Монтгомери на основе черно-белой гравюры с отдельной пластины неизвестного происхождения.

Часть I

Понимание сети Lightning

Обзор сети Lightning, подходящий для всех, кто заинтересован в понимании базовых концепций и использовании сети Lightning.

Глава 1

Введение

Добро пожаловать в книгу «Освоение сети Lightning»!

Сеть Lightning (часто сокращенно LN от англ. Lightning Network) меняет способ обмена стоимостями в интернете, и это одно из самых захватывающих достижений в истории системы Bitcoin. Сегодня, в 2021 году, сеть Lightning все еще находится в зачаточном состоянии. Сеть Lightning – это протокол для использования системы Bitcoin разумным и неочевидным способом. Это технология, накладываемая вторым слоем поверх системы Bitcoin.

Концепция сети Lightning была предложена в 2015 году, а первая ее имплементация была запущена в 2018 году. Начиная с 2021 года мы только начинаем видеть возможности, которые сеть Lightning предоставляет системе Bitcoin, включая улучшенную конфиденциальность, скорость и масштабирование. Обладая базовыми знаниями о сети Lightning, вы поможете сформировать будущее данной сети, одновременно создавая возможности для себя.

Мы исходим из того, что у вас уже есть некоторые базовые знания о системе Bitcoin, но если нет, то не волнуйтесь – в приложении A мы объясним наиболее важные концепции системы Bitcoin, которые вы должны знать, чтобы понять сеть Lightning. Если вы хотите узнать о системе Bitcoin больше, то можете прочитать книгу Андреаса М. Антонопулоса «Освоение системы Bitcoin», 2-е издание (Andreas M. Antonopoulos, Mastering Bitcoin, 2nd edition, O'Reilly), доступную бесплатно онлайн[3].

Хотя большая часть данной книги написана для программистов, первые несколько глав написаны так, чтобы быть доступными любому, независимо от технического опыта. В этой главе мы начнем с некоторой терминологии, затем перейдем к рассмотрению концепции доверия и ее применения в этих системах и, наконец, обсудим историю и будущее сети Lightning. Давайте начнем.

Базовые понятия сети Lightning

Когда мы займемся разведывательным анализом того, как на самом деле работает сеть Lightning, то столкнемся с некоторой технической терминологией, которая поначалу может немного дезориентировать. Хотя все эти понятия и термины будут подробно объяснены по мере продвижения по книге и определены в глоссарии, ознакомление с несколькими базовыми определениями

[3] См. https://github.com/bitcoinbook/bitcoinbook.

сейчас облегчит понимание концепций в следующих двух главах. Если вы еще не понимаете всех слов в этих определениях, то не беда. Вы будете понимать все больше по мере продвижения по тексту.

Блочная цепь, блокчейн

Распределенный реестр транзакций, создаваемый сетью компьютеров. Bitcoin, например, – это система, которая создает блочную цепь. Сеть Lightning сама по себе не является блочной цепью и не создает блочную цепь. Это сеть, которая опирается на существующую внешнюю блочную цепь для обеспечения своей безопасности.

Цифровая подпись

Цифровая подпись – это математическая схема для верифицирования подлинности цифровых сообщений или документов. Валидная цифровая подпись дает получателю основание полагать, что сообщение было создано известным отправителем, что отправитель не может отрицать отправку сообщения и что сообщение не было изменено при передаче.

Хеш-функция

Криптографическая функция хеширования – это математический алгоритм, который соотносит данные произвольного размера с битовой строкой фиксированного размера (хешем) и предназначен для однопутной функции, то есть функции, которую невозможно инвертировать.

Узел

Компьютер, который участвует в сети. Узел Lightning – это компьютер, который участвует в сети Lightning. Узел Bitcoin – это компьютер, который участвует в сети Bitcoin. Обычно пользователь LN выполняет узел Lightning и узел Bitcoin.

Внутри цепи и вне цепи

Платеж происходит внутри цепи, если он зарегистрирован как транзакция в сети Bitcoin (или другой базовой) блочной цепи. Платежи, отправляемые по платежным каналам между узлами Lightning и которые не видны в базовой блочной цепи, называются платежами вне цепи. Обычно в сети Lightning единственными транзакциями внутри цепи являются транзакции, используемые для открытия и закрытия платежного канала Lightning. Существует третий тип транзакции, модифицирующий канал, именуемый склеиванием (splicing), который можно использовать для добавления/удаления суммы средств, зафиксированных в канале.

Платеж

Когда стоимость обменивается в сети Lightning, мы называем это «платежом» по сравнению с «транзакцией» в блочной цепи Bitcoin.

Платежный канал

Финансовая взаимосвязь между двумя узлами в сети Lightning, обычно имплементируемая с помощью мультиподписных Bitcoin-транзакций, которые имеют совместный контроль над биткойном между двумя узлами Lightning.

Маршрутизация по сравнению с отправкой

В отличие от системы Bitcoin, где транзакции «отправляются» путем их широковещательной передачи всем, Lightning – это маршрутизированная сеть, в которой платежи «маршрутизируются» по одному или нескольким платежным каналам по пути от отправителя к получателю.

Транзакция

Структура данных, которая регистрирует передачу контроля над некоторыми средствами (например, некоторыми биткойнами). Сеть Lightning опирается на транзакции Bitcoin (или транзакции другой блочной цепи) для осуществления контроля над средствами.

Более подробные определения этих и многих других терминов можно найти в глоссарии. На протяжении всей книги мы будем объяснять смысл этих терминов и то, как на самом деле работают данные технологии.

На протяжении всей этой книги вы будете встречать слово «Bitcoin» с заглавной первой буквой, которое относится к системе Bitcoin и является именем собственным. Вы также будете встречать слово «биткойн» со строчной буквой «б», которое относится к денежной единице. Каждый биткойн далее подразделяется на 100 миллионов единиц, каждая из которых называется «сатоши» (в единственном и множественном числе на русском пишется одинаково).

Теперь, когда вы знакомы с этими базовыми терминами, давайте перейдем к понятию, которое вам уже знакомо: доверие (trust).

Доверие в децентрализованных сетях

Вы часто будете слышать, как люди называют систему Bitcoin и сеть Lightning «бездоверительными» (trustless). На первый взгляд это сбивает с толку. В конце концов, разве доверие – это не хорошо? Банки даже используют его в своих названиях! Разве «бездоверительная» система, система, лишенная доверия, не является чем-то плохим?

Использование слова «бездоверительный» предназначено для того, чтобы передать идею способности работать, не нуждаясь в доверии к другим участникам системы. В такой децентрализованной системе, как Bitcoin, вы всегда можете заключить сделку с тем, кому доверяете. Однако система также гарантирует, что вас не обманут, даже если вы не можете доверять другой стороне транзакции. Доверие – это не обязательное, а приятное свойство системы.

Сравните это с традиционными системами, такими как банковское обслуживание, где вы должны доверять третьей стороне, поскольку она контролирует ваши деньги. Если банк нарушит ваше доверие, то вы можете обратиться в регулирующий орган или суд, но это потребует огромных затрат времени, денег и усилий.

Бездоверительность не означает отсутствие доверия. Она означает, что доверие не является необходимым условием для всех транзакций и что вы можете совершать транзакции даже с людьми, которым вы не доверяете, потому что система предотвращает обман.

Прежде чем мы перейдем к тому, как работает сеть Lightning, важно понять одну базовую концепцию, которая лежит в основе системы Bitcoin, сети Lightning и многих других подобных систем: то, что мы называем протоколом справедливости. Протокол справедливости – это способ достижения справедливых исходов между участниками, которым не нужно доверять друг другу, без необходимости в центральном органе власти, и он является основой децентрализованных систем, таких как Bitcoin.

Справедливость без центральной власти

Когда у людей есть конкурирующие интересы, то как они могут установить достаточное доверие, чтобы участвовать в каком-то совместном или транзакционном поведении? Ответ на этот вопрос лежит в основе нескольких научных и гуманистических дисциплин, таких как экономика, социология, поведенческая психология и математика. Некоторые из этих дисциплин дают нам «мягкие» ответы, которые зависят от таких понятий, как репутация, справедливость, мораль и даже религия. Другие дисциплины дают нам конкретные ответы, которые зависят только от допущения о том, что участники этих взаимодействий будут действовать рационально, руководствуясь своими личными интересами в качестве главной цели.

В общих чертах, существует несколько способов обеспечивать справедливые исходы во взаимодействии между людьми, которые могут иметь конкурирующие интересы:

Требовать доверия

Вы взаимодействуете только с теми людьми, которым уже доверяете, благодаря предыдущим взаимодействиям, репутации или семейным отношениям. Это достаточно хорошо работает в малых масштабах, в особенности в семьях и небольших группах, что является наиболее распространенной основой для кооперативного поведения. К сожалению, это не масштабируется и страдает от трайбалистского предубеждения (внутри группы).

Верховенство закона

Установить правила взаимодействия, которые будут соблюдаться учреждением. Это масштабируется лучше, но не может масштабироваться глобально из-за различий в обычаях и традициях, а также неспособности масштабировать институты правоприменения. Одним из неприятных побочных эффектов этого решения является то, что по мере своего роста институты становятся все более и более мощными, а это может привести к коррупции.

Доверенные третьи стороны

Поставить посредника в каждом взаимодействии, чтобы обеспечивать справедливость. В сочетании с «верховенством закона», обеспечивающим надзор за посредниками, это лучше масштабируется, но страдает от того же дисбаланса власти: посредники становятся очень влиятельными и могут привлечь коррупцию. Концентрация власти приводит к системному риску и системному провалу («слишком большой, чтобы позволить ему обанкротиться»).

Теоретико-игровые протоколы справедливости

Эта последняя категория возникает в результате сочетания интернета и криптографии и является предметом данного раздела. Давайте посмотрим, как она работает и в чем ее преимущества и недостатки.

Доверительные протоколы без посредников

Криптографические системы, такие как Bitcoin и сеть Lightning, – это системы, позволяющие совершать транзакции с людьми (и компьютерами), которым вы не доверяете. Такую работу часто называют «бездоверительной», хотя на самом деле она не является бездоверительной. Вы должны доверять выполняемому программному обеспечению и должны верить, что протокол, имплементированный этой программой, приведет к справедливым исходам.

Большое различие между подобного рода криптографической системой и традиционной финансовой системой заключается в том, что в традиционных финансах у вас есть доверенная третья сторона, например банк, для обеспечения справедливости исходов. Существенная проблема с такими системами заключается в том, что они передают слишком много власти третьей стороне, а также уязвимы перед единой точкой отказа. Если доверенная третья сторона сама нарушает доверие или пытается обмануть, то основание для доверия нарушается.

Изучая криптографические системы, вы заметите определенную закономерность: вместо того чтобы полагаться на доверенную третью сторону, эти системы пытаются предотвратить несправедливые исходы, используя систему положительных и отрицательных стимулов. В криптографических системах вы доверяете протоколу, фактически представляющему собой систему с набором правил, которые при правильной разработке будут правильно применять желаемые положительные и отрицательные стимулы. Преимущество такого подхода двоякое: вы не только избегаете доверия третьей стороне, но и уменьшаете необходимость обеспечения справедливых исходов. До тех пор, пока участники подчиняются согласованному протоколу и остаются в рамках системы, механизм стимулирования в этом протоколе обеспечивает справедливые исходы без контроля за его исполнением.

Использование положительных и отрицательных стимулов для достижения справедливых исходов является одним из аспектов раздела математики, именуемого теорией игр, предметом которого является изучение «моделей стратегического взаимодействия между лицами, принимающими рациональные решения»[4]. Криптографические системы, которые контролируют финансовые взаимодействия между участниками, такие как Bitcoin и сеть Lightning, в значительной степени опираются на теорию игр, чтобы предотвращать обман участников и позволять участникам, которые не доверяют друг другу, добиваться справедливых исходов.

Хотя теория игр и ее использование в криптографических системах на первый взгляд могут показаться запутанными и незнакомыми, скорее всего, вы уже знакомы с этими системами в своей повседневной жизни; вы просто еще

[4] Статья в Википедии (https://en.wikipedia.org/wiki/Game_theory), посвященная теории игр, содержит дополнительную информацию.

не узнаете их. В следующем далее разделе мы задействуем простой пример из детства, который поможет определить базовую закономерность. Как только вы поймете базовую закономерность, вы увидите ее повсюду в пространстве блочной цепи и научитесь распознавать ее быстро и интуитивно.

В этой книге мы называем эту закономерность протоколом справедливости, определяемым как процесс, в котором используется система положительных и/или отрицательных стимулов в целях обеспечения справедливых исходов для участников, которые не доверяют друг другу. Соблюдение протокола справедливости необходимо только для того, чтобы участники не могли избежать положительных или отрицательных стимулов.

Протокол справедливости в действии

Давайте рассмотрим пример протокола справедливости, с которым вы, возможно, уже знакомы.

Представьте себе семейный обед с родителем и двумя детьми. Дети привередливы в еде, и единственное, что они согласятся съесть, – это жареную картошку. Родитель приготовил чашу жареного картофеля («картофель фри» или «чипсы», в зависимости от того, какой английский диалект вы используете). Брат и сестра должны разделить между собой тарелку с чипсами. Родитель должен обеспечить справедливое распределение чипсов между каждым ребенком; в противном случае родителю придется выслушивать постоянные жалобы (возможно, весь день), и всегда есть вероятность того, что несправедливая ситуация перерастет в насилие. Что должен делать родитель?

В этом стратегическом взаимодействии между двумя детьми, которые не доверяют друг другу и имеют конкурирующие интересы, существует несколько разных способов достижения справедливости. Наивный, но часто используемый метод заключается в том, что родители используют свою власть в качестве доверенной третьей стороны: они делят миску с чипсами на две порции. Это похоже на традиционные финансы, где банк, бухгалтер или юрист выступают в качестве доверенной третьей стороны, чтобы предотвратить любой обман между двумя сторонами, которые хотят совершить сделку.

Проблема этого сценария заключается в том, что он возлагает большую власть и ответственность на доверенную третью сторону. В этом примере родитель несет полную ответственность за равное распределение чипсов, а стороны просто ждут, наблюдают и жалуются. Дети обвиняют родителей в том, что они занимают сторону любимчика и несправедливо распределяют чипсы. Они дерутся из-за чипсов, крича, что «этот чипс больше!», втягивая родителя в свою перепалку. Это звучит ужасно, не так ли? Должен ли родитель кричать громче? Убрать все чипсы? Угрожать никогда больше не делать чипсы и отправить детей из-за стола голодными?

Существует гораздо более оптимальное решение: детей учат играть в игру под названием «дели и выбирай». За каждым обедом один ребенок делит чашу чипсов на две порции, а *другой* выбирает ту порцию, которую он хочет. Почти сразу же дети начнут понимать динамику этой игры. Если один из них совершает ошибку или пытается обмануть, то другой может его «оштрафовать», выбрав чашу побольше. Играть честно – в интересах обоих детей, но в особенности того, кто делит чашу. В этом сценарии проигрывает только обманщик.

Родителю даже не нужно использовать свою власть или обеспечивать справедливость. Родителю лишь нужно *обеспечивать соблюдение протокола*; до тех пор, пока дети не смогут избежать назначенных им ролей «делющего» и «выбирающего», протокол сам по себе обеспечивает справедливый результат без необходимости какого-либо вмешательства. Родитель не может занимать сторону любимчика или искажать исход.

Хотя печально известные битвы за чипы 1980-х годов четко иллюстрируют эту точку зрения, любое сходство между описанным выше сценарием и реальным детским опытом любого из авторов со своими двоюродными братьями совершенно случаен … или нет?

Примитивы безопасности как строительные блоки

Для того чтобы подобный протокол справедливости работал, должны существовать определенные гарантии, или *примитивы безопасности*, которые можно комбинировать для обеспечения соблюдения. Первый примитив безопасности – это строгая *временная упорядоченность/последовательность*: действие «деление» должно происходить до действия «выбор». Это не сразу очевидно, но если вы не сможете гарантировать, что действие A произойдет до действия B, тогда протокол развалится. Второй элемент безопасности – это *обязательство без возможности отказа*. Каждый ребенок должен определиться со своим выбором роли: либо делющий, либо выбирающий. Кроме того, как только деление завершено, делющий привязан к созданному им делению – он не может отказаться от этого выбора и повторить попытку.

Криптографические системы предлагают ряд примитивов безопасности, которые могут объединяться разными способами для строительства протокола справедливости. В дополнение к упорядоченности и обязательству мы также можем использовать целый ряд других инструментов:

- хеш-функции для генерирования отпечатков данных как форма обязательства или как основа для цифровой подписи;
- цифровые подписи для аутентификации, неразглашения и подтверждения права собственности на секрет;
- шифрование/дешифрование для ограничения доступа к информации только авторизованным участникам.

Это лишь небольшой список целого «зверинца» используемых средств обеспечения безопасности и криптографии. Все время изобретаются более простые примитивы и комбинации.

В нашем примере из реальной жизни мы увидели одну из форм протокола справедливости под названием «дели и выбирай». Это всего лишь один из множества различных протоколов справедливости, которые могут быть построены путем комбинирования строительных блоков примитивов безопасности различными способами. Но базовая закономерность всегда одна и та же: два или более участника взаимодействуют, не доверяя друг другу, выполняя ряд шагов, которые являются частью согласованного протокола. Шаги протокола устанавливают положительные и отрицательные стимулы для обеспечения

того, чтобы, если участники рациональны, обман был контрпродуктивным, а справедливость – автоматическим исходом.

Контроль за соблюдением не является необходимым для получения справедливых исходов – он необходим только для того, чтобы участники не нарушали согласованный протокол.

Теперь, когда вы понимаете эту базовую закономерность, вы начнете видеть его повсюду в системе Bitcoin, сети Lightning и многих других системах. Давайте рассмотрим несколько конкретных примеров далее.

Пример протокола справедливости

Наиболее ярким примером протокола справедливости является консенсусный алгоритм системы Bitcoin под названием *Доказательство работы* (Proof of Work, аббр. PoW). В системе Bitcoin майнеры соревнуются за верификацию транзакций и их агрегирование в блоки. В целях обеспечения того, чтобы майнеры не обманывали без наделения их полномочиями, в Bitcoin используется подсистема положительных и отрицательных стимулов. Майнеры должны использовать электричество и выделять оборудование для выполнения «работы», которая встроена в качестве «доказательства» внутрь каждого блока. Это достигается благодаря свойству хеш-функций, при котором выходное значение случайно распределяется по всему диапазону возможных выходов. Если майнерам удается произвести валидный блок достаточно быстро, то они получают вознаграждение, зарабатывая блочное вознаграждение за этот блок. Принуждение майнеров использовать много электроэнергии до того, как сеть рассмотрит их блок, означает, что у них есть стимул правильно проверять транзакции в блоке. Если они обманывают или совершают какую-либо ошибку, то их блок отклоняется, и электричество, которое они использовали, чтобы «доказать» это, тратится впустую. Никому не нужно заставлять майнеров производить валидные блоки; вознаграждение и наказание стимулируют их к этому. Протоколу лишь нужно обеспечить, чтобы принимались только валидные блоки с доказательством работы.

Закономерность протокола справедливости также можно найти во многих разных аспектах сети Lightning:

- те, кто финансирует каналы, обеспечивают, чтобы у них была подписана транзакция возврата средств, прежде чем публиковать финансовую транзакцию;
- всякий раз, когда канал переводится в новое состояние, старое состояние «отзывается», обеспечивая, что если кто-либо попытается выполнить его широковещательную передачу, то он потеряет весь остаток средств и будет оштрафован;
- те, кто пересылают платежи, знают, что если они подтверждают/фиксируют пересылку средств вперед, то могут либо получить возврат, либо получить оплату от узла, предшествующего им.

Снова и снова мы видим эту закономерность. Соблюдение справедливых исходов не обеспечивается никакими органами власти. Они возникают как естественное следствие протокола, который поощряет справедливость и штрафует обман, протокола справедливости, в котором задействуются личные интересы, направляя их на справедливые исходы.

Система Bitcoin и сеть Lightning являются имплементациями протоколов справедливости. Тогда зачем нужна сеть Lightning? Разве системы Bitcoin недостаточно?

Мотивация для сети Lightning

Bitcoin – это система, которая регистрирует транзакции в глобально реплицируемом публичном реестре. Каждая транзакция видна, валидируется и хранится каждым участвующим компьютером. Как нетрудно себе представить, это генерирует большой объем данных, и его трудно масштабировать.

По мере роста системы Bitcoin и спроса на транзакции число транзакций в каждом блоке увеличивается, пока в конечном итоге не достигает предельного размера блока. Как только блоки «заполняются», избыточные транзакции остаются ждать в очереди. Многие пользователи будут увеличивать сумму комиссионных, которые они готовы платить, чтобы купить место для своих транзакций в следующем блоке.

Если спрос продолжает опережать емкость сети, то все большее число транзакций пользователей остаются неподтвержденными. Конкуренция за комиссионные также увеличивает стоимость каждой транзакции, делая многие транзакции с меньшей стоимостью (например, микротранзакции) совершенно неэкономичными в периоды особенно высокого спроса.

В целях решения этой проблемы мы могли бы увеличить лимит на размер блока, чтобы создать пространство для большего числа транзакций. Увеличение в «предложении» блочного пространства приведет к более низкому ценовому равновесию для транзакционных комиссий.

Однако увеличение размера блока перекладывает затраты на операторов узлов и требует, чтобы они тратили больше ресурсов на валидирование и хранение блочной цепи. Поскольку блочные цепи являются эпидемическими протоколами обмена сообщениями, каждый узел должен знать и валидировать каждую отдельную транзакцию, которая происходит в сети. Более того, после валидации каждая транзакция и блок должны быть распространены на «соседей» узла, умножая потребность в емкости. Таким образом, чем больше размер блока, тем выше потребность в емкости, обработке и хранении каждого отдельного узла. Увеличение емкости транзакций в таком ключе приводит к нежелательному эффекту централизации системы за счет сокращения числа узлов и операторов узлов. Поскольку операторы узлов не получают компенсации за работу узлов, если работа узлов обходится очень дорого, то только несколько хорошо финансируемых операторов узлов будут продолжать выполнять узлы.

Масштабирование блочных цепей

Как показывают несколько расчетов, побочные эффекты увеличения размера блока или уменьшения времени блока по отношению к централизации сети являются серьезными.

Предположим, что использование системы Bitcoin растет настолько, что сеть должна обрабатывать 40 000 транзакций в секунду, что является приблизительным уровнем транзакционной обработки сети Visa во время пиковой используемости.

Исходя из того, что на транзакцию в среднем приходится 250 байт, это приведет к потоку данных 10 мегабайт в секунду (МБ/с) или 80 мегабит в секунду (Мбит/с) только для того, чтобы иметь возможность получать все транзакции. Сюда не входят накладные расходы на трафик, связанные с пересылкой информации о транзакции другим одноранговым узлам. Хотя 10 МБ/с и не кажутся экстремальными в контексте высокоскоростных оптоволоконных и мобильных скоростей 5G, это фактически исключило бы любого, кто не может удовлетворить данное требование, из работающего узла, в особенности в странах, где высокопроизводительный интернет недоступен или недоступен широко.

У пользователей также есть много других требований к пропускной способности, и нельзя ожидать, что они будут тратить столько лишь на получение транзакций.

Более того, хранение этой информации локально привело бы к 864 гигабайтам в день. Это примерно один терабайт данных, или размер жесткого диска.

Верифицирование 40 000 подписей алгоритма цифровой подписи на основе эллиптической кривой (Elliptic Curve Digital Signature Algorithm, аббр. ECDSA) в секунду также едва выполнимо (см. соответствущу статью на StackExchange[5]), что делает скачивание начального блока (initial block download, аббр. IBD) блочной цепи Bitcoin (синхронизирование и верифицирование всего, начиная с генезисного (первичного) блока) практически невозможным без очень дорогого оборудования.

В то время как 40 000 транзакций в секунду кажутся большими, они достигают паритета с традиционными финансовыми платежными сетями только в пиковые моменты. Инновации в межмашинных платежах, микротранзакциях и других приложениях, скорее всего, повысят спрос на много порядков.

Проще говоря: масштабировать блочную цепь для валидирования транзакций по всему миру децентрализованным способом невозможно.

Но что, если бы каждый узел не был бы обязан знать и проверять каждую отдельную транзакцию? Что, если бы существовал способ удерживать масштабируемые транзакции вне цепи без потери безопасности сети Bitcoin?

В феврале 2015 года Джозеф Пун (Joseph Poon) и Тадеуш Дриджа (Thaddeus Dryja) предложили возможное решение проблемы масштабируемости системы Bitcoin, опубликовав работу под названием «Сеть Lightning в рамках системы Bitcoin: масштабируемые мгновенные платежи вне цепи» (The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments)[6].

В (ныне устаревшем) техническом документе Пун и Дриджа подсчитали, что для того, чтобы система Bitcoin достигла 47 000 транзакций в секунду, обрабатываемых системой Visa на пике, потребуется 8 ГБ блоков. Это сделало бы опе-

[5] См. https://bitcoin.stackexchange.com/questions/95339/how-many-bitcoin-transactions-can-be verified-per-second.

[6] *Джозеф Пун и Тадеуш Дриджа.* Сеть Lightning в рамках системы Bitcoin: масштабируемые мгновенные платежи вне цепи. ЧЕРНОВАЯ версия 0.5.9.2. 14 января 2016 года (Joseph Poon and Thaddeus Dryja. “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”. DRAFT Version 0.5.9.2. January 14, 2016). https://lightning.network/lightning-network-paper.pdf.

рирование узлом совершенно неприемлемым для кого бы то ни было, кроме крупных предприятий и операций промышленного уровня. Результатом стала бы сеть, в которой только несколько пользователей могли бы фактически валидировать состояние реестра. Система Bitcoin опирается на то, что пользователи сами валидируют реестр, не доверяя третьим лицам в явной форме, чтобы оставаться децентрализованной. Взимание комиссии с пользователей за оперирование узлами вынудило бы среднестатистического пользователя доверять третьим сторонам, чтобы узнавать состояние реестра, что в конечном итоге нарушило бы доверительную модель в системе Bitcoin.

Сеть Lightning предлагает новую сеть, второй слой, где пользователи могут осуществлять платежи друг другу в одноранговом режиме без необходимости публикации транзакции в блочной цепи системы Bitcoin для каждого платежа. Пользователи могут платить друг другу в сети Lightning столько раз, сколько захотят, без создания дополнительных транзакций Bitcoin или взимания комиссии внутри цепи. Они используют блочную цепь системы Bitcoin только для первоначальной загрузки биткойна в сеть Lightning и для расчетов, то есть для удаления биткойна из сети Lightning. Как следствие гораздо больше платежей Bitcoin может происходить вне цепи, при этом только транзакции начальной загрузки и окончательного расчета должны валидироваться и храниться узлами Bitcoin. Помимо снижения нагрузки на узлы, платежи в сети Lightning дешевле для пользователей, потому что им не нужно платить комиссию за блочную цепь, и более приватны для пользователей, потому что они не публикуются для всех участников сети и, кроме того, не хранятся постоянно.

Хотя сеть Lightning изначально была задумана для системы Bitcoin, она может быть имплементирована в любой блочной цепи, которая отвечает некоторым базовым техническим требованиям. Другие блочные цепи, такие как Litecoin, уже поддерживают сеть Lightning. Кроме того, несколько других блочных цепей разрабатывают аналогичные второслойные решения, чтобы помочь им масштабироваться.

Определяющие признаки сети Lightning

Сеть Lightning – это сеть, которая работает как протокол второго слоя поверх системы Bitcoin и других блочных цепей. Сеть Lightning обеспечивает быстрые, безопасные, приватные, бездоверительные и не требующие разрешения платежи. Вот несколько признаков сети Lightning:

- пользователи сети Lightning могут маршрутизировать платежи друг другу по низкой цене и в реальном времени;
- пользователям, которые обмениваются стоимостями через сеть Lightning, не нужно ждать подтверждения блока для платежей;
- как только платеж в сети Lightning завершен, обычно в течение нескольких секунд он является окончательным и не может быть отменен. Как и транзакция в системе Bitcoin, платеж в сети Lightning может быть возвращен только получателем;
- в отличие от внутрицепных транзакций Bitcoin, которые передаются широковещательно и проверяются всеми узлами сети, платежи, маршрути-

зируемые в сети Lightning, передаются между парами узлов и не видны всем, что обеспечивает гораздо большую конфиденциальность;
- в отличие от транзакций в сети Bitcoin, платежи, маршрутизируемые в сети Lightning, не нуждаются в постоянном хранении. Таким образом, Lightning потребляет меньше ресурсов и, следовательно, дешевле. Это свойство также имеет преимущества для конфиденциальности;
- сеть Lightning использует луковичную маршрутизацию, аналогичную протоколу, используемому сетью Tor (от англ. The Onion Router, луковичный маршрутизатор), работающей на основе технологии обеспечения конфиденциальности, в результате чего даже участвующие в маршрутизации платежа узлы напрямую знают только о своем предшественнике и преемнике в маршруте платежа;
- при использовании поверх системы Bitcoin сеть Lightning использует настоящий биткойн, который всегда находится во владении (хранении) и полностью контролируется пользователем. Lightning – это не отдельный токен или монета, это действительно биткойн.

Примеры использования сети Lightning, пользователи и их истории

В целях более глубокого понимания того, как на самом деле работает сеть Lightning и почему люди ее используют, мы проследим за рядом пользователей и их историями.

В наших примерах некоторые люди уже систему Bitcoin использовали, а другие в ней абсолютные новички. Каждый человек и его приведенная ниже история иллюстрируют один или несколько конкретных вариантов применения. Мы будем возвращаться к ним на протяжении всей этой книги:

Потребитель

Алиса – пользователь системы Bitcoin, который хочет совершать быстрые, безопасные, дешевые и приватные платежи за небольшие розничные покупки. Она покупает кофе за биткойны, используя сеть Lightning.

Торговец

Боб владеет кафе «Кофейня Боба». Внутрицепные платежи Bitcoin не масштабируются для малых сумм, таких как чашка кофе, поэтому он использует сеть Lightning для приема платежей Bitcoin почти мгновенно и за низкие комиссионные.

Бизнес по предоставлению программно-информационных услуг

Чан – китайский предприниматель, который продает информационные услуги, связанные с сетью Lightning, а также системой Bitcoin и другими криптовалютами. Чан продает эти информационные услуги через интернет, осуществляя микроплатежи через сеть Lightning. Кроме того, Чан имплементировал службу поставщика ликвидности, которая арендует емкость входящего канала в сети Lightning, взимая небольшие комиссионные в биткойне за каждый период аренды.

Игрок

Дина – геймер-подросток из России. Она играет во много разных компьютерных игр, но ее любимые – это те, в которых есть «внутриигровая экономика», основанная на реальных деньгах. Играя в игры, она также зарабатывает деньги, приобретая и продавая виртуальные внутриигровые предметы. Сеть Lightning позволяет ей совершать небольшие сделки по внутриигровым предметам, а также зарабатывать небольшие суммы за выполнение квестов.

Вывод

В этой главе мы поговорили о фундаментальной концепции, которая лежит в основе как системы Bitcoin, так и сети Lightning: протоколе справедливости.

Мы рассмотрели историю сети Lightning и мотивы, лежащие в основе технических решений для масштабирования на втором слое для системы Bitcoin и других сетей, основанных на блочной цепи (блокчейне).

Мы изучили базовую терминологию, включая узел, платежный канал, транзакции внутри цепи и платежи вне цепи.

Наконец, мы познакомились с Алисой, Бобом, Чаном и Диной, за которыми будем следить на протяжении всей остальной части книги. В следующей главе мы встретимся с Алисой и рассмотрим ее мыслительный процесс, когда она выбирает кошелек Lightning и готовится совершить свой первый платеж Lightning, чтобы купить чашку кофе в кофейне Боба.

Глава 2

Приступаем к работе

В этой главе мы начнем с того, с чего начинает большинство людей, впервые сталкиваясь с сетью Lightning, – с выбора программного обеспечения для участия в экономике LN. Мы рассмотрим варианты выбора двух пользователей, которые представляют распространенный вариант использования сети Lightning, и будем учиться на примере. Алиса, посетительница кофейни, будет использовать кошелек Lightning на своем мобильном устройстве, чтобы купить кофе в кофейне Боба. Боб, торговец, будет использовать узел Lightning и кошелек для работы системы кассовых расчетов в своей кофейне, чтобы иметь возможность принимать платежи через сеть Lightning.

Первый кошелек Lightning Алисы

Алиса – давний пользователь системы Bitcoin. Впервые мы встретились с Алисой в главе 1 книги «Освоение системы Bitcoin»[7], когда она купила чашку кофе в кофейне Боба, используя транзакцию Bitcoin. Если вы еще незнакомы с тем, как работают транзакции Bitcoin, или нуждаетесь в освежении своих знаний, то, пожалуйста, ознакомьтесь с книгой «Освоение системы Bitcoin» или кратким описанием в приложении А.

Алиса узнала, что кофейня Боба недавно начала принимать платежи LN! Алисе не терпится узнать о сети Lightning и поэкспериментировать с ней; она хочет стать одним из первых клиентов LN Боба. Для этого Алиса сначала должна выбрать кошелек Lightning, который соответствует ее потребностям.

Алиса не хочет доверять хранение своего биткойна третьим лицам. Она узнала о криптовалюте уже достаточно, чтобы знать, как пользоваться кошельком. Ей также нужен мобильный кошелек, чтобы она могла использовать его для небольших платежей на ходу, поэтому она выбирает кошелек Eclair, популярный неопекаемый (некустодиальный)[8] мобильный кошелек Lightning. Давайте узнаем больше о том, как и почему она сделала этот выбор.

---

[7] *Андреас М. Антонопулос.* Освоение системы Bitcoin. 2-е изд., глава 1 (Andreas M. Antonopoulos, Mastering Bitcoin, 2nd Edition, Chapter 1, O'Reilly, https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch01.asciidoc).

[8] Неопекаемым, или некастодиальным (noncustodial), можно считать криптовалютный кошелек, сохраняющий за пользователем возможность полностью контролировать ключи и свои средства. К данной категории можно отнести аппаратные, мобильные, бумажные, настольные и веб-кошельки. – *Прим. перев.*

Узлы Lightning

Доступ к сети Lightning осуществляется посредством программных приложений, которые могут обмениваться данными по протоколу LN. Узел сети Lightning (узел LN или просто узел) – это программное приложение, обладающее тремя важными характеристиками. Во-первых, узлы Lightning являются кошельками, поэтому они отправляют и получают платежи через сеть Lightning, а также в сети Bitcoin. Во-вторых, узлы должны взаимодействовать на одноранговой основе с другими узлами Lightning, создавая сеть. Наконец, узлам Lightning также необходим доступ к блочной цепи Bitcoin (или другим блочным цепям для других криптовалют) с целью защиты средств, используемых для платежей.

Пользователи имеют наивысшую степень контроля, управляя своим собственным узлом Bitcoin и узлом Lightning. Однако узлы Lightning также могут использовать облегченный клиент Bitcoin, обычно именуемый упрощенной верификацией платежей (simplified payment verification, аббр. SPV), в целях взаимодействия с блочной цепью Bitcoin.

Проводники Lightning

Проводники LN – это полезные инструменты для показа статистики узлов, каналов и емкости сети.

Ниже приведен неисчерпывающий список:

- проводник 1ML[9] по сети Lightning;
- проводник ACINQ[10] по сети Lightning с причудливой визуализацией;
- проводник Amboss Space[11] по сети Lightning с метриками сообщества и интуитивно понятными визуализациями;
- проводник Fiatjaf[12] по сети Lightning с многочисленными диаграммами;
- проводник hashXP[13] по сети Lightning.

[9] См. https://1ml.com/.

[10] См. https://explorer.acinq.co/.

[11] См. https://amboss.space/.

[12] См. https://ln.bigsun.xyz/.

[13] См. https://hashxp.org/lightning/node.

Обратите внимание, что при использовании проводников Lightning, как и в случае с другими проводниками по блокам, конфиденциальность может быть проблемой. Если пользователи проявляют небрежность, то веб-сайт может отслеживать их IP-адреса и собирать данные об их поведении (например, интересующие пользователей узлы).

Также следует отметить, что поскольку нет глобального консенсуса относительно текущего графа Lightning или текущего состояния какой-либо существующей политики канала, пользователям никогда не следует полагаться на проводников по сети Lightning для получения самой актуальной информации. Кроме того, по мере того как пользователи открывают, закрывают и обновляют каналы, граф будет меняться, и отдельные проводники по сети Lightning могут быть не в курсе последних событий. Используйте проводники Lightning для визуализации сети или сбора информации, но не в качестве авторитетного источника того, что происходит в сети Lightning. Для того чтобы иметь авторитетное представление о сети Lightning, запустите свой собственный узел Lightning. Он построит граф каналов и соберет различную статистику, которую вы сможете просматривать с помощью веб-интерфейса.

Кошельки Lightning

Термин «*кошелек Lightning*» несколько двусмыслен, поскольку он может описывать широкий спектр компонентов в сочетании с некоторым пользовательским интерфейсом. Наиболее распространенные компоненты программного обеспечения кошелька Lightning включают:

- хранилище ключей, в котором хранятся секреты, такие как приватные ключи;
- узел LN (узел Lightning), который взаимодействует в одноранговой сети, как описано ранее;
- узел Bitcoin, который хранит данные блочной цепи и взаимодействует с другими узлами Bitcoin;
- «карту» узлов и каналов, генерируемую на основе базы данных, которые объявлены в сети Lightning;
- менеджер каналов, который может открывать и закрывать каналы LN;
- систему в увеличенном масштабе, которая может отыскивать путь подсоединенных каналов от источника платежа до места назначения платежа.

Кошелек Lightning может содержать все эти функции, действуя как «полноценный» кошелек, не полагаясь на какие-либо сторонние службы. Либо один или несколько из этих компонентов могут полагаться (частично или полностью) на сторонние службы, которые выполняют эти функции.

Ключевое различие (преднамеренный каламбур) заключается в том, является функция хранилища ключей внутренней или переданной на аутсорсинг. В блочных цепях контроль над ключами определяет характер опекунства над средствами, о чем свидетельствует фраза «ваши ключи – ваши монеты; не ваши ключи – не ваши монеты». Любой кошелек, который передает управление ключами на аутсорсинг, называется *опекаемым*, или кустодиальным, кошельком, потому что третья сторона, действующая в качестве опекуна, конт-

ролирует средства пользователя, а не самого пользователя. Неопекаемый, или *самоопекаемый*, кошелек, для сравнения, – это кошелек, в котором хранилище ключей является частью кошелька, а ключи контролируются непосредственно пользователем. Термин «*неопекаемый* кошелек» просто подразумевает, что хранилище ключей является локальным и находится под контролем пользователя. Однако один или несколько других компонентов кошелька могут передаваться или не передаваться на аутсорсинг и полагаться на доверенные третьи стороны.

Блочные цепи, в особенности открытые блочные цепи, такие как Bitcoin, пытаются свести к минимуму или ослабить доверие к третьим сторонам и расширить возможности пользователей. Такой подход нередко именуется «бездоверительным», хотя больше подходит термин «минимально доверительный». В таких системах пользователь доверяет правилам программного обеспечения, а не третьим лицам. Поэтому вопрос контроля над ключами является принципиальным соображением при выборе кошелька Lightning.

Любой другой компонент кошелька Lightning предполагает аналогичные соображения по поводу доверия. Если все компоненты находятся под контролем пользователя, то степень доверия к третьим лицам сводится к минимуму, обеспечивая максимальную власть пользователю. Разумеется, это приводит к прямому компромиссу, потому что с этой властью приходит соответствующая ответственность за управление сложным программным обеспечением.

Каждый пользователь должен учитывать свои собственные технические навыки, прежде чем решать, какой тип кошелька Lightning использовать. Те, кто обладают сильными техническими навыками, должны использовать кошелек Lightning, который ставит все компоненты под непосредственный контроль пользователя. Тем, у кого меньше технических навыков, но есть желание контролировать свои средства, следует выбрать нестандартный кошелек Lightning. Часто доверие в этих случаях связано с конфиденциальностью. Если пользователи решают передать некоторые функции на аутсорсинг третьей стороне, то они обычно отказываются от некоторой конфиденциальности, поскольку третья сторона будет узнавать о них некоторую информацию.

Наконец, те, кто ищет простоту и удобство, даже в ущерб контролю и безопасности, могут выбрать опекаемый кошелек Lightning. Это наименее сложный с технической точки зрения вариант, но он подрывает доверительную модель криптовалюты и поэтому должен рассматриваться только как ступенька к большему контролю и самообеспечению.

Кошельки можно характеризовать или классифицировать несколькими способами. Наиболее важные вопросы, которые следует задать о конкретном кошельке, таковы:

1. Имеет ли этот кошелек Lightning полноценный узел Lightning или же он использует сторонний узел Lightning?
2. Имеет ли этот кошелек Lightning полноценный узел Bitcoin или же он использует сторонний узел Bitcoin?
3. Хранит ли этот кошелек Lightning свои собственные ключи под контролем пользователя (самоопека) или ключи хранятся сторонним опекуном?

Если кошелек Lightning использует сторонний узел Lightning, то именно этот сторонний узел Lightning решает, как взаимодействовать с системой Bitcoin. Следовательно, использование стороннего узла Lightning подразумевает, что вы также используете сторонний узел Bitcoin. Только когда кошелек Lightning использует свой собственный узел Lightning, существует выбор между полноценным узлом Bitcoin и сторонним узлом Bitcoin.

На самом высоком уровне абстракции вопросы 1 и 3 являются самыми элементарными. Из этих двух вопросов можно вывести четыре возможные категории. Эти четыре категории можно поместить в квадрант, как показано в табл. 2-1. Но помните, что это всего лишь один из способов классифицирования кошельков Lightning.

Таблица 2-1. Квадрант кошельков Lightning

	Полноценный узел Lightning	Сторонний узел Lightning
Самоопекаемый	Вопрос 1: высокие технические навыки, наименьшее доверие к третьим сторонам, наибольшее число разрешений	Вопрос 2: технические навыки ниже среднего, доверие к третьим сторонам ниже среднего, требуются некоторые разрешения
Опекаемый	Вопрос 3: технические навыки выше среднего, доверие к третьим сторонам выше среднего, требует некоторых разрешений	Вопрос 4: низкие технические навыки, высокое доверие к третьим сторонам, наименьшее число разрешений

Квадрант 3 (вопрос 3), где используется полный узел Lightning, но ключи хранятся у опекуна, в настоящее время не распространен. Будущие кошельки из этого квадранта могут позволить пользователю беспокоиться об операционных аспектах своего узла, но затем делегировать доступ к ключам третьей стороне, которая в основном использует холодное хранилище.

Кошельки Lightning могут инсталлироваться на различных устройствах, включая ноутбуки, серверы и мобильные устройства. Для оперирования полноценным узлом Lightning вам потребуется использовать сервер или настольный компьютер, поскольку мобильные устройства и ноутбуки обычно недостаточно мощны с точки зрения емкости, обработки, времени автономной работы и подключаемости.

Категория сторонних узлов Lightning снова может быть подразделена:

Облегченный

Это означает, что кошелек не управляет узлом Lightning и, следовательно, должен получать информацию о сети Lightning через интернет от чужого узла Lightning.

Никакой

Это означает, что не только узел Lightning управляется третьей стороной, но и бóльшая часть кошелька управляется третьей стороной в облаке. Этот кошелек предназначен для хранения средств, в котором кто-то другой опекает средства.

Эти подкатегории используются в табл. 2-2.

Другие термины, которые нуждаются в пояснении в табл. 2-2 в столбце «Узел Bitcoin», таковы:

Neutrino

Этот кошелек не управляет узлом Bitcoin. Вместо этого доступ к узлу Bitcoin, управляемому кем-то другим (третьей стороной), осуществляется по протоколу Neutrino.

Electrum

Этот кошелек не управляет узлом Bitcoin. Вместо этого доступ к узлу Bitcoin, управляемому кем-то другим (третьей стороной), осуществляется по протоколу Electrum.

Bitcoin Core

Это (референтная) имплементация узла Bitcoin.

btcd

Это еще одна имплементация узла Bitcoin.

В табл. 2-2 мы видим несколько примеров популярных в настоящее время приложений узлов и кошелков Lightning для различных типов устройств. Список отсортирован сначала по типу устройства, а затем в алфавитном порядке.

Таблица 2-2. Примеры популярных кошельков Lightning

Приложение	Устройство	Узел Lightning	Узел Bitcoin	Хранилище ключей
Blue Wallet	Мобильное	Никакой	Никакой	Опекаемый
Breez Wallet	Мобильное	Полноценный узел	Neutrino	Самоопекаемый
Eclair Mobile	Мобильное	Облегченный	Electrum	Самоопекаемый
Lntxbot	Мобильное	Никакой	Никакой	Опекаемый
Muun	Мобильное	Облегченный	Neutrino	Самоопекаемый
Phoenix Wallet	Мобильное	Облегченный	Electrum	Самоопекаемый
Zeus	Мобильное	Полноценный узел	Bitcoin Core/btcd	Самоопекаемый
Electrum	Настольное	Полноценный узел	Bitcoin Core/ Electrum	Самоопекаемый
Zap Desktop	Настольное	Полноценный узел	Neutrino	Самоопекаемый
c-lightning	Сервер	Полноценный узел	Bitcoin Core	Самоопекаемый
Eclair Server	Сервер	Полноценный узел	Bitcoin Core/ Electrum	Самоопекаемый
lnd	Сервер	Полноценный узел	Bitcoin Core/btcd	Самоопекаемый

Тестовая сеть Bitcoin

Система Bitcoin предлагает альтернативную цепь для целей тестирования, именуемую testnet, в отличие от «обычной» цепи Bitcoin, которая называется mainnet. В testnet валютой является *testnet-биткойн* (tBTC), который представляет собой бесполезную копию биткойна, используемую исключительно для тестирования. Каждая функция системы Bitcoin воспроизводится в точности, но деньги ничего не стоят, так что вам буквально нечего терять!

Некоторые кошельки Lightning также могут работать в testnet, позволяя совершать платежи Lightning с помощью биткойна testnet, не рискуя реальными

средствами. Это отличный способ безопасно поэкспериментировать с сетью Lightning. Кошелек Eclair Mobile, который Алиса использует в этой главе, является одним из примеров кошелька Lightning, поддерживающего работу testnet.

Немного tBTC для игры можно получить с помощью вентиля биткойнов testnet, который выдает бесплатные tBTC по запросу. Вот несколько вентилей testnet:

https://coinfaucet.eu/en/btc-testnet
https://testnet-faucet.mempool.co
https://bitcoinfaucet.uo1.net
https://testnet.help/en/btcfaucet/testnet

Все примеры в этой книге можно точно воспроизвести в testnet с tBTC, так что если пожелаете, то можете последовать по пятам, не рискуя реальными деньгами.

Уравновешивание сложности и контроля

Кошельки Lightning должны соблюдать тщательное равновесие между сложностью и контролем со стороны пользователя. Те, которые дают пользователю наибольший контроль над своими средствами, высочайшую степень конфиденциальности и наибольшую независимость от сторонних служб, с неизбежностью являются более сложными и трудными в эксплуатации. По мере развития технологий некоторые из этих компромиссов станут менее очевидными, и пользователи смогут получать больший контроль без дополнительных сложностей. Однако на данный момент разные компании и проекты занимаются разведкой разных позиций в этом спектре сложности управления, надеясь найти «золотую» середину для пользователей, на которых они ориентированы.

Выбирая кошелек, имейте в виду, что даже если вы не видите этих компромиссов, они все равно существуют. Например, многие кошельки будут пытаться снимать бремя управления каналами со своих пользователей. Для этого они вводят центральные хабовые узлы, к которым автоматически подключаются все их кошельки. Хотя этот компромисс упрощает пользовательский интерфейс и пользовательский опыт, он вводит единую точку отказа (point of failure, аббр. SPoF), поскольку эти хабовые узлы становятся незаменимыми для работы кошелька. Более того, использование такого «хаба» может снизить конфиденциальность пользователя, так как хаб знает отправителя и потенциально (при построении маршрута платежа от имени пользователя) также получателя каждого платежа, производимого кошельком пользователя.

В следующем далее разделе мы вернемся к нашему первому пользователю, Алисе, и рассмотрим ее первую настройку кошелька Lightning. Она выбрала кошелек, который является более изощренным, чем более простые опекаемые кошельки. Это позволяет нам показать некоторые основные сложности и представить некоторые внутренние механизмы продвинутого кошелька. Вы, возможно, обнаружите, что ваш первый идеальный кошелек ориентирован на простоту использования, принимая некоторые компромиссы между контролем и конфиденциальностью. Или же, вероятно, вы более опытный пользователь и хотите выполнять свои собственные узлы Lightning и Bitcoin как часть технического решения для вашего кошелька.

Скачивание и инсталляция кошелька Lightning

При поиске нового криптовалютного кошелька вы должны быть очень осторожны при выборе безопасного источника программно-информационного обеспечения.

К сожалению, многие поддельные приложения для кошельков крадут ваши деньги, и некоторые из них даже попадают на авторитетные и предположительно проверенные веб-сайты программного обеспечения, такие как магазины приложений Apple и Google. Независимо от того, устанавливаете ли вы свой первый или десятый кошелек, всегда проявляйте крайнюю осторожность. Мошенническое приложение может не просто украсть любые деньги, которые вы ему доверите, но оно также может украсть ключи и пароли из других приложений, поставив под угрозу операционную систему вашего мобильного устройства.

Алиса использует устройство Android и будет использовать магазин Google Play для скачивания и инсталлирования кошелька Eclair. Выполнив поиск в Google Play, она находит запись для «Eclair Mobile», как показано на рис. 2-1.

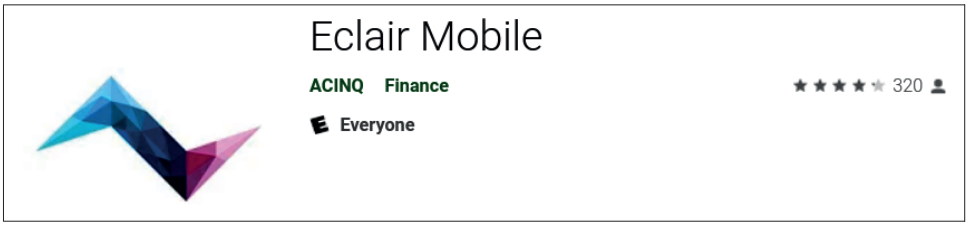


Рис. 2-1. Eclair Mobile в магазине Google Play

Используя testnet-биткойны, можно поэкспериментировать и протестировать все программное обеспечение, ориентированное на систему Bitcoin, с нулевым риском (за исключением вашего собственного времени). Вы также можете, зайдя в магазин Google Play, скачать кошелек Eclair testnet, чтобы попробовать Lightning (в testnet).

Алиса замечает на этой странице несколько разных элементов, которые помогают ей убедиться в том, что это, скорее всего, правильный кошелек «Eclair Mobile», который она ищет. Во-первых, в качестве разработчика этого мобильного кошелька указана организация ACINQ2[14], которая, как Алиса знает из своих исследований, является правильным разработчиком. Во-вторых, кошелек был установлен «10 000+» раз и имеет более 320 положительных отзывов. Маловероятно, что это мошенническое приложение, которое проникло в магазин Google Play. В качестве третьего шага она переходит на веб-сайт ACINQ[15]. Она проверяет безопасность веб-страницы, убедившись, что адрес начинается с https или в некоторых браузерах с префикса в виде замка. На веб-сайте она переходит в раздел скачивания или ищет ссылку на магазин приложений Google. Она находит ссылку и нажимает на нее. Она убеждается, что эта ссылка

[14] ACINQ: разработчики мобильного кошелька Eclair Lightning.

[15] См. https://acinq.co/.

приводит ее к тому же самому приложению в магазине приложений Google. Удовлетворенная этими выводами, Алиса инсталлирует приложение Eclair на свое мобильное устройство.

Все примеры в этой книге могут быть точно воспроизведены в testnet с помощью tBTC, так что, если хотите, можете продолжить работу там, не рискуя реальными деньгами.

Всегда проявляйте большую осторожность при инсталлировании программного обеспечения на любое устройство. Существует много поддельных криптовалютных кошельков, которые не только украдут ваши деньги, но и могут скомпрометировать все другие приложения на вашем устройстве.

Создание нового кошелька

Когда Алиса впервые открывает мобильное приложение Eclair, ей предлагается выбрать «Create a New Wallet» (Создать новый кошелек) или «Import an Existing Wallet» (Импортировать существующий кошелек). Алиса создаст новый кошелек, но давайте сначала обсудим, почему здесь представлены эти опции и что значит импортировать существующий кошелек.

Ответственность за хранение ключей

Как мы упоминали в начале этого раздела, Eclair – это неопекаемый кошелек, а это означает, что Алиса единолично владеет ключами, используемыми для управления ее биткойном. Это также означает, что Алиса отвечает за защиту и резервное копирование этих ключей. Если Алиса потеряет ключи, то никто не сможет помочь ей вернуть биткойн, и ключи будут потеряны навсегда.

С мобильным кошельком Eclair Алиса опекает и контролирует ключи и, следовательно, несет полную ответственность за сохранность ключей и их резервное копирование. Если она потеряет ключи, то она потеряет биткойн, и никто не сможет помочь ей оправиться от этой потери!

Мнемонические слова

Подобно большинству кошельков Bitcoin, Eclair Mobile предоставляет мнемоническую фразу (также иногда именуемую «посевом» или «начальной фразой»), для того чтобы Алиса оставила их в резерве. Мнемоническая фраза состоит из 24 английских слов, выбранных программой случайным образом и используемых в качестве основы для ключей, которые генерируются кошельком. Алиса может использовать мнемоническую фразу для восстановления всех транзакций и средств в мобильном кошельке Eclair в случае потери мобильного устройства, ошибки программного обеспечения или повреждения памяти.

Правильный термин для этих резервных слов – «мнемоническая фраза». Мы избегаем использования термина «посев» (seed) для обозначения мнемонической фразы, потому что, хотя его использование и распространено, оно неверно.

Когда Алиса решит создать новый кошелек, она увидит экран со своей мнемонической фразой, которая выглядит как снимок экрана на рис. 2-2.

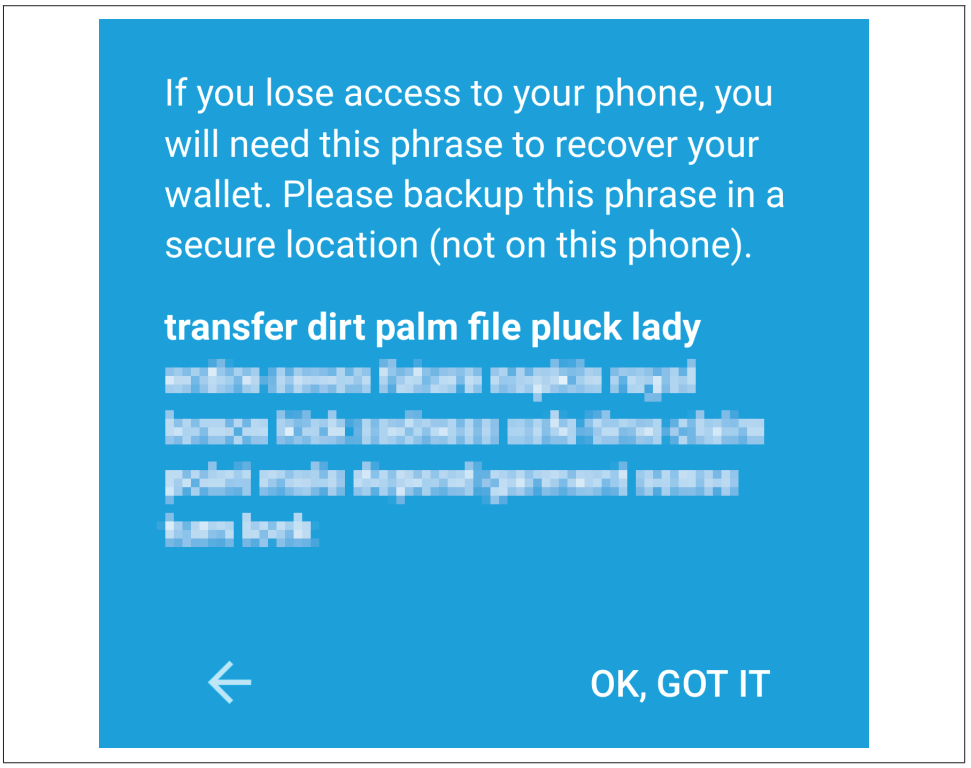


Рис. 2-2. Мнемоническая фраза нового кошелька

На рис. 2-2 мы намеренно скрыли часть мнемонической фразы, чтобы читатели этой книги не могли использовать мнемонику повторно.

Безопасное хранение мнемоники

Алисе не следует забывать, что мнемоническую фразу нужно хранить таким образом, чтобы предотвратить ее кражу, а также избегать случайной потери. Рекомендуемый способ правильно сбалансировать эти риски – написать две копии мнемонической фразы на бумаге, пронумеровав каждое из слов – порядок имеет значение.

Как только Алиса запишет мнемоническую фразу, после нажатия кнопки **OK, GOT IT** на своем экране ей будет представлен тест, чтобы убедиться, что она правильно записала мнемоническую фразу. В тесте будет предложено выбрать три или четыре слова наугад. Для Алисы этот тест был неожиданным, но так как она правильно записала мнемонику, она проходит его без каких-либо трудностей.

После того как Алиса запишет мнемоническую фразу и пройдет тест, она должна положить каждую копию в отдельном безопасном месте, например в запертом ящике стола или в несгораемом сейфе.

Никогда не пытайтесь использовать доморощенную схему «обеспечения безопасности», которая каким-либо образом отклоняется от рекомендаций лучших образцов практики в разделе «Безопасное хранение мнемоники» на стр. 50. Не разрезайте свою мнемонику пополам, не делайте скриншоты, не храните ее на USB-накопителях или облачных накопителях, не шифруйте и не пытайтесь использовать любой другой нестандартный метод. Вы будете склонять чашу весов в сторону риска необратимой потери. Многие люди теряли средства не из-за кражи, а из-за того, что они попробовали нестандартное решение, не имея опыта, чтобы сбалансировать связанные с ним риски. Рекомендации лучших образцов практики тщательно рассматриваются экспертами и подходят для подавляющего большинства пользователей.

После того как Алиса инициализирует свой мобильный кошелек Eclair, она увидит краткое руководство, в котором освещаются различные элементы пользовательского интерфейса. Мы не будем здесь повторять учебное пособие, но мы рассмотрим все эти элементы, следуя за попыткой Алисы купить чашку кофе!

Загрузка биткойна в кошелек

У Алисы теперь есть кошелек Lightning. Но он – пуст! Сейчас она сталкивается с одним из наиболее сложных аспектов этого эксперимента: она должна найти способ приобрести немного биткойна и загрузить его в свой кошелек Eclair.

Если у Алисы уже есть биткойн в другом кошельке, то она может отправить этот биткойн на свой кошелек Eclair, вместо того чтобы приобретать новый биткойн для загрузки на свой новый кошелек.

Приобретение биткойна

Алиса может приобрести биткойн несколькими способами:

- она может обменять часть своей национальной валюты (например, доллары США) на бирже криптовалют;
- она может купить немного у друга/подруги или знакомого на Bitcoin-встрече в обмен на наличные;
- она может найти Bitcoin-банкомат в своем районе, который действует как торговый автомат, продавая биткойны за наличные;
- она может предложить свои навыки или продукт, который она продает, и принять оплату в биткойне;
- она может попросить своего работодателя или клиентов заплатить ей в биткойне.

Все эти методы имеют разную степень сложности, и многие из них будут связаны с оплатой комиссионных. Некоторые также потребуют, чтобы Алиса предоставила документы, удостоверяющие личность, в соответствии с местными банковскими правилами. Однако, используя все эти методы, Алиса сможет получать биткойн.

Получение биткойна

Давайте предположим, что Алиса нашла местный Bitcoin-банкомат и решила купить немного биткойна в обмен на наличные. Пример Bitcoin-банкомата, собранного компанией Lamassu, показан на рис. 2-3. Такие Bitcoin-банкоматы принимают национальную валюту (наличные) через слот для наличных и отправляют биткойн на Bitcoin-адрес, отсканированный из кошелька пользователя с помощью встроенной камеры.

Рис. 2-3. Bitcoin-банкомат Lamassu

Для того чтобы получить биткойн в свой кошелек Eclair Lightning, Алисе нужно будет предоставить Bitcoin-банкомату Bitcoin-адрес из кошелька Eclair Lightning. Затем банкомат может отправить недавно приобретенный биткойн Алисы на этот Bitcoin-адрес.

Для того чтобы увидеть Bitcoin-адрес в кошельке Eclair, Алиса должна провести пальцем по левой колонке под названием **YOUR BITCOIN ADDRESS** (Ваш Bitcoin-адрес) (см. рис. 2-4), где она увидит квадратный штрих-код (т. н. QR-код) и строку букв и цифр под ним.

QR-код содержит ту же строку букв и цифр, что и под ним, в удобном для сканирования формате. Таким образом, Алисе не нужно вводить Bitcoin-адрес. На скриншоте (рис. 2-4) мы намеренно размыли оба, чтобы читатели не могли случайно отправить биткойн на этот адрес.

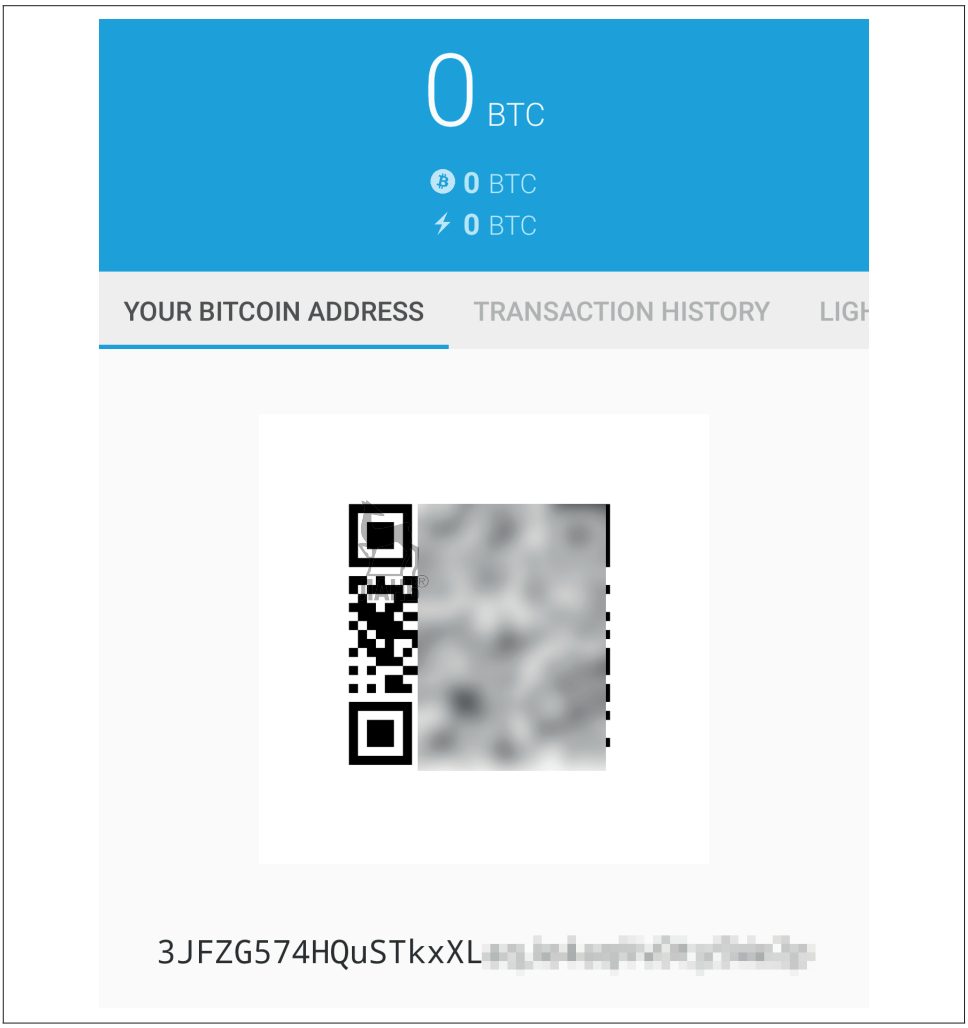


Рис. 2-4. Bitcoin-адрес Алисы, показанный в Eclair

Как Bitcoin-адреса, так и QR-коды содержат информацию подсистемы обнаружения ошибок, которая предотвращает любые ошибки ввода или сканирования, приводящие к «неправильному» Bitcoin-адресу. Если в адресе есть ошибка, то любой кошелек Bitcoin заметит ошибку и откажется принимать Bitcoin-адрес как валидный.

Алиса может поднести свое мобильное устройство к банкомату и показать его встроенной камере, как показано на рис. 2-5. Вставив немного наличных в слот, она получит биткойн в Eclair!

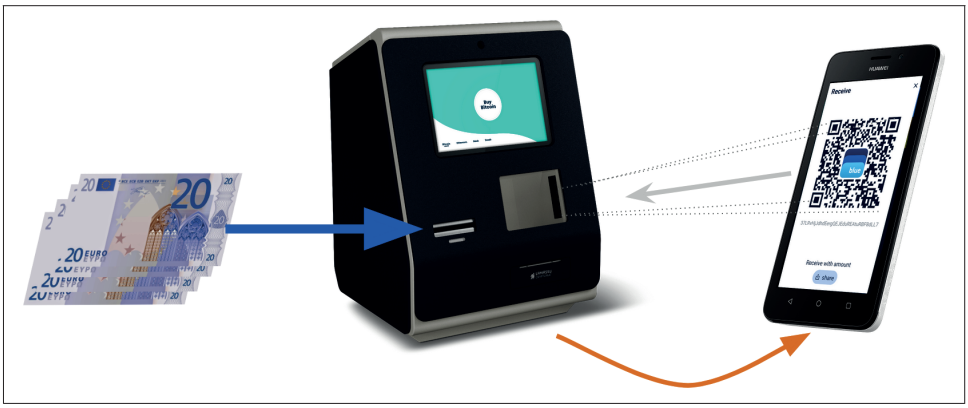


Рис. 2-5. Bitcoin-банкомат сканирует QR-код

Алиса увидит транзакцию из банкомата на вкладке **TRANSACTION HISTORY** (История транзакций) кошелька Eclair. Хотя Eclair обнаружит транзакцию Bitcoin всего за несколько секунд, потребуется примерно один час, чтобы транзакция Bitcoin была «подтверждена» в блочной цепи Bitcoin. Как видно на рис. 2-6, кошелек Eclair Алисы показывает «6+ conf» под транзакцией, указывая на то, что транзакция получила необходимый минимум из шести подтверждений, и ее средства теперь готовы к использованию.

Число подтверждений транзакции – это число блоков, добытых после блока, который содержал эту транзакцию (включая этот блок). Шесть подтверждений – лучшая практика, но разные кошельки Lightning могут считать канал открытым после любого числа подтверждений. Некоторые кошельки даже увеличивают число ожидаемых подтверждений на монетарную стоимость канала.

Хотя в этом примере Алиса использовала банкомат для приобретения своего первого биткойна, те же базовые концепции будут применяться, даже если она использовала один из других методов из раздела «Приобретение биткойна» на стр. 51. Например, если Алиса хотела продать товар или предоставить профессиональную услугу в обмен на биткойн, то ее клиенты могли бы отсканировать Bitcoin-адрес своими кошельками и заплатить ей в биткойне.

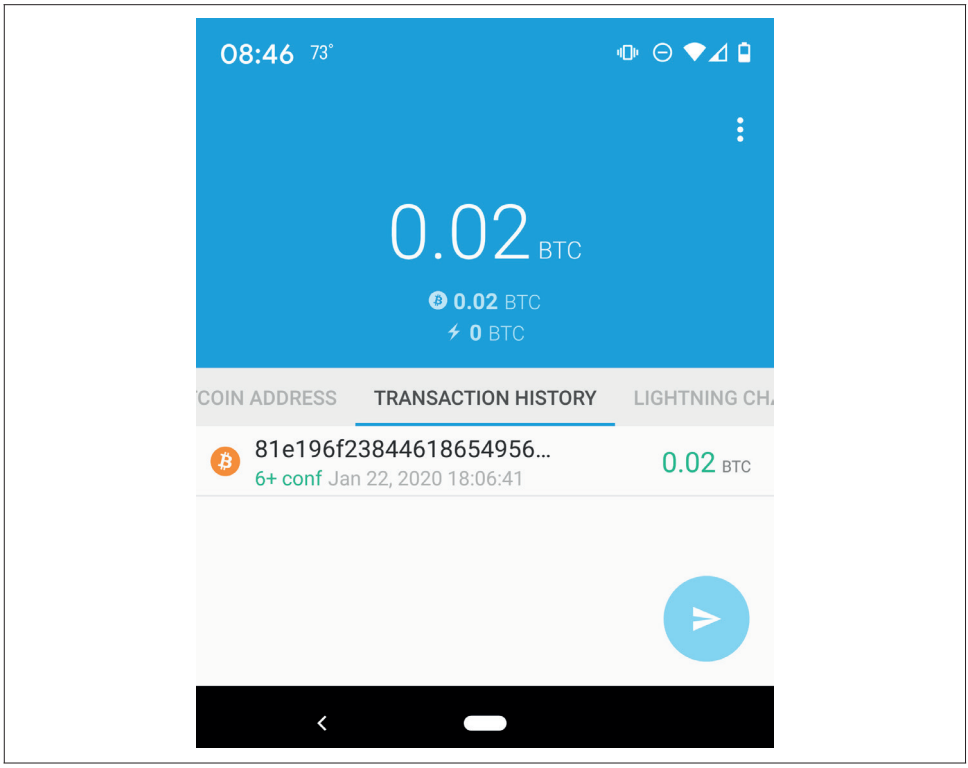


Рис. 2-6. Алиса получает биткойн

Точно так же, если бы она выставила счет клиенту за услугу, предлагаемую через интернет, то Алиса могла бы отправить электронное письмо или мгновенное сообщение с Bitcoin-адресом или QR-кодом своему клиенту, и он мог бы вставить или отсканировать информацию в кошелек Bitcoin, чтобы ей заплатить.

Алиса могла бы даже распечатать QR-код, прикрепить его к вывеске и выставить на всеобщее обозрение, чтобы получать чаевые. Например, она могла бы прикрепить QR-код к своей гитаре и получать чаевые во время выступления на улице[16]!

Наконец, если Алиса купила биткойн на бирже криптовалют, то она могла (и должна была) бы «вывести» биткойн, вставив свой Bitcoin-адрес на вебсайте биржи. Затем биржа отправила бы биткойн непосредственно на ее адрес.

[16] Как правило, не рекомендуется реиспользовать один и тот же Bitcoin-адрес для нескольких платежей, поскольку все транзакции Bitcoin являются публичными. Проходящий мимо любопытный человек может отсканировать QR-код Алисы и посмотреть, сколько денег Алиса уже получила по этому адресу в блочной цепи Bitcoin. К счастью, сеть Lightning предлагает более приватные решения этой проблемы, которые обсуждаются далее в нашей книге!

Из системы Bitcoin в сеть Lightning

Биткойн Алисы теперь контролируется ее кошельком Eclair и был зарегистрирован в блочной цепи Bitcoin. На данный момент биткойн Алисы находится *внутри цепи*, и, стало быть, транзакция была передана широковещательно по всей сети Bitcoin, была верифицирована всеми узлами Bitcoin и добыта (зарегистрирована) в блочной цепи Bitcoin.

До сих пор мобильный кошелек Eclair вел себя только как кошелек Bitcoin, и Алиса не использовала функциональность Eclair, связанную с сетью Lightning. Как и в случае со многими кошельками Lightning, Eclair соединяет систему Bitcoin и сеть Lightning мостом, выступая одновременно в качестве кошелька Bitcoin и кошелька Lightning.

Теперь Алиса готова начать использовать сеть Lightning путем выведения своего биткойна из цепи, чтобы воспользоваться преимуществами быстрых, дешевых и приватных платежей, которые предлагает сеть Lightning.

Каналы сети Lightning

Проведя пальцем вправо, Алиса получает доступ к разделу **LIGHTNING CHANNELS** (Каналы Lightning) в Eclair. Здесь она может управлять каналами, которые будут подключать ее кошелек к сети Lightning.

В этом месте давайте вернемся к определению канала LN, чтобы немного прояснить ситуацию. Во-первых, слово «канал» – это метафора *финансовой взаимосвязи* между кошельком Lightning Алисы и другим кошельком Lightning. Мы называем ее каналом, потому что это средство внецепного обмена множеством платежей между кошельком Алисы и другим кошельком в сети Lightning без совершения внутрицепных транзакций в блочной цепи Bitcoin.

Кошелек или узел, к которому Алиса открывает канал, называется ее канальным одноранговым узлом. После «открытия» канал можно использовать для отправки множества платежей туда и обратно между кошельком Алисы и ее канальным одноранговым узлом.

Кроме того, канальный одноранговый узел Алисы может пересылать платежи по другим каналам дальше в сети Lightning. Благодаря этому Алиса может маршрутизировать платеж на любой кошелек (например, кошелек Lightning Боба) до тех пор, пока кошелек Алисы сможет найти приемлемый *путь*, перепрыгивая с канала на канал, вплоть до кошелька Боба.

Не все канальные одноранговые узлы являются *хорошими* одноранговыми узлами для маршрутизации платежей. Хорошо соединенные одноранговые узлы смогут маршрутизировать платежи по более коротким путям к месту назначения, увеличивая шансы на успех. А канальные одноранговые узлы с достаточными средствами смогут маршрутизировать более крупные платежи.

Другими словами, Алисе нужен один или несколько каналов, которые соединяют ее с одним или несколькими другими узлами сети Lightning. Ей не нужен канал для подсоединения своего кошелька к кофейне Боба напрямую, чтобы отправить Бобу платеж, хотя она и может решить открыть прямой канал. Любой узел в сети Lightning может быть использован для первого канала Алисы. Чем лучше связ-

ность узла, тем большего числа людей Алиса сможет достичь. Поскольку в этом примере мы хотим также продемонстрировать маршрутизацию платежей, мы не будем заставлять Алису открывать канал прямо к кошельку Боба. Вместо этого мы попросим Алису открыть канал на хорошо соединенный узел, а затем позже использовать этот узел для пересылки своего платежа, маршрутизируя его через любые другие узлы по мере необходимости, чтобы связаться с Бобом.

Сначала открытых каналов нет, поэтому, как мы видим на рис. 2-7, на вкладке **LIGHTNING CHANNELS** (Каналы Lightning) показан пустой список. Если вы заметили, в правом нижнем углу есть символ плюса (+), который является кнопкой для открытия нового канала.

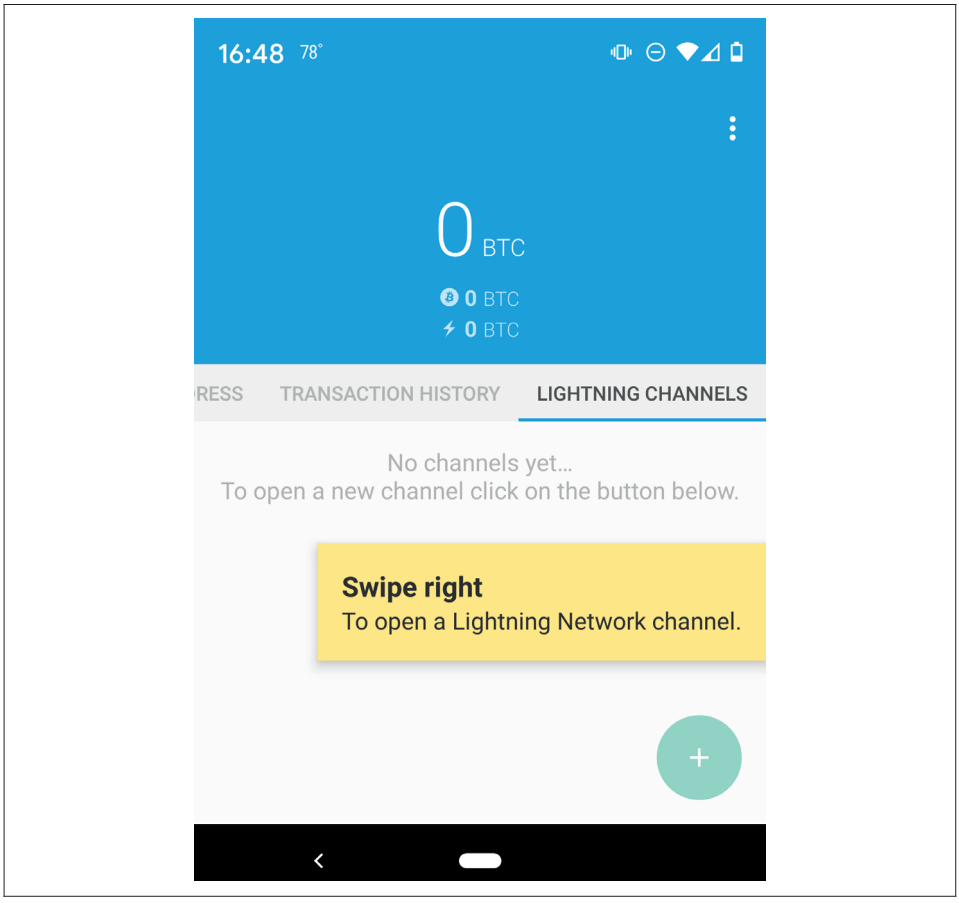


Рис. 2-7. Вкладка **LIGHTNING CHANNELS**

Алиса нажимает на символ плюса, и ей предлагается четыре возможных способа открыть канал:

- вставить URI-адрес узла;
- отсканировать URI-адрес узла;
- случайный узел;
- узел ACINQ.

«URI-адрес узла» – это универсальный идентификатор ресурса (URI), который идентифицирует конкретный узел Lightning. Алиса может либо вставить такой URI-адрес из своего буфера обмена, либо отсканировать QR-код, содержащий ту же информацию. Пример URI-адреса узла показан в виде QR-кода на рис. 2-8, а затем в виде текстовой строки.

Рис. 2-8. URI-адрес узла в виде QR-кода

```
0237fefbe8626bf888de0cad8c73630e32746a22a2c4faa91c1d9877a3826e1174@1.
ln.aanto-nop.com:9735
```

Хотя Алиса может выбрать конкретный узел Lightning или использовать опцию **Случайный узел**, чтобы кошелек Eclair выбирал узел случайным образом, она выберет опцию **Узел ACINQ** для подсоединения к одному из хорошо соединенных узлов Lightning ACINQ.

Выбор узла ACINQ немного снизит конфиденциальность Алисы, поскольку это даст ACINQ возможность видеть все транзакции Алисы. Это также создаст единую точку отказа, поскольку у Алисы будет только один канал, и если узел ACINQ не будет доступен, то Алиса не сможет совершать платежи. Для того чтобы поначалу все было просто, мы примем эти компромиссы. В последующих главах мы постепенно узнаем, как добиться большей независимости и идти на меньшее число компромиссов!

Алиса выбирает узел ACINQ и готова открыть свой первый канал в сети Lightning.

Открытие канала Lightning

Когда Алиса выбирает узел для открытия нового канала, ее просят выбрать, сколько биткойна она хочет выделить на этот канал. В последующих главах мы обсудим последствия этого выбора, но на данный момент Алиса выделит каналу почти все свои средства. Поскольку ей придется заплатить комиссию за транзакцию, чтобы открыть канал, она выберет сумму, немного меньшую, чем ее суммарный остаток средств[17].

Алиса выделяет на свой канал 0.018 BTC из своих 0.020 BTC и принимает дефолтную ставку комиссии, как показано на рис. 2-9.

[17] Кошелек Eclair не предлагает опцию автоматического расчета необходимых комиссионных и выделения максимальной суммы средств каналу, поэтому Алисе приходится рассчитывать это самостоятельно.

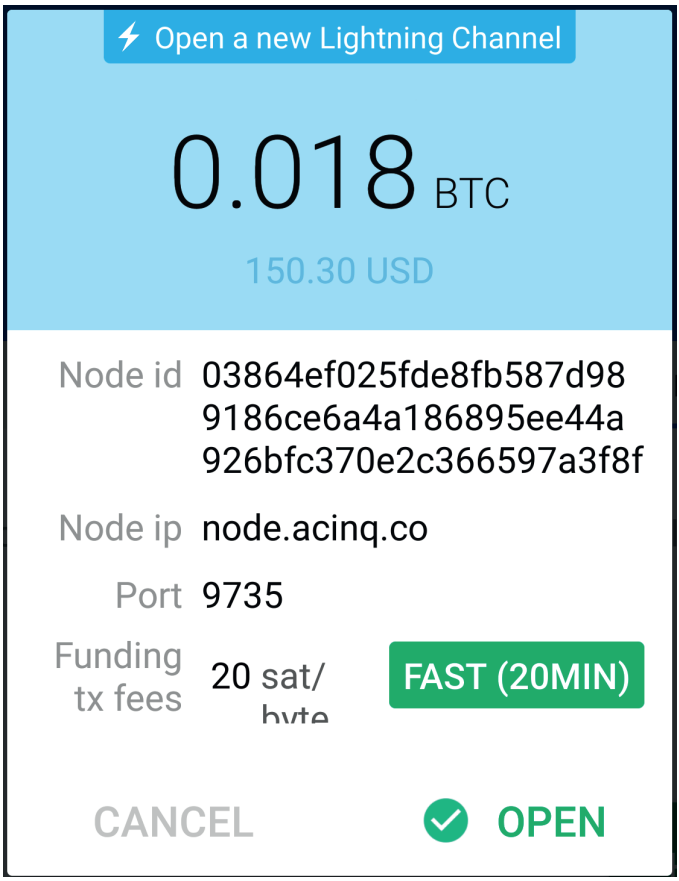


Рис. 2-9. Открытие канала Lightning

Как только она нажимает кнопку **OPEN** (Открыть), ее кошелек создает специальную транзакцию Bitcoin, которая открывает канал Lightning, именуемую *финансовой транзакцией*. Внутрицепная финансовая транзакция отправляется в сеть Bitcoin для подтверждения.

Теперь Алисе снова приходится ждать (см. рис. 2-10) до тех пор, пока транзакция не будет записана в блочной цепи Bitcoin. Как и в случае с первоначальной транзакцией Bitcoin, которую она использовала для приобретения своего биткойна, ей приходится ждать шести или более подтверждений (примерно один час).

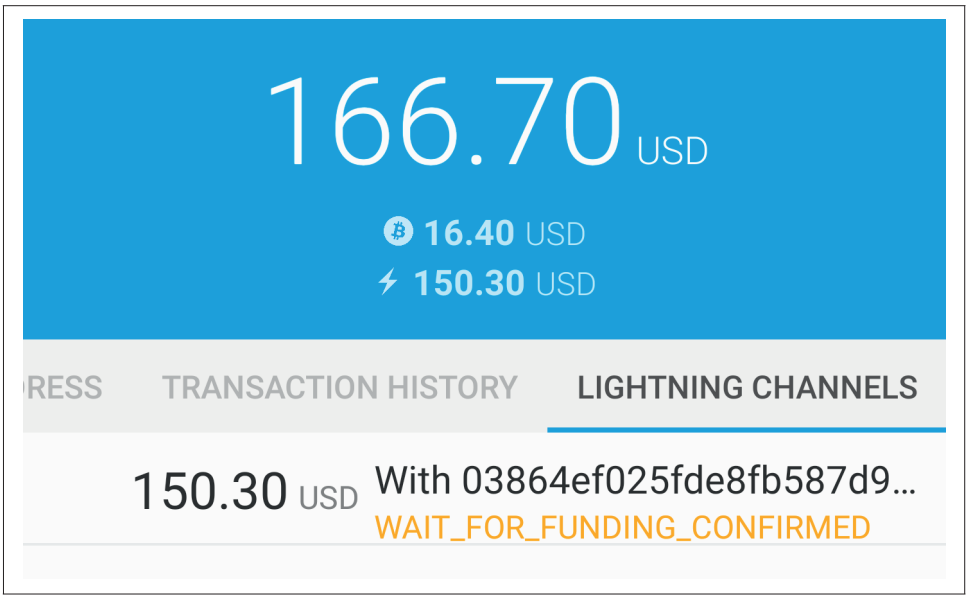


Рис. 2-10. Ожидание финансовой транзакции, чтобы открыть канал

После того как финансовая транзакция будет подтверждена, канал Алисы к узлу ACINQ будет открыт, профинансирован и готов, как показано на рис. 2-11.

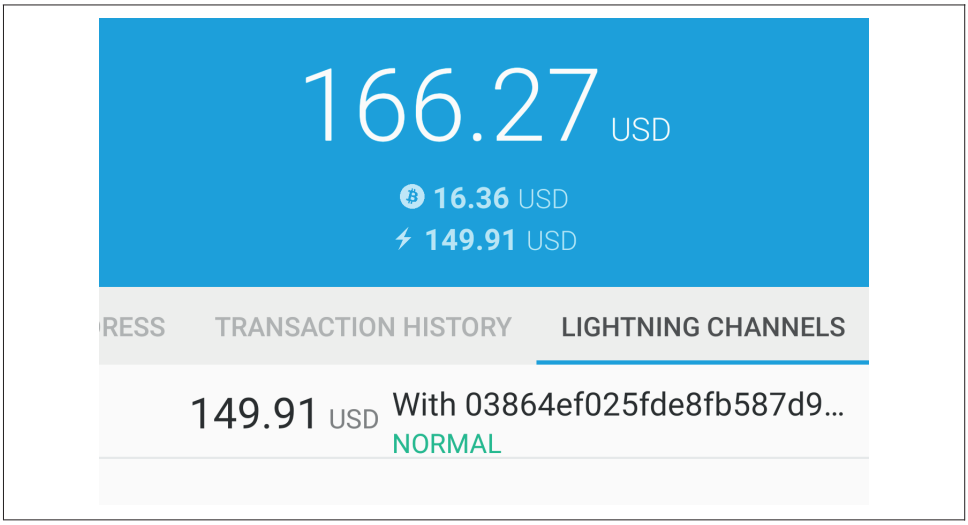


Рис. 2-11. Канал открыт

Вы заметили, что денежная сумма канала, похоже, изменилась? Это не так: канал содержит все те же 0.018 BTC, но за время между снимками экрана курс обмена BTC изменился, поэтому стоимость в долларах США будет другой. Вы можете выбрать отображение остатка в BTC или долларах США, но имейте в виду, что значения в долларах США рассчитываются в реальном времени и будут меняться!

Покупка чашки кофе с помощью сети Lightning

Теперь у Алисы все готово для того, чтобы начать использовать сеть Lightning. Как вы видите, это заняло немного работы и немного времени в ожидании подтверждений. Однако теперь последующие действия выполняются быстро и легко. Сеть Lightning позволяет осуществлять платежи без необходимости ждать подтверждений, так как средства зачисляются за считанные секунды.

Алиса хватает свое мобильное устройство и бежит в кофейню Боба по соседству. Ей не терпится попробовать свой новый кошелек Lightning и использовать его, чтобы что-нибудь купить!

Кофейня Боба

У Боба есть простое приложение для торговых точек (point-of-sale, аббр. PoS) для использования любым клиентом, который хочет расплатиться биткойнами через сеть Lightning. Как мы увидим в следующей главе, Боб использует популярную платформу с открытым исходным кодом BTCPay Server, которая содержит все необходимые компоненты технического решения для электронной коммерции или розничной торговли, такие как:

- узел Bitcoin, использующий программное обеспечение Bitcoin Core;
- узел Lightning, использующий программное обеспечение c-lightning;
- простое PoS-приложение для планшета.

Сервер BTCPay упрощает инсталлирование всего необходимого программного обеспечения, закачку фотографий и цен на товары на сервер, а также быстрый запуск магазина.

На прилавке в кофейне Боба есть планшетное устройство, показывающее то, что вы видите на рис. 2-12.

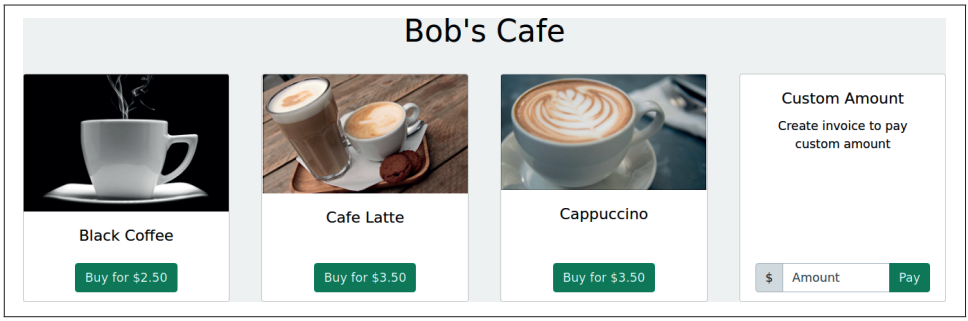


Рис. 2-12. Приложение Боба для торговых точек

Счет Lightning

Алиса выбирает на экране опцию **Кофе латте** и получает *счет Lightning* (также именуемый «платежным запросом»), как показано на рис. 2-13.

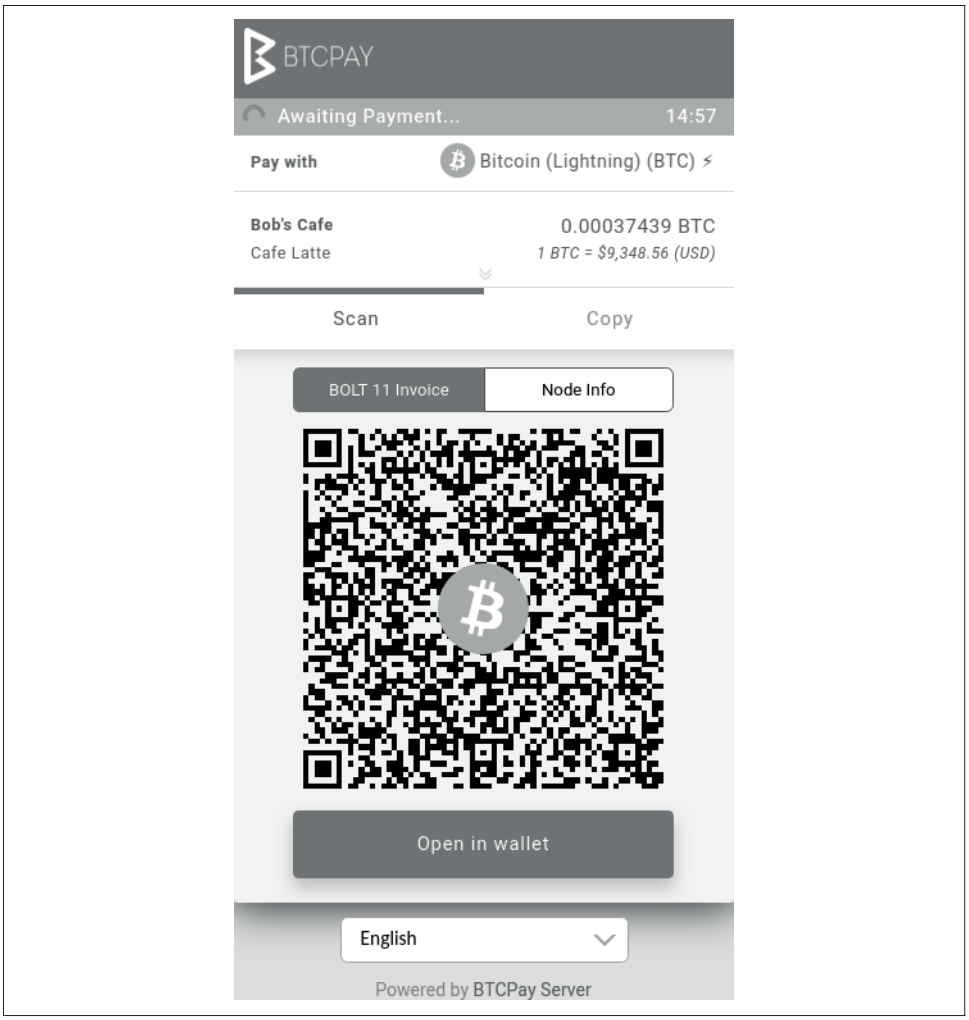


Рис. 2-13. Счет Lightning за латте Алисы

В целях оплаты счета Алиса открывает свой кошелек Eclair и нажимает кнопку **Send** (Отправить) (которая выглядит как стрелка вверх) на вкладке **TRANSACTION HISTORY** (История транзакций), как показано на рис. 2-14.

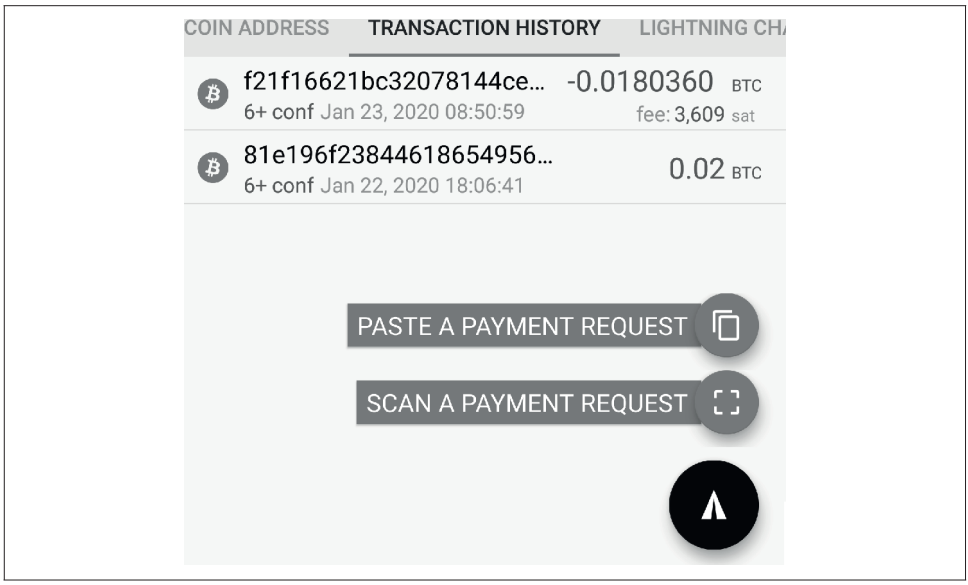


Рис. 2-14. Алиса, нажимающая кнопку **Send** (Отправить)

Термин «платежный запрос» может относиться к запросу на оплату в системе Bitcoin или счету Lightning, и термины «счет» и «платежный запрос» (или запрос на оплату) часто используются взаимозаменяемо. Правильным техническим термином является «счет Lightning», независимо от того, как он называется в кошельке.

Алиса выбирает опцию **scan a payment request** (сканировать платежный запрос), сканирует QR-код, отображаемый на экране планшета (см. рис. 2-13), и ей предлагается подтвердить свой платеж, как показано на рис. 2-15.

Алиса нажимает **PAY** (Оплатить), и через секунду планшет Боба показывает успешный платеж. Алиса завершила свой первый платеж LN! Это было быстро, недорого и просто. Теперь она может наслаждаться своим латте, который был куплен за биткойн через быструю, дешевую и децентрализованную платежную систему. Отныне Алиса может просто выбирать товар на экране планшета Боба, сканировать QR-код с помощью своего мобильного телефона, нажимать **PAY**, и ей подадут кофе, и все это будет происходить в течение нескольких секунд, и все это без внутрицепной транзакции.

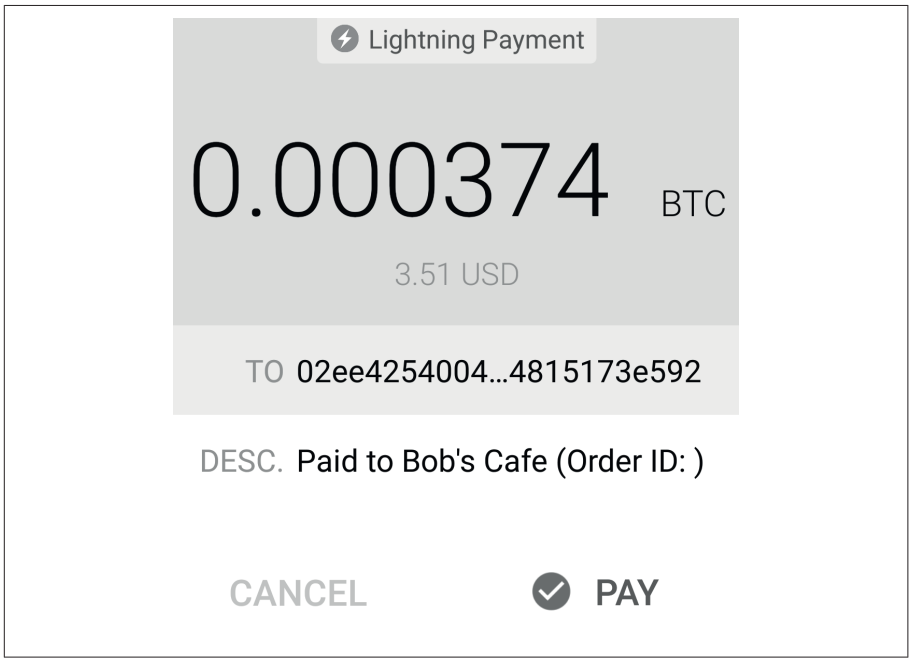


Рис. 2-15. Подтверждение выполненной Алисой отправки

Платежи Lightning лучше и для Боба. Он уверен, что ему заплатят за латте Алисы, не дожидаясь внутрицепного подтверждения. В будущем, всякий раз, когда Алисе захочется выпить кофе в кофейне Боба, она сможет выбирать оплату биткойном в сети Bitcoin или сети Lightning. Как вы думаете, какой из них она выберет?

Вывод

В этой главе мы проследили за Алисой, когда она скачала и инсталлировала свой первый кошелек Lightning, приобрела и перевела немного биткойна, открыла свой первый канал Lightning и купила чашку кофе, совершив свой первый платеж в сети Lightning. В следующих главах мы рассмотрим «под капотом», как работает каждый компонент сети Lightning и как платеж Алисы дошел до кофейни Боба.

Глава 3

Как работает сеть Lightning

Теперь, когда мы проследили за Алисой, когда она создавала кошелек Lightning и покупала кофе у Боба, мы заглянем под капот и распакуем разные компоненты сети Lightning, участвующие в этом процессе. В этой главе будет дан высокоуровневый обзор, не углубляясь во все технические детали. Цель скорее состоит в том, чтобы помочь вам ознакомиться с наиболее важными концепциями и строительными блоками сети Lightning.

Если у вас есть опыт в области информатики, криптографии, системы Bitcoin и разработки протоколов, то этой главы должно быть достаточно, чтобы вы смогли самостоятельно расставить связующие детали. Если у вас опыта меньше, то эта глава даст вам достаточно хороший обзор, чтобы вам было легче понять официальные спецификации протокола, известные как спецификации BOLT (Базис технологии Lightning). Если вы – новичок, то эта глава поможет вам лучше понять технические главы книги.

Если вам нужно освежить знания об основах системы Bitcoin, то вы можете найти краткий обзор следующих ниже тем в приложении А:

- ключи и адреса;
- хеш-функции;
- цифровые подписи;
- структура транзакций;
- входы и выходы транзакций;
- выстраивание транзакций в цепь;
- Bitcoin Script;
- мультиподписные адреса и скрипты;
- привязки ко времени;
- сложные скрипты.

Мы начнем с краткого определения того, что такое сеть Lightning, и разберем ее в деталях в оставшейся части этой главы.

Сеть Lightning – это одноранговая сеть платежных каналов, имплементированная в виде умных контрактов на блочной цепи Bitcoin, а также протокол обмена информацией, который определяет то, как участники настраивают и исполняют эти умные контракты.

Что такое платежный канал?

В зависимости от контекста есть несколько способов описания платежного канала. Давайте начнем с высокого уровня, а затем добавим еще немного деталей.

Платежный канал – это *финансовая взаимосвязь* между двумя узлами сети Lightning, именуемыми партнерами по каналу. Финансовая взаимосвязь распределяет *остаток средств* (выраженный в миллисатоши) между двумя партнерами по каналу.

Платежный канал управляется *криптографическим протоколом*, и, значит, партнеры по каналу используют предопределенный процесс, основанный на криптографии, с целью перераспределения остатка канала в пользу того или иного партнера по каналу. Криптографический протокол обеспечивает, чтобы один партнер по каналу не смог обмануть другого, вследствие чего партнерам не нужно доверять друг другу.

Криптографический протокол устанавливается путем финансирования *мультиподписного адреса* «2 из 2», который требует сотрудничества двух партнеров по каналу и не позволяет ни одному из партнеров по каналу тратить средства в одностороннем порядке.

Подводя итог: платежный канал – это финансовая взаимосвязь между узлами, распределяющая средства с мультиподписного адреса по строго определенному криптографическому протоколу.

Основы платежного канала

В основе платежного канала лежит просто мультиподписной адрес «2 из 2» в блочной цепи Bitcoin, для которого вы владеете одним ключом, а ваш партнер по каналу владеет другим ключом.

Вы и ваш партнер по каналу согласовываете последовательность транзакций, которые расходуются с этого мультиподписного адреса. Вместо того чтобы передавать и записывать эти транзакции в блочную цепь Bitcoin, вы оба храните их неизрасходованными.

Последняя транзакция в этой последовательности кодирует остаток канала и определяет то, как этот остаток делится между вами и вашим партнером по каналу.

Таким образом, добавление новой транзакции в эту последовательность эквивалентно перемещению некоторой части остатка канала от одного партнера по каналу к другому без ведома сети Bitcoin. Когда вы согласовываете каждую новую транзакцию, изменяя распределение средств в канале, вы также отзываете предыдущую транзакцию, чтобы ни одна из сторон не смогла вернуться к предыдущему состоянию.

Каждая транзакция в последовательности использует скриптовый язык Bitcoin Script, и, таким образом, согласование средств между вами и вашим партнером по каналу управляется умным контрактом системы Bitcoin. Умный контракт настроен штрафовать участника канала, если он пытается отправлять ранее отозванное состояние канала.

Если у вас есть неопубликованная транзакция с мультиподписного адреса «2 из 2», которая выплачивает вам часть остатка, то подпись другой стороны обеспечивает вашу возможность независимо опубликовать эту транзакцию в любое время, добавив свою собственную подпись.

Возможность удерживать частично подписанную транзакцию офлайновой и неопубликованной, с возможностью публикации и владения этим остатком в любое время, является основой сети Lightning.

Маршрутизирование платежей по каналам

При условии что у нескольких участников есть каналы из одной стороны в другую, платежи также могут «пересылаться» с платежного канала на платежный канал, установив по сети *путь*, соединяющий несколько платежных каналов вместе.

Например, Алиса может отправить деньги Чарли, если у Алисы есть канал с Бобом, а у Боба есть канал с Чарли.

Благодаря конструкции сети Lightning существует возможность расширять умные контракты, которые управляют каналом, чтобы у Боба не было возможности украсть средства, которые пересылаются по его каналу.

Точно так же, как умный контракт защищает партнеров по каналу, поэтому им нет надобности доверять друг другу, вся сеть защищает участников, чтобы они могли пересылать платежи, не доверяя никому из других участников.

Поскольку каналы строятся из мультиподписных адресов, а транзакции обновления остатка являются транзакциями Bitcoin с предварительной подписью, все доверие, необходимое для работы сети Lightning, исходит из доверия к децентрализованной сети Bitcoin!

Вышеупомянутые инновации, безусловно, являются основными прорывами, позволившими создать сеть Lightning. Однако сеть Lightning – это гораздо больше, чем криптографические протоколы поверх языка Bitcoin Script. Это изощренный протокол связи, который позволяет одноранговым узлам обмениваться сообщениями Lightning с целью передачи биткойна. Протокол связи определяет то, как сообщения Lightning шифруются, и то, как ими обмениваются.

Сеть Lightning также использует эмидемический протокол (протокол сплетен) для распространения публичной информации о каналах (топологии сети) среди всех участников.

Алисе, например, нужна информация о топологии сети, чтобы знать о канале между Бобом и Чарли, благодаря которой она имеет возможность построить маршрут к Чарли.

И последнее, но не менее важное: важно понимать, что сеть Lightning – это не что иное, как приложение поверх системы Bitcoin, использующее Bitcoin-транзакции и Bitcoin Script. Не существует ни «Lightning-монеты», ни «блочной цепи Lightning». Помимо всех технических примитивов, протокол LN – это творческий подход к тому, чтобы получать больше выгод от системы Bitcoin, позволяя осуществлять произвольное число мгновенных платежей с мгновенными расчетами без необходимости доверять кому-либо еще, кроме сети Bitcoin.

Платежные каналы

Как мы видели в предыдущей главе, Алиса использовала программное обеспечение своего кошелька для создания платежного канала между собой и другим участником LN.

Канал ограничен только тремя вещами:

- во-первых, время, необходимое интернету для передачи нескольких сотен байт данных, необходимых протоколу для перемещения средств с одного конца канала на другой;
- во-вторых, емкость канала, то есть сумма биткойна, которая фиксируется на канале при его открытии;
- в-третьих, лимит максимального размера Bitcoin-транзакции также лимитирует число незавершенных (продолжающихся) маршрутизируемых платежей, которые могут осуществляться одновременно по каналу.

Платежные каналы обладают несколькими очень интересными и полезными свойствами:

- поскольку время обновления канала в первую очередь зависит от скорости связи в интернете, совершение платежа по платежному каналу может быть практически мгновенным;
- если канал открыт, то для совершения платежа не требуется подтверждение Bitcoin-блоков. На самом деле – до тех пор, пока вы и ваш партнер по каналу подчиняетесь протоколу – это не требует никакого взаимодействия с сетью Bitcoin или кем-либо еще, кроме вашего партнера по каналу;
- криптографический протокол построен таким образом, что между вами и вашим партнером по каналу практически не требуется доверия. Если ваш партнер становится невосприимчивым или пытается вас обмануть, то вы можете попросить систему Bitcoin выступить в качестве «суда» и вынести решение по умному контракту, о котором вы и ваш партнер ранее договорились;
- платежи, произведенные по платежному каналу, известны только вам и вашему партнеру. В этом смысле вы получаете конфиденциальность по сравнению с Bitcoin, где каждая транзакция является публичной. В блочной цепи Bitcoin станет видимым только конечный остаток, который представляет собой совокупность всех платежей в этом канале.

Системе Bitcoin было около пяти лет, когда талантливые разработчики впервые выяснили, как можно создавать двунаправленные, неограниченные по сроку, маршрутизируемые платежные каналы, и к настоящему времени известно по меньшей мере три разных метода.

В этой главе основное внимание будет уделено методу строительства канала, впервые описанному в техническом документе сети Lightning[18] Джозефом Пуном и Тадеушом Дриджей в 2015 году. Такие каналы называются каналами Пуна–Дриджа и являются методом строительства каналов, который в настоящее время используется в сети Lightning. Два других предложенных метода – это дуплексные микроплатежные каналы, введенные Кристианом Декером

[18] См. https://lightning.network/lightning-network-paper.pdf.

(Christian Decker) примерно в то же время, что и каналы Пуна–Дриджа и каналы eltoo, представленные в статье «eltoo: простой второслойный протокол для Bitcoin»[19] Кристианом Декером, Расти Расселом и (соавтором этой книги) Олаолувой Осунтокун в 2018 году.

Каналы eltoo обладают некоторыми интересными свойствами и упрощают имплементацию платежных каналов. Однако каналы eltoo требуют изменения языка Bitcoin Script и, следовательно, не могут быть имплементированы в главной сети (mainnet) Bitcoin по состоянию на 2020 год.

Мультиподписной адрес

Платежные каналы строятся поверх мультиподписных адресов «2 из 2».

В кратком изложении, мультиподписной адрес – это место, к чему биткойн привязан, а для его отвязывания и расходования требуется несколько подписей. В мультиподписном адресе «2 из 2», используемом в сети Lightning, участвуют два подписанта, и обоим необходимо поставить подпись, чтобы потратить средства.

Мультиподписные скрипты и мультиподписные адреса более подробно описаны в разделе «Мультиподписные скрипты» на стр. 392.

Финансовая транзакция

Основополагающим строительным блоком платежного канала является мультиподписной адрес «2 из 2». Один из двух партнеров канала будет финансировать платежный канал, отправляя биткойн на мультиподписной адрес. Такая транзакция называется *финансовой транзакцией* и регистрируется в блочной цепи Bitcoin[20].

Несмотря на то что финансовая транзакция является публичной, то, что это платежный канал Lightning, не очевидно до тех пор, пока он не будет закрыт, если только канал не будет объявлен публично. Каналы обычно объявляются публично маршрутизационными узлами, которые хотят пересылать платежи. Однако еще существуют неконвертируемые каналы, которые обычно создаются мобильными узлами, которые не принимают активного участия в маршрутизации. Более того, канальные платежи по-прежнему не видны никому, кроме партнеров по каналу, равно как и распределение остатка канала между ними.

Сумма, внесенная на мультиподписной адрес, называется *емкостью канала* и задает максимальную сумму, которая может быть отправлена по платежному каналу. Однако, поскольку средства могут отправляться туда и обратно, емкость канала не является верхним пределом того, сколько средств может проходить по каналу. Это связано с тем, что если емкость канала исчерпана платежами в одном направлении, то его можно снова использовать для отправки платежей в противоположном направлении.

---

[19] См. eltoo: A Simple Layer2 Protocol for Bitcoin, Christian Decker, Rusty Russel, https://blockstream.com/eltoo.pdf.

[20] Хотя в первоначальном техническом документе Lightning описывались каналы, финансируемые обоими партнерами канала, текущая спецификация, начиная с 2020 года, исходит из того, что только один партнер выделяет средства на канал. По состоянию на май 2021 года каналы Lightning с двойным финансированием являются экспериментальными в рамках имплементаци c-lightning сети LN.

Средства, отправленные на мультиподписной адрес, в финансовой транзакции иногда называются «привязанными в канале Lightning». Однако на практике средства в канале Lightning не «привязаны», а скорее «высвобождены». Средства канала Lightning более ликвидны, чем средства в блочной цепи Bitcoin, поскольку их можно потратить быстрее, дешевле и более конфиденциально. У переноса средств в сеть Lightning есть несколько недостатков (например, необходимость хранить их в «горячем» кошельке), но идея «привязывания средств» в Lightning абсолютно дезориентирует.

Пример плохой процедуры открытия канала

Если внимательно подумать о мультиподписных адресах «2 из 2», то можно понять, что внесение средств на такой адрес, по-видимому, сопряжено с некоторым риском. Что делать, если ваш партнер по каналу отказывается подписывать транзакцию, чтобы высвободить средства? Неужели они застряли навсегда? Давайте теперь рассмотрим этот сценарий и то, как протокол LN позволяет его избегать.

Алиса и Боб хотят создать платежный канал. Каждый из них создает пару из приватного/публичного ключей, а затем обменивается публичными ключами. Теперь они могут создать мультиподпись «2 из 2» с двумя публичными ключами, образовав основу для своего платежного канала.

Затем Алиса создает Bitcoin-транзакцию, отправляя несколько mBTC на мультиподписной адрес, созданный из публичных ключей Алисы и Боба. Если Алиса не предпримет никаких дополнительных шагов и просто выполнит широковещательную передачу этой транзакции, то она должна доверять Бобу, что тот предоставит свою подпись для оплаты с мультиподписного адреса. Боб, с другой стороны, имеет возможность шантажировать Алису, скрывая свою подпись и отказывая Алисе в доступе к ее средствам.

В целях предотвращения такой ситуации Алисе нужно будет создать дополнительную транзакцию, которая расходуется с мультиподписного адреса, выполняя возврат ее mBTC. Затем Алиса просит Боба подписать транзакцию возврата средств, *прежде* чем выполнить широковещательную передачу ее финансовой транзакции в сеть Bitcoin. Благодаря этому Алиса может вернуть средства, даже если Боб исчезнет или откажется сотрудничать.

Транзакция «возврата средств», которая защищает Алису, является первой из класса транзакций, именуемых *фиксационными транзакциями*, которые мы рассмотрим подробнее далее.

Фиксационная транзакция

Фиксационная транзакция (commitment transaction) – это транзакция, которая выплачивает каждому партнеру по каналу его остаток канала и обеспечивает, чтобы партнеры по каналу не обязаны были доверять друг другу. Подписывая фиксационную транзакцию, каждый партнер по каналу «фиксирует» текущий остаток и дает другому партнеру по каналу возможность вернуть свои средства в любое удобное для него время.

Удерживая подписанную фиксационную транзакцию, каждый партнер канала может получить свои средства даже без сотрудничества с другим партнером

канала. Это защищает его от исчезновения другого партнера по каналу, отказа сотрудничать или попытки обмануть, нарушив протокол платежного канала.

Фиксационная транзакция, которую Алиса подготовила в предыдущем примере, представляла собой возврат ее первоначального платежа на мультиподписной адрес. В более общем случае, однако, фиксационная транзакция разбивает средства платежного канала, выплачивая двум партнерам по каналу в соответствии с распределением (остатком), которое у каждого из них есть. Сначала Алиса владеет всем остатком, так что это простой возврат средств. Но по мере того, как средства поступают от Алисы к Бобу, они будут обмениваться подписями под новыми фиксационными транзакциями, то есть новым распределением остатка, при этом часть средств выплачивается Алисе, а часть – Бобу.

Давайте допустим, что Алиса открывает канал с емкостью 100 000 сатоши с Бобом. Первоначально Алисе принадлежит 100 000 сатоши, все средства на канале. Вот как работает протокол платежного канала.

1. Алиса создает новую пару из приватного/публичного ключа и сообщает Бобу, что она хочет открыть канал посредством сообщения `open_channel` (сообщение в протоколе LN).
2. Боб также создает новую пару из приватного/публичного ключа и соглашается принять канал от Алисы, отправляя свой публичный ключ Алисе через сообщение `accept_channel`.
3. Теперь Алиса создает финансовую транзакцию из своего кошелька, которая отправляет 100k сатоши на мультиподписной адрес с привязывающим скриптом: `2 <Pubkey Alice> <pubkey Bob> 2 CHECKMULTISIG`.
4. Алиса еще не выполнила широковещательную передачу этой финансовой транзакции, но отправляет Бобу ИД транзакции в сообщении `funding_created` вместе со своей подписью под фиксационной транзакцией Боба.
5. И Алиса, и Боб создают свою версию фиксационной транзакции. Эта транзакция будет потрачена из финансовой транзакции и отправит весь биткойн обратно на адрес, контролируемый Алисой.
6. Алисе и Бобу не нужно обмениваться этими фиксационными транзакциями, поскольку каждый из них знает, как они собраны, и может строить обе независимо (потому что они договорились о каноническом порядке входов и выходов). Им нужно только обменяться подписями.
7. Боб предоставляет подпись под фиксационной транзакцией Алисы и отправляет ее назад Алисе посредством сообщения `funding_signed`.
8. Теперь, когда был произведен обмен подписями, Алиса выполнит широковещательную передачу финансовой транзакции в сеть Bitcoin.

Следуя этому протоколу, Алиса не отказывается от владения своими 100k сатоши, даже если средства отправляются на мультиподписной адрес «2 из 2», для которого Алиса контролирует только один ключ. Если Боб перестанет отвечать Алисе, то она сможет выполнить широковещательную передачу своей фиксационной транзакции и получить свои средства назад. Ее единственные расходы – это комиссионные за внутрицепные транзакции. До тех пор, пока она подчиняется протоколу, это ее единственный риск при открытии канала.

После этого первоначального обмена фиксационные транзакции создаются всякий раз, когда изменяется остаток канала. Другими словами, всякий раз,

когда платеж отправляется между Алисой и Бобом, создаются новые фиксационные транзакции и происходит обмен подписями. Каждая новая фиксационная транзакция кодирует последний остаток средств между Алисой и Бобом.

Если Алиса захочет отправить 30k сатоши Бобу, то оба создадут новую версию своих фиксационных транзакций, которые теперь будут выплачивать 70k сатоши Алисе и 30k сатоши Бобу. Кодируя новый остаток для Алисы и Боба, новые фиксационные транзакции являются средством, с помощью которого платеж «отправляется» по каналу.

Теперь, когда мы понимаем фиксационные транзакции, давайте рассмотрим несколько более тонких деталей. Вы, возможно, заметили, что этот протокол оставляет Алисе или Бобу возможность обмануть.

Обман с предыдущим состоянием

Сколько фиксационных транзакций Алиса удерживает после того, как она заплатит Бобу 30k сатоши? У нее есть две: первоначальная, которая платит ей 100k сатоши, и более поздняя, которая платит ей 70k сатоши, и Бобу 30k сатоши.

В канальном протоколе, который мы видели до сих пор, Алисе ничто не мешает опубликовать предыдущую фиксационную транзакцию. Желая обмануть, Алиса может опубликовать фиксационную транзакцию, которая предоставляет ей 100k сатоши. Поскольку эта фиксационная транзакция была подписана Бобом, то он не сможет помешать Алисе их перечислить.

Необходим какой-то механизм запрета, который запрещал бы Алисе публиковать старую фиксационную транзакцию. Давайте теперь выясним, как этого можно достичь и как это позволяет сети Lightning работать, не требуя никакого доверия между Алисой и Бобом.

Поскольку система Bitcoin устойчива к цензуре, никто не сможет помешать кому-либо опубликовать старую фиксационную транзакцию. В целях предотвращения этой формы обмана фиксационные транзакции строятся таким образом, что в случае передачи старой транзакции обманщик может быть оштрафован. Делая штраф достаточно большим, мы создаем сильный стимул против обмана, и это делает систему безопасной.

Применение штрафа заключается в предоставлении обманутой стороне возможности потребовать остаток обманщика. Таким образом, если кто-то попытается обмануть путем выполнения широковещательной передачи старой фиксационной транзакции, в которой ему выплачивается более высокий остаток средств, чем ему причитается, другая сторона может его оштрафовать, забрав как свой собственный остаток, так и остаток обманщика. Обманщик теряет все.

Вы, возможно, заметили, что если Алиса почти полностью истощит остаток своего канала, то она может попробовать обмануть с небольшим риском. Штраф Боба не был бы таким болезненным, если бы ее остаток канала был низким. Для того чтобы это предотвратить, протокол Lightning требует, чтобы каждый партнер по каналу поддерживал минимальный остаток в канале (именуемый *резервом*), чтобы у них всегда была «личная заинтересованность».

Давайте еще раз пройдемся по сценарию строительства канала, добавив механизм штрафных санкций для защиты от обмана.

1. Алиса создает канал с Бобом и вкладывает в него 100k сатоши.
2. Алиса отправляет Бобу 30k сатоши.
3. Алиса пытается обмануть Боба, лишая его заработанных 30k сатоши, опубликовав старую фиксационную транзакцию, требующую для себя полных 100k сатоши.
4. Боб обнаруживает обман и штрафует Алису, забирая себе все 100k сатоши.
5. Боб в итоге получает 100k сатоши, получая 70k сатоши за то, что поймал Алису на обмане.
6. Алиса остается с 0 сатоши.
7. Пытаясь обмануть Боба на 30k сатоши, она теряет 70k сатоши, которыми владела.

Благодаря сильному механизму штрафных санкций у Алисы нет соблазна обманывать, публикуя старую фиксационную транзакцию, потому что она рискует потерять весь свой остаток средств.

В главе 12 книги «Освоение системы Bitcoin» Андреас Антонопулос (соавтор данной книги) излагает это следующим образом: «Ключевой характеристикой системы Bitcoin является то, что как только транзакция становится валидной, она остается валидной и не истекает. Единственный способ отменить транзакцию – это дважды потратить ее входы с помощью еще одной транзакции до того, как она будет добыта».

Теперь, когда мы понимаем, зачем нужен механизм штрафов и как он предотвратит обман, давайте посмотрим, как он работает, в деталях.

Обычно фиксационная транзакция имеет по меньшей мере два выхода, оплачивая каждому партнеру по каналу. Мы изменяем это, добавляя к одному из платежей *задержку привязки ко времени* и *отзывной секрет*. Привязка ко времени не позволяет владельцу выхода тратить его немедленно, как только фиксационная транзакция была включена в блок. Отзывной секрет позволяет любой стороне немедленно потратить этот платеж, минуя привязку ко времени.

Итак, в нашем примере Боб удерживает фиксационную транзакцию, которая немедленно выполняет платеж Алисе, но его собственный платеж задерживается и может быть отозван. Алиса также удерживает фиксационную транзакцию, но у нее все наоборот: она немедленно платит Бобу, но ее собственный платеж задерживается и может быть отозван.

Два партнера по каналу держат половину отзывного секрета, в результате чего ни один из них не знает всего секрета. Если они делятся своей половиной, то другой партнер по каналу имеет полный секрет и может использовать его для исполнения условия отзыва. При подписании новой фиксационной транзакции каждый партнер канала отзывает предыдущую фиксацию, предоставляя другой стороне свою половину отзывного секрета.

Мы рассмотрим механизм отзыва подробнее в разделе «Отзыв и рефиксация» на стр. 197, где узнаем подробности о том, как создаются и используются отзывные секреты.

Проще говоря, Алиса подписывает новую фиксационную транзакцию Боба только в том случае, если Боб предлагает свою половину отзывного секрета для предыдущей фиксации. Боб подписывает новую фиксационную транзакцию Алисы, только если она предоставит ему свою половину отзывного секрета из предыдущей фиксации.

С каждой новой фиксацией они обмениваются необходимым «штрафным» секретом, который позволяет им эффективно *отзывать* предыдущую фиксационную транзакцию, делая ее невыгодной для передачи. По сути, они лишают возможности использовать старые фиксации (в сущности, обязательства), подписывая новые. Мы имеем в виду то, что, хотя технически все еще возможно использовать старые фиксации, механизм штрафных санкций делает это экономически нерациональным.

Привязка ко времени устанавливается равной числу блоков вплоть до 2016 (примерно две недели). Если один из партнеров канала опубликует фиксационную транзакцию, не сотрудничая с другим партнером, то ему придется подождать этого числа блоков (к примеру, две недели), чтобы получить свой остаток средств. Другой партнер по каналу может в любое время потребовать свой собственный остаток. Кроме того, если опубликованная ими фиксация была ранее отозвана, то партнер по каналу также может немедленно истребовать остаток средств обманщика, минуя привязку ко времени и оштрафовав обманщика.

Привязка ко времени регулируется и может согласовываться между партнерами по каналу. Обычно для каналов с большей емкостью требуется больше времени, а для каналов меньшего объема – меньше, чтобы выровнять стимулы со стоимостью средств.

Для каждого нового обновления остатка канала необходимо создавать и сохранять новые фиксационные транзакции и новые отзывные секреты. До тех пор, пока канал остается открытым, необходимо сохранять все отзывные секреты, когда-либо созданные для канала, поскольку они могут понадобиться в будущем. К счастью, секретов довольно мало, и хранить их нужно только партнерам по каналу, а не всей сети. Более того, благодаря умному механизму выведения, используемому для получения отзывных секретов, нам нужно хранить только самый последний секрет, потому что из него могут быть выведены предыдущие секреты (см. «Обман и наказание на практике» на стр. 197).

Тем не менее управление и хранение отзывных секретов являются одной из наиболее сложных частей узлов Lightning, которая требует, чтобы операторы узлов поддерживали резервные копии.

Такие технологии, как службы сторожевых вышек или смена протокола строительства канала на протокол eltoo, могут стать будущими стратегиями по смягчению этих проблем и уменьшению необходимости в отзывных секретах, штрафных транзакциях и резервном копировании каналов.

Алиса может закрыть канал в любое время, если Боб не ответит, истребовав свою справедливую долю остатка средств. После публикации *последней* фиксационной транзакции внутри цепи Алиса должна дождаться истечения срока привязки, прежде чем она сможет потратить свои средства из фиксационной

транзакции. Как мы увидим позже, есть более простой способ закрыть канал без ожидания, если Алиса и Боб оба находятся онлайн и кооперируют в закрытии канала с правильным распределением остатка. Но фиксационные транзакции, хранимые каждым партнером по каналу, действуют как сейф, обеспечивая невозможность потери средств, в случае если возникнут проблемы с их партнером по каналу.

Объявление канала

Партнеры канала могут согласиться объявить о своем канале всей сети Lightning, сделав его *публичным каналом*. В целях объявления о канале они используют эпидемический протокол сети Lightning, чтобы сообщить другим узлам о существовании, емкости и комиссионных канала.

Публичное объявление каналов позволяет другим узлам использовать их для маршрутизации платежей, тем самым также генерируя комиссионные за маршрутизацию для партнеров канала.

В противоположность этому партнеры канала могут решить не объявлять канал, сделав его *необъявленным* каналом.

Вы, возможно, встретите термин «приватный канал», используемый для описания необъявленного канала. Мы избегаем применять этот термин, потому что он вводит в заблуждение и создает ложное ощущение конфиденциальности. Хотя необъявленный канал не будет известен другим, пока он используется, его существование и емкость будут раскрыты, когда канал закроется, потому что эти детали будут видны внутри цепи в окончательной расчетной транзакции. Его существование также может просочиться многими другими путями, поэтому мы избегаем называть его «приватным».

Необъявленные каналы по-прежнему используются для маршрутизации платежей, но только узлами, которые знают об их существовании или получают «маршрутизационные подсказки» о пути, в который входит необъявленный канал.

Когда канал и его емкость публично объявляются с использованием эпидемического протокола, объявление может также включать информацию о канале (метаданные), такую как комиссионные за маршрутизацию и продолжительность привязки ко времени.

Когда новые узлы присоединяются к сети Lightning, они собирают сообщения о каналах, передаваемые по эпидемическому протоколу от своих одноранговых узлов, создавая внутреннюю карту сети Lightning. Затем эту карту можно использовать для отыскания путей для платежей, соединяя каналы друг с другом из конца в конец.

Закрытие канала

Лучший способ закрыть канал… – его не закрывать! Для открытия и закрытия каналов требуется внутрицепная транзакция, которая повлечет за собой комиссионные за транзакцию. Поэтому лучше всего держать каналы открытыми как можно дольше. Вы можете продолжать использовать свой канал для со-

вершения и пересылки платежей до тех пор, пока у вас на вашем конце канала есть достаточная емкость. Но даже если вы отправите весь остаток на другой конец канала, то вы можете затем использовать канал для получения платежей от вашего партнера по каналу. Эта концепция использования канала в одном направлении, а затем использования его в противоположном направлении называется «перебалансировкой», и мы рассмотрим ее подробнее в другой главе. За счет перебалансировки канала его можно поддерживать открытым практически неограниченное время и использовать для практически неограниченного числа платежей.

Однако иногда закрытие канала желательно или необходимо. Например:

- вы хотите уменьшить остаток средств, хранящийся в ваших каналах Lightning, по соображениям безопасности и хотите отправить средства в «холодное хранилище»;
- ваш партнер по каналу перестает отвечать на запросы в течение длительного времени, и вы больше не можете использовать канал;
- канал используется нечасто, потому что узел вашего партнера по каналу не имеет хорошей связности, поэтому вы хотите использовать средства для другого канала, где узел имеет хорошую связность;
- ваш партнер по каналу нарушил протокол из-за ошибки программного обеспечения либо намеренно, вынудив вас закрыть канал с целью защиты ваших средств.

Есть три пути закрытия платежного канала:

- взаимное закрытие (хороший путь);
- принудительное закрытие (плохой путь);
- нарушение протокола (уродливый путь).

Каждый из этих методов полезен в различных обстоятельствах, которые мы рассмотрим в следующих разделах данной главы. Например, если ваш партнер по каналу находится в офлайне, то вы не сможете последовать по «хорошему пути», потому что взаимное закрытие невозможно без сотрудничающего партнера. Обычно ваше программное обеспечение LN автоматически выбирает наилучший механизм закрытия, доступный в данных обстоятельствах.

Взаимное закрытие (хороший путь)

Взаимное закрытие – это когда оба партнера по каналу соглашаются закрыть канал, и это предпочтительный метод закрытия канала.

Когда вы решите, что хотите закрыть канал, ваш узел LN сообщит вашему партнеру по каналу о вашем намерении. Теперь и ваш узел, и узел партнера по каналу работают вместе, чтобы закрыть канал. Никакие новые попытки маршрутизации не будут приниматься ни от одного из партнеров канала, и любые текущие попытки маршрутизации будут улажены или удалены по истечении тайм-аута. Финализация попыток маршрутизации требует времени, поэтому для завершения взаимного закрытия также может потребоваться некоторое время.

После того как больше не будет ожидающих своей очереди попыток маршрутизации, узлы кооперируют в подготовке *завершающей транзакции*. Эта транзакция аналогична фиксационной транзакции: она кодирует последний остаток канала, но выходы НЕ обременены привязкой ко времени.

Внутрицепные транзакционные комиссионные за закрывающую транзакцию оплачиваются партнером канала, который открыл канал, а не тем, кто инициировал процедуру закрытия. Используя оценщик внутрицепных комиссионных, партнеры канала договариваются о соответствующей комиссии и оба подписывают закрывающую транзакцию.

После выполнения широковещательной передачи и подтверждения закрывающей транзакции сетью Bitcoin канал фактически закрывается, и каждый партнер канала получает свою долю остатка канала. Несмотря на время ожидания, взаимное закрытие обычно происходит быстрее, чем принудительное закрытие.

Принудительное закрытие (плохой путь)

Принудительное закрытие – это когда один партнер по каналу пытается закрыть канал без согласия другого партнера по каналу.

Обычно это происходит, когда один из партнеров по каналу недоступен, поэтому взаимное закрытие невозможно. В этом случае вы инициируете принудительное закрытие, чтобы в одностороннем порядке закрыть канал и «освободить» средства.

В целях инициирования принудительного закрытия вы можете просто опубликовать последнюю фиксационную транзакцию, имеющуюся у вашего узла. В конце концов, именно для этого и нужны фиксационные транзакции – они дают гарантию того, что вам не нужно доверять своему партнеру по каналу, чтобы получить остаток вашего канала.

Как только вы выполните широковещательную передачу последней фиксационной транзакции в сеть Bitcoin и она будет подтверждена, она создаст два расходуемых выхода, один для вас и один для вашего партнера. Как мы обсуждали ранее, сеть Bitcoin не имеет возможности узнать, была ли эта транзакция самой последней фиксационной транзакцией или старой, которая была опубликована для кражи у вашего партнера. Следовательно, эта фиксационная транзакция даст небольшое преимущество вашему партнеру. Партнер, инициировавший принудительное закрытие, будет иметь свои выходы, обремененные привязкой ко времени, а выходы другого партнера будут немедленно израсходованы. В случае если вы выполнили широковещательную передачу более ранней фиксационной транзакции, задержка привязки ко времени дает вашему партнеру возможность оспорить транзакцию, используя отзывной секрет, и оштрафовать вас за обман.

При публикации фиксационной транзакции во время принудительного закрытия внутрицепные комиссионные будут выше, чем при взаимном закрытии по нескольким причинам.

1. Когда была согласована фиксационная транзакция, партнеры по каналу не знали, какими будут внутрицепные комиссионные в будущем, когда будет выполнятся широковещательная передача транзакции. Поскольку комиссионные нельзя изменить без изменения выходов фиксационной транзакции (для которой требуются обе подписи) и поскольку принудительное закрытие происходит, когда партнер по каналу недоступен для подписания, разработчики протокола решили быть очень щедрыми в отношении комиссионного тарифа, включаемого в фиксационные транзакции. Он может быть в пять раз выше, чем предполагают оценщики комиссионных на момент согласования фиксационной транзакции.

2. Фиксационная транзакция включает дополнительные выходы для любых попыток маршрутизации, ожидающих своей очереди, с использованием контрактов с привязкой к хешу и времени (hash time-locked contract, аббр. HTLC), что делает фиксационную транзакцию больше (в байтах), чем транзакция взаимного закрытия. Более крупные транзакции требуют бóльших комиссионных.
3. Любые незавершенные попытки маршрутизации должны быть урегулированы внутри цепи, что приводит к дополнительным внутрицепным транзакциям.

Контракты с привязкой к хешу и времени (HTLC) будут подробно рассмотрены в разделе «Контракты с привязкой к хешу и времени» на стр. 216. На данный момент мы будем исходить из того, что это платежи, которые маршрутизируются через сеть Lightning, а не платежи, осуществляемые непосредственно между двумя участниками канала. Указанные HTLC-контракты передаются в качестве дополнительных выходов в фиксационных транзакциях, тем самым увеличивая размер транзакций и внутрицепных комиссионных.

В общем случае принудительное закрытие не рекомендуется без меньшей необходимости. Ваши средства будут привязаны на более длительный срок, и человеку, который открыл канал, придется платить более высокие комиссионные. Кроме того, вам, возможно, придется заплатить комиссионные за прерывание или урегулирование попыток маршрутизации, даже если вы не открывали канал.

Если вам известен партнер по каналу, то вы можете подумать о том, чтобы связаться с этим физическим лицом или компанией, дабы узнать, почему их узел Lightning не работает, и попросить их перезапустить его, чтобы иметь возможность добиться взаимного закрытия канала.

Принудительное закрытие следует рассматривать только в качестве последнего средства.

Нарушение протокола (уродливый путь)

Нарушение протокола – это когда ваш партнер по каналу пытается вас обмануть, намеренно или нет, публикуя устаревшую фиксационную транзакцию в блочной цепи Bitcoin, по сути, инициируя (нечестное) принудительное закрытие со своей стороны.

Для того чтобы обнаруживать такую ситуацию, ваш узел должен быть онлайн и следить за новыми блоками и транзакциями в блочной цепи Bitcoin.

Поскольку платеж вашего партнера по каналу будет привязан привязкой ко времени, у вашего узла есть некоторое время, чтобы обнаружить нарушение протокола и опубликовать *штрафную транзакцию* до того, как срок привязки истечет.

Если вы успешно обнаружите нарушение протокола и наложите штраф, то получите все средства в канале, включая средства вашего партнера по каналу.

В этом сценарии закрытие канала будет довольно быстрым. Вам придется заплатить внутрицепные комиссионные за публикацию штрафной транзакции, но ваш узел может установить эти комиссионные в соответствии с оценкой комиссионных и не переплачивать. Как правило, вы захотите платить более вы-

сокие комиссионные, чтобы как можно скорее гарантировать подтверждение. Однако, поскольку вы в конечном итоге получите все средства обманщика, по сути, именно обманщик и будет платить за эту транзакцию.

Если вам не удастся обнаружить нарушение протокола и срок привязки истечет, то вы получите только те средства, которые были выделены вам в соответствии с фиксационной транзакцией, опубликованной вашим партнером. Любые средства, которые вы получили после этого, будут украдены вашим партнером. Если вам выделен какой-либо остаток, то вам придется заплатить внутрицепные комиссионные, чтобы забрать этот остаток.

Как и в случае принудительного закрытия, все попытки маршрутизации, ожидающие своей очереди, также должны быть урегулированы в фиксационной транзакции.

Нарушение протокола может быть исполнено быстрее, чем взаимное закрытие, потому что вы не ждете, чтобы договориться со своим партнером о закрытии, и быстрее, чем принудительное закрытие, потому что вам не нужно ждать истечения срока вашей привязки ко времени.

Теория игр предсказывает, что обман не является привлекательной стратегией, потому что обманщика легко обнаружить, и обманщик рискует потерять *все* свои средства, хотя он и может получить, но только то, что у него было в более раннем состоянии. Кроме того, по мере развития сети Lightning и широкого распространения сторожевых вышек обманщики будут обнаруживаться третьей стороной, даже если обманутый партнер по каналу находится в офлайновом режиме.

Таким образом, мы не рекомендуем обманывать. Однако мы рекомендуем всем, кто поймает обманщика, его штрафовать, забирая его средства.

Тогда как отлавливать обманщика или нарушение протокола в своей повседневной деятельности? Это делается с помощью программного обеспечения, которое выполняет мониторинг публичной блочной цепи Bitcoin в отношении внутрицепных транзакций, соответствующих любым фиксационным транзакциям по любому из ваших каналов. Это программное обеспечение относится к одному из трех типов:

- должным образом обслуживаемый узел Lightning, работающий 24/7;
- универсальный узел сторожевой вышки, который вы выполняете для просмотра своих каналов;
- сторонний узел сторожевой вышки, которому вы платите за надзор за вашими каналами.

Помните, что фиксационная транзакция имеет тайм-аут, указанный в заданном числе блоков, максимум до 2016 блоков. В случае если вы запустите свой узел Lightning хотя бы раз до истечения тайм-аута, он будет перехватывать все попытки обмана. Не рекомендуется идти на подобного рода риск; важно поддерживать бесперебойную работу узла в хорошем состоянии (см. «Почему для оперирования узлом Lightning важна надежность?» на стр. 126).

Счета

Большинство платежей в сети Lightning начинаются со счета, сгенерированного получателем платежа. В нашем предыдущем примере Боб создает счет, чтобы запросить платеж у Алисы.

Существует способ отправить незапрошенный платеж без счета, используя обходной путь в протоколе под названием `keysend`. Мы рассмотрим его в разделе «Спонтанные платежи keysend» на стр. 279.

Счет – это простая платежная инструкция, содержащая такую информацию, как уникальный идентификатор платежа (именуемый *платежным хешем*), получателя, сумму и необязательное текстовое описание.

Наиболее важной частью счета является платежный хеш, который позволяет платежу проходить по нескольким каналам *атомарным* путем. Атомарный в информатике означает любое действие или изменение состояния, которое либо завершается успешно, либо вообще не выполняется – промежуточное состояние или частичное действие невозможно. В сети Lightning это означает, что платеж либо проходит весь путь, либо не проходит совсем. Он не может быть завершен частично таким образом, чтобы промежуточный узел на пути мог получить платеж и его сохранить. Не существует такого понятия, как «частичный платеж» или «частично успешный платеж».

Счета не передаются по сети Lightning. Вместо этого они общаются «вне полосы», используя любой другой механизм связи. Это похоже на то, как Bitcoin-адреса передаются отправителям за пределами сети Bitcoin: в виде QR-кода, по электронной почте или в виде текстового сообщения. Например, Боб может предоставить Алисе счет Lightning в виде QR-кода, по электронной почте или по любому другому каналу обмена сообщениями.

Счета обычно кодируются либо в виде длинной строки в кодировке bech32, либо в виде QR-кода, который сканируется с помощью кошелька Lightning на смартфоне. Счет содержит запрашиваемую сумму биткойна и подпись получателя. Отправитель использует подпись для извлечения публичного ключа (также именуемого ИД узла) получателя, чтобы отправитель знал, куда отправлять платеж.

Вы заметили, как это контрастирует с системой Bitcoin и как используются разные термины? В системе Bitcoin получатель передает адрес отправителю. В сети Lightning получатель создает счет и отправляет счет отправителю. В Bitcoin отправитель отправляет средства на определенный адрес. В Lightning отправитель оплачивает счет, и платеж направляется получателю. Биткойн основан на концепции «адреса», а Lightning – это платежная сеть, основанная на концепции «счета». В Bitcoin мы создаем «транзакцию», тогда как в Lightning мы отправляем «платеж».

Платежный хеш и прообраз

Наиболее важной частью счета является платежный хеш. При создании счета Боб создаст платежный хеш следующим образом:

Боб выбирает случайное число r. Это случайное число называется прообразом, или платежным секретом.

Боб использует SHA-256 для вычисления хеша H из r, именуемого платежным хешем: H = SHA-256(r).

Термин «*прообраз*» происходит из математики. В любой функции $y = f(x)$ множество входов, которые производят определенное значение y, называется прообразом y. В этом случае функция представляет собой алгоритм хеширования SHA-256, и любое значение r, которое создает хеш H, называется прообразом.

Не существует известного способа найти значение, обратное SHA-256 (т. е. вычислить прообраз из хеша). Только Боб знает значение r, так что это секрет Боба. Но как только Боб раскрывает r, любой, у кого есть хеш H, может проверить, что r является правильным секретом, вычислив SHA-256 (r) и убедившись, что он соответствует H.

Процесс оплаты в сети Lightning является безопасным только в том случае, если r выбирается полностью случайным образом и не является предсказуемым. Эта безопасность основана на том факте, что хеш-функции не могут быть инвертированы или практически решены атакой грубой силой, и, следовательно, никто не может найти r из H.

Дополнительные метаданные

Счета могут дополнительно включать другие полезные метаданные, такие как краткое текстовое описание. Если у пользователя есть несколько счетов для оплаты, то пользователь может прочитать описание и получить напоминание о том, о чем идет речь в счете.

Счет также может содержать некоторые маршрутизационные подсказки, которые позволяют отправителю использовать необъявленные каналы для построения маршрута к получателю. Маршрутизационные подсказки также могут использоваться для указания публичных каналов, например каналов, которые, как известно получателю, имеют достаточную емкость для маршрутизирования платежа.

В случае если узел Lightning отправителя не сможет отправить платеж по сети Lightning, то счета могут дополнительно включать внутрицепной Bitcoin-адрес в качестве запасного варианта.

Хотя всегда можно «откатить назад» к внутрицепной Bitcoin-транзакции, на самом деле вместо этого лучше открыть новый канал для получателя. Если вам приходится взимать внутрицепные комиссионные за совершение платежа, то вы также можете взимать эти комиссионные, чтобы открыть канал и произвести платеж через Lightning. После совершения платежа у вас остается открытый канал, который имеет ликвидность на стороне получателя и может быть использован для маршрутизирования платежей обратно на ваш узел Lightning в будущем. Одна внутрицепная транзакция дает вам платеж и канал для дальнейшего использования.

Счета Lightning содержат дату истечения срока. Поскольку получатель должен сохранять прообраз r для каждого выставленного счета, полезно, чтобы срок действия счетов истекал, чтобы эти прообразы не нужно было хранить вечно. После того как срок действия счета истекает или он оплачен, получатель может отказаться от прообраза.

Доставка платежа

Мы видели, как получатель создает счет, содержащий платежный хеш. Этот платежный хеш будет использоваться для перемещения платежа по ряду платежных каналов от отправителя к получателю, даже если между ними нет прямого платежного канала.

В следующих нескольких разделах мы углубимся в идеи и методы, которые используются для доставки платежей по сети Lightning, и будем применять все концепции, которые мы представили до сих пор.

Во-первых, давайте рассмотрим коммуникационный протокол сети Lightning.

Эпидемический протокол обмена сообщениями между одноранговыми узлами

Как мы упоминали ранее, когда создается платежный канал, партнеры канала имеют возможность сделать его публичным, объявив о его существовании и деталях для всей сети Lightning.

Объявления канала передаются по одноранговому *эпидемическому протоколу* обмена сообщениями (протоколу сплетен). Одноранговый протокол – это коммуникационный протокол, в котором каждый узел подсоединяется к случайной подборке других узлов в сети, обычно через TCP/IP. Каждый узел, который напрямую подсоединен (через TCP/IP) к вашему узлу, называется вашим одноранговым узлом. Ваш узел, в свою очередь, является одним из его одноранговых узлов. Имейте в виду, что когда мы говорим, что ваш узел подсоединен к другим одноранговым узлам, мы не имеем в виду, что у вас есть платежные каналы, а только то, что вы подсоединены по эпидемическому протоколу.

После открытия канала узел может выбрать отправку объявления о канале посредством сообщения channel_announcement своим одноранговым узлам. Каждый одноранговый узел валидирует информацию из сообщения channel_announcement и верифицирует, что финансовая транзакция подтверждается в блочной цепи Bitcoin. После верификации узел перешлет эпидемическое сообщение своим собственным одноранговым узлам, а они перешлют его своим одноранговым узлам и т. д., распространяя объявление по всей сети. Во избежание чрезмерного обмена информацией объявление канала пересылается каждым узлом только в том случае, если он еще не пересылал это объявление ранее.

Эмидемический протокол также используется для объявления информации об известных узлах с помощью сообщения node_announcement. Для того чтобы это сообщение было переадресовано, узел должен иметь по меньшей мере один публичный канал, объявленный в эмидемическом протоколе, опять же, чтобы избежать чрезмерного коммуникационного трафика.

Платежные каналы имеют различные метаданные, которые полезны для других участников сети. Эти метаданные в основном используются для принятия решений о маршрутизации. Поскольку узлы могут временами изменять метаданные своих каналов, эта информация передается в сообщении

channel_update. Эти сообщения будут пересылаться примерно только четыре раза в день (в расчете на канал), чтобы предотвратить чрезмерный обмен информацией. Эмидемический протокол также содержит целый ряд запросов и сообщений для первоначальной синхронизации узла с представлением сети или для обновления представления узла после непродолжительного перехода в офлайн.

Основная проблема для участников сети Lightning заключается в том, что топологическая информация, передаваемая эпидемическим протоколом, является лишь частичной. Например, емкость платежных каналов передается по эпидемическому протоколу посредством сообщения channel_announcement. Однако эта информация не так полезна, как фактическое распределение емкости с точки зрения локального остатка между двумя партнерами по каналу. Узел может пересылать только столько биткойна, сколько ему фактически принадлежит (локальный остаток) в пределах этого канала.

Хотя сеть Lightning и могла бы быть сконструирована для обмена информацией об остатке каналов и точной топологии, это не было сделано по нескольким причинам:

- для того чтобы защитить конфиденциальность пользователей, в нем не раскрываются все финансовые транзакции и платежи. Обновление остатка канала покажет, что платеж переместился по каналу. Эта информация может быть сопоставлена, чтобы выявить все источники и пункты назначения платежей;
- масштабировать объем платежей, которые могут осуществляться с помощью сети Lightning Network. Помните, что сеть Lightning была создана в первую очередь потому, что уведомление каждого участника о каждом платеже плохо масштабируется. Таким образом, сеть Lightning не может быть сконструирована таким образом, чтобы обмениваться обновлениями остатка каналов между участниками;
- сеть Lightning – это динамическая система. Она меняется постоянно и часто. Добавляются узлы, отключаются другие узлы, меняются остатки и т. д. Даже если все всегда передается, информация будет действовать только в течение короткого промежутка времени. На самом деле информация часто устаревает к моменту ее получения.

Мы рассмотрим детали эмидемического протокола в следующей главе.

А пока важно знать только то, что эпидемический протокол существует и что он используется для обмена информацией о топологии сети Lightning. Эта информация о топологии имеет решающее значение для доставки платежей через сеть платежных каналов.

Отыскание пути и маршрутизация

Платежи в сети Lightning пересылаются по *пути*, состоящему из каналов, связывающих одного участника с другим, от источника платежа до места назначения платежа. Процесс нахождения пути от источника к месту назначения называется *отысканием пути*. Процесс использования этого пути для совершения платежа называется *маршрутизацией*.

Частая критика сети Lightning заключается в том, что маршрутизация не решена, либо даже что эта задача «неразрешима». На самом деле маршрутизация тривиальна. С другой стороны, отыскание пути – задача действительно сложная. Эти два термина часто путают, и их необходимо четко определить, чтобы установить, какую задачу мы пытаемся решить.

Как мы увидим далее, сеть Lightning в настоящее время использует протокол на основе источника для отыскания путей и протокол луковичной маршрутизации для маршрутизирования платежей. «Основанный на источнике» означает, что отправитель платежа должен найти путь через сеть к намеченному месту назначения. «Луковично маршрутизируемый» означает, что элементы пути являются многослойными (как у луковицы), причем каждый слой зашифрован так, что его может видеть только один узел за раз. Мы обсудим луковичную маршрутизацию в следующем разделе.

Отыскание пути на основе источника

Если бы мы знали точные остатки каналов каждого канала, то мы могли бы легко вычислить путь оплаты, используя любой из стандартных алгоритмов отыскания путей, преподаваемых на любом курсе информатики. Это можно было бы даже решить таким образом, чтобы оптимизировать комиссионные, выплачиваемые узлам за пересылку платежа.

Однако информация об остатке всех каналов не является и не может быть известна всем участникам сети. Нам нужны более инновационные стратегии отыскания пути.

Имея лишь частичную информацию о топологии сети, отыскание пути является реальной проблемой, и в этой части сети Lightning все еще проводятся активные исследования. Тот факт, что задача отыскания пути в сети Lightning не «решена полностью», является основным предметом критики в отношении технологии.

Одно из распространенных критических замечаний в отношении отыскания пути в сети Lightning заключается в том, что эта задача неразрешима, поскольку она эквивалентна NP-полной задаче о коммивояжере (traveling salesperson problem, аббр. TSP), фундаментальной задаче в теории вычислительной сложности. На самом деле отыскание пути в Lightning не эквивалентно TSP и относится к другому классу задач. Мы успешно решаем такого рода задачи (отыскание путей на графе с неполной информацией) всякий раз, когда просим Google указать нам направление движения, чтобы избежать заторов на дороге. Мы также успешно решаем эту задачу всякий раз, когда направляем платеж по сети Lightning.

Отыскание пути и маршрутизация могут быть имплементированы несколькими разными способами, и в сети Lightning могут сосуществовать многочисленные алгоритмы отыскания пути и маршрутизации, подобно тому, как в интернете существуют многочисленные алгоритмы отыскания пути и маршрутизации. Отыскание пути на основе источника является одним из многих возможных решений и успешным в текущем масштабе сети Lightning.

Стратегия отыскания пути, имплементируемая в настоящее время узлами Lightning, заключается в итеративном переборе путей до тех пор, пока не бу-

дет найден один, обладающий достаточной ликвидностью для пересылки платежа. Это итеративный процесс проб и ошибок до тех пор, пока не будет достигнут успех или не будет найден путь. Алгоритм в настоящее время не обязательно приводит к пути с наименьшими комиссионными. Хотя он и не оптимален и, безусловно, может быть улучшен, даже эта упрощенная стратегия работает довольно хорошо.

Это «прощупывание» выполняется узлом или кошельком Lightning и не видно непосредственно пользователю. Пользователь, возможно, поймет, что происходит прощупывание, только в том случае, если платеж не завершится мгновенно.

В интернете мы используем интернет-протокол и алгоритм IP-пересылки для пересылки интернет-пакетов от отправителя к получателю. Хотя эти протоколы обладают приятным свойством, позволяющим интернет-хостам совместно находить путь для передачи информации через интернет, мы не можем реиспользовать и принять этот протокол для пересылки платежей в сети Lightning. В отличие от интернета, платежи Lightning должны быть атомарными, а остатки каналов должны оставаться приватными. Кроме того, емкость канала в Lightning часто меняется, в отличие от интернета, где емкость (пропускная способность) соединения относительно статична. Эти ограничения требуют новых стратегий.

Конечно, отыскание пути тривиально, если мы хотим заплатить нашему партнеру по прямому каналу и у нас для этого достаточно средств на нашей стороне канала. Во всех остальных случаях наш узел использует для отыскания пути информацию из эпидемического протокола. Сюда входят известные в настоящее время публичные платежные каналы, известные узлы, известная топология (каким образом известные узлы подсоединены), известная емкость канала и известная политика взимания комиссионных, установленная владельцами узлов.

Луковичная маршрутизация

Сеть Lightning использует протокол луковичной маршрутизации, аналогичный знаменитой сети Tor (The Onion Router, т. е. луковичный маршрутизатор). Используемый в Lightning протокол луковичной маршрутизации называется форматом SPHINX Mix[21], который будет подробно объяснен в следующей главе.

Формат SPHINX Mix луковичной маршрутизации в Lightning похож на маршрутизацию сети Tor только по концепции, но как протокол, так и имплементация полностью отличаются от тех, которые используются в сети Tor.

[21] *Джордж Данезис и Ян Голдберг.* Sphinx: компактный и доказуемо безопасный формат Mix (George Danezis and Ian Goldberg, “Sphinx: A Compact and Provably Secure Mix Format”, in IEEE Symposium on Security and Privacy (New York: IEEE, 2009), 269–282).

Платежный пакет, используемый для маршрутизации, называется «луковицей»[22].

Давайте воспользуемся аналогией с луковицей, чтобы проследить за маршрутизируемым платежом. На своем маршруте от отправителя платежа (плательщика) к получателю платежа (конечному получателю) луковица передается от узла к узлу по пути. Отправитель создает всю луковицу целиком, начиная с центра наружу. Сначала отправитель создает платежную информацию для (конечного) получателя платежа и шифрует ее с помощью слоя шифрования, который может быть дешифрован только получателем. Затем отправитель оборачивает этот слой инструкциями для узла в пути, *непосредственно предшествующем конечному получателю*, и шифрует слоем, который может быть дешифрован только этим узлом.

Слои создаются с помощью инструкций, работающих в обратном направлении до тех пор, пока весь путь не будет закодирован в слоях. Затем отправитель передает полную луковицу первому узлу в пути, который может читать только самый внешний слой. Каждый узел отслаивает слой, находит внутри инструкции, раскрывающие следующий узел в пути, и передает луковицу дальше. Поскольку каждый узел отслаивает один слой, он не может прочитать остальную часть луковицы. Он знает лишь то, откуда луковица только что пришла и куда она направляется дальше, без каких-либо указаний на то, кто является первоначальным отправителем или конечным получателем.

Это продолжается до тех пор, пока сообщение не достигнет места назначения платежа (получателя платежа). Затем узел назначения открывает луковицу и обнаруживает, что дополнительных слоев для дешифровки нет, и может прочитать платежную информацию внутри.

В отличие от реальной луковицы, при отслаивании каждого слоя узлы добавляют некоторые зашифрованные дополнения, чтобы размер луковицы оставался неизменным для следующего узла. Как мы увидим, за счет этого любой из промежуточных узлов не способен что-либо узнать о размере (длине) пути, о том, сколько узлов участвует в маршрутизации, сколько узлов предшествовало им или сколько следует за ними. Это повышает конфиденциальность за счет предотвращения тривиальных атак с анализом трафика.

Используемый в Lightning протокол луковичной маршрутизации обладает следующими ниже свойствами:

- промежуточный узел может видеть, только по какому каналу он получил луковицу и по какому каналу переслать луковицу. Это означает, что ни один узел маршрутизации не может знать, кто инициировал платеж и кому предназначен платеж. Это самое важное свойство, которое обеспечивает высокую степень конфиденциальности;

[22] Термин «луковица» (onion) первоначально использовался в проекте Tor. Более того, сеть Tor также называется луковичной сетью, и проект использует луковицу в качестве своего логотипа. Доменным именем верхнего уровня, используемым службами Tor в интернете, является *луковица*.

- данные достаточно малы, чтобы поместиться в один пакет TCP/IP и даже в кадр слоя связи (например, Ethernet). Это значительно усложняет анализ трафика платежей, что еще больше повышает конфиденциальность;
- луковицы строятся таким образом, что они всегда будут иметь одинаковую длину независимо от положения обрабатывающего узла вдоль пути. По мере того как каждый слой «отслаивается» (отшелушивается), луковица заполняется зашифрованными «мусорными» данными, чтобы размер луковицы оставался неизменным. За счет этого промежуточные узлы знают свое положение в пути;
- луковицы в каждом слое имеют программный код аутентификации сообщений на основе хеша (hash-based message authentication code, аббр. HMAC), так что манипуляции с луковицами не допускаются и практически невозможны;
- в луковице может быть до 26 переходов (чешуек, или луковичных слоев, или переходных узлов, если вы предпочитаете). Это позволяет использовать достаточно длинные пути. Точная доступная длина пути зависит от числа байтов, выделяемых полезному грузу маршрутизации в каждом переходе;
- в шифровании луковицы для каждой чешуйки используются разные эфемерные ключи шифрования. В случае утечки ключа (в частности, приватного ключа узла) в какой-то момент времени злоумышленник не сможет их дешифровать. Проще говоря, в целях достижения большей безопасности ключи никогда не реиспользуются;
- ошибки могут отправляться обратно с ошибочного узла изначальному отправителю, используя тот же протокол луковичной маршрутизации. Для внешних наблюдателей и промежуточных узлов ошибочные луковицы неотличимы от маршрутизационных луковиц. Маршрутизация ошибок позволяет использовать метод «прощупывание» путем проб и ошибок для отыскания пути, обладающего достаточной емкостью для успешной маршрутизации платежа.

Луковичная маршрутизация будет подробно рассмотрена в главе 10.

Алгоритм пересылки платежей

После того как отправитель платежа нашел возможный путь по сети и собрал луковицу, платеж пересылается каждым узлом в пути. Каждый узел обрабатывает один слой луковицы и пересылает его следующему узлу в пути.

Каждый промежуточный узел получает сообщение Lightning под названием update_add_htlc с платежным хешем и луковицей. Промежуточный узел исполняет серию шагов, именуемых алгоритмом пересылки платежей.

1. Узел дешифрует внешний слой луковицы и проверяет целостность сообщения.
2. Он подтверждает, что может выполнять маршрутизационные подсказки, основываясь на комиссионных за канал и доступной емкости исходящего канала.
3. Он работает со своим партнером по каналу на входящем канале, чтобы обновить состояние канала.
4. Он вносит некоторые дополнения в конец луковицы, чтобы сохранить постоянную длину, так как он удалил некоторые данные с самого начала.

5. Он следует маршрутизационным подсказкам, чтобы переслать измененный луковичный пакет по исходящему платежному каналу, также отправив сообщение `update_add_htlc`, которое содержит тот же платежный хеш и луковицу.
6. Он работает со своим партнером по каналу на исходящем канале, чтобы обновить состояние канала.

Разумеется, эти шаги прерываются и абортируются при обнаружении ошибки, и сообщение об ошибке отправляется обратно отправителю сообщения `update_add_htlc`. Сообщение об ошибке также форматируется как луковица и отправляется обратно по входящему каналу.

Поскольку ошибка распространяется в обратном направлении по каждому каналу вдоль пути, партнеры канала удаляют ожидающий платеж, откатывая платеж в обратном направлении, с которого он начался.

Хотя вероятность отказа платежа высока, если он не будет выполнен быстро, узел никогда не должен инициировать другую попытку платежа по другому пути до того, как луковица вернется с ошибкой. Отправитель заплатит дважды, если обе попытки оплаты в конечном итоге увенчаются успехом.

Шифрование одноранговой обмена сообщениями

Протокол LN в основном представляет собой одноранговый протокол между его участниками. Как мы видели в предыдущих разделах, в сети есть две накладывающиеся друг на друга функции, образующие две логические сети, которые вместе составляют *сеть Lightning*:

1) широкая одноранговая сеть, которая использует эмидемический протокол для распространения информации о топологии, где одноранговые узлы соединяются друг с другом случайным образом. Одноранговые узлы не обязательно имеют платежные каналы между собой, поэтому они не всегда являются партнерами по каналам;
2) сеть платежных каналов между партнерами по каналу. Партнеры по каналу также «сплетничают» о топологии, то есть они являются одноранговыми узлами в эмидемическом протоколе.

Вся связь между одноранговыми узлами передается с помощью сообщений, именуемых *сообщениями Lightning*. Все эти сообщения зашифрованы с использованием каркаса криптографической связи, именуемой *Каркасом криптосвязи на основе протокола Noise* (Noise Protocol Framework). Структура протокола Noise позволяет создавать криптографические протоколы связи, которые обеспечивают аутентификацию, шифрование, прямую секретность и конфиденциальность личных данных. Структура протокола Noise также используется в ряде популярных сквозных зашифрованных коммуникационных систем, таких как WhatsApp, WireGuard и I2P. Более подробную информацию можно найти на веб-сайте каркаса Noise Protocol Framework[23].

Использование структуры протокола Noise в сети Lightning обеспечивает, чтобы каждое сообщение в сети одновременно аутентифицировалось и шифровалось, повышая конфиденциальность сети и ее устойчивость к анализу

[23] См. https://noiseprotocol.org/.

трафика, глубокой инспекции пакетов и прослушиванию. Однако, как побочный эффект, это немного усложняет разработку и тестирование протокола, поскольку невозможно просто наблюдать за сетью с помощью инструмента захвата пакетов или сетевого анализа, такого как Wireshark. Вместо этого разработчикам приходится использовать специализированные подключаемые модули (плагины), которые дешифруют протокол с точки зрения одного узла, такие как диссектор сети Lightning[24], плагин Wireshark.

Мысли о доверии

До тех пор, пока человек подчиняется протоколу и его узел защищен, нет серьезного риска потери денежных средств при участии в сети Lightning. Тем не менее при открытии канала существует стоимость оплаты внутрицепных комиссионных. Любая стоимость должна сопровождаться соответствующей выгодой. В нашем случае награда Алисе за то, что она взяла на себя расходы по открытию канала, заключается в том, что Алиса может отправлять и, переместив часть монет на другой конец канала, получать платежи в биткойне в сети Lightning в любое время и что она может получать комиссионные в биткойне, пересылая платежи для других людей. Алиса знает, что теоретически Боб может закрыть канал сразу после открытия, что для Алисы будет означать увеличение комиссионных за закрытие канала. Алисе нужно будет немного доверять Бобу. Алиса была в кофейне Боба, и очевидно, что Боб заинтересован в продаже ей кофе, так что в этом смысле Алиса может доверять Бобу. Взаимная выгода есть как для Алисы, так и для Боба. Алиса решает, что вознаграждения ей достаточно, чтобы взять на себя расходы по внутрицепным комиссионным за создание канала для Боба. В отличие от этого, Алиса не откроет канал кому-то неизвестному, кто только что без приглашения отправил ей электронное письмо с просьбой открыть новый канал.

Сравнение с системой Bitcoin

В то время как сеть Lightning построена поверх Bitcoin и наследует многие его функциональности и свойства, существуют важные различия, о которых пользователи обеих сетей должны знать.

Некоторые из этих различий связаны с различиями в терминологии. Существуют также архитектурные различия и различия в пользовательском опыте. В следующих нескольких разделах мы рассмотрим различия и сходства, объясним терминологию и скорректируем наши ожидания.

Адреса против счетов, транзакции против платежей

В типичном платеже с использованием системы Bitcoin пользователь получает Bitcoin-адрес (например, сканируя QR-код на веб-странице либо получая его в сообщении Lightning или по электронной почте от друга). Затем они используют свой кошелек Bitcoin, чтобы создать транзакцию для отправки средств на этот адрес.

[24] См. https://github.com/nayutaco/lightning-dissector.

В сети Lightning получатель платежа создает счет. Счет Lightning можно рассматривать как аналог Bitcoin-адреса. Намеченный получатель передает отправителю счет Lightning в виде QR-кода либо символьной строки, точно так же, как Bitcoin-адрес.

Отправитель использует свой кошелек Lightning для оплаты счета, копируя текст счета либо сканируя QR-код счета. Платеж Lightning аналогичен Bitcoin-«транзакции».

Однако есть некоторые различия в пользовательском опыте. Bitcoin-адрес можно *реиспользовать*. Срок действия Bitcoin-адресов никогда не истекает, и если владелец адреса все еще владеет ключами, хранящиеся внутри средства всегда доступны. Отправитель может отправить любую сумму биткойна на ранее используемый адрес, а получатель может отправить один статический адрес для получения многочисленных платежей. Хотя это противоречит лучшим образцам практики по соображениям конфиденциальности, это технически возможно и на самом деле довольно распространено.

Однако в Lightning каждый счет может быть использован только один раз для определенной суммы платежа. Нельзя заплатить больше или меньше, нельзя использовать счет снова, и в счет встроено время истечения срока. В Lightning получатель должен генерировать новый счет для каждого платежа, заранее указывая сумму платежа. Из этого правила есть исключение – механизм под названием keysend, который мы рассмотрим в разделе «Спонтанные платежи keysend» на стр. 279.

Выбор выходов против отыскания пути

Для того чтобы произвести платеж в сети Bitcoin, отправителю необходимо использовать один или несколько неизрасходованных транзакционных выходов (unspent transaction output, аббр. UTXO). Если у пользователя несколько UTXO-выходов, то ему (или, скорее, его кошельку) для отправки нужно будет выбрать один или несколько UTXO-выходов. Например, пользователь, делающий оплату в размере 1 BTC, может использовать один выход стоимостью 1 BTC, два выхода стоимостью 0.25 BTC и 0.75 BTC или четыре выхода стоимостью 0.25 BTC каждый.

В Lightning платежи не требуют потребления входов. Вместо этого каждая оплата приводит к обновлению остатка канала, перераспределяя его между двумя партнерами по каналу. Отправитель воспринимает это как «перемещение» остатка канала со своего конца канала на другой конец, к своему партнеру по каналу. Платежи Lightning используют серию каналов для маршрутизирования от отправителя к получателю. Каждый из этих каналов должен обладать достаточной емкостью для маршрутизирования платежа.

Поскольку для совершения платежа можно использовать множество возможных каналов и путей, выбор каналов и путей пользователем Lightning в некоторой степени аналогичен выбору UTXO-выхода пользователем Bitcoin.

С помощью таких технологий, как атомарные многопутевые платежи (atomic multipath payments, аббр. AMP) и многокомпонентные платежи (multipart payments, аббр, MPP), которые мы рассмотрим в последующих главах, несколько путей Lightning могут агрегироваться в один атомарный платеж, точно так же, как несколько биткойновых UTXO-выходов могут агрегироваться в одну атомарную Bitcoin-транзакцию.

Выходы со сдачей в Bitcoin против отсутствия сдачи в Lightning

Для того чтобы произвести платеж в сети Bitcoin, отправителю необходимо использовать один или несколько неизрасходованных транзакционных выходов (UTXO). UTXO-выходы могут быть потрачены только полностью; их нельзя разделить и израсходовать частично. Таким образом, если пользователь хочет потратить 0.8 BTC, но у него есть только UTXO-выход с 1 BTC, то ему необходимо потратить весь 1 BTC UTXO-выход, отправив 0.8 BTC получателю и 0.2 BTC обратно себе в качестве сдачи. Платеж со сдачей в размере 0.2 BTC создает новый UTXO-выход, именуемый «выход со сдачей».

В Lightning финансовая транзакция расходует некоторый UTXO-выход системы Bitcoin, создавая мультиподписной UTXO-выход для открытия канала. Как только биткойн будет привязан в этом канале, его части могут отправляться туда и обратно по каналу без необходимости создавать какую-либо сдачу. Это связано с тем, что партнеры по каналу просто обновляют остаток канала и создают новый UTXO-выход только тогда, когда канал в конечном итоге закрывается с помощью транзакции закрытия канала.

Майнинговые комиссионные против маршрутизационных комиссионных

В сети Bitcoin пользователи платят майнерам комиссионные за включение своих транзакций в блок. Эти комиссонные выплачиваются майнеру, который добывает этот конкретный блок. Размер комиссионных зависит от размера транзакции в байтах, которую транзакция использует в блоке, а также от того, как быстро пользователь хочет, чтобы эта транзакция была добыта. Поскольку майнеры обычно сначала добывают наиболее прибыльные транзакции, пользователь, который хочет, чтобы его транзакция была добыта немедленно, будет платить более высокие комиссионные за байт, тогда как пользователь, который не спешит, будет платить меньшие комиссионные за байт.

В сети Lightning пользователи платят комиссионные другим пользователям (промежуточному узлу) за маршрутизацию платежей по их каналам. В целях маршрутизирования платежа промежуточный узел должен будет перенести средства в два или более каналов, которыми он владеет, а также передать данные для платежа отправителя. Как правило, пользователь маршрутизации взимает плату с отправителя в зависимости от суммы платежа, установив минимальные *базовые комиссионные* (фиксированные комиссионные за каждый платеж) и *комиссионный тариф* (соответствующие комиссионные, пропорционально стоимости платежа). Таким образом, маршрутизация платежей с более высокой стоимостью будет стоить дороже, и формируется рынок ликвидности, где разные пользователи взимают разные комиссионные за маршрутизацию по своим каналам.

Комиссионные, варьирующиеся в зависимости от трафика, против объявленных комиссионных

В сети Bitcoin майнеры стремятся к получению прибыли и обычно включают в блок как можно больше транзакций, оставаясь при этом в пределах емкости блока, именуемой *весом блока*.

Если в очереди (именуемой *mempool*) больше транзакций, чем может поместиться в блоке, они начинают добывать транзакции, которые платят самые высокие комиссионные за единицу (байты) *веса транзакции*. Таким образом, когда в очереди транзакций много, пользователям приходится платить более высокие комиссионные за включение в следующий блок, либо им приходится ждать до тех пор, пока в очереди не станет меньше транзакций. Это, естественно, приводит к появлению рынка комиссионных, где пользователи платят в зависимости от того, насколько срочно им нужно, чтобы их транзакция была включена в следующий блок.

Дефицитным ресурсом в сети Bitcoin является пространство в блоках. Пользователи системы Bitcoin конкурируют за блочное пространство, а рынок комиссионных в системе Bitcoin основан на доступном блочном пространстве. Дефицитными ресурсами в сети Lightning являются *ликвидность канала* (объем средств, доступных для маршрутизирования по каналам) и *связность канала* (число узлов с хорошей связностью, к которым каналы могут быть подсоединены). Пользователи Lightning конкурируют за емкость и возможности подсоединения; поэтому рынок комиссионных в сети Lightning определяется емкостью и возможностями подсоединения.

В сети Lightning пользователи платят комиссионные пользователям, которые маршрутизируют их платежи. Маршрутизация платежа, с экономической точки зрения, – это не что иное, как предоставление и распределение емкости отправителю. Естественно, маршрутизаторы, которые взимают более низкие комиссионные за ту же емкость, будут более привлекательными для маршрутизации. Таким образом, существует рынок комиссионных, где маршрутизаторы конкурируют друг с другом за комиссионные, которые они взимают за маршрутизирование платежей по своим каналам.

Публичные Bitcoin-транзакции против приватных платежей Lightning

В сети Bitcoin каждая транзакция видна публично в блочной цепи Bitcoin. Хотя соответствующие адреса являются псевдонимными и обычно не привязаны к идентификатору, они все равно видны и валидируются всеми последующими пользователями в сети. В дополнение к этому компании по надзору за блочной цепью массово собирают и анализируют эти данные и продают их заинтересованным сторонам, таким как частные фирмы, правительства и спецслужбы.

LN-платежи, с другой стороны, являются почти полностью приватными. В типичной ситуации только отправитель и получатель полностью осведомлены об источнике, пункте назначения и сумме транзакции в конкретном платеже. Более того, получатель может даже не знать источник платежа. Поскольку платежи являются луковично-маршрутизируемыми, пользователи, которые маршрутизируют платеж, знают только о сумме платежа и не могут определить ни источник, ни пункт назначения.

Таким образом, Bitcoin-транзакции транслируются в широковещательном порядке публично и хранятся вечно. Платежи Lightning выполняются между несколькими выбранными одноранговыми узлами, и информация о них хранится в частном порядке только до тех пор, пока канал не будет закрыт. Соз-

дать инструменты массового наблюдения и анализа, эквивалентные тем, которые используются в системе Bitcoin, будет намного сложнее для сети Lightning.

Ожидание подтверждений против денежного расчета Lightning

В сети Bitcoin транзакции обрабатываются только после того, как они были включены в блок, и в таком случае считается, что они «подтверждены» в этом блоке. По мере того как добывается больше блоков, транзакция получает больше «подтверждений» и считается более безопасной.

В сети Lightning подтверждения имеют значение только для открытия и закрытия каналов внутри цепи. Как только финансовая транзакция наберет подходящее число подтверждений (например, 3), партнеры по каналу будут считать канал открытым. Поскольку биткойн в канале защищен умным контрактом, который управляет этим каналом, платежи осуществляются *мгновенно* после получения конечным получателем. С практической точки зрения мгновенный расчет означает, что выполнение и расчет платежей занимают всего несколько секунд. Как и в случае с системой Bitcoin, платежи Lightning необратимы.

Наконец, когда канал закрывается, в сети Bitcoin совершается транзакция; как только эта транзакция подтверждается, канал считается закрытым.

Отправка произвольных сумм против ограничений по емкости

В сети Bitcoin пользователь может отправить любую сумму биткойна, которым он владеет, другому пользователю без ограничений по емкости. Одна транзакция теоретически может в качестве платежа отправить до 21 миллиона биткойна.

В сети Lightning пользователь может отправить партнеру по каналу только столько биткойна, сколько в настоящее время существует на его стороне конкретного канала. Например, если пользователь владеет одним каналом с 0.4 BTC на своей стороне и другим каналом с 0.2 BTC на своей стороне, то максимум, который он может отправить одним платежом, составляет 0.4 BTC. Это верно независимо от того, сколько биткойна в настоящее время находится у пользователя в его кошельке Bitcoin.

Многокомпонентные платежи (MPP) – это функциональность, которая в предыдущем примере позволяет пользователю совмещать оба своих канала в размере 0.4 BTC и 0.2 BTC для отправки максимум 0.6 BTC одним платежом. MPP-платежи в сети Lightning в настоящее время тестируются, и ожидается, что к моменту завершения написания этой книги они будут широко доступны и использоваться. Более подробную информацию о MPP можно получить в разделе «Многокомпонентные платежи» на стр. 314.

Если платеж маршрутизируется, то каждый маршрутизационный узел вдоль пути маршрутизации должен иметь каналы с емкостью, по меньшей мере такой же, как сумма маршрутизируемого платежа. Это должно быть справедливо для каждого отдельного канала, по которому проходит платеж. Емкость наименее емкого канала в пути задает верхний лимит емкости всего пути.

Следовательно, емкость и возможности подсоединения являются критически важными и дефицитными ресурсами в сети Lightning.

Стимулы для крупных платежей против малых платежей

Структура комиссионных в Bitcoin не зависит от стоимости транзакции. Транзакция в размере 1 миллиона долларов имеет ту же комиссию, что и транзакция в Bitcoin в размере 1 доллара, при условии аналогичного размера транзакции в байтах (конкретнее, «виртуальных» байтах после SegWit [протокола Сегрегированного свидетеля]). В Lightning комиссионные представляют собой фиксированные базовые комиссионные плюс процент от стоимости транзакции. Таким образом, в Lightning комиссионные за оплату увеличиваются с увеличением стоимости платежа. Эти противоположные структуры комиссионных создают разные стимулы и приводят к разной используемости в отношении стоимости транзакции. Транзакция большей стоимости будет дешевле в системе Bitcoin; следовательно, пользователи будут предпочитать Bitcoin для транзакций с большой стоимостью. Аналогичным образом на другом конце шкалы пользователи предпочтут Lightning для транзакций с малой стоимостью.

Использование блочной цепи в качестве реестра против судебной системы

В сети Bitcoin каждая транзакция в конечном итоге записывается в блок в блочной цепи. Таким образом, блочная цепь формирует полную историю каждой транзакции с момента создания Bitcoin и способ полного аудита каждого существующего биткойна. После того как транзакция включается в блочную цепь, она становится окончательной. Следовательно, никаких споров возникнуть не может, и однозначно ясно, насколько биткойн контролируется конкретным адресом в конкретной точке блочной цепи.

В сети Lightning остаток в канале в определенное время известен только двум партнерам по каналу и становится видимым для остальной части сети только тогда, когда канал закрыт. Когда канал закрывается, окончательный остаток канала передается в блочную цепь Bitcoin, и каждый партнер получает свою долю биткойна в этом канале. Например, если начальный остаток составлял 1 BTC, оплаченный Алисой, и Алиса произвела платеж в размере 0.3 BTC Бобу, то окончательный остаток канала составляет 0.7 BTC для Алисы и 0.3 BTC для Боба. Если Алиса попытается обмануть, предъявив состояние открытия канала в блочную цепь Bitcoin с 1 BTC для Алисы и 0 BTC для Боба, то Боб может нанести ответный удар, отправив истинное окончательное состояние канала, а также создать штрафную транзакцию, которая отдает ему весь биткойн в канале. Для сети Lightning блочная цепь Bitcoin действует как судебная система. Подобно роботизированному судье, Bitcoin записывает начальный и конечный остатки каждого канала и утверждает штрафы, если одна из сторон попытается обмануть.

Офлайн против онлайна, асинхронность против синхронности

Когда пользователь Bitcoin отправляет средства на адрес назначения, ему не нужно ничего знать о получателе. Получатель может быть в офлайновом или онлайновом режиме, и никакого взаимодействия между отправителем

и получателем не требуется. Взаимодействие происходит между отправителем и блочной цепью Bitcoin. Получение биткойна в блочной цепи Bitcoin – это *пассивное* и *асинхронное* действие, которое не требует какого-либо взаимодействия со стороны получателя или того, чтобы получатель был онлайн в любое время. Bitcoin-адреса могут даже генерироваться в офлайновом режиме и никогда не «регистрируются» в сети Bitcoin. Только расходование биткойна требует взаимодействия.

В Lightning получатель должен находиться онлайн, чтобы завершить платеж до истечения его срока. Получатель должен выполнять узел или иметь кого-то, кто управляет узлом от его имени (сторонний опекун). Выражаясь точнее, во время оплаты оба узла, отправитель и получатель, должны находиться онлайн и должны координировать свои действия. Получение платежа Lightning – это *активное* и *синхронное* действие между отправителем и получателем без участия большей части сети Lightning или сети Bitcoin (за исключением промежуточных маршрутизационных узлов, если таковые имеются).

Синхронная и всегда онлайновая природа сети Lightning, вероятно, является самой большой разницей в пользовательском опыте, и это часто сбивает с толку пользователей, привыкших к сети Bitcoin.

Сатоши против миллисатоши

В сети Bitcoin наименьшая сумма – это сатоши, которую нельзя разделить дальше. Lightning немного гибче, и узлы Lightning работают с миллисатоши (тысячными долями сатоши). Это позволяет отправлять крошечные платежи через Lightning. Один платеж в миллисатоши может быть отправлен по платежному каналу, и его сумма настолько мала, что ее следует охарактеризовать надлежаще как *наноплатеж*.

Единицу измерения миллисатоши, конечно же, невозможно рассчитать в блочной цепи Bitcoin с такой степенью гранулярности. После закрытия канала остатки округляются до ближайшего сатоши. Но за время существования канала возможны миллионы наноплатежей на уровне миллисатоши. Сеть Lightning преодолевает барьер микроплатежей.

Общие черты сетей Bitcoin и Lightning

В то время как сеть Lightning отличается от Bitcoin по нескольким направлениям, в том числе архитектурой и пользовательским опытом, она построена на основе Bitcoin и сохраняет многие стержневые функциональные свойства сети Bitcoin.

Денежная единица

И сеть Bitcoin, и сеть Lightning используют одни и те же денежные единицы: биткойн. В платежах Lightning используется тот же биткойн, что и в Bitcoin-транзакциях. Как следствие, поскольку денежная единица одна и та же, денежный лимит тот же: менее 21 миллиона биткойна. В сети Bitcoin из этой общей суммы в 21 миллион биткойна некоторые уже распределены по мультиподписным адресам «2 из 2» в рамках платежных каналов в сети Lightning.

Необратимость и окончательность платежей

Как Bitcoin-транзакции, так и платежи Lightning являются необратимыми и немутируемыми. Ни в одной из систем нет операции «отмена» или «возвратный платеж». Как отправитель любого из них, вы должны действовать ответственно, но также, как получателю, вам гарантируется окончательность ваших транзакций.

Доверие и риск контрагента

Как и в случае с Bitcoin, Lightning требует, чтобы пользователь доверял только математике, шифрованию и чтобы в программном обеспечении не было никаких критических дефектов. Ни Bitcoin, ни Lightning не требуют, чтобы пользователь доверял человеку, компании, учреждению или правительству. Поскольку Lightning строится поверх Bitcoin и опирается на Bitcoin в качестве опорного базового слоя, ясно, что модель безопасности Lightning сводится к модели безопасности Bitcoin. Это означает, что Lightning обеспечивает в целом ту же безопасность, что и Bitcoin, при большинстве обстоятельств, лишь с небольшим снижением безопасности в некоторых узких обстоятельствах.

Безразрешительная работа

Как Bitcoin, так и Lightning могут использоваться любым человеком, имеющим доступ к интернету и соответствующему программному обеспечению, например узлу и кошельку. Ни одна из сетей не требует, чтобы пользователи получали разрешение, проверку или авторизацию от третьих сторон, компаний, учреждений или правительства. Правительства могут запретить Bitcoin или Lightning в пределах своей юрисдикции, но не могут предотвратить их глобальное использование.

Открытый исходный код и открытая система

И Bitcoin, и Lightning – это программные системы с открытым исходным кодом, созданные децентрализованным глобальным сообществом добровольцев, доступные по открытым лицензиям. Оба основаны на открытых и совместимых протоколах, которые работают как открытые системы и открытые сети. Глобально, открыто и бесплатно.

Вывод

В этой главе мы рассмотрели, как на самом деле работает сеть Lightning и все составляющие ее компоненты. Мы рассмотрели каждый шаг в создании, эксплуатировании и закрытии канала. Мы рассмотрели то, как проходят платежи, и, наконец, сравнили Lightning с Bitcoin и проанализировали их различия и общие черты.

В следующих нескольких главах мы вернемся ко всем этим темам, но гораздо более подробно.

Глава 4

Программное обеспечение узла Lightning

Как мы видели в предыдущих главах, узел Lightning – это компьютерная система, которая участвует в сети Lightning. Сеть Lightning – это не продукт или компания; это набор открытых стандартов, которые определяют базовый уровень для взаимодействия. Таким образом, программное обеспечение узла Lightning было создано разнообразными компаниями и общественными группами. Подавляющее большинство программного обеспечения Lightning имеет открытый исходный код, а значит, исходный код общедоступен и лицензирован таким образом, чтобы обеспечивать совместную работу, совместное использование и участие сообщества в процессе разработки. Аналогичным образом все имплементации узла Lightning, которые мы представим в этой главе, имеют открытый исходный код и разрабатываются совместно.

В отличие от системы Bitcoin, где стандарт определяется референтной имплементацией в программном обеспечении (Bitcoin Core), в Lightning стандарт определяется серией стандартизирующих документов, именуемых Базисом технологии Lightning (Basis of Lightning Technology, аббр. BOLT), которые можно найти в репозитории lightning-rfc[25].

Референтной имплементации сети Lightning не существует, но существует несколько конкурирующих, совместимых со спецификацией BOLT и совместимых между собой имплементаций, разработанных разными коллективами и организациями. Коллективы, разрабатывающие программное обеспечение для сети Lightning, также вносят свой вклад в разработку и эволюцию стандартов BOLT.

Еще одно важное различие между программным обеспечением узла Lightning и программным обеспечением узла Bitcoin заключается в том, что узлы Lightning не должны работать в соответствии с консенсусными правилами и могут иметь расширенную функциональность, выходящую за рамки базовоуровневой версии BOLT. Поэтому разные коллективы могут использовать разные экспериментальные функциональные свойства, которые в случае успеха и широкого внедрения позже могут стать частью спецификаций BOLT.

[25] См. https://github.com/lightningnetwork/lightning-rfc.

В этой главе вы узнаете, как настроить каждый из программных пакетов наиболее популярных имплементаций узла Lightning. Мы представили их в алфавитном порядке, чтобы подчеркнуть, что мы, как правило, не предпочитаем и не одобряем одно в ущерб другому. У каждого есть свои сильные и слабые стороны, и выбор одного из них будет зависеть от множества факторов. Поскольку они разработаны на разных языках программирования (например, Go, C и т. д.), ваш выбор также может зависеть от вашего уровня знакомства и опыта с конкретным языком и набором инструментов разработки.

Среда разработки Lightning

Если вы – разработчик, то вы захотите настроить среду разработки со всеми инструментами, библиотеками и вспомогательным программным обеспечением для написания и оперирования программным обеспечением Lightning. В данной сугубо технической главе мы рассмотрим этот процесс шаг за шагом. Если материал становится слишком плотным, или вы на самом деле не настраиваете среду разработки, тогда можете смело перейти к следующей главе, которая имеет менее технический характер.

Использование командной строки

В примерах этой главы и в более широком плане в большей части нашей книги используется терминал командной строки. Это означает, что вы вводите команды в терминал и получаете текстовые ответы. Кроме того, примеры демонстрируются в операционной системе, основанной на ядре Linux и программной системе GNU, в частности в последней долгосрочной стабильной версии Ubuntu (Ubuntu 20.04 LTS). Большинство примеров можно воспроизвести в других операционных системах, таких как Windows или macOS, с небольшими изменениями в командах. Самое большое различие между операционными системами заключается в *пакетном менеджере*, который устанавливает различные программные библиотеки и их пререквизиты. В приведенных примерах мы будем использовать `apt`, то есть пакетный менеджер в Ubuntu. В macOS обычным пакетным менеджером, используемым для разработки с открытым исходным кодом, является Homebrew[26], доступ к которому осуществляется с помощью команды `brew`.

В большинстве приведенных здесь примеров мы будем создавать программное обеспечение непосредственно из исходного кода. Хотя это бывает довольно сложно, это дает нам максимальную власть и контроль. Вместо этого вы можете использовать Docker-контейнеры, предварительно скомпилированные пакеты или другие механизмы инсталляции, если вы застряли!

[26] См. https://brew.sh/.

Во многих примерах в этой главе мы будем использовать интерфейс командной строки операционной системы (также именуемый оболочкой), доступ к которому осуществляется через терминальное приложение. Оболочка сначала покажет приглашение в качестве индикатора того, что она готова к вашей команде. Затем вы набираете команду и нажимаете клавишу **Enter**, на что командная оболочка отвечает некоторым текстом и новым приглашением ввести следующую команду. В вашей системе приглашение может выглядеть по-другому, но в следующих ниже примерах оно обозначается символом `$`. Когда в примерах вы видите текст после символа `$`, не печатайте символ `$`, а набирайте команду сразу после него. Затем нажмите клавишу **Enter**, чтобы исполнить команду. В примерах строки, следующие за каждой командой, являются ответами операционной системы на эту команду. Когда вы увидите следующий префикс `$`, вы поймете, что это новая команда, и вам следует повторить процесс.

В целях поддержания согласованности во всех примерах командной строки мы используем оболочку `bash`. В то время как другие оболочки будут вести себя аналогичным образом и вы сможете выполнять все примеры без нее, некоторые скрипты оболочки написаны специально для оболочки `bash` и могут потребовать некоторых изменений или настроек для выполнения в другой оболочке. В целях обеспечения согласованности вы можете инсталлировать оболочку `bash` в Windows и macOS, и она инсталлируется по умолчанию в большинстве систем Linux.

Скачивание репозитория книги

Все примеры исходного кода доступны в онлайновом репозитории книги. Поскольку репозиторий будет обновляться настолько, насколько это возможно, вы всегда должны искать в онлайновом репозитории последнюю версию, а не копировать ее из печатной или электронной книги.

Репозиторий можно скачать в виде ZIP-пакета, посетив GitHub[27] и нажав зеленую кнопку **Code** (Код) справа.

В качестве альтернативы можно использовать команду `git`, чтобы создать версионно-контролируемый клон репозитория на вашем локальном компьютере. Git – это распределенная система версионного контроля, которая используется большинством разработчиков для совместной разработки программного обеспечения и отслеживания изменений в репозиториях программного обеспечения. Скачайте и установите git, следуя инструкциям из проекта Git[28].

В целях создания локальной копии репозитория на вашем компьютере выполните команду `git` следующим образом:

```
$ git clone https://github.com/lnbook/lnbook.git
```

[27] См. https://github.com/lnbook/lnbook.

[28] См. https://git-scm.com/.

Теперь у вас есть полная копия репозитория книги в папке под названием `lnbook`. Вы захотите перейти в только что загруженный каталог, выполнив:

```
$ cd lnbook
```

Во всех последующих примерах мы будем исходить из того, что вы выполняете команды из этой папки.

Docker-контейнеры

Многие разработчики используют контейнер, представляющий собой разновидность виртуальной машины, для инсталлирования предварительно настроенной операционной системы и приложений со всеми необходимыми зависимостями. Бóльшая часть программного обеспечения Lightning также может быть проинсталлирована с помощью контейнерной системы, такой как Docker, которую можно найти на домашней странице Docker[29]. Контейнерные инсталляции намного проще, в особенности для тех, кто не привык к среде командной строки.

Репозиторий книги содержит коллекцию Docker-контейнеров, которые можно использовать для настройки согласованной среды разработки, чтобы практиковаться и воспроизводить примеры в любой системе. Поскольку контейнер представляет собой полноценную операционную систему, которая работает с согласованной конфигурацией, вы можете быть уверены, что примеры будут работать на вашем компьютере без необходимости беспокоиться о зависимостях, версиях библиотек или различиях в конфигурации.

Docker-контейнеры часто оптимизированы таким образом, чтобы быть небольшими, т. е. занимать минимальное место на диске. Однако в этой книге мы используем контейнеры для *стандартизирования* среды и обеспечения ее согласованности для всех читателей. Кроме того, эти контейнеры не предназначены для того, чтобы использоваться для оперирования службами в фоновом режиме. Вместо этого они предназначены для тестирования примеров и обучения, взаимодействуя с программным обеспечением. По этим причинам контейнеры довольно большие и поставляются со множеством инструментов разработки и утилит. Обычно дистрибутив Alpine используется для контейнеров Linux из-за их уменьшенного размера. Тем не менее мы предлагаем контейнеры, собранные в Ubuntu, потому что все больше разработчиков знакомы с Ubuntu, и это знакомство для нас важнее, чем размер.

Инсталляция и использование Docker и его команд подробно описаны в приложении B. Если вы незнакомы с Docker, то сейчас самое подходящее время быстро просмотреть этот раздел.

Последние определения контейнеров и конфигурации сборки можно найти в репозитории книги в папке *code/docker*. Каждый контейнер находится в отдельной папке, как видно из следующего ниже:

[29] См. https://docker.com/.

```
$ tree -F --charset=asciii code/docker
code/docker
|-- bitcoind/
|   |-- bashrc
|   |-- bitcoind/
|   |   |-- bitcoin.conf
|   |   `-- keys/
|   |       |-- demo_address.txt
|   |       |-- demo_mnemonic.txt
|   |       `-- demo_privkey.txt
|   |-- bitcoind-entrypoint.sh
|   |-- cli
|   |-- Dockerfile
|   `-- mine.sh*
|-- c-lightning/
|   |-- bashrc
|   |-- cli
|   |-- c-lightning-entrypoint.sh
|   |-- devkeys.pem
|   |-- Dockerfile
|   |-- fund-c-lightning.sh
|   |-- lightningd/
|   |   `-- config
|   |-- logtail.sh
|   `-- wait-for-bitcoind.sh
|-- eclair/
|   |-- bashrc
|   |-- cli
|   |-- Dockerfile
|   |-- eclair/
|   |   `-- eclair.conf
|   |-- eclair-entrypoint.sh
|   |-- logtail.sh
|  `-- wait-for-bitcoind.sh
|-- lnd/
|   |-- bashrc
|   |-- cli
|   |-- Dockerfile
|   |-- fund-lnd.sh
|   |-- lnd/
|   |   `-- lnd.conf
|   |-- lnd-entrypoint.sh
|   |-- logtail.sh
|   `-- wait-for-bitcoind.sh
|-- check-versions.sh
|-- docker-compose.yml
|-- Makefile
`-- run-payment-demo.sh*
```

Как мы увидим в следующих нескольких разделах, вы можете собирать эти контейнеры локально или извлекать их из репозитория книги в Docker Hub[30]. В следующих далее разделах мы исходим из того, что вы проинсталлировали Docker и знакомы с базовым использованием команды `docker`.

[30] См. https://hub.docker.com/orgs/lnbook.

Bitcoin Core и regtest

Большинству имплементаций узлов Lightning для работы требуется доступ к полноценному узлу Bitcoin.

Инсталлирование полноценного узла Bitcoin и синхронизирование блочной цепи Bitcoin выходят за рамки этой книги и сами по себе являются относительно сложной задачей. Если вы хотите это попробовать, то обратитесь к главе 3 «Bitcoin Core: референтная имплементация» книги «Освоение системы Bitcoin»[31], в которой обсуждаются инсталляция и эксплуатация узла Bitcoin.

Узел Bitcoin может работать в режиме regtest, где он создает локальную симулированную блочную цепь Bitcoin для целей тестирования. В следующих ниже примерах мы будем использовать режим regtest, который позволит нам продемонстрировать Lightning без необходимости синхронизировать узел Bitcoin или рисковать какими-либо средствами.

Контейнером для Bitcoin Core является bitcoind. Он сконфигурирован на запуск Bitcoin Core в режиме regtest и добычу 6 новых блоков каждые 10 секунд. Его порт дистанционного (удаленного) вызова процедур (RPC) открыт на порту 18443 и доступен для RPC-вызовов с пользовательским именем regtest и паролем regtest. Вы также можете получить к нему доступ с помощью интерактивной оболочки и запускать команды bitcoin-cli локально.

Сборка контейнера Bitcoin Core

Давайте подготовим контейнер `bitcoind`. Самый простой способ – извлечь последний контейнер из Docker Hub:

```
$ docker pull lnbook/bitcoind
Using default tag: latest
latest: Pulling from lnbook/bitcoind
35807b77a593: Pull complete
e1b85b9c5571: Pull complete
[...]
288f1cc78a00: Pull complete
Digest: sha256:861e7e32c9ad650aa367af40fc5acff894e89e47aff4bd400691ae18f1b550e2
Status: Downloaded newer image for lnbook/bitcoind:latest
docker.io/lnbook/bitcoind:latest
```

В качестве альтернативы вы можете собрать контейнер самостоятельно из определения локального контейнера, которое находится в *code/docker/bitcoind/Dockerfile*.

Вам не нужно собирать контейнер, если вы ранее использовали команду pull для его извлечения из Docker Hub.

Локальная сборка контейнера будет использовать немного меньшую пропускную способность вашей сети, но на сборку потребуется больше процессорного времени. Для его сборки мы используем команду `docker build`:

[31] См. https://github.com/bitcoinbook/bitcoinbook.

```
$ cd code/docker
$ docker run -it --name bitcoind lnbook/bitcoind
Starting bitcoind...
Bitcoin Core starting
Waiting for bitcoind to start
bitcoind started
================================================
Imported demo private key
Bitcoin address: 2NBKgwSWY5qEmfN2Br4WtMDGuamjpuUc5q1
Private key: cSaejkcWwU25jMweWEewRSsrVQq2FGTij1xjXv4x1XvxVRF1ZCr3
================================================
================================================
Balance: 0.00000000
================================================
Mining 101 blocks to unlock some bitcoin
[
  "34c744207fd4dd32b70bac467902bd8d030fba765c9f240a2e98f15f05338964",
  "64d82721c641c378d79b4ff2e17572c109750bea1d4eddbae0b54f51e4cdf23e",

 [...]

  "7a8c53dc9a3408c9ecf9605b253e5f8086d67bbc03ea05819b2c9584196c9294",
  "39e61e50e34a9bd1d6eab51940c39dc1ab56c30b21fc28e1a10c14a39b67a1c3",
  "4ca7fe9a55b0b767d2b7f5cf4d51a2346f035fe8c486719c60a46dcbe33de51a"
]
Mining 6 blocks every 10 seconds
Balance: 50.00000000
[
  "5ce76cc475e40515b67e3c0237d1eef597047a914ba3f59bbd62fc3691849055",
  "1ecb27a05ecfa9dfa82a7b26631e0819b2768fe5e6e56c7a2e1078b078e21e9f",
  "717ceb8b6c329d57947c950dc5668fae65bddb7fa03203984da9d2069e20525b",
  "185fc7cf3557a6ebfc4a8cdd1f94a8fa08ed0c057040cdd68bfb7aee2d5be624",
  "59001ae237a3834ebe4f6e6047dcec8fd67df0352ddc70b6b02190f982a60384",
  "754c860fe1b9e0e7292e1de96a65eaa78047feb4c72dbbde2a1d224faa1499dd"
]
```

Как вы видите, `bitcoind` запускается и добывает 101 симулированный блок, чтобы запустить цепь. Это связано с тем, что в соответствии с консенсусными правилами системы Bitcoin недавно добытый биткойн не подлежит расходованию до тех пор, пока не пройдет 100 блоков. Добывая 101 блок, мы делаем монетарную базу первого блока доступной для использования. После этого начального действия по добыче каждые 10 секунд добывается 6 новых блоков, чтобы цепь продолжала двигаться вперед.

На данный момент транзакций нет. Но у нас есть немного тестового биткойна, который был добыт в кошельке и доступен для использования. Когда мы подсоединим несколько узлов Lightning к этой цепи, мы отправим немного биткойна в их кошельки, чтобы иметь возможность открыть несколько каналов Lightning между узлами Lightning.

Взаимодействие с контейнером bitcoin core

Тем временем мы также можем взаимодействовать с контейнером `bitcoind`, отправляя ему команды оболочки. Контейнер отправляет журнальный файл на терминал, отображающий процесс добычи в рамках процесса `bitcoind`. Для

взаимодействия с оболочкой можно выдавать команды в еще одном терминале, используя команду `docker exec`. Поскольку мы ранее назвали работающий контейнер с помощью аргумента `name`, мы можем ссылаться на него по этому имени при исполнении команды `docker exec`. Сначала давайте запустим интерактивную оболочку `bash`:

```
$ docker exec -it bitcoind /bin/bash
root@e027fd56e31a:/bitcoind# ps x
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 /bin/bash /usr/local/bin/mine.sh
    7 ?        Ssl    0:03 bitcoind -datadir=/bitcoind -daemon
   97 pts/1    Ss     0:00 /bin/bash
  124 pts/0    S+     0:00 sleep 10
  125 pts/1    R+     0:00 ps x
root@e027fd56e31a:/bitcoind#
```

Запуск интерактивной оболочки помещает нас «внутрь» контейнера. Она регистрируется в системе как пользователь `root`, как можно видеть из префикса `root@` в новом приглашении командной строки `root@e027fd56e31a:/bitcoind#`. Если выдать команду `ps x`, чтобы увидеть, какие процессы работают, то можно увидеть, что и `bitcoind`, и скрипт `mine.sh` работают в фоновом режиме. Для того чтобы выйти из этой оболочки, нажмите **Ctrl**-**D** или наберите `exit`, и вы вернетесь к приглашению операционной системы.

Вместо запуска интерактивной оболочки также можно выдать одну команду, которая выполняется внутри контейнера. Для удобства команда `bitcoin-cli` имеет псевдоним «`cli`», который передает правильную конфигурацию. Итак, давайте ее выполним, чтобы спросить Bitcoin Code о блочной цепи. Мы выполняем `cli getblockchaininfo`:

```
$ docker exec bitcoind cli getblockchaininfo
{
  "chain": "regtest",
  "blocks": 131,
  "headers": 131,
  "bestblockhash":
    "2cf57aac35365f52fa5c2e626491df634113b2f1e5197c478d57378e5a146110",

  [...]

  "warnings": ""
}
```

Команда `cli` в контейнере `bitcoind` позволяет выдавать команды RPC-вызовов узлу Bitcoin Core и получать результаты в кодировке JavaScript Object Notation (JSON).

Вдобавок во всех наших Docker-контейнерах прединсталлирован кодировщик/декировщик JSON командной строки под названием `jq`. Кодировщик/декодировщик `jq` помогает обрабатывать данные в формате JSON посредством командной строки или из внутренних скриптов. Результат любой команды в формате JSON можно отправить в `jq`, используя символ `|`. Этот символ, а также эта операция называется «конвейером». Давайте применим конвейер и `jq` к приведенной выше команде следующим образом:

```
$ docker exec bitcoind bash -c "cli getblockchaininfo | jq .blocks"
197
```

`jq .blocks` поручает декодировщику JSON `jq` извлекать поля `blocks` из результата `getblockchaininfo`. В нашем случае он извлекает и печатает значение 197, которое мы могли бы использовать в последующей команде.

Как вы увидите в последующих разделах, можно запускать несколько контейнеров одновременно, а затем взаимодействовать с ними по отдельности. Можно выдавать команды для извлечения информации, такой как публичный ключ узла Lightning, или для выполнения таких действий, как открытие канала Lightning для другого узла. В целях создания работающей сети Lightning, в которой сочетается множество различных имплементаций узлов, нам нужны лишь команды `docker run` и `docker exec` вместе с `jq` для декодирования JSON, и это все. Благодаря этому мы получаем возможность проводить разнообразные эксперименты на нашем собственном компьютере.

Проект c-lightning узла Lightning

`c-lightning` – это облегченная, легко настраиваемая под собственные нужды и соответствующая стандартам имплементация протокола LN, разработанная Blockstream в рамках проекта Elements. Указанный проект имеет открытый исходный код и разработан совместно на GitHub[32].

В следующих разделах мы соберем Docker-контейнер, который оперирует узлом c-lightning, подсоединяющимся к контейнеру bitcoind, который мы собрали ранее. Мы также покажем, как конфигурировать и собирать программное обеспечение c-lightning непосредственно из исходного кода.

Сборка c-lightning в качестве Docker-контейнера

Дистрибутив программного обеспечения `c-lightning` имеет Docker-контейнер, но он предназначен для оперирования `c-lightning` в производственных системах и рядом с узлом `bitcoind`. Мы будем использовать несколько более простой контейнер, сконфигурированный для оперирования `c-lightning` в демонстрационных целях.

Давайте извлечем контейнер c-lightning из репозитория Docker Hub книги:

```
$ docker pull lnbook/c-lightning
Using default tag: latest
latest: Pulling from lnbook/c-lightning

[...]

Digest: sha256:bdefcefe8a9712e7b3a236dcc5ab12d999c46fd280e209712e7cb649b8bf0688
Status: Downloaded image for lnbook/c-lightning:latest
docker.io/lnbook/c-lightning:latest
```

В качестве альтернативы можно собрать Docker-контейнер `c-lightning` из файлов книги, которые вы ранее скачали в каталог с именем `lnbook`. Как и прежде, мы будем использовать команду `docker build` в подкаталоге *code/docker*. Пометим образ контейнера тегом `lnbook/c-lightning` следующим образом:

[32] См. https://github.com/ElementsProject/lightning.

```
$ cd code/docker
$ docker build -t lnbook/c-lightning c-lightning
Sending build context to Docker daemon 91.14kB
Step 1/34 : ARG OS=ubuntu
Step 2/34 : ARG OS_VER=focal
Step 3/34 : FROM ${OS}:${OS_VER} as os-base
 ---> fb52e22af1b0

 [...]

Step 34/34 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Running in 8d3d6c8799c5
Removing intermediate container 8d3d6c8799c5
 ---> 30b6fd5d7503
Successfully built 30b6fd5d7503
Successfully tagged lnbook/c-lightning:latest
```

Наш контейнер теперь собран и готов к работе. Однако, прежде чем мы запустим контейнер `c-lightning`, необходимо запустить контейнер `bitcoind` в еще одном терминале, потому что `c-lightning` зависит от `bitcoind`. Нам также нужно будет настроить сеть Docker, которая позволит контейнерам подсоединяться друг к другу, как если бы они находились в одной локальной сети.

Docker-контейнеры могут «разговаривать» друг с другом по виртуальной локальной сети, управляемой системой Docker. Каждый контейнер может иметь пользовательское имя, и другие контейнеры могут использовать это имя для определения его IP-адреса и легкого к нему подсоединения.

Настройка сети Docker

После найстройки сети Docker он будет активировать сеть на нашем локальном компьютере при каждом запуске Docker, например после перезагрузки. Таким образом, нам нужно настроить сеть только один раз с помощью команды `docker network create`. Имя сети само по себе не важно, но на нашем компьютере оно должно быть уникальным. По умолчанию Docker имеет три сети с именами `host`, `bridge` и `none`. Мы назовем нашу новую сеть `lnbook` и создадим ее следующим образом:

```
$ docker network create lnbook
ad75c0e4f87e5917823187febedfc0d7978235ae3e88eca63abe7e0b5ee81bfb
$ docker network ls
NETWORK ID    NAME    DRIVER  SCOPE
7f1fb63877ea  bridge  bridge  local
4e575cba0036  host    host    local
ad75c0e4f87e  lnbook  bridge  local
ee8824567c95  none    null    local
```

Как вы видите, выполнение команды `docker network ls` дает нам список сетей Docker. Была создана наша сеть `lnbook`. Идентификатор сети можно игнорировать, потому что он управляется автоматически.

Оперирование контейнерами bitcoind и c-lightning

Следующий шаг состоит в запуске контейнеров `bitcoind` и `c-lightning` и их подсоединении к сети `lnbook`. В целях запуска контейнера в конкретной сети необходимо передать аргумент `network` команде `docker run`. Для того чтобы контейнерам было легко друг друга находить, мы также назначим каждому из них имя с помощью аргумента `name`. Запускаем `bitcoind` следующим образом:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Вы должны увидеть, что `bitcoind` будет запущен, и он начнет добывать блоки каждые 10 секунд. Оставьте его работающим и откройте новое окно терминала, чтобы запустить `c-lightning`. Мы используем аналогичную команду `docker run` с аргументами `network` и `name` для запуска `c-lightning` следующим образом:

```
$ docker run -it --network lnbook --name c-lightning lnbook/c-lightning
Waiting for bitcoind to start...
Waiting for bitcoind to mine blocks...
Starting c-lightning...
2021-09-12T13:14:50.434Z UNUSUAL lightningd: Creating configuration directory
/lightningd/regtest
Startup complete
Funding c-lightning wallet
8a37a183274c52d5a962852ba9f970229ea6246a096ff1e4602b57f7d4202b31
lightningd: Opened log file /lightningd/lightningd.log
lightningd: Creating configuration directory /lightningd/regtest
lightningd: Opened log file /lightningd/lightningd.log
```

Контейнер `c-lightning` запускается и подсоединяется к контейнеру `bitcoind` через сеть Docker. Сначала наш узел `c-lightning` будет ждать запуска `bitcoind`, а затем он будет ждать до тех пор, пока `bitcoind` не добудет немного биткойна в свой кошелек. Наконец, в рамках запуска контейнера скрипт отправит команду RPC-вызова на узел `bitcoind`, который создаст транзакцию. Данная транзакция пополнит кошелек `c-lightning` 10 тестовыми BTC. Теперь наш узел `c-lightning` не только работает, но у него даже есть немного тестового биткойна для игры!

Как мы продемонстрировали с контейнером `bitcoind`, нашему контейнеру `c-lightning` можно выдавать команды в другом терминале для извлечения информации, открытия каналов и т. д. Команда, которая позволяет выдавать инструкции командной строки узлу `c-lightning`, называется `lightning-cli`. Указанная команда `lightning-cli` также имеет псевдоним `cli` внутри этого контейнера. В целях получения информации об узле `c-lightning` используйте следующую ниже команду `docker exec` в другом окне терминала:

```
$ docker exec c-lightning cli getinfo
{
  "id": "026ec53cc8940df5fed5fa18f8897719428a15d860ff4cd171fca9530879c7499e",
  "alias": "IRATEARTIST",
  "color": "026ec5",
  "num_peers": 0,
  "num_pending_channels": 0,

 [...]

  "version": "0.10.1",
```

```
    "blockheight": 221,
    "network": "regtest",
    "msatoshi_fees_collected": 0,
    "fees_collected_msat": "0msat",
    "lightning-dir": "/lightningd/regtest"
}
```

Теперь у нас есть первый узел Lightning, работающий в виртуальной сети и взаимодействующий с тестовой блочной цепью Bitcoin. Позже в этой главе мы запустим больше узлов и подсоединим их друг к другу, чтобы совершить несколько платежей Lightning.

В следующем разделе мы также рассмотрим, как скачивать, конфигурировать и компилировать `c-lightning` непосредственно из исходного кода. Этот необязательный и продвинутый шаг научит вас пользоваться сборочными инструментами и позволит вносить изменения в исходный код `c-lightning`. Обладая этими знаниями, вы сможете писать некий исходный код, исправлять возможные дефекты или создавать плагин для `c-lightning`.

Если вы не планируете углубляться в исходный код или программирование узла Lightning, то можете пропустить следующий раздел целиком. Только что собранного нами Docker-контейнера будет достаточно для большинства примеров в книге.

Инсталлирование c-lightning из исходного кода

Разработчики `c-lightning` предоставили подробные инструкции по сборке `c-lightning` из исходного кода. Мы будем следовать инструкциям из GitHub[33].

Инсталлирование необходимых библиотек и пакетов

В этих инструкциях по инсталляции предполагается, что вы собираете `c-lightning` в Linux или аналогичной системе с помощью сборочных инструментов GNU. Если это не так, то поищите инструкции для вашей операционной системы в репозитории проекта Elements.

Обычно первым шагом является инсталляция необходимых библиотек. Для их инсталлирования мы используем пакетный менеджер `apt`:

```
$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://eu-north-1b.clouds.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://eu-north-1b.clouds.archive.ubuntu.com/ubuntu bionic-updates InRe-
lease
[88.7 kB]

[...]

Fetched 18.3 MB in 8s (2,180 kB/s)
Reading package lists... Done

$ sudo apt-get install -y \
  autoconf automake build-essential git libtool libgmp-dev \
```

[33] См. https://github.com/ElementsProject/lightning/blob/master/doc/INSTALL.md.

```
    libsqlite3-dev python python3 python3-mako net-tools zlib1g-dev \
    libsodium-dev gettext

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  autotools-dev binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-7
dpkg-dev fakeroot g++ g++-7 gcc gcc-7 gcc-7-base libalgorithm-diff-perl

  [...]

Setting up libsigsegv2:amd64 (2.12-2) ...
Setting up libltdl-dev:amd64 (2.4.6-14) ...
Setting up python2 (2.7.17-2ubuntu4) ...
Setting up libsodium-dev:amd64 (1.0.18-1) ...

[...]
$
```

Через несколько минут и большой оживленности на экране у вас будут проинсталлированы все необходимые пакеты и библиотеки. Многие из этих библиотек также используются другими пакетами Lightning и необходимы для разработки программного обеспечения в целом.

Копирование исходного кода c-lightning

Далее мы скопируем последнюю версию `c-lightning` из репозитория исходного кода. Для этого будем использовать команду `git clone`, которая клонирует версионно-контролируемую копию на ваш локальный компьютер, тем самым позволяя вам синхронизировать ее с последующими изменениями без необходимости повторной загрузки всего репозитория:

```
$ git clone --recurse https://github.com/ElementsProject/lightning.git
Cloning into 'lightning'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 53192 (delta 5), reused 5 (delta 2), pack-reused 53168
Receiving objects: 100% (53192/53192), 29.59 MiB | 19.30 MiB/s, done.
Resolving deltas: 100% (39834/39834), done.

$ cd lightning
```

Теперь у нас есть копия `c-lightning`, клонированная во вложенную папку *lightning*, и мы использовали команду `cd` (изменить каталог) для входа в эту вложенную папку.

Компилирование исходного кода c-lightning

Далее мы используем набор *сборочных скриптов*, которые обычно доступны во многих проектах с открытым исходным кодом. В этих сборочных скриптах используются команды `configure` и `make`, которые позволяют нам:

- выбирать опции сборки и проверять необходимые зависимости (`configure`);
- собирать и инсталлировать исполняемые файлы и библиотеки (`make`).

Выполнение команды configure с опцией help покажет все доступные опции:

```
$ ./configure --help
Usage: ./configure [--reconfigure] [setting=value] [options]

Options include:
  --prefix= (default /usr/local)
    Prefix for make install
  --enable/disable-developer (default disable)
    Developer mode, good for testing
  --enable/disable-experimental-features (default disable)
    Enable experimental features
  --enable/disable-compat (default enable)
    Compatibility mode, good to disable to see if your software breaks
  --enable/disable-valgrind (default (autodetect))
    Run tests with Valgrind
  --enable/disable-static (default disable)
    Static link sqlite3, gmp and zlib libraries
  --enable/disable-address-sanitizer (default disable)
    Compile with address-sanitizer
```

В этом примере нам не нужно изменять какие-либо дефолтные значения. Следовательно, мы снова выполняем configure без каких-либо опций, чтобы применить дефолтные значения:

```
$ ./configure

Compiling ccan/tools/configurator/configurator...done
checking for python3-mako... found
Making autoconf users comfortable... yes
checking for off_t is 32 bits... no
checking for __alignof__ support... yes

[...]

Setting COMPAT... 1
PYTEST not found
Setting STATIC... 0
Setting ASAN... 0
Setting TEST_NETWORK... regtest
$
```

Затем мы используем команду make, чтобы собрать библиотеки, компоненты и исполняемые файлы проекта c-lightning. Работа этой части займет несколько минут и приведет к интенсивному использованию процессора и диска вашего компьютера. Ожидайте некоторого шума от вентиляторов! Выполните make:

```
$ make

cc -DBINTOPKGLIBEXECDIR="\"../libexec/c-lightning\"" -Wall -Wundef -Wmis...

[...]

cc -Og ccan-asort.o ccan-autodata.o ccan-bitmap.o ccan-bitops.o ccan-...
```

Если все пройдет хорошо, то вы не увидите никакого сообщения об ошибке (ERROR), останавливающего исполнение приведенной выше команды. Программ-

ный пакет `c-lightning` был скомпилирован из исходного кода, и теперь мы готовы установить исполняемые компоненты, которые собрали на предыдущем шаге:

```
sudo make install

mkdir -p /usr/local/bin
mkdir -p /usr/local/libexec/c-lightning
mkdir -p /usr/local/libexec/c-lightning/plugins
mkdir -p /usr/local/share/man/man1
mkdir -p /usr/local/share/man/man5
mkdir -p /usr/local/share/man/man7
mkdir -p /usr/local/share/man/man8
mkdir -p /usr/local/share/doc/c-lightning
install cli/lightning-cli lightningd/lightningd /usr/local/bin
[...]
```

В целях верификации, что команды `lightningd` и `lightning-cli` были проинсталлированы правильно, мы запросим у каждого исполняемого файла информацию о его версии:

```
$ lightningd --version
v0.10.1-34-gfe86c11
$ lightning-cli --version
v0.10.1-34-gfe86c11
```

Версия состоит из последней версии выпуска (v0.10.1), за которой следует число изменений с момента выпуска (34) и, наконец, хеш, определяющий, какая именно это версия (fe86c11). Вы, возможно, увидите версию, отличную от показанной выше, поскольку программное обеспечение продолжит эволюционировать еще долго после публикации этой книги. Однако независимо от того, какую версию вы видите, тот факт, что команды исполняются и отвечают информацией о версии, означает, что вы преуспели в создании программного обеспечения `c-lightning`.

Проект демона узла сети Lightning

Демон сети Lightning (LND) – это полноценная имплементация узла LN от Lightning Labs. Проект LND предоставляет ряд исполняемых приложений, включая `lnd` (сам демон) и `lncli` (утилита командной строки). LND имеет несколько подключаемых бэкендовых цепных служб, включая `btcd` (полный узел), `bitcoind` (Bitcoin Core) и `Neutrino` (новый экспериментальный легкий клиент). LND написан на языке программирования Go. Указанный проект имеет открытый исходный код и разработан совместно на GitHub[34].

В следующих нескольких разделах мы соберем Docker-контейнер для запуска LND, соберем LND из исходного кода и узнаем, как конфигурировать и оперировать LND.

Docker-контейнер LND

Пример Docker-контейнера LND можно извлечь из репозитория Docker Hub книги:

[34] См. https://github.com/LightningNetwork/lnd.

```
$ docker pull lnbook/lnd
Using default tag: latest
latest: Pulling from lnbook/lnd
35807b77a593: Already exists
e1b85b9c5571: Already exists
52f9c252546e: Pull complete

[...]

Digest: sha256:e490a0de5d41b781c0a7f9f548c99e67f9d728f72e50cd4632722b3ed3d85952
Status: Downloaded newer image for lnbook/lnd:latest
docker.io/lnbook/lnd:latest
```

В качестве альтернативы можно собрать контейнер LND локально. Контейнер находится в *code/docker/lnd*. Мы меняем рабочий каталог на *code/docker* и выполняем

```
$ cd code/docker
$ docker build -t lnbook/lnd lnd
Sending build context to Docker daemon 9.728kB
Step 1/29 : FROM golang:1.13 as lnd-base
 ---> e9bdcb0f0af9
Step 2/29 : ENV GOPATH /go

[...]

Step 29/29 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Using cache
 ---> 397ce833ce14
Successfully built 397ce833ce14
Successfully tagged lnbook/lnd:latest
```

Теперь наш контейнер готов к работе. Как и в случае с контейнером `c-lightning`, который мы собрали ранее, контейнер LND также зависит от запущенного экземпляра Bitcoin Core. Как и прежде, нам нужно запустить контейнер `bitcoind` в другом терминале и подсоединить к нему LND через сеть Docker. Мы уже настроили сеть Docker под названием `lnbook` и будем использовать ее здесь снова.

Обычно каждый оператор узла оперирует своим собственным узлом Lightning и своим собственным узлом Bitcoin на своем собственном сервере. Для нас контейнер с одним контейнером `bitcoind` может обслуживать большое число узлов Lightning. В нашей симулированной сети мы можем оперировать несколькими узлами Lightning, все из которых подсоединяются к одному узлу Bitcoin в режиме `regtest`.

Оперирование контейнерами bitcoind и LND

Как и прежде, мы запускаем контейнер `bitcoind` в одном терминале, а LND – в другом. Если у вас уже запущен контейнер `bitcoind`, то перезапускать его не нужно. Просто оставьте его работающим и пропустите следующий шаг. В целях запуска `bitcoind` в сети `lnbook` команда `docker run` используется следующим образом:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Далее мы запускаем контейнер LND, который только что собрали. Как и раньше, нам нужно подсоединить его к сети `lnbook` и назначить ему имя:

```
$ docker run -it --network lnbook --name lnd lnbook/lnd
Waiting for bitcoind to start...
Waiting for bitcoind to mine blocks...
Starting lnd...
Startup complete
Funding lnd wallet
{"result":"dbd1c8e2b224e0a511c11efb985dabd84d72d935957ac30935ec4211d28beacb","
er
ror":null,"id":"lnd-run-container"}
[INF] LTND: Version: 0.13.1-beta commit=v0.13.1-beta, build=production, log
ging=default, debuglevel=info
[INF] LTND: Active chain: Bitcoin (network=regtest)
[INF] RPCS: Generating TLS certificates...
```

Контейнер LND запускается и подсоединяется к контейнеру `bitcoind` через сеть Docker. Сначала наш узел LND будет ждать запуска `bitcoind`, а затем он будет ждать до тех пор, пока `bitcoind` не добудет немного биткойна в свой кошелек. Наконец, в рамках запуска контейнера скрипт отправит команду RPC-вызова на узел `bitcoind`, тем самым создав транзакцию, которая пополнит кошелек LND 10 тестовыми BTC.

Как мы демонстрировали ранее, команды нашему контейнеру можно выдавать в другом терминале, чтобы извлекать информацию, открывать каналы и т. д. Команда, которая позволяет выдавать инструкции командной строки демону `lnd`, называется `lncli`. Опять-таки, в этом контейнере мы предоставили псевдоним `cli`, который запускает `lncli` со всеми надлежащими параметрами. Давайте получим информацию об узле с помощью команды `docker exec` в другом окне терминала:

```
$ docker exec lnd cli getinfo
{
  "version": "0.13.1-beta commit=v0.13.1-beta",
  "commit_hash": "596fd90ef310cd7abbf2251edaae9ba4d5f8a689",
  "identity_pubkey":
  "02d4545dccbeda29a10f44e891858940f4f3374b75c0f85dcb7775bb922fdeaa14",

 [...]

}
```

Теперь у нас есть еще один узел Lightning, который работает в сети `lnbook` и взаимодействует с `bitcoind`. Если вы все еще используете контейнер `c-lightning`, то сейчас работают два узла. Они еще не подсоединены друг к другу, но мы скоро их подсоединим.

При желании в одной и той же сети Lightning можно оперировать любой комбинацией узлов LND и `c-lightning`. Например, чтобы работать со вторым узлом LND, необходимо исполнить команду `docker run` с другим именем контейнера, например:

```
$ docker run -it --network lnbook --name lnd2 lnbook/lnd
```

В приведенной выше команде мы запускаем еще один контейнер LND, назвав его `lnd2`. Имена полностью зависят от вас, если они уникальны. Если вы не укажете имя, то Docker создаст уникальное имя, случайным образом объединив два английских слова, таких как «naughty_einstein» (шаловливый Эйнштейн). Это было фактическое название, которое Docker выбрал для нас, когда мы писали этот абзац. Забавно, не правда?!

В следующем разделе мы рассмотрим, как скачивать и компилировать LND непосредственно из исходного кода. Это необязательный и продвинутый шаг, который научит вас использовать сборочные инструменты языка Go и позволит вносить изменения в исходный код LND. Обладая этими знаниями, вы сможете писать какой-нибудь код или исправлять возможные дефекты.

Если вы не планируете углубляться в исходный код или программирование узла Lightning, то можете пропустить следующий раздел целиком. Только что собранного нами Docker-контейнера будет достаточно для большинства примеров в книге.

Инсталлирование LND из исходного кода

В этом разделе мы соберем LND с нуля. LND написан на языке программирования Go. Если вы хотите узнать о Go больше, то выполните поиск по слову golang вместо go, чтобы избежать нерелевантных результатов. Поскольку он написан на Go, а не на C или C++, в нем используется другой «сборочный» каркас, чем каркас GNU autotools/make, применение которого мы видели ранее в `c-lightning`. Однако не волнуйтесь – инсталлировать и использовать инструменты golang довольно просто, и мы покажем здесь все шаги. Go – фантастический язык для совместной разработки программного обеспечения, потому что он создает очень согласованный, точный и легко читаемый исходный код, независимо от числа авторов. Go сфокусирован и «мини-оптимизирован» таким образом, чтобы обеспечивать согласованность между версиями языка. Как компилируемый язык он также довольно эффективен. Давайте нырнем внутрь.

Мы будем следовать инструкциям по инсталляции, приведенным в проектной документации LND[35].

Сначала проинсталлируем пакет golang и связанные с ним библиотеки. Мы строго требуем Go версии 1.13 или более поздней. Официальные пакеты языка Go распространяются в виде двоичных файлов из проекта Go[36]. Для удобства они также упакованы как пакеты Debian, доступные посредством команды apt. Вы можете последовать инструкциям из проекта Go или же использовать следующие ниже команды apt в системе Debian/Ubuntu Linux, как описано на вики-странице GitHub по языку Go[37]:

```
$ sudo apt install golang-go
```

Убедитесь, что у вас проинсталлирована правильная версия и она готова к использованию, выполнив команду:

[35] См. https://github.com/lightningnetwork/lnd/blob/master/docs/INSTALL.md.

[36] См. https://golang.org/dl.

[37] См. https://github.com/golang/go/wiki/Ubuntu.

```
$ go version
go version go1.13.4 linux/amd64
```

У нас версия 1.13.4, так что все готово к работе. Далее необходимо указывать любым программам, где найти код Go. Это достигается путем задания средовой переменной GOPATH. Обычно код Go находится в каталоге с именем *gocode* непосредственно в домашнем каталоге пользователя. С помощью следующих двух команд мы соответствующим образом задаем GOPATH и убеждаемся, что ваша оболочка добавляет его в исполняемый путь PATH. Обратите внимание, что домашний каталог пользователя в командной оболочке обозначается как ~.

```
$ export GOPATH=~/gocode
$ export PATH=$PATH:$GOPATH/bin
```

Во избежание необходимости задавать эти переменные среды всякий раз, когда вы открываете оболочку, эти две строки можно добавить в конец вашего конфигурационного файла *.bashrc* оболочки bash в домашнем каталоге, используя выбранный вами редактор.

Копирование исходного кода LND

Как и во многих проектах с открытым исходным кодом в настоящее время, исходный код для LND находится на GitHub (www.github.com). Команда `go get` может извлечь его напрямую, используя протокол Git:

```
$ go get -d github.com/lightningnetwork/lnd
```

Как только команда `go get` завершится, у вас будет подкаталог под GOPATH, содержащий исходный код LND.

Компилирование исходного кода LND

В LND используется сборочная система `make`. В целях сборки проекта мы меняем каталог на каталог исходного кода LND, а затем применяем `make` следующим образом:

```
$ cd $GOPATH/src/github.com/lightningnetwork/lnd
$ make && make install
```

Через несколько минут у вас будут проинсталлированы две новые команды, `lnd` и `lncli`. Попробуйте их и проверьте их версию, чтобы убедиться, что они проинсталлированы:

```
$ lnd --version
lnd version 0.10.99-beta commit=clock/v1.0.0-106-
gc1ef5bb908606343d2636c8cd345169e064bdc91
$ lncli --version
lncli version 0.10.99-beta commit=clock/v1.0.0-106-
gc1ef5bb908606343d2636c8cd345169e064bdc91
```

Скорее всего, вы увидите версию, отличную от показанной выше, поскольку программное обеспечение продолжит эволюционировать еще долго после публикации этой книги. Однако, независимо от того, какую версию вы видите, тот факт, что команды исполняются и показывают вам информацию о версии, означает, что вы преуспели в создании программного обеспечения LND.

Проект узла Lightning Eclair

Eclair (по-французски «молния») – это имплементация сети Lightning на языке Scala, созданная ACINQ. Eclair также является одним из самых популярных и новаторских мобильных кошельков Lightning, которые мы использовали для демонстрации платежей Lightning в главе 2. В этом разделе мы рассмотрим проект сервера Eclair, который оперирует узлом Lightning. Eclair – это проект с открытым исходным кодом, который можно найти на GitHub[38].

В следующих нескольких разделах мы соберем Docker-контейнер для работы с Eclair, как делали ранее с c-lightning и LND. Мы также соберем Eclair непосредственно из исходного кода.

Docker-контейнер Eclair

Давайте извлечем контейнер Eclair из репозитория Docker Hub книги:

```
$ docker pull lnbook/eclair
Using default tag: latest
latest: Pulling from lnbook/eclair
35807b77a593: Already exists
e1b85b9c5571: Already exists

[...]

c7d5d5c616c2: Pull complete
Digest: sha256:17a3d52bce11a62381727e919771a2d5a51da9f91ce2689c7ecfb03a6f028315
Status: Downloaded newer image for lnbook/eclair:latest
docker.io/lnbook/eclair:latest
```

В качестве альтернативы вместо этого можно собрать контейнер локально. К настоящему времени вы – уже почти эксперт в основных операциях Docker! В этом разделе мы повторим многие из ранее рассмотренных команд по сборке контейнера Eclair. Контейнер находится в каталоге *code/docker/eclair*. Мы начинаем с терминала, переключив рабочий каталог на *code/docker* и исполнив команду `docker build`:

```
$ cd code/docker
$ docker build -t lnbook/eclair eclair
Sending build context to Docker daemon 11.26kB
Step 1/27 : ARG OS=ubuntu
Step 2/27 : ARG OS_VER=focal
Step 3/27 : FROM ${OS}:${OS_VER} as os-base
 ---> fb52e22af1b0

[...]

Step 27/27 : CMD ["/usr/local/bin/logtail.sh"]
 ---> Running in fe639120b726
Removing intermediate container fe639120b726
 ---> e6c8fe92a87c
Successfully built e6c8fe92a87c
Successfully tagged lnbook/eclair:latest
```

[38] См. https://github.com/ACINQ/eclair.

Теперь наш образ готов к работе. Контейнер Eclair также зависит от работающего экземпляра Bitcoin Core. Как и прежде, необходимо запустить контейнер `bitcoind` в другом терминале и подсоединить к нему Eclair через сеть Docker. Мы уже настроили сеть Docker под названием `lnbook` и будем реиспользовать ее здесь.

Одно из заметных различий между Eclair и LND или `c-lightning` заключается в том, что Eclair не содержит отдельного биткойнового кошелька, а вместо этого полагается непосредственно на биткойновый кошелек в Bitcoin Core. Напомним, что с помощью LND мы профинансировали его биткойновый кошелек, выполнив транзакцию по переводу биткойна с кошелька Bitcoin Core на биткойновый кошелек LND. Этот шаг не является необходимым при использовании Eclair. При оперировании Eclair кошелек Bitcoin Core используется непосредственно в качестве источника средств для открытия каналов. Как следствие, в отличие от контейнеров LND или `c-lightning`, контейнер Eclair не содержит скрипта для перевода биткойна в свой кошелек при запуске.

Оперирование контейнерами bitcoind и Eclair

Как и прежде, мы запускаем контейнер `bitcoind` в одном терминале, а контейнер Eclair – в другом. Если у вас уже работает контейнер `bitcoind`, то перезапускать его не нужно. Просто оставьте его работающим и пропустите следующий шаг. В целях запуска `bitcoind` в сети `lnbook` мы используем команду `docker run` следующим образом:

```
$ docker run -it --network lnbook --name bitcoind lnbook/bitcoind
```

Затем мы запускаем контейнер Eclair, который только что собрали. Нам нужно будет подсоединить его к сети `lnbook` и назначить ему имя, как мы это делали с другими контейнерами:

```
$ docker run -it --network lnbook --name eclair lnbook/eclair
Waiting for bitcoind to start...
Waiting for bitcoind to mine blocks...
Starting eclair...
Eclair node started
INFO o.b.Secp256k1Context - secp256k1 library successfully loaded
INFO fr.acinq.eclair.Plugin - loading 0 plugins
INFO a.e.slf4j.Slf4jLogger - Slf4jLogger started
INFO fr.acinq.eclair.Setup - hello!
INFO fr.acinq.eclair.Setup - version=0.4.2 commit=52444b0

[...]
```

Контейнер Eclair запускается и подсоединяется к контейнеру `bitcoind` через сеть Docker. Сначала узел Eclair будет ждать запуска `bitcoind`, а затем он будет ждать до тех пор, пока `bitcoind` не добудет немного биткойна в свой кошелек.

Как мы демонстрировали ранее, контейнеру можно выдавать команды в другом терминале, чтобы извлекать информацию, открывать каналы и т. д. Команда, которая позволяет выдавать инструкции командной строки демону `eclair`, называется `eclair-cli`. Как и прежде, в этом контейнере мы предоставили для `eclair-cli` полезный псевдоним, именуемый просто `cli`, который предлагает необходимые аргументы и параметры. Используя команду `docker exec` в другом окне терминала, мы получаем информацию об узле из Eclair:

```
$ docker exec eclair cli getinfo
{
  "version": "0.4.2-52444b0",
  "nodeId":
  "02fa6d5042eb8098e4d9c9d99feb7ebc9e257401ca7de829b4ce757311e0301de7",
  "alias": "eclair",
  "color": "#49daaa",
  "features": {

[...]

  },
  "chainHash":
    "06226e46111a0b59caaf126043eb5bbf28c34f3a5e332a1fc7b2b73cf188910f",
  "network": "regtest",
  "blockHeight": 779,
  "publicAddresses": [],
  "instanceId": "01eb7a68-5db0-461b-bdd0-29010df40d73"
}
```

Теперь у нас есть еще один узел Lightning, работающий в сети `lnbook` и взаимодействующий с `bitcoind`. В одной и той же сети Lightning можно оперировать любым числом и любой комбинацией узлов Lightning, и может сосуществовать любое число узлов Eclair, LND и `c-lightning`. Например, для того чтобы работать со вторым узлом Eclair, необходимо исполнить команду `docker run` с другим именем контейнера следующим образом:

```
$ docker run -it --network lnbook --name eclair2 lnbook/eclair
```

В приведенной выше команде мы запускаем еще один контейнер Eclair с именем `eclair2`.

В следующем разделе мы также рассмотрим, как скачивать и компилировать Eclair непосредственно из исходного кода. Это необязательный и продвинутый шаг, который научит вас использовать инструменты сборки на языках Scala и Java и позволит вносить изменения в исходный код Eclair. Обладая этими знаниями, вы сможете писать некоторый код или исправлять возможные дефекты.

> Если вы не планируете углубляться в исходный код или программирование узла Lightning, то можете пропустить следующий раздел целиком. Только что собранного нами Docker-контейнера будет достаточно для большинства примеров в книге.

Инсталлирование Eclair из исходного кода

В этом разделе мы соберем Eclair с нуля. Eclair написан на языке программирования Scala, который компилируется с помощью компилятора Java. Для работы с Eclair сначала нужно проинсталлировать Java и его сборочные инструменты. Мы будем следовать инструкциям, приведенным в документе *BUILD.md* проекта Eclair[39].

Требуемый компилятор Java является частью OpenJDK 11. Нам также понадобится сборочный каркас под названием Maven версии 3.6.0 или выше.

[39] См. https://github.com/ACINQ/eclair/blob/master/BUILD.md.

Для инсталлирования OpenJDK 11 и Maven в системе Debian/Ubuntu Linux используется команда apt, как показано ниже:

```
$ sudo apt install openjdk-11-jdk maven
```

Убедитесь, что у вас проинсталлирована правильная версия, выполнив:

```
$ javac -version

javac 11.0.7
$ mvn -v
Apache Maven 3.6.1
Maven home: /usr/share/maven
Java version: 11.0.7, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdkamd64
```

Мы имеем OpenJDK 11.0.7 и Maven 3.6.1, так что все готово.

Копирование исходного кода Eclair

Исходный код Eclair находится на GitHub. Команда `git clone` может создать для нас локальную копию. Давайте перейдем в домашний каталог и запустим ее там:

```
$ cd ~
$ git clone https://github.com/ACINQ/eclair.git
```

Как только команда `git clone` завершится, у вас будет подкаталог *eclair*, содержащий исходный код сервера Eclair.

Компилирование исходного кода Eclair

В Eclair используется сборочная система Maven. В целях сборки проекта мы меняем рабочий каталог на каталог исходного кода Eclair, а затем применяем пакет `mvn` следующим образом:

```
$ cd eclair
$ mvn package
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] --------------------< fr.acinq.eclair:eclair_2.13 >---------------------
[INFO] Building eclair_2.13 0.4.3-SNAPSHOT [1/4]
[INFO] --------------------------------[ pom ]---------------------------------

[...]

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:06 min
[INFO] Finished at: 2020-12-12T09:43:21-04:00
[INFO] ------------------------------------------------------------------------
```

Через несколько минут сборка пакета Eclair должна завершиться. Однако действие «package» (пакет) также будет запускать тесты, и некоторые из них подключаются к интернету и могут завершиться отказом. Если вы хотите пропустить тесты, то добавьте в команду опцию `-DskipTests`.

Теперь распакуйте и запустите сборочный пакет, следуя инструкциям по инсталляции Eclair с GitHub.

Поздравляю! Вы создали Eclair из исходного кода и готовы кодировать, тестировать, исправлять дефекты и вносить свой вклад в этот проект!

Сборка полной сети из разнообразных узлов Lightning

Наш завершающий пример, представленный в этом разделе, сведет все собранные нами различные компоненты, чтобы сформировать сеть Lightning, состоящую из различных имплементаций узлов (LND, `c-lightning`, Eclair). Мы составим сеть, соединив узлы вместе и открыв каналы от одного узла к другому. В качестве последнего шага мы выполним маршрутизацию платежа по этим каналам!

В этом примере мы построим демонстрационную сеть Lightning, состоящую из четырех узлов Lightning с именами Алиса (Alice), Боб (Bob), Чан (Chan) и Дина (Dina). Мы свяжем Алису с Бобом, Боба с Чаном, а Чана с Диной. Это показано на рис. 4-1.

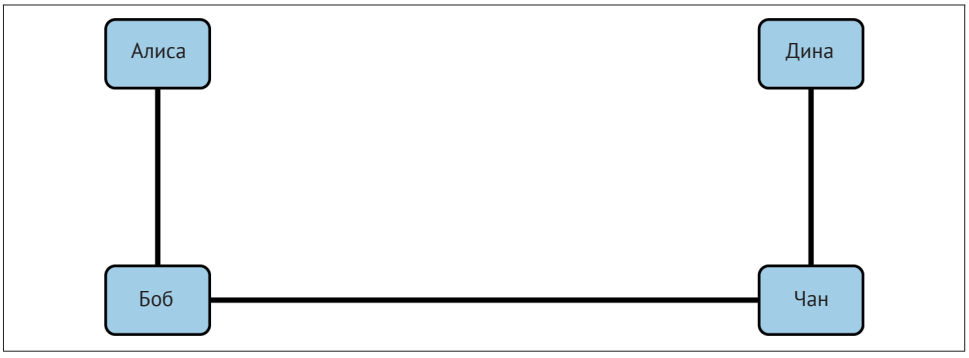


Рис. 4-1. Небольшая демонстрационная сеть из четырех узлов

Наконец, мы попросим Дину создать счет, и Алиса оплатит этот счет. Поскольку Алиса и Дина не подсоединены напрямую, платеж будет промаршрутизирован как HTLC-контракт по всем платежным каналам.

Использование docker-compose для оркестрирования Docker-контейнеров

Для приведения этого примера в рабочее состояние мы будем использовать инструмент оркестровки контейнеров, который имеется в виде команды `docker-compose`. Эта команда позволяет указывать приложение, состоящее из нескольких контейнеров, и запускать приложение, запуская все кооперирующие контейнеры вместе.

Сначала давайте проинсталлируем docker-compose. Инструкции[40] зависят от вашей операционной системы.

После завершения инсталляции можно подтвердить свою инсталляцию, выполнив docker-compose следующим образом:

[40] См. https://docs.docker.com/compose/install.

```
$ docker-compose version
docker-compose version 1.21.0, build unknown
[...]
```

Наиболее распространенными командами `docker-compose`, которые мы будем использовать, являются `up` и `down`, например `docker-compose up`.

Конфигурация docker-compose

Конфигурационный файл для `docker-compose` находится в каталоге *code/docker* и называется *docker-compose.yml*. Он содержит спецификацию сети и каждого из четырех контейнеров. Верхняя часть выглядит так:

```
version: "3.3"
networks:
  lnnet:

services:
  bitcoind:
    container_name: bitcoind
    build:
        context: bitcoind
    image: lnbook/bitcoind:latest
    networks:
      - lnnet
    expose:
      - "18443"
      - "12005"
      - "12006"
  Alice:
    container_name: Alice
```

Приведенный выше фрагмент определяет сеть под названием `lnnet` и контейнер под названием `bitcoind`, который будет подсоединяться к сети `lnnet`. Контейнер – тот же, который мы собрали в начале этой главы. Мы предоставляем доступ к трем портам контейнера, что позволяет отправлять на него команды и отслеживать блоки и транзакции. Далее в конфигурации указывается контейнер LND с именем «Алиса». Потом вы также увидите спецификации для контейнеров под названием «Боб» (`c-lightning`), «Чан» (Eclair) и «Дина» (снова LND).

Поскольку все эти разнообразные имплементации соответствуют спецификации BOLT и были тщательно протестированы на предмет совместимости, им нетрудно работать вместе, строя сеть Lightning.

Запуск образца сети Lightning

Прежде чем начать, необходимо убедиться, что мы уже не работаем ни с одним из контейнеров. Если новый контейнер имеет то же имя, что и уже работающий, то он не запустится. Используйте `docker ps`, `docker stop` и `docker rm` по мере необходимости, чтобы остановить и удалить все работающие в данный момент контейнеры!

Поскольку мы используем одни и те же имена для этих оркестрированных Docker-контейнеров, нам может потребоваться выполнить «очистку», чтобы избежать любых конфликтов имен.

В целях запуска примера мы переключаемся в каталог, содержащий конфигурационный файл *docker-compose.yml*, и исполняем команду `docker-compose up`:

```
$ cd code/docker
$ docker-compose up
Creating Chan     ... done
Creating Dina     ... done
Creating bitcoind ... done
Creating Bob      ... done
Creating Alice    ... done
Attaching to Chan, Dina, Alice, bitcoind, Bob
Alice      | Waiting for bitcoind to start...
Bob        | Waiting for bitcoind to start...
Dina       | Waiting for bitcoind to start...
Chan       | Waiting for bitcoind to start...
Bitcoind   | Starting bitcoind...
bitcoind   | Waiting for bitcoind to start
bitcoind   | bitcoind started
bitcoind   | ================================================

[...]

Chan       | Starting eclair...
Dina       | Starting lnd...
Chan       | Eclair node started
Alice      | ...Waiting for bitcoind to mine blocks...
Bob        | ...Waiting for bitcoind to mine blocks...
Alice      | Starting lnd...
Bob        | Starting c-lightning...

[...]
```

После запуска вы увидите целый поток журнальных файлов, когда каждый узел запускается и сообщает о своем продвижении. На вашем экране это может выглядеть довольно беспорядочно, но каждая строка выводимого результата имеет префикс имени контейнера, как было показано ранее. Если вы хотите просмотреть журналы только из одного контейнера, то можете это сделать в другом окне терминала, используя команду `docker-compose logs` с флагом `f` (*follow*) и конкретным именем контейнера:

```
$ docker-compose logs -f Alice
```

Открытие каналов и маршрутизирование платежа

Теперь наша сеть Lightning должна заработать. Как мы увидели в предыдущих разделах этой главы, выдавать команды работающему Docker-контейнеру можно с помощью команды `docker exec`. Независимо от того, запустили ли мы контейнер с помощью `docker run` или запустили их несколько с помощью

`docker-compose up`, мы все равно можем получить доступ к контейнерам по отдельности, используя команды Docker.

Демонстрация оплаты содержится в скрипте оболочки Bash, который называется *run-payment-demo.sh*. В целях выполнения этого демонстрационного примера на вашем компьютере должна быть проинсталлирована оболочка Bash. В большинстве Linux и Unix-подобных систем (например, macOS) оболочка `bash` прединсталлирована. Пользователи Windows могут проинсталлировать подсистему Windows для Linux и использовать дистрибутив Linux, такой как Ubuntu, чтобы получить собственную команду `bash` на своем компьютере.

Давайте выполним скрипт, чтобы увидеть его эффект, а затем посмотрим, как он работает внутри. Мы используем `bash`, чтобы выполнить ее как команду:

```
$ cd code/docker
$ bash run-payment-demo.sh
Starting Payment Demo
======================================================
Waiting for nodes to startup
- Waiting for bitcoind startup...
- Waiting for bitcoind mining...
- Waiting for Alice startup...
- Waiting for Bob startup...
- Waiting for Chan startup...
- Waiting for Dina startup...
All nodes have started
======================================================
Getting node IDs
- Alice: 0335e200756e156f1e13c3b901e5ed5a28b01a3131cd0656a27ac5cc20d4e71129
- Bob: 033e9cb673b641d2541aaaa821c3f9214e8a11ada57451ed5a0eab2a4afbce7daa
- Chan: 02f2f12182f56c9f86b9aa7d08df89b79782210f0928cb361de5138364695c7426
- Dina: 02d9354cec0458e0d6dee5cfa56b83040baddb4ff88ab64960e0244cc618b99bc3
======================================================

[...]

Setting up connections and channels
- Alice to Bob
- Open connection from Alice node to Bob's node
- Create payment channel Alice->Bob

[...]

Get 10k sats invoice from Dina
- Dina invoice:
lnbcrt100u1psnuzzrpp5rz5dg4wy27973yr7ehwns5ldeusceqdaq0hguu8c29n4nsqkznjsdqqc-
qzpgxqyz5vqsp5vdpehw33fljnmmexa6ljk55544f3syd8nfttqlm3
ljewu4r0q20q9qyyssqxh5nhkpjgfm47yxn4p9ecvndz7zddlsgpufnpyjl0kmnq227tdujlm0acd-
v39hcuqp2vhs40aav70c9yp0tee6tgzk8ut79mr877q0cpkjcfvr
======================================================
Attempting payment from Alice to Dina
Successful payment!
```

Как видно из результата, скрипт сначала получает идентификаторы узлов (публичные ключи) для каждого из четырех узлов. Затем он соединяет узлы и настраивает канал с 1 000 000 сатоши от каждого узла к следующему в сети.

Наконец, он выставляет счет на 10 000 сатоши с узла Дины и оплачивает счет с узла Алисы.

Если скрипт завершится отказом, то вы можете попробовать запустить его снова с самого начала. Или же вручную поочередно исполнить команды, находящиеся в скрипте, и посмотреть на результаты.

В этом скрипте есть многое, к чему придется время от времени возвращаться, но по мере того, как вы будете лучше понимать лежащую в основе технологию, все больше и больше этой информации станет ясной. Предлагаем вам вернуться к этому примеру позже.

Конечно же, с помощью данной тестовой сети можно делать гораздо больше, чем трехканальная четырехузловая оплата. Вот несколько идей для ваших экспериментов:

- создать более сложную сеть, запустив гораздо больше узлов различных типов. Отредактируйте файл *docker-compose.yml* и скопируйте разделы, переименовывая контейнеры по мере необходимости;
- соединить узлы в более сложных топологиях: кольцевые маршруты, ступица и спица или полная сетка;
- выполнить многочисленные платежи, чтобы исчерпать емкость канала. Затем выполните платежи в противоположном направлении, чтобы сбалансировать каналы. Посмотрите, как алгоритм маршрутизации адаптируется;
- изменить комиссионные за канал, чтобы увидеть, как алгоритм маршрутизации согласовывает несколько маршрутов и какие оптимизации он применяет. Будет ли дешевый, длинный маршрут лучше, чем дорогой, короткий маршрут?
- запустить циклический платеж от узла обратно к самому себе, чтобы сбалансировать свои собственные каналы. Посмотрите, как это влияет на все остальные каналы и узлы;
- создать сотни или тысячи небольших счетов в цикле, а затем оплачивать их как можно быстрее в другом цикле. Измерьте, сколько транзакций в секунду вы можете выжать из этой тестовой сети.

Lightning Polar[41] позволяет визуализировать сеть, с которой вы экспериментировали, используя Docker.

Вывод

В этой главе мы рассмотрели различные проекты, имплементирующие спецификации BOLT. Мы создали контейнеры для запуска образцовой сети Lightning и узнали, как создавать каждый проект из исходного кода. Теперь вы готовы провести разведку дальше и копать глубже.

[41] См. https://lightningpolar.com/.

Глава 5

Оперирование узлом сети Lightning

Прочитав до этого места, вы, вероятно, создали кошелек Lightning. В этой главе мы сделаем еще один шаг вперед и настроим полноценный узел Lightning. В дополнение к настройке узла мы научимся им оперировать и поддерживать его работу в течение долгого времени.

Есть много причин, по которым вы, возможно, захотите создать свой собственный узел Lightning.

Они таковы:

- быть полноправным, активным участником сети Lightning, а не просто конечным пользователем;
- управлять магазином электронной коммерции или получать доход посредством платежей Lightning;
- получать доход от комиссионных за маршрутизирование Lightning или за аренду ликвидности канала;
- разрабатывать новые службы, приложения или плагины для сети Lightning;
- повышать свою финансовую конфиденциальность при использовании Lightning;
- использовать некоторые приложения, созданные поверх Lightning, например приложения для обмена мгновенными сообщениями на базе Lightning;
- финансовая свобода, независимость и суверенитет.

С оперированием узла LN связаны расходы. Вам нужен компьютер, постоянное подключение к интернету, много места на диске и много времени! Эксплуатационные расходы будут включать расходы на электроэнергию.

Но навыки, которые вы приобретете благодаря этому опыту, бесценны и могут быть применены ко множеству других задач.

Давайте начнем!

Важно правильно задать свои собственные ожидания на основе точных фактов. Если вы планируете оперировать узлом Lightning исключительно для получения дохода за счет платы за маршрутизирование, то, пожалуйста, сначала тщательно выполните свою домашнюю работу. Вести прибыльный бизнес, оперируя узлом Lightning, определенно непросто. Рассчитайте все ваши первоначальные и текущие расходы в электронной таблице. Внимательно изучите статистику LN. Каков текущий объем платежей? Каков объем на узел? Каковы текущие средние комиссионные за маршрутизацию? Консультируйтесь на форумах и обращайтесь за советом или отзывами к другим членам сообщества, которые уже приобрели реальный опыт. Сформируйте свое собственное обоснованное мнение только после того, как вы выполнили это упражнение по надлежащей осмотрительности. Большинство людей найдут свою мотивацию для оперирования узлом не в финансовой выгоде, а в чем-то другом.

Выбор своей платформы

Оперировать узлом Lightning можно самыми разными путями, начиная от небольшого мини-ПК, размещенного у вас дома, или выделенного сервера и заканчивая внешним сервером в облаке. Выбранный вами метод будет зависеть от имеющихся у вас ресурсов и от того, сколько денег вы хотите потратить.

Почему для оперирования узлом Lightning важна надежность?

В системе Bitcoin вычислительное оборудование не особенно важно, если только кто-то специально не оперирует майнинговым узлом. Программным обеспечением узла Bitcoin Core можно оперировать на любой машине, которая соответствует его минимальным требованиям, и ему не нужно быть онлайн для получения платежей – только для их отправки. Если узел Bitcoin выходит из строя на длительный период времени, то пользователь может просто перезагрузить узел, и после подсоединения к остальной сети он выполнит ресинхронизацию блочной цепи.

Однако в Lightning пользователь должен быть онлайн как для отправки, так и для получения платежей. Если узел Lightning находится офлайн, то он не может получать какие-либо платежи от кого-либо, и, следовательно, его открытые счета не могут быть исполнены. Более того, открытые каналы офлайнового узла не могут использоваться для маршрутизирования платежей. Ваши партнеры по каналу заметят, что вы находитесь в офлайновом режиме, и не смогут с вами связаться, чтобы промаршрутизировать платеж. Если вы слишком часто находитесь в офлайновом режиме, то они, возможно, посчитают, что биткойн, привязанный в их каналах с вами, недостаточно используется, и, возможно, закроют эти каналы. Мы уже обсуждали случай атаки на протокол, при которой ваш партнер по каналу пытается вас обмануть, отправляя более раннюю фиксационную транзакцию. Если вы находитесь в офлайновом режиме и ваши каналы не мониторятся, то попытка кражи

может увенчаться успехом, и у вас не будет возможности обратиться за помощью по истечении срока привязки. Следовательно, для узла Lightning надежность чрезвычайно важна.

Существуют также проблемы с отказом оборудования и потерей данных. В Bitcoin аппаратный отказ может быть тривиальной проблемой, если у пользователя есть резервная копия своей мнемонической фразы или секретных ключей. Кошелек Bitcoin и биткойн внутри кошелька можно легко восстановить с помощью приватных ключей на новом компьютере. Бóльшая часть информации может быть скачана повторно из блочной цепи.

В отличие от этого, в Lightning информация о каналах пользователя, включая фиксационные транзакции и отзывные секреты, не является публичной и хранится только на оборудовании отдельного пользователя. Таким образом, программные и аппаратные отказы в сети Lightning могут легко приводить к потере средств.

Типы аппаратных узлов Lightning

Существует три главных типа аппаратных узлов Lightning.

Общецелевые компьютеры

Узел LN может работать на домашнем компьютере или ноутбуке под управлением Windows, macOS или Linux. Обычно он работает вместе с узлом Bitcoin.

Выделенное оборудование

Узел Lightning также может работать на выделенном оборудовании, таком как Raspberry Pi, Rock64 или мини-ПК. Эта конфигурация обычно выполняет программный стек, включающий узел Bitcoin и другие приложения. Указанная конфигурация популярна, потому что оборудование предназначено только для ведения и обслуживания узла Lightning и чаще настраивается с помощью инсталляционного «помощника».

Предварительно сконфигурированное оборудование

Узел LN также может работать на специально собранном оборудовании, специально выбранном и настроенном под него. Сюда будут входить готовые к применению технические решения по организации узла Lightning, которые можно приобрести в виде комплекта или системы «под ключ».

Оперирование в «облаке»

Виртуальный частный сервер (Virtual private server, аббр. VPS) и службы облачных вычислений, такие как Microsoft Azure, Google Cloud, Amazon Web Services (AWS) или DigitalOcean, вполне доступны по цене и могут быть настроены очень быстро. Узел Lightning может быть размещен в такой службе за сумму от 20 до 40 долларов в месяц.

Однако, как говорится, «облако – это просто компьютеры других людей». Использование этих служб означает оперирование вашим узлом на компьютерах других людей. Это влечет за собой соответствующие преимущества и недостатки. Ключевыми преимуществами являются удобство, эффектив-

ность, время безотказной работы и, возможно, даже стоимость. Облачный оператор управляет и руководит узлом на высочайшем уровне, автоматически обеспечивая вам удобство и эффективность. Он обеспечивает превосходное время безотказной работы и доступность, часто намного лучшие, чем то, чего может достичь отдельный человек дома. Если учесть, что только затраты на электроэнергию для работы сервера во многих западных странах составляют около 10 долларов в месяц, а затем добавьте к этому стоимость пропускной способности сети и самого оборудования, то предложение по VPS становится финансово конкурентноспособным. Наконец, с VPS не требуется место для домашнего компьютера, и у вас нет никаких проблем с шумом или перегревом компьютера. С другой стороны, есть несколько заметных недостатков. Работающий в «облаке» узел Lightning всегда будет менее безопасным и менее конфиденциальным, чем узел, работающий на вашем собственном компьютере. Кроме того, указанные службы облачных вычислений очень централизованы. Подавляющее большинство узлов Bitcoin и Lightning, работающих на таких службах, расположены в нескольких центрах обработки данных в Вирджинии, Саннивейле, Сиэтле, Лондоне и Франкфурте. Когда в сетях или центрах обработки данных этих провайдеров возникают проблемы с обслуживанием, это затрагивает тысячи узлов в так называемых «децентрализованных» сетях.

Если у вас есть возможность и способность оперировать узлом на вашем собственном компьютере дома или в офисе, то это может быть предпочтительнее, чем оперировать им в облаке. Тем не менее если оперирование вашим собственным сервером не является вариантом, то непременно рассмотрите возможность его работы на VPS.

Оперирование узлом дома

Если у вас дома или в офисе есть подключение к интернету с разумной пропускной способностью, то вы, безусловно, можете оперировать узлом Lightning там. Любого «широкополосного» соединения достаточно для оперирования облегченным узлом, а быстрое соединение позволит вам также оперировать полноценным узлом Bitcoin.

Хотя вы и можете оперировать узлом Lightning (и даже узлом Bitcoin) на своем ноутбуке, это довольно быстро станет раздражать. Указанные программы потребляют ресурсы вашего компьютера и должны работать 24/7. Ваши пользовательские приложения, такие как браузер или электронная таблица, будут конкурировать с фоновыми службами Lightning за ресурсы вашего компьютера. Другими словами, ваш браузер и другие рабочие нагрузки на рабочем столе будут замедлены. И когда ваше приложение по обработке текстов зависнет на ноутбуке, ваш узел Lightning тоже выйдет из строя, в результате чего вы не сможете получать транзакции и будете потенциально уязвимы для атак. Кроме того, вы никогда не должны выключать свой ноутбук. Все это в совокупности приводит к тому, что такой расклад не является идеальным. То же самое относится и к вашему персональному настольному компьютеру, которым вы пользуетесь ежедневно.

Вместо этого большинство пользователей предпочтут оперировать узлом на выделенном компьютере. К счастью, для этого не нужен компьютер класса

«сервер». Можно оперировать узлом Lightning на одноплатном компьютере, таком как Raspberry Pi, или на мини-ПК (обычно продаваемом как ПК для домашнего кинотеатра). Это простые компьютеры, которые чаще всего используются в качестве центра домашней автоматизации или медиасервера. Они относительно недороги по сравнению с ПК или ноутбуком. Преимущество выделенного устройства в качестве платформы для узлов Lightning и Bitcoin заключается в том, что оно может работать непрерывно, бесшумно и незаметно в вашей домашней сети, спрятанное за вашим маршрутизатором или телевизором. Никто даже не узнает, что эта маленькая коробочка на самом деле является частью глобальной банковской системы!

Не рекомендуется использовать узел в 32-битовой операционной системе и/или на 32-битовом процессоре, поскольку программное обеспечение узла может столкнуться с проблемами, связанными с ресурсами, что приведет к отказу и, возможно, потере средств.

Какое оборудование требуется для работы узла Lightning?

Как минимум для работы узла Lightning требуется следующее:

Центральный процессор

Для работы узла Bitcoin нужна достаточная вычислительная мощность, которая будет непрерывно скачивать и валидировать новые блоки. При настройке нового узла Bitcoin пользователь также должен учитывать начальное скачивание блока (initial block download, аббр. IBD), которое может занимать от нескольких часов до нескольких дней. Рекомендуется использовать 2-ядерный или 4-ядерный процессор.

Оперативная память

Система с 2 Гб оперативной памяти едва ли сможет поддерживать работу как узла Bitcoin, так и узла Lightning. Она будет работать намного лучше, по меньшей мере с 4 Гб оперативной памяти. IBD будет особенно сложным с объемом оперативной памяти менее 4 Гб. Более 8 Гб оперативной памяти не требуется, поскольку для этих типов служб центральный процессор является более узким местом вследствие криптографических операций, таких как валидирование подписей.

Накопитель для хранения данных

Это может быть жесткий диск (HDD) или твердотельный накопитель (SSD). Твердотельный накопитель будет значительно быстрее (но дороже) для оперирования узлом. Бóльшая часть хранилища используется для блочной цепи Bitcoin, размер которой составляет сотни гигабайт. Справедливый компромисс (между стоимостью и сложностью) заключается в покупке небольшого твердотельного накопителя для самозагрузки операционной системы и жесткого диска большего размера для хранения крупных объектов данных (в основном баз данных).

Raspberry Pi являются распространенным вариантом выбора для оперирования программным обеспечением узла из-за стоимости и доступности запчастей. Работающая на устройстве операционная система обычно самозагружается с защищенной цифровой (SD) карты. Для большинства случаев использования это не проблема, но Bitcoin Core печально известен своей интенсивностью ввода-вывода. Вы должны убедиться, что разместили блочную цепь Bitcoin и каталог данных Lightning на другом диске, потому что длительный интенсивный ввод-вывод может привести к сбою SD-карты.

Подключение к интернету

Надежное подключение к интернету требуется для скачивания новых блоков Bitcoin, а также для связи с другими одноранговыми узлами Lightning. Во время работы предполагаемое использование данных колеблется от 10 до 100 Гб в месяц, в зависимости от конфигурации. При запуске полноценный узел Bitcoin скачивает полную блочную цепь.

Источник питания

Требуется надежный источник питания, поскольку узлы Lightning должны быть постоянно подключены к сети. Сбой питания приведет к сбою текущих платежей. Для высоконагруженных маршрутизационных узлов в случае перебоев в подаче электроэнергии целесообразно иметь резервный источник питания или источник бесперебойного питания. В идеале вы также должны подключить к этому источнику свой интернет-маршрутизатор.

Резервная копия

Резервное копирование имеет решающее значение, поскольку сбой может привести к потере данных и, следовательно, к потере средств. Вам захочется рассмотреть какое-нибудь техническое решение по резервному копированию данных. Это может быть автоматическое резервное копирование в облаке на сервер или веб-служба, которые вы контролируете. В качестве альтернативы это может быть автоматическое резервное копирование на локальное оборудование, например второй жесткий диск. Для достижения наилучших результатов можно комбинировать как локальное, так и дистанционное резервное копирование.

Переключение серверной конфигурации в облаке

При аренде облачного сервера часто экономически выгодно менять конфигурацию между двумя фазами эксплуатации. Во время IBD-скачивания (например, в первый день) потребуется более быстрый процессор и более быстрое хранилище. После синхронизации блочной цепи требования к скорости процессора и хранилища значительно снижаются, поэтому производительность может быть снижена до более экономичного уровня.

Например, в облаке Amazon мы бы для IBD-скачивания использовали 8–16 Гб оперативной памяти, 8-ядерный процессор (например, t3-large или m3.large) и более быстрый твердотельный накопитель емкостью 400 Гб (более 1000 операций ввода-вывода в секунду [IOPS]), сократив его время всего до

6–8 часов. Как только это будет завершено, мы переключим экземпляр сервера на 2 Гб оперативной памяти, 2-ядерный процессор (например, t3.small) и накопитель на общецелевом жестком диске объемом 1 Тб. Это будет стоить примерно столько же, сколько если бы вы все время выполняли его на более медленном сервере, но это позволит вам достичь рабочего состояния менее чем за день, вместо того чтобы ждать IBD-скачивания почти неделю.

Постоянное хранилище данных (накопитель)

Если вы используете мини-ПК или арендуете сервер, то хранение может быть самой дорогой частью, которая стоит столько же, сколько компьютер и подключение (данные) вместе взятые.

Давайте взглянем на разные доступные варианты. Во-первых, существует два главных типа накопителей: жесткие диски и твердотельные накопители. Жесткие диски дешевле, а твердотельные накопители быстрее, но и те, и другие справляются со своей работой.

В самых быстрых из имеющихся сегодня твердотельных накопителях используется интерфейс энергонезависимой памяти Express (NVMe). Твердотельные накопители с протоколом доступа NVMe работают быстрее на высокопроизводительных машинах, но также и дороже. Традиционные твердотельные накопители на базе SATA дешевле, но не так быстры. Твердотельные накопители SATA работают достаточно хорошо для настройки вашего узла. Небольшие компьютеры, возможно, не смогут воспользоваться преимуществами твердотельных накопителей NVMe. Например, Raspberry Pi 4 не может извлечь из них выгоду из-за ограниченной пропускной способности своего USB-порта.

Для того чтобы выбрать размер, давайте посмотрим на блочную цепь Bitcoin. По состоянию на август 2021 года ее размер составляет 360 Гб, включая индекс транзакций, и увеличивается примерно на 60 Гб в год. Если вы хотите иметь некоторый запас для будущего роста или для инсталлирования других данных на свой узел, то приобретите диск объемом не менее 512 Гб, а еще лучше – диск объемом 1 Тб.

Использование инсталлятора или помощника

Инсталлирование узла Lightning или узла Bitcoin может оказаться сложной задачей, если вы незнакомы со средой командной строки. К счастью, существует ряд проектов, которые создают «помощников», то есть программное обеспечение, которое инсталлирует и конфигурирует различные компоненты за вас. Вам все равно нужно будет изучить некоторые заклинания командной строки для взаимодействия с вашим узлом, но бóльшая часть начальной работы делается за вас.

RaspiBlitz

Одним из самых популярных и полных «помощников» является *RaspiBlitz* (рис. 5-1), проект, созданный Кристианом Ротцоллом (Christian Rotzoll). Он предназначен для инсталлирования на Raspberry Pi 4. RaspiBlitz поставляется с рекомендуемым комплектом оборудования, который вы можете собрать

за считанные часы или максимум за выходные. Если вы посещаете «хакатон» Lightning в своем городе, то, скорее всего, увидите много людей, которые работают над настройкой RaspiBlitz, обмениваются советами и помогают друг другу. Проект RaspiBlitz можно найти на GitHub[42].

В дополнение к узлу Bitcoin и Lightning проект RaspiBlitz может инсталлировать ряд дополнительных служб, таких как:

- Tor (запускается как скрытая служба);
- ElectRS (сервер Electrum в Rust);
- сервер BTCPay (процессор криптовалютных платежей);
- BTC RPC Explorer (проводник по блочной цепи Bitcoin);
- Ride The Lightning (графический интерфейс управления узлом Lightning);
- LNbits (система кошельков / учетных записей Lightning);
- рабочий стол Spectre (мультиподписной Trezor, Ledger, кошелек Coldcard и Spectre-DIY);
- lndmanage (интерфейс командной строки для расширенного управления каналами);
- петля (служба подводных свопов);
- JoinMarket (служба CoinJoin).

Рис. 5-1. Узел RaspiBlitz

[42] См. https://github.com/rootzoll/raspiblitz.

myNode

myNode[43] – еще один популярный «вспомогательный» проект с открытым исходным кодом, включающий целый ряд программного обеспечения, связанного с Bitcoin. Он легко инсталлируется: вы «прошиваете» инсталлятор на SD-карту и самозагружаете свой мини-ПК с SD-карты. Для использования myNode не нужен какой-либо монитор, поскольку инструменты администрирования доступны дистанционно из браузера. Если на вашем мини-ПК нет монитора, мыши или клавиатуры, то вы можете управлять им с другого компьютера или даже со своего смартфона. После инсталляции перейдите на адрес http://mynode.local и создайте кошелек Lightning и узел в два клика.

В дополнение к узлу Bitcoin и Lightning проект myNode может дополнительно инсталлировать целый ряд дополнительных служб, таких как:

- Ride The Lightning (графический интерфейс управления узлом Lightning);
- OpenVPN (поддержка виртуальной частной сети [VPN] для дистанционного управления или кошелька);
- lndmanage (интерфейс командной строки для продвинутого управления каналами);
- BTC RPC Explorer (проводник по блочной цепи Bitcoin).

Umbrel

Известный своим UX/UI (показан на рис. 5-2), Umbrel предоставляет очень простой и доступный способ привести в рабочее состояние и запустить ваш узел Bitcoin и Lightning в кратчайшие сроки, в особенности для начинающих. Его весьма отличительной особенностью является то, что во время IBD-скачивания в Umbrel используется Neutrino/SPV, вследствие чего вы можете мгновенно начать использовать свой узел. Как только Bitcoin Core полностью синхронизируется в фоновом режиме, он автоматически переключается и отключает режим SPV. Umbrel ОС поддерживает Raspberry Pi 4, а также может быть инсталлирован в любую ОС на базе Linux или в виртуальную машину в macOS или Windows. Вы также можете подключить любой кошелек, который поддерживает Bitcoin Core P2P, Bitcoin Core RPC, протокол Electrum или lndconnect.

Нет необходимости ждать дождливого дня – вы можете отправиться прямо в Umbrel[44], чтобы узнать больше.

[43] См. https://mynodebtc.com/.

[44] См. https://getumbrel.com/.

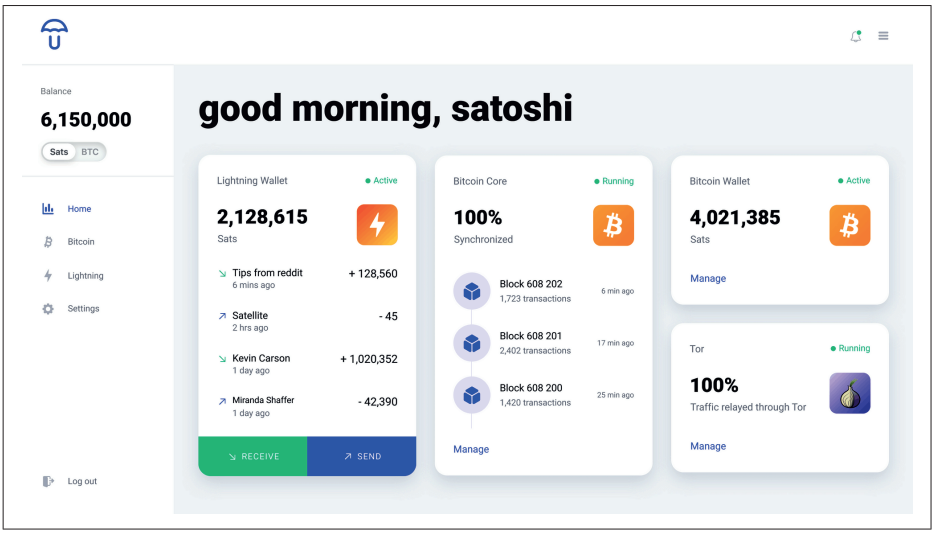


Рис. 5-2. Веб-интерфейс Umbrel

В дополнение к узлу Bitcoin и Lightning в программное обеспечение Umbrel имплементирован магазин приложений Umbrel, где можно легко инсталлировать дополнительные службы, такие как:

- терминал Lightning (интерфейс для управления ликвидностью канала, вывод средств на внутрицепные места назначения и пополнения кошельков из внутрицепных источников);
- Ride The Lightning (графический интерфейс управления узлом Lightning);
- рабочий стол Spectre (координатор только для наблюдения для мультиподписных кошельков Bitcoin с одним ключом);
- сервер BTCPay (процессор криптовалютных платежей);
- BTC RPC Explorer (проводник по блочной цепи Bitcoin);
- ThunderHub (мониторинг и управление вашим узлом);
- Sphinx Relay (управление подсоединением и хранением для чата Sphinx);
- mempool.space (визуализатор очереди mempool и проводник по блокам);
- LNbits (система кошельков / учетных записей Lightning).

Umbrel в настоящее время все еще находится в стадии бета-тестирования и не считается безопасным.

BTCPay Server

Несмотря на то что изначально платформа электронной коммерции и платежей BTCPay Server не была сконструирована как инсталляционный «помощник», она имеет невероятно простую систему инсталляции, в которой используются Docker-контейнеры и команда `docker-compose` для инсталлирования узла Bitcoin, узла Lightning и платежного шлюза, среди многих других служб. Она может инсталлироваться на различных аппаратных платформах, от простого Raspberry Pi 4 (рекомендуется 4 Гб) до мини-ПК или старого ноутбука, настольного компьютера или сервера.

BTCPay Server – это полнофункциональная автономная самоопекаемая платформа электронной коммерции, которая может интегрироваться со многими платформами электронной коммерции, такими как WordPress WOOCommerce и др. Инсталляция полноценного узла – это всего лишь шаг инсталляции платформы электронной коммерции. Хотя первоначально она была разработана как функциональная замена коммерческой платежной службы и API BitPay, однако эволюционировала в полноценную платформу для служб BTC и Lightning, связанных с электронной коммерцией. Для многих продавцов или магазинов это полностью готовое решение по электронной коммерции по типу «все в одном магазине».

В дополнение к узлу Bitcoin и Lightning сервер BTCPay также может инсталлировать различные службы, в том числе:

- узел Lightning c-lightning или LND;
- поддержку Litecoin;
- поддержку Monero;
- сервер Spark (веб-кошелек c-lightning);
- сервер взимания комиссионных (c-lightning ecommerce API);
- Ride The Ligh (веб-интерфейс для управления узлами Lightning);
- множество ответвлений BTC;
- BTCTransmuter (служба автоматизации «событие–действие», поддерживающая обмен валюты).

Число дополнительных служб и функциональностей быстро растет, поэтому приведенный выше список представляет собой лишь небольшую часть из того, что доступно на серверной платформе BTCPay.

Узел Bitcoin или облегченный узел Lightning

Одним из важнейших вариантов вашей настройки будет выбор узла Bitcoin и его конфигурации. Bitcoin Core, то есть референтная имплементация, является наиболее распространенным вариантом, но не единственным из имеющихся. Одним из альтернативных вариантов является btcd, имплементация узла Bitcoin на языке Go. btcd поддерживает некоторые функциональные свойства, которые полезны для работы Lightning-узла LND и недоступны в Bitcoin Core.

Второе соображение заключается в том, будете ли вы оперировать архивным узлом Bitcoin с полной копией блочной цепи (около 350 Гб на середину 2021 года) или же сокращенной блочной цепью, в которой хранятся только самые последние блоки. Сокращенная блочная цепь может сэкономить вам немного места на диске, но вам все равно нужно будет скачать полную блочную цепь хотя бы один раз (во время IBD-скачивания). Следовательно, это не сэкономит вам никакого сетевого трафика. Использование подрезанного узла для оперирования узлом Lightning все еще является экспериментальной возможностью и может не поддерживать всю функциональность. Однако многие люди успешно оперируют таким узлом.

Наконец, у вас также есть возможность вообще не оперировать узлом Bitcoin. Вместо этого можно оперировать Lightning-узлом LND в «облегченном» режиме, используя протокол Neutrino для извлечения информации о блочной цепи из публичных узлов Bitcoin, оперируемых другими. Подобная работа означает, что вы берете ресурсы из сети Bitcoin, не предлагая ничего взамен. Вмес-

то этого вы предлагаете свои ресурсы и вносите свой вклад в сообщество LN. В случае малых узлов Lightning это, как правило, сокращает сетевой трафик по сравнению с оперированием полным узлом Bitcoin.

Имейте в виду, что оперирование узлом Bitcoin позволяет поддерживать другие службы, помимо и поверх узла Lightning. Для этих других служб может потребоваться архивный (неподрезанный) узел Bitcoin, и нередко не получается работать без узла Bitcoin. Заранее подумайте о том, какие другие службы вы, возможно, захотите поддерживать сейчас или в будущем, чтобы принять обоснованное решение о типе выбранного вами узла Bitcoin.

Суть этого решения такова: если вы можете позволить себе диск размером более 500 Гб, то запустите полный архивный узел Bitcoin. Вы будете вносить ресурсы в систему Bitcoin и помогать другим, кто не может себе этого позволить. Если вы не можете позволить себе такой большой диск, то запустите подрезанный узел. Если вы не можете позволить себе диск или пропускную способность даже для подрезанного узла, то запустите облегченный узел LND поверх Neutrino.

Выбор операционной системы

Следующий шаг – выбрать операционную систему для вашего узла. Подавляющее большинство интернет-серверов работают на том или ином варианте Linux. Linux является предпочтительной платформой для интернета, потому что это мощная операционная система с открытым исходным кодом. Linux, однако, имеет крутую кривую обучения и требует знакомства со средой командной строки. Это часто отпугивает новых пользователей.

В конечном счете большинство служб могут выполняться в любой современной операционной системе POSIX, которая включает macOS, Windows и, конечно же, Linux. Ваш выбор должен быть обусловлен больше вашим знакомством и удобством работы с операционной системой и вашими учебными целями. Если вы хотите расширить свои знания и научиться управлять системой Linux, то это отличная возможность сделать это с помощью конкретного проекта и четкой цели. Если вы просто хотите привести узел в рабочее состояние и его запустить, то используйте то, что вы знаете.

В настоящее время многие службы также предоставляются в виде контейнеров, обычно на основе системы Docker. Эти контейнеры можно разворачивать в различных операционных системах, абстрагируясь от опорной операционной системы. Тем не менее вам, возможно, потребуется изучить некоторые команды Linux CLI, так как большинство контейнеров внутри выполняют какой-либо вариант Linux.

Выбор имплементации узла Lightning

Как и в случае выбора операционной системы, ваш выбор имплементации узла Lightning должен зависеть в первую очередь от вашего знакомства с языком программирования и инструментами разработки, используемыми в проектах. Хотя между различными имплементациями узлов существуют некоторые небольшие различия в функциональных возможностях, они относительно незначительны, и большинство имплементаций сходятся на общих стандартах, определенных в спецификациях BOLT.

С другой стороны, знакомство с языком программирования и сборочной системой является хорошей основой для выбора узла. Это связано с тем, что инсталлирование, конфигурирование, текущее сопровождение и устранение неполадок будут связаны со взаимодействием с различными инструментами, используемыми сборочной системой. Сюда входят:

- Make, Autotools и утилиты GNU для c-lightning;
- утилиты Go для LND;
- Java/Maven для Eclair.

Язык программирования влияет не только на выбор сборочной системы, но и на многие другие аспекты программы. Каждый язык программирования имеет целую философию конструирования и влияет на многие другие аспекты, такие как:

- формат и синтаксис конфигурационных файлов;
- расположение файлов (в файловой системе);
- аргументы командной строки и их синтаксис;
- форматирование сообщения об ошибке;
- необходимые библиотеки;
- интерфейсы дистанционного вызова процедур.

Когда вы выбираете свой узел Lightning, вы также выбираете все вышеупомянутые характеристики. Таким образом, ваше знакомство с этими инструментами и философией конструирования облегчит оперирование узлом. Или усложнит, если вы приземляетесь в незнакомой области.

С другой стороны, если это ваш первый опыт работы с командной строкой и средой сервера/служб, вы окажетесь незнакомыми с любой имплементацией и получите возможность узнать что-то совершенно новое. В этом случае вы можете принять решение, основываясь на ряде других факторов, таких как:

- качество форумов поддержки и чатов;
- качество документации;
- степень интеграции с другими инструментами, которыми вы хотите оперировать.

В качестве последнего соображения вы, возможно, захотите проинспектировать производительность и надежность разных имплементаций узлов. Это особенно важно, если вы будете использовать этот узел в производственной среде и ожидаете интенсивного трафика и высоких требований к надежности. Это может иметь место, если вы планируете вести на нем платежную систему магазина.

Инсталлирование узла Bitcoin или Lightning

Вы решили не использовать инсталляционный «помощник», а вместо этого погрузиться в командную строку операционной системы Linux? Это смелое решение, и мы постараемся помочь вам воплотить его в жизнь. Если вы предпочитаете не пытаться делать это вручную, то рассмотрите возможность использования приложения, которое поможет вам проинсталлировать программное обеспечение узла, или технического решения на основе контейнеров, как описано в разделе «Использование инсталлятора или помощника» на стр. 131.

В этом разделе мы углубимся в продвинутую тему системного администрирования из командной строки. Администрирование Linux является отдельным набором навыков, который выходит за рамки этой книги. Это сложная тема, и в ней есть много подводных камней. Действуйте с осторожностью!

В следующих нескольких разделах мы кратко опишем, как проинсталлировать и сконфигурировать узлы Bitcoin и Lightning в операционной системе Linux. Вам нужно будет ознакомиться с инструкциями по инсталляции конкретных приложений узлов Bitcoin и Lightning, которые вы решили использовать. Обычно вы найдете их в файле с именем *INSTALL* или в подкаталоге *docs* каждого проекта. Мы опишем только некоторые общие шаги, применимые ко всем таким службам, и предлагаемые нами инструкции неизбежно будут неполными.

Фоновые службы

Для тех, кто привык выполнять приложения на своем рабочем столе или в смартфоне, приложение всегда имеет графический пользовательский интерфейс, даже если иногда может работать в фоновом режиме. Однако приложения узла Bitcoin и Lightning сильно отличаются. Эти приложения не имеют встроенного графического пользовательского интерфейса. Вместо этого они работают как безголовые фоновые службы, то есть они всегда работают в фоновом режиме и не взаимодействуют с пользователем напрямую.

Это может создать некоторую путаницу для пользователей, которые не привыкли выполнять фоновые службы. Как узнать, работает ли такая служба в данный момент? Как ее запускать и останавливать? Как с ней взаимодействовать? Ответы на эти вопросы зависят от используемой вами операционной системы. На данный момент мы будем считать, что вы используете какой-то вариант Linux, и ответим на них в этом контексте.

Изоляция процесса

Фоновые службы обычно работают в рамках определенной учетной записи пользователя, чтобы изолировать их от операционной системы и друг от друга. Например, Bitcoin Core сконфигурирован под выполнение от пользовательского имени `bitcoin`. Вам нужно будет использовать командную строку, чтобы создавать пользователя для каждой из управляемых вами служб.

Кроме того, если вы подключили внешний диск, то вам нужно будет сообщить операционной системе, чтобы та переместила домашний каталог пользователя на этот диск. Это связано с тем, что такая служба, как Bitcoin Core, будет создавать файлы в домашнем каталоге пользователя. Если вы настраиваете ее под скачивание полной блочной цепи Bitcoin, то эти файлы будут занимать несколько сотен гигабайт. Здесь мы исходим из того, что вы подключили внешний диск и он находится в пути */external_drive/* операционной системы.

В большинстве систем Linux нового пользователя можно создать с помощью команды useradd, например:

```
$ sudo useradd -m -d /external_drive/bitcoin -s /dev/null bitcoin
```

Флаги `m` и `d` создают домашний каталог пользователя, как в данном случае задано каталогом */external_drive/bitcoin*. Флаг `s` задает интерактивную оболочку

пользователя. В данном случае мы устанавливаем для нее значение */dev/null*, чтобы отключить использование интерактивной оболочки. Последний аргумент – это пользовательское имя `bitcoin` нового пользователя.

Запуск узла

Как для служб узла Bitcoin, так и для служб узла Lightning «инсталляция» также включает в себя создание так называемого *скрипта запуска*, чтобы обеспечить запуск узла во время самозагрузки компьютера. Запуск и завершение работы фоновых служб обрабатываются процессом операционной системы, который в Linux называется `init` или `systemd`. Обычно скрипт запуска системы находится в подкаталоге `contrib` каждого проекта. Например, если вы используете современную ОС Linux, использующую `systemd`, то вы найдете скрипт под названием *bitcoind.service*, который может запускать и останавливать службу узла Bitcoin Core.

Ниже приведен пример того, как выглядит скрипт запуска узла Bitcoin, взятый из репозитория Bitcoin Core:

```
[Unit]
Description=Bitcoin daemon
After=network.target

[Service]
ExecStart=/usr/bin/bitcoind -daemon \
                            -pid=/run/bitcoind/bitcoind.pid \
                            -conf=/etc/bitcoin/bitcoin.conf \
                            -datadir=/var/lib/bitcoind

# Обеспечить, чтобы каталог конфигурации был читаемым пользователем службы
PermissionsStartOnly=true
ExecStartPre=/bin/chgrp bitcoin /etc/bitcoin

# Управление процессом
######################

Type=forking
PIDFile=/run/bitcoind/bitcoind.pid
Restart=on-failure
TimeoutStopSec=600

# Создание каталога и разрешения
################################

# Выполнять как bitcoin:bitcoin
User=bitcoin
Group=bitcoin

# /run/bitcoind
RuntimeDirectory=bitcoind
RuntimeDirectoryMode=0710

# /etc/bitcoin
ConfigurationDirectory=bitcoin
ConfigurationDirectoryMode=0710
```

```
# /var/lib/bitcoind
StateDirectory=bitcoind
StateDirectoryMode=0710

[...]

[Install]
WantedBy=multi-user.target
```

Проинсталлируйте скрипт как корневой (`root`) пользователь, скопировав его в папку */lib/systemd/system/* службы `systemd`, а потом перезагрузите `systemd`:

```
$ sudo systemctl daemon-reload
```

Затем активируйте службу:

```
$ sudo systemctl enable bitcoind
```

Теперь вы можете запускать и останавливать службу. Пока не запускайте ее, так как мы еще не настроили узел Bitcoin.

```
$ sudo systemctl start bitcoind
$ sudo systemctl stop bitcoind
```

Конфигурирование узла

В целях конфигурирования своего узла вам необходимо создать конфигурационный файл и на него ссылаться. По традиции этот файл обычно создается в */etc* в каталоге с именем программы. Например, конфигурации Bitcoin Core и LND обычно хранятся соответственно в */etc/bitcoin/bitcoin.conf* и */etc/lnd/lnd.conf*.

Эти конфигурационные файлы представляют собой текстовые файлы, каждая строка которых выражает одну конфигурационную опцию и ее значение. Для всего, что не определено в конфигурационном файле, принимаются дефолтные значения. То, какие опции можно задать в конфигурации, можно увидеть двумя способами. Во-первых, выполнение приложения узла с аргументом help покажет опции, которые можно определить в командной строке. Эти же опции можно определить в конфигурационном файле. Во-вторых, обычно пример конфигурационного файла со всеми дефолтными опциями можно найти в репозитории программного кода.

По одному примеру конфигурационного файла можно найти в каждом образе Docker, которые мы использовали в главе 4. Например, файл *code/docker/bitcoind/bitcoind/bitcoin.conf*:

```
regtest=1
server=1
debuglogfile=debug.log
debug=1
txindex=1
printtoconsole=0

[regtest]
fallbackfee=0.000001
port=18444
noconnect=1
dnsseed=0
```

```
dns=0
upnp=0
onlynet=ipv4
rpcport=18443
rpcbind=0.0.0.0
rpcallowip=0.0.0.0/0
rpcuser=regtest
rpcpassword=regtest
zmqpubrawblock=tcp://0.0.0.0:12005
zmqpubrawtx=tcp://0.0.0.0:12006
```

Этот конкретный конфигурационный файл конфигурирует Bitcoin Core для работы в качестве узла `regtest` и предоставляет слабое пользовательское имя и пароль для дистанционного доступа, поэтому вам не следует использовать его для конфигурирования вашего узла. Однако он служит для иллюстрации синтаксиса конфигурационного файла, и вы можете внести в него изменения в Docker-контейнере, чтобы поэкспериментировать с разными опциями. Посмотрите, можете ли вы использовать команду `bitcoind -help`, чтобы понять, что делает каждая опция в контексте сети Docker, которую мы создали в главе 4.

Нередко дефолтных значений достаточно, и с помощью нескольких изменений программное обеспечение вашего узла может быть сконфигурировано быстро. Для того чтобы узел Bitcoin Core работал с минимальной индивидуальной настройкой, вам нужно всего четыре строки конфигурации:

```
server=1
daemon=1
txindex=1
rpcuser=ПОЛЬЗОВАТЕЛЬНОЕ_ИМЯ
rpcpassword=ПАРОЛЬ
```

Даже опция `txindex` не является строго необходимой, хотя она обеспечивает, чтобы ваш узел Bitcoin создавал индекс всех транзакций, что требуется для некоторых приложений. Опция `txindex` не требуется для оперирования узлом Lightning.

Для Lightning-узла `c-lightning`, работающего на том же сервере, тоже требуется всего несколько строк в конфигурации:

```
network=mainnet
bitcoin-rpcuser=ПОЛЬЗОВАТЕЛЬНОЕ_ИМЯ
bitcoin-rpcpassword=ПАРОЛЬ
```

В целом неплохо сводить к минимуму объем конфигурирования этих систем. Дефолтная конфигурация тщательно продумана с целью поддержания наиболее распространенных развертываний. Если вы измените дефолтное значение, то это может вызвать проблемы позже или снизить производительность вашего узла. Одним словом, модифицируйте только тогда, когда это необходимо!

Конфигурирование сети

Конфигурирование сети обычно не является проблемой во время конфигурирования нового приложения. Однако одноранговые сети, такие как Bitcoin и Lightning, создают для конфигурирования сети несколько уникальных проблем.

В централизованной службе ваш компьютер подсоединяется к «большим серверам» какой-либо корпорации, а не наоборот. Ваше домашнее интернет-соединение фактически сконфигурировано исходя из допущения о том, что вы просто являетесь потребителем служб, предоставляемых другими. Но в одноранговой системе каждый одноранговый узел одновременно потребляет и предоставляет службы другим узлам. Если вы используете узел Bitcoin или Lightning у себя дома, то вы предоставляете службу другим компьютерам в интернете. Ваша интернет-служба по умолчанию не сконфигурирована так, чтобы вы могли управлять серверами, и вам, возможно, потребуется некоторое дополнительное конфигурирование, чтобы другие могли достигать вашего узла.

Если вы хотите оперировать узлом Bitcoin или Lightning, то вам необходимо предоставить возможность другим узлам в интернете подсоединяться к вам. Это означает обеспечение возможности входящих TCP-подключений к порту Bitcoin (по умолчанию порт 8333) или порту Lightning (по умолчанию порт 9735). Хотя вы можете иметь узел Bitcoin без входящего подключения, это нельзя сделать с узлом Lightning. Узел Lightning должен быть доступен для других пользователей извне вашей сети.

По умолчанию ваш домашний интернет-маршрутизатор не ожидает входящих соединений извне, и фактически входящие соединения блокируются. IP-адрес вашего интернет-маршрутизатора является единственным IP-адресом, доступным извне, и все компьютеры, которые вы используете в своей домашней сети, используют этот единственный IP-адрес. Это достигается с помощью механизма, именуемого *трансляцией сетевых адресов* (Network Address Translation, аббр. NAT), который позволяет вашему интернет-маршрутизатору выступать в качестве посредника для всех исходящих соединений. Если вы хотите разрешить входящее соединение, то вам необходимо настроить *переадресацию портов*, которая сообщает вашему интернет-маршрутизатору, что входящие соединения по определенным портам должны перенаправляться на определенные компьютеры внутри сети. Это можно сделать вручную, изменив конфигурацию вашего интернет-маршрутизатора, или, если ваш маршрутизатор это поддерживает, с помощью механизма автоматической переадресации портов, именуемого *Универсальной автоматической настройкой сетевых устройств* (Universal Plug and Play, аббр. UPnP).

Альтернативным механизмом переадресации портов является активирование луковичного маршрутизатора (The Onion Router, аббр. Tor), который обеспечивает своего рода наложение виртуальной частной сети, позволяющее осуществлять входящие подсоединения к *луковичному адресу*. Если вы используете Tor, то вам не нужно выполнять переадресацию портов или активировать входящие подсоединения к портам Bitcoin или Lightning. Если вы оперируете своими узлами с помощью Tor, то весь трафик проходит через Tor, и никакие другие порты не используются.

Давайте рассмотрим разные способы, которыми вы можете предоставлять возможность другим пользователям подсоединяться к вашему узлу. Мы рассмотрим эти альтернативы по порядку, от самых простых до самых сложных.

Это просто работает!

Существует вероятность того, что ваш интернет-провайдер или маршрутизатор по умолчанию сконфигурирован на поддержку UPnP, и все просто работа-

ет автоматически. Давайте сначала попробуем этот подход, на всякий случай, если нам повезет.

Исходя из того, что у вас уже работает узел Bitcoin или Lightning, мы попытаемся выяснить, доступны ли они извне.

Для того чтобы этот тест сработал, у вас должен быть либо узел Bitcoin, либо узел Lightning (или и то, и другое), настроенный и работающий в вашей домашней сети. Если ваш маршрутизатор поддерживает UPnP, то входящий трафик будет автоматически перенаправляться на соответствующие порты компьютера, на котором работает узел.

Можно задействовать несколько очень популярных и полезных веб-сайтов, чтобы узнать ваш внешний IP-адрес и выяснить, разрешает и перенаправляет ли он входящие соединения на известный порт. Вот два надежных из них:

- https://canyouseeme.org;
- https://www.whatismyip.com/port-scanner.

По умолчанию эти службы позволяют вам проверять входящие соединения только к тому IP-адресу, с которого вы подсоединяетесь. Это делается для того, чтобы вы не могли использовать службу для сканирования сетей и компьютеров других людей. Вы увидите внешний IP-адрес вашего маршрутизатора и поле ввода номера порта. Если вы не изменили дефолтные порты в конфигурации вашего узла, то попробуйте порт 8333 (Bitcoin) и/или 9735 (Lightning).

На рис. 5-3 вы видите результат проверки порта 9735 на сервере, который оперирует узлом Lightning, с помощью инструмента сканирования портов *whatismyip.com*. Он показывает, что сервер принимает входящие подсоединения к порту Lightning. Если вы видите такой результат, то у вас все готово!

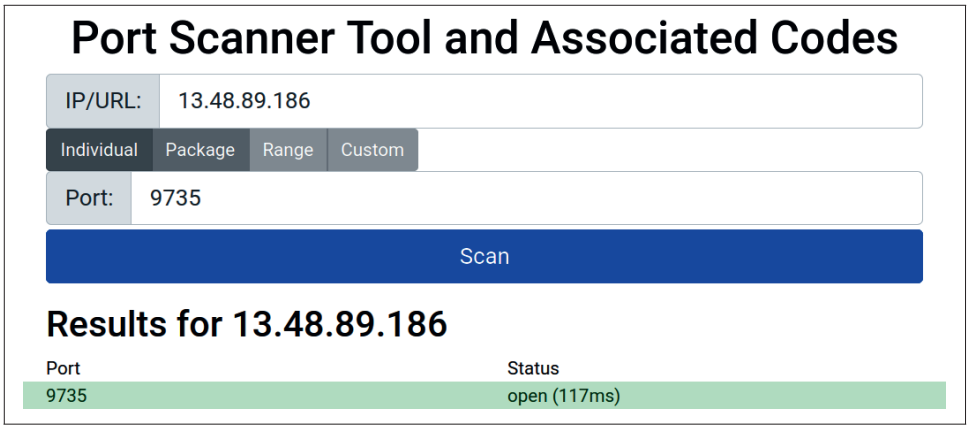


Рис. 5-3. Проверка входящего порта 9735

Автоматическая переадресация портов с использованием UPnP

Иногда, даже если ваш интернет-маршрутизатор поддерживает UPnP, он может быть по умолчанию отключен. В этом случае вам необходимо изменить

конфигурацию вашего интернет-маршрутизатора в его веб-интерфейсе администрирования.

1. Подключитесь к веб-сайту настройки вашего интернет-маршрутизатора. Обычно это можно сделать, подключившись к *шлюзовому адресу* вашей домашней сети с помощью веб-браузера. Шлюзовой адрес можно найти, просмотрев IP-конфигурацию любого компьютера в вашей домашней сети. Нередко это первый адрес в одной из немаршрутизируемых сетей, такой как 192.168.0.1 или 10.0.0.1. Проверьте все стикеры на вашем маршрутизаторе, а также шлюзовой адрес. Если найдете, то откройте браузер и введите IP-адрес в URL-поле браузера / поле поиска, например «192.168.0.1» или «http://192.168.0.1».
2. Найдите пользовательское имя и пароль администратора для конфигурационной веб-панели маршрутизатора. Это нередко написано на наклейке на самом маршрутизаторе и может быть просто «admin» и «password». Быстрый поиск в интернете вашего интернет-провайдера и модели маршрутизатора также поможет вам найти эту информацию.
3. Найдите настройку UPnP и включите ее.

Перезапустите свой узел Bitcoin и/или Lightning и повторите тест открытого порта с одним из веб-сайтов, которые мы использовали в предыдущем разделе.

Использование Tor для входящих соединений

Луковичный маршрутизатор (The Onion Router, аббр. Tor) – это VPN со специальным свойством, заключающимся в том, что он шифрует сообщения между луковичными слоями (переходами или переходными узлами – hops), вследствие чего любой промежуточный узел не может определить источник или пункт назначения пакета. Узлы и Bitcoin, и Lightning поддерживают работу через Tor, в силу чего вы получаете возможность управлять узлом, не раскрывая свой IP-адрес или местоположение. Следовательно, он обеспечивает высокий уровень конфиденциальности вашего сетевого трафика. Дополнительным преимуществом работы Tor является то, что, поскольку он работает как VPN, он решает задачу переадресации портов с вашего интернет-маршрутизатора. Входящие соединения принимаются по туннелю Tor, и ваш узел можно найти по специально сгенерированному луковичному адресу вместо IP-адреса.

Активирование Tor требует двух шагов. Во-первых, на свой компьютер необходимо проинсталлировать маршрутизатор и прокси Tor. Во-вторых, необходимо активировать использование прокси Tor в вашей конфигурации Bitcoin или Lightning.

Для инсталлирования Tor в системе Ubuntu Linux, использующей пакетный менеджер apt, выполните:

```
sudo apt install tor
```

Далее мы конфигурируем наш узел Lightning на использование Tor для его внешней связности. Вот пример конфигурации для LND:

```
[Tor]
tor.active=true
tor.v3=true
tor.streamisolation=true
listen=localhost
```

Это позволит активировать Tor (`tor.active`), установить луковичную службу v3 (`tor.v3=true`), использовать другой луковичный поток для каждого соединения (`tor.streamisolation`) и ограничить прослушивание соединений только локальным хостом, чтобы избежать утечки вашего IP-адреса (`isten=localhost`).

Проверить правильность инсталлирования и работы Tor можно, выполнив простую однострочную команду. Эта команда должна работать на большинстве версий Linux:

```
curl --socks5 localhost:9050 --socks5-hostname localhost:9050 -s https://
check.torproject.org/ | cat | grep -m 1 Congratulations | xargs
```

Если все работает правильно, то ответом на эту команду должно быть «`Congratulations. This browser is configured to use Tor`» (Поздравляем. Этот браузер настроен на использование Tor).

Из-за природы Tor вы не можете с легкостью применить внешнюю службу, чтобы проверить доступность вашего узла по луковичному адресу. Тем не менее вы должны увидеть свой луковичный адрес Tor в журналах вашего узла Lightning. Это длинная строка из букв и цифр, за которой следует суффикс `.onion`. Теперь ваш узел должен быть доступен из интернета с дополнительным бонусом конфиденциальности!

Ручная переадресация портов

Это самый сложный процесс, требующий немалых технических навыков. Детали зависят от типа вашего интернет-маршрутизатора, настроек и политик вашего поставщика служб и множества других факторов. Прежде чем пробовать этот гораздо более трудный механизм, сначала попробуйте UPnP или Tor.

Базовые шаги заключаются в следующем:

1. Найдите IP-адрес компьютера, на котором находится ваш узел. Обычно он распределяется динамически протоколом динамического конфигурирования хоста (Dynamic Host Configuration Protocol, аббр. DHCP) и часто находится где-то в диапазоне 192.168.x.x или 10.x.x.x.
2. Найдите MAC-адрес управления доступом к среде (media access control, аббр. MAC) сетевого интерфейса вашего узла. Его можно найти в настройках интернета этого компьютера.
3. Назначьте статический IP-адрес вашему узлу, чтобы он всегда был одним и тем же. Вы можете использовать IP-адрес, который у него есть в данный момент. На вашем интернет-маршрутизаторе найдите «Static Leases» (Статическая аренда) в разделе конфигурации DHCP. Соотнесите MAC-адрес с выбранным вами IP-адресом. Теперь вашему узлу всегда будет выделяться этот IP-адрес. В качестве альтернативы можно обратиться к конфигурации DHCP вашего маршрутизатора и узнать, каков диапазон его DHCP-адресов. Выберите неиспользуемый адрес за пределами диапазона адресов DHCP. Затем на сервере настройте сеть на прекращение использования DHCP и жестко закодируйте выбранный IP-адрес, отличный от DHCP, в конфигурации сети операционной системы.
4. Наконец, настройте «Port Forwarding» (Переадресацию портов) на вашем интернет-маршрутизаторе для маршрутизирования входящего трафика по определенным портам на выбранный IP-адрес вашего сервера.

После завершения переконфигурирования повторите проверку порта, используя один из веб-сайтов из предыдущих разделов.

Безопасность вашего узла

Узел Lightning – это, по определению, *горячий кошелек*. Это означает, что средства (как внутри цепи, так и вне цепи), контролируемые узлом Lightning, напрямую контролируются ключами, которые загружаются в память узла или хранятся на жестком диске узла. Если узел Lightning скомпрометирован, то можно очень легко создавать транзакции внутри цепи или вне цепи, чтобы истощить его средства. Поэтому крайне важно его защитить от несанкционированного доступа.

Безопасность – это целостное усилие, в том смысле, что вы должны обеспечить безопасность каждого уровня системы. Как говорится: цепь настолько прочна, насколько прочно самое слабое звено. Это важная концепция информационной безопасности, и мы применим ее к нашему узлу.

Несмотря на все меры безопасности, которые вы предпримете, помните, что сеть Lightning – это экспериментальная технология на ранней стадии, и в исходном коде любого используемого вами проекта могут быть дефекты, которые можно использовать. Не вкладывайте в сеть Lightning больше денег, чем вы готовы рискнуть потерять.

Безопасность операционной системы

Защита операционной системы – это обширная тема, выходящая за рамки нашей книги. Однако мы можем сформировать несколько базовых принципов.

В целях обеспечения безопасности вашей операционной системы следует учитывать несколько главных пунктов:

Происхождение

Начните с того, что убедитесь, что вы скачиваете правильный образ операционной системы, и проверьте все подписи или контрольные суммы перед его инсталлированием. Распространите это на любое инсталлируемое вами программное обеспечение. Перепроверяйте любой источник или URL-адрес, с которого вы скачиваете, дважды. Проверьте целостность и правильность скачанного программного обеспечения с помощью проверки подписи и контрольной суммы.

Техническое сопровождение

Обеспечьте поддержание своей операционной системы в актуальном состоянии. Активируйте автоматическую ежедневную или еженедельную инсталляцию обновлений безопасности. Наименьшие привилегии: настройте пользователей для определенных процессов и предоставьте им наименьший доступ, необходимый для оперирования службой. Не запускайте процессы с правами администратора (например, `root`).

Изоляция процесса

Используйте функциональности операционной системы по изолированию процессов друг от друга.

Разрешения файловой системы

Тщательно настройте файловую систему по принципу наименьших привилегий. Не делайте файлы доступными для чтения или записи для всех.

Надежная аутентификация

Используйте надежные случайно сгенерированные пароли либо, по возможности, аутентификацию с публичным ключом. Например, безопаснее вместо пароля использовать защищенную оболочку (SSH) с парой криптографических ключей.

Двухфакторная аутентификация (2FA)

Используйте двухфакторную аутентификацию везде, где это возможно, включая Универсальный второй фактор (Universal 2nd Factor, аббр. U2F) с аппаратными ключами безопасности. Это относится ко всем внешним службам, которые вы можете использовать, таким как ваш поставщик облачных служб. Ее также можно применить к вашей собственной настройке, например к вашей собственной конфигурации SSH. Используйте 2FA и для косвенных служб. Например, предположим, что вы используете облачную службу. Вы указали своему поставщику облачных служб адрес электронной почты, поэтому вам также следует защитить свой адрес электронной почты с помощью 2FA.

Резервное копирование

Создавайте резервные копии своей системы и также обеспечьте защиту резервных копий с помощью шифрования. Периодически выполняйте эти резервные копирования. По меньшей мере один раз проверьте, что можете реконструировать свою резервную копию и что она является полной и доступной. Если возможно, храните одну копию своих резервных копий на другом диске, чтобы избежать отказа единственного жесткого диска, который уничтожит как ваш активный узел, так и ваши резервные копии.

Управление уязвимостью и внешним воздействием

Используйте дистанционное сканирование, чтобы минимизировать атакуемую поверхность вашей системы. Закройте все ненужные службы или порты. Инсталлируйте только то программное обеспечение и пакеты, которые вам действительно нужны и которые вы используете. Удалите пакеты, которые вы больше не используете. Рекомендуется не использовать свой узловой компьютер для действий, не связанных с узлом, которые вы можете выполнять на другом своем компьютере. В первую очередь, если можете, не используйте свой узловой компьютер для поиска, серфинга в интернете или чтения вашей электронной почты.

Это список самых базовых мер безопасности. Он ни в коем случае не является исчерпывающим.

Доступ к узлу

Ваш узел Lightning будет выставлять наружу API дистанционного (удаленного) вызова процедур (remote procedure call, аббр. RPC). Это означает, что вашим узлом можно управлять дистанционно с помощью команд, отправляемых на

определенный TCP-порт. Контроль доступа к этому API RPC-вызовов достигается с помощью некоторой формы аутентификации пользователя. В зависимости от типа настроенного вами узла Lightning это будет сделано либо с помощью аутентификации по пользовательскому имени /паролю, либо с помощью механизма, именуемого аутентификационным «макаруном» (macaroon, миндальное печенье). Как следует из названия, это более сложный вид печенья. В отличие от файла cookie, он имеет криптографическую подпись и может выражать набор возможностей доступа.

Например, LND использует «макаруны» для предоставления доступа к API RPC-вызовов. По умолчанию программное обеспечение LND создает три макаруна с разными уровнями доступа, которые называются admin, invoice и readonly. В зависимости от того, какой макарун вы копируете и используете в своем RPC-клиенте, у вас есть доступ только для чтения, доступ к счетам (который включает возможности только для чтения) или доступ администратора, который дает вам полный контроль. В LND также есть функция bakery (выпечки) макаруна, которая может строить макаруны с любой комбинацией возможностей с очень тонким контролем.

Если вы используете модель аутентификации по пользовательскому имени/паролю, убедитесь, что вы выбрали длинный и случайный пароль. Вам не придется часто вводить этот пароль, потому что он будет сохранен в конфигурационных файлах. Поэтому вы должны выбрать тот, который нельзя угадать. Многие из примеров, которые вы увидите, включают плохо подобранные пароли, и часто люди копируют их в свои собственные системы, предоставляя легкий доступ любому. Не делайте этого! Используйте парольный менеджер для создания длинного случайного буквенно-цифрового пароля. Поскольку некоторые специальные символы, такие как $?/!*\&%‘"’, могут мешать работе командной строки, лучше избегать их в паролях, которые будут использоваться в среде командной оболочки. Во избежание проблем придерживайтесь длинных случайных буквенно-цифровых паролей.

Обычно достаточно простой буквенно-цифровой последовательности длиной более 12 символов, сгенерированной случайным образом. Если вы планируете хранить на своем узле Lightning крупные суммы денег и обеспокоены дистанционными атаками грубой силой, выберите пароль длиной более 20 символов, чтобы сделать такие атаки практически невозможными.

Резервное копирование узла и каналов

Очень важным соображением при оперировании узлом Lightning является задача резервного копирования. В отличие от кошелька Bitcoin, где мнемоническая фраза BIP-39 может восстанавливать все состояние кошелька, в Lightning это не так.

В кошельках Lightning действительно используется резервная копия мнемонической фразы BIP-39, но только для внутрицепного кошелька. Однако из-за того, как строятся каналы, для восстановления узла Lightning мнемонической фразы недостаточно. Необходим дополнительный слой резервных копий, который называется *статической резервной копией каналов* (static channel backup, аббр. SCB). Без SCB-копии оператор узла Lightning может потерять все

средства, находящиеся в каналах, в случае если он потеряет хранилище данных узла Lightning.

Не финансируйте каналы до тех пор, пока вы не создадите систему постоянного резервного копирования состояния вашего канала. Ваши резервные копии должны быть перемещены «за пределы сайта» в другую систему и местоположение вдали от вашего узла, чтобы они могли пережить различные системные отказы (отключение питания, повреждение данных и т. д.) или стихийные бедствия (наводнение, пожар и т. д.).

SCB-копии не являются панацеей. Во-первых, необходимо создавать резервные копии состояния каждого канала всякий раз, когда происходит новая фиксационная транзакция. Во-вторых, реконструировать из резервной копии канала опасно. Если у вас нет последней фиксационной транзакции и вы случайно широковещательно передаете старую (отозванную) фиксацию, то ваш одноранговый узел будет исходить из того, что вы пытаетесь обмануть и потребовать весь остаток канала с помощью штрафной транзакции. Для того чтобы быть уверенными в закрытии канала, вам нужно выполнить *кооперативное закрытие*. Но во время этого кооперативного закрытия злонамеренный одноранговый узел может дезориентировать ваш узел, заставив его выполнить широковещательную передачу старой отозванной фиксации, тем самым обманывая вас и заставляя ваш узел непреднамеренно пытаться обмануть.

Кроме того, резервные копии ваших каналов должны быть зашифрованы для обеспечения вашей конфиденциальности и безопасности вашего канала. В противном случае любой, кто найдет резервные копии, сможет не только увидеть все ваши каналы, но и использовать резервные копии для закрытия всех ваших каналов таким образом, чтобы передать остаток вашим коллегам по каналу. Другими словами, злоумышленник, получивший доступ к вашим резервным копиям, может привести к потере всех средств вашего канала.

Как вы видите, SCB-копии не являются надежной защитой. Они представляют собой слабый компромисс, поскольку меняют один тип риска (повреждение или потеря данных) на другой тип риска (вредоносный одноранговый узел). Для того чтобы выполнить реконструкцию из SCB-копии, вы должны взаимодействовать со своими одноранговыми узлами по каналу и надеяться, что они не попытаются вас обмануть, дав вам старую фиксацию либо обманув ваш узел, заставив его выполнить широковещательную передачу отозванной фиксации, чтобы он мог вас оштрафовать. Несмотря на недостатки SCB-копии, SCB-копии все-таки имеют смысл, и вы должны их делать. Если вы не сделаете SCB-копий и потеряете данные своего узла, то вы навсегда потеряете средства своего канала. Гарантировано! Однако если вы сделаете SCB-копии и потеряете данные своего узла, то у вас есть разумный шанс, что некоторые из ваших коллег честны и что вы сможете вернуть часть средств своего канала. Если вам повезет, то вы сможете вернуть все свои средства. В заключение для вас лучше всего делать непрерывные SCB-копии на диске, отличном от первичного жесткого диска узла.

Механизмы резервного копирования каналов все еще находятся в стадии разработки и являются слабым местом в большинстве имплементаций Lightning.

На момент написания этой главы только LND предлагал встроенный механизм SCB-копирования. Eclair имеет аналогичный механизм для развертываний на стороне сервера, хотя Eclair Mobile предлагает дополнительное резервное копирование на Google Диск. В c-lightning недавно были объединены необходимые интерфейсы для плагина в целях имплементирования резервного копирования каналов. К сожалению, в разных имплементациях узлов не существует согласованного, выверенного механизма резервного копирования.

Резервное копирование баз данных узла Lightning на основе файлов в лучшем случае является частичным решением, поскольку вы рискуете создать резервную копию несогласованного состояния базы данных. В дополнение к этому вы, возможно, будете ненадежно отслеживать последние фиксации состояния. Гораздо лучше иметь механизм резервного копирования, который срабатывает всякий раз, когда происходит изменение состояния канала, тем самым обеспечивая согласованность данных.

В целях настройки SCB-копий в LND задайте параметр backupfilepath в командной строке либо в конфигурационном файле. Затем LND сохранит SCB-файл по этому пути к каталогу. Разумеется, это только первый шаг к решению задачи. Теперь вам нужно настроить механизм, который отслеживает этот файл на предмет изменений. Всякий раз, когда файл меняется, механизм резервного копирования должен копировать этот файл на другой, предпочтительно внешний диск. Такие механизмы резервного копирования выходят за рамки данной книги. Тем не менее любое сложное решение по резервному копированию должно быть способно справиться с этим сценарием. Напомним, что файлы резервных копий тоже должны быть зашифрованы.

Риск со стороны горячего кошелька

Как мы уже обсуждали ранее, сеть Lightning состоит из сети *горячих кошельков*. Средства, которые вы храните в кошельке Lightning, постоянно находятся онлайн. Это делает их уязвимыми. Следовательно, вы не должны хранить крупные суммы в кошельке Lightning. Крупные суммы должны храниться в холодном кошельке, который не находится онлайн и который может совершать транзакции только внутри цепи.

Даже если вы начнете с малого, со временем вы все равно, возможно, обнаружите, что у вас в кошельке Lightning лежит значительная сумма денег. Это типичный сценарий для владельцев магазинов. Если вы используете узел Lightning для операций электронной коммерции, то ваш кошелек, скорее всего, будет получать средства часто, но отправлять их редко. Таким образом, в конечном итоге у вас возникнут две проблемы одновременно. Во-первых, ваши каналы будут несбалансированными, с большими локальными остатками, перевешивающими малые дистанционные остатки. Во-вторых, у вас будет слишком много денег в кошельке. К счастью, вы также можете решить обе эти проблемы одновременно.

Давайте рассмотрим некоторые решения, которые можно использовать, чтобы уменьшить сумму средств, размещенных в горячем кошельке.

Зачистка средств

Если остаток вашего кошелька Lightning станет слишком большим для вашей рискоустойчивости, то вам нужно будет «зачистить» (sweep) средства из ко-

шелька. Это можно сделать тремя способами: зачисткой внутри цепи, зачисткой вне цепи и петли вывода. Давайте рассмотрим каждый из этих вариантов в следующих нескольких разделах.

Внутрицепная зачистка

Зачистка средств внутри цепи осуществляется путем перевода средств с кошелька Lightning на кошелек Bitcoin. Это делается путем закрытия каналов. Когда вы закрываете канал, все средства с вашего локального остатка «зачищаются» на адрес Bitcoin. Адрес Bitcoin для внутрицепных средств обычно генерируется вашим кошельком Lightning, и поэтому это тот же горячий кошелек. Возможно, вам потребуется выполнить дополнительную внутрицепную транзакцию, чтобы перевести средства на более безопасный адрес, например сгенерированный на вашем аппаратном кошельке.

Закрытие каналов повлечет за собой внутрицепные комиссионные и снизит емкость и возможности подсоединения вашего узла Lightning. Однако если вы оперируете популярным узлом электронной коммерции, то у вас не будет недостатка в емкости, и вы сможете стратегически закрывать каналы с большими локальными остатками, по сути, «комплектуя» ваши средства для перемещения внутри цепи. Возможно, вам потребуется использовать некоторые методы перебалансировки каналов (см. раздел «Перебалансировка каналов» на стр. 161) перед закрытием каналов, чтобы максимизировать преимущества этой стратегии.

Внецепная зачистка

Вы можете задействовать еще один метод – оперирование вторым узлом Lightning, который не рекламируется в сети. Вы можете установить каналы большой емкости из вашего публичного узла (например, того, на котором работает ваш магазин) на ваш неразрекламированный (скрытый) узел и на регулярной основе «зачищать» средства, совершая платеж Lightning на свой скрытый узел.

Преимущество этого метода заключается в том, что узел Lightning, который получает платежи за ваш магазин, будет публично известен. Это делает его мишенью для хакеров, поскольку предполагается, что любой узел Lightning, связанный с магазином, имеет большой остаток. Второй узел, который не связан с вашим магазином, будет нелегко идентифицировать как ценную цель.

В качестве дополнительной меры безопасности можно сделать свой второй узел скрытой службой Tor, чтобы ее IP-адрес не был известен. Это еще больше снижает вероятность атак и повышает вашу конфиденциальность.

Вам нужно будет настроить скрипт, который выполняется через регулярные промежутки времени. Цель этого скрипта – создать счет на вашем скрытом узле и оплатить этот счет с узла вашего магазина, тем самым переведя средства на ваш скрытый узел.

Имейте в виду, что этот метод не перемещает средства в холодное хранилище. Оба узла Lightning являются горячими кошельками. Целевая задача этой зачистки – перевести средства с очень известного горячего кошелька на малоизвестный горячий кошелек.

Зачистка на основе подводного свопа

Еще один способ уменьшить остаток вашего горячего кошелька Lightning – это использовать метод, именуемый *подводным свопом*. Подводные (субмаринные)

свопы, разработанные соавтором Олаолувой Осунтокуном и Алексом Босвортом (Alex Bosworth), позволяют обменивать внутрицепные биткойны на платежи Lightning и наоборот. По сути, подводные свопы – это атомарные свопы между внецепными средствами Lightning и внутрицепными средствами Bitcoin.

Оператор узла может инициировать подводный своп и отправить все доступные остатки каналов другой стороне, которая в обмен отправит ему внутрицепной биткойн.

В будущем это может стать платной службой, предлагаемой узлами сети Lightning, которые рекламируют курсы обмена или взимают фиксированные комиссионные за конвертацию.

Преимущество подводного свопа для зачистки средств заключается в том, что ни один канал не нужно закрывать. Это означает, что мы сохраняем наши каналы, только перебалансируя остаток наших каналов с помощью этой операции. Когда мы отправляем платеж Lightning, то переводим часть остатка с локального на дистанционный по одному или нескольким нашим каналам. Это не только уменьшает остаток, открытый в горячем кошельке нашего узла, но и увеличивает остаток, доступный для будущих входящих платежей.

Вы могли бы сделать это, доверив посреднику действовать в качестве шлюза, но это имеет риск кражи ваших монет. Однако в случае подводного свопа операция не требует доверия. Подводный своп – это неопекаемые *атомарные* операции, и, стало быть, контрагент в вашем подводном свопе не может украсть ваши средства, потому что внутрицепной платеж зависит от завершения внецепного платежа, и наоборот.

Подводные свопы с помощью петли

Одним из примеров службы подводного свопа является *петля* (Loop) от Lightning Labs, той же компании, которая строит LND. Петля поставляется в двух вариантах: петля ввода (Loop In) и петля вывода (Loop Out). Петля ввода принимает внутрицепной Bitcoin-платеж и конвертирует его во внецепной Lightning-платеж. Петля вывода конвертирует Lightning-платеж в Bitcoin-платеж[45].

Для того чтобы использовать указанную службу петлевого перевода средств, у вас должен работать Lightning-узел LND.

С целью уменьшения остатка вашего горячего кошелька Lightning вы могли бы воспользоваться службой петлевого вывода – Loop Out. В целях использования службы Loop необходимо проинсталлировать на свой узел некоторое дополнительное программное обеспечение. Программное обеспечение Loop

[45] Из документации Lightning Labs: петля вывода предназначена для коммерсантов, служб и пользователей, которые в основном получают средства через Lightning. Петля вывода служит связующим звеном, позволяя отправлять средства из сети Lightning во «внутрицепные» пункты назначения, такие как биржевые счета или системы холодного хранения. Петля ввода позволяет обычным пользователям «пополнять» свои кошельки Lightning, когда средства заканчиваются. В этих случаях пользователи «переводят» средства в кошельки Lightning из «внутрицепных» источников (например, бирж или холодного хранилища). – *Прим. перев.*

работает вместе с вашим узлом LND и предоставляет некоторые инструменты командной строки для исполнения подводных свопов. Программное обеспечение Loop и инструкции по инсталляции можно найти на GitHub[46].

После того как будет проинсталлировано и запущено программное обеспечение, операция петлевого вывода Loop Out будет сводиться к выполнению одной команды:

```
loop out --amt 501000 --conf_target 400
Max swap fees for 501000 sat Loop Out: 25716 sat
Regular swap speed requested, it might take up to 30m0s for the swap to be ex-
ecuted.
CONTINUE SWAP? (y/n), expand fee detail (x): x

Estimated on-chain sweep fee:        149 sat
Max on-chain sweep fee:            14900 sat
Max off-chain swap routing fee:    10030 sat
Max no show penalty (prepay):       1337 sat
Max off-chain prepay routing fee:     36 sat
Max swap fee: 750 sat
CONTINUE SWAP? (y/n): y
Swap initiated

Run `loop monitor` to monitor progress.
```

Обратите внимание, что ваш максимальный размер комиссионных, который представляет наихудший сценарий, будет зависеть от выбранной вами целевой задачи подтверждения.

Время безотказной работы и доступность узла Lightning

В отличие от узлов Bitcoin, узлы Lightning должны находиться онлайн почти непрерывно. Ваш узел должен быть онлайн, чтобы получать платежи, открывать каналы, закрывать каналы (кооперативно) и отслеживать нарушения протокола. В сети Lightning доступность узлов является настолько важным требованием, что показатель доступности используется различными инструментами автоматического управления каналами (например, автопилотом) для определения того, с какими узлами открывать каналы. Показатель «доступность» можно также увидеть в качестве метрики узла в популярных проводниках по узлам (см. «Проводники Lightning» на стр. 42), таких как 1ML.

Доступность узла особенно важна для смягчения и устранения потенциальных нарушений протокола (т. е. отозванных фиксаций). Хотя вы и можете позволить себе короткие перерывы от часа до одного или двух дней, вы не можете отключать свой узел в течение более длительных периодов времени без риска потери средств.

Поддерживать узел онлайн непросто, так как различные дефекты и ограничения ресурсов могут и иногда будут приводить к простою. В особенности если вы используете загруженный и популярный узел. В этом случае вы столкнетесь с ограничениями памяти, свопового пространства, числа открытых файлов, дискового пространства и т. д. Целый ряд самых разных проблем будет приводить к аварийному отказу вашего узла или вашего сервера.

[46] См. https://github.com/lightninglabs/loop.

Допускайте неисправности и автоматизируйте

Если у вас есть время и навыки, то вам следует протестировать некоторые основные сценарии отказа в тестовой сети Lightning. Из тестовой сети вы извлечете ценные уроки, не рискуя никакими средствами. Любой шаг, который вы выполните для автоматизации вашей системы, повысит вашу доступность.

Автоматический перезапуск компьютерного сервера

Что произойдет, когда ваш сервер или операционная система выходит из строя? Что произойдет при отключении электроэнергии? Просимулируйте эту неисправность, нажав кнопку **сброс** на вашем компьютере или отсоединив кабель питания. После аварийного отказа, сброса или сбоя питания компьютер должен автоматически перезагрузиться. У некоторых компьютеров в BIOS есть настройка, указывающая, как компьютер должен реагировать на сбои питания. Проверьте ее, чтобы убедиться, что при сбое питания компьютер перезагружается действительно автоматически без вмешательства человека.

Автоматический перезапуск узла

Что произойдет, когда ваш узел или один из ваших узлов выходит из строя? Просимулируйте эту неисправность, отключив соответствующие процессы узла. Если узел аварийно отказывает, то он должен автоматически перезапуститься. Протестируйте его, чтобы убедиться, что узел или узлы действительно перезапускаются при сбое автоматически без вмешательства человека. Если это не так, то, скорее всего, ваш узел неправильно настроен как служба операционной системы.

Автоматическое переподключение к сети

Что произойдет, если ваша сеть «ляжет»? Что произойдет, когда ваш интернет-провайдер временно отключится? Что произойдет, когда ваш интернет-провайдер назначит новый IP-адрес вашему маршрутизатору или вашему компьютеру? Когда сеть возвращается, автоматически ли подключаются к сети узлы, которыми вы оперируете? Просимулируйте эту неисправность, отсоединив, а затем повторно подключив кабель Ethernet от устройства, на котором размещены ваши узлы. Узлы должны автоматически переподсоединяться и продолжать работу без вмешательства человека.

Конфигурирование своих журнальных файлов

Все описанные выше сбои должны оставлять текстовые записи в соответствующих журнальных файлах. Увеличьте детализацию журналирования, если это необходимо. Найдите эти записи об ошибках в журнальных файлах и используйте их для мониторинга.

Мониторинг доступности узла

Мониторинг вашего узла является важной частью поддержания его работоспособности. Вам необходимо следить не только за доступностью самого компьютера, но и за доступностью и корректной работой программного обеспечения узла Lightning.

Это можно делать несколькими способами, но большинство из них требуют некоторой индивидуальной настройки. Вы можете использовать генерические инструменты мониторинга инфраструктуры или мониторинга приложений, но вы должны настроить их специально для опрашивания API узла Lightning, чтобы обеспечить исправную работу узла, его синхронизацию с блочной цепью и подсоединение к одноранговым каналам.

Lightning.watch[47] предоставляет специализированную службу, которая предлагает мониторинг узлов Lightning. Она использует Telegram-бота, чтобы уведомлять вас о любых перебоях в службе. Это бесплатная служба, хотя вы можете заплатить (через Lightning, конечно), чтобы получать более быстрые оповещения.

Со временем мы ожидаем, что все больше сторонних служб будут предоставлять специализированный мониторинг узлов Lightning, оплачиваемый с помощью микроплатежей. Возможно, такие службы и их API станут стандартизированными и однажды будут напрямую поддерживаться программным обеспечением узла Lightning.

Сторожевые вышки

Сторожевые вышки (Watchtowers) – это механизм для передачи мониторинга на аутсорсинг и вынесения решений о штрафных санкциях за нарушения протокола Lightning.

Как упоминалось в предыдущих главах, протокол Lightning обеспечивает безопасность с помощью механизма штрафных санкций. Если один из ваших партнеров по каналу выполняет широковещательную передачу старой фиксационной транзакции, то вашему узлу необходимо будет исполнить отзывное условие и выполнить широковещательную передачу штрафной транзакции, чтобы избежать потери денег. Но если ваш узел выйдет из строя во время нарушения протокола, то вы можете потерять деньги.

В целях решения этой проблемы мы можем использовать одну или несколько сторожевых вышек для передачи на аутсорсинг работы по мониторингу нарушений протокола и выдаче штрафных транзакций. Настройка сторожевой башни состоит из двух частей: сервер сторожевой вышки (или просто сторожевая вышка), который отслеживает блочную цепь, и клиент сторожевой вышки, который запрашивает у сервера сторожевой вышки эту службу мониторинга.

Технология сторожевой вышки все еще находится на ранних стадиях разработки и не пользуется широкой поддержкой. Однако в следующем ниже отрывке мы перечислим несколько экспериментальных имплементаций, которые вы можете попробовать.

Программное обеспечение LND включает в себя как сторожевой сервер, так и сторожевой клиент. Вы можете активировать сторожевой сервер, добавив следующие ниже опции конфигурации:

```
[watchtower]
watchtower.active=1
watchtower.towerdir=/путь_к_каталогу_сторожевых_данных
```

[47] См. https://lightning.watch/.

Сторожевой клиент LND можно задействовать, активировав его в конфигурации, а затем с помощью командной строки подключив его к сторожевому серверу. Конфигурация такова:

```
[wtclient]
wtclient.active=1
```

Клиент командной строки `lncli` приложения LND показывает следующие ниже опции по управлению сторожевым клиентом:

```
$ lncli wtclient

NAME:
   lncli wtclient - Interact with the watchtower client.

USAGE:
   lncli wtclient command [command options] [arguments...]

COMMANDS:
     add     Register a watchtower to use for future sessions/backups.
     remove  Remove a watchtower to prevent its use for future sessions/back-
ups.
     towers  Display information about all registered watchtowers.
     tower   Display information about a specific registered watchtower.
     stats   Display the session stats of the watchtower client.
     policy  Display the active watchtower client policy configuration.

OPTIONS:
   --help, -h show help
```

`c-lightning` имеет API-интерфейсы, необходимые для плагина сторожевого клиента, хотя такой плагин еще не имплементирован.

Наконец, популярным автономным сторожевым сервером является Eye of Satoshi (TEOS). Его можно найти на GitHub[48].

Управление каналами

Как оператор узла Lightning, одной из повторяющихся задач, которые вам необходимо будет выполнять, является управление вашими каналами. Это означает открытие исходящих каналов с вашего узла на другие узлы, а также предоставление другим узлам возможности открывать входящие каналы для вашего узла. В будущем может стать возможным кооперативное строительство каналов, в результате чего вы сможете открывать симметричные каналы, в которых средства выделяются на обоих концах при создании. На данный момент, однако, у новых каналов есть средства только на одном конце, на стороне создателя. Следовательно, чтобы ваш узел был сбалансирован как по входящей, так и по исходящей емкости, вам необходимо открывать каналы для других и побуждать других открывать каналы для вашего узла.

[48] См. https://github.com/talaia-labs/python-teos.

Открытие исходящих каналов

После того как вы приведете свой узел Lightning в рабочее состояние, вы сможете пополнить его кошелек Bitcoin, а затем начать открывать каналы с помощью этих средств.

Вы должны тщательно выбирать партнеров по каналу, потому что способность вашего узла отправлять платежи зависит от того, кем являются ваши партнеры по каналу и насколько хорошо они связаны с остальной частью сети Lightning. Вы также захотите иметь более одного канала, чтобы избежать подверженности одной точке отказа. Поскольку Lightning теперь поддерживает многокомпонентные платежи, вы можете разделять свои первоначальные средства на несколько каналов и маршрутизировать более крупные платежи, совмещая их емкости. В то же время избегайте того, чтобы ваши каналы были слишком малыми. Поскольку вам необходимо платить комиссионные за Bitcoin-транзакции, чтобы открывать и закрывать канал, остаток канала не должен быть настолько мал, чтобы внутрицепные комиссионные занимали значительную часть. Все дело в остатке!

Подводя итог:

- соединяйтесь с каналами, имеющими хорошую связность;
- открывайте более одного канала;
- не открывайте слишком много каналов;
- не делайте каналы слишком малыми.

Один из способов найти каналы с хорошей связностью – открыть канал для популярного продавца, продающего товары в сети Lightning. Эти узлы, как правило, хорошо финансируются и имеют хорошую связность. Таким образом, когда вы будете готовы купить что-то онлайн через Lightning, вы можете открыть канал непосредственно к узлу продавца. Идентификатор узла продавца будет указан в счете, который вы получите, когда попытаетесь что-то купить. Это облегчает задачу.

Еще один способ найти узлы с хорошей связностью состоит в использовании проводника по сети Lightning (см. «Проводник Lightning» на стр. 42), например 1ML[49], и просмотреть список узлов, отсортированных по емкости канала и числу каналов. Не выбирайте самые большие узлы, потому что это способствует централизации. Выбирайте узел в середине списка, чтобы вы могли помочь им расти. Еще одним фактором, который следует учитывать, может быть промежуток времени, в течение которого узел был в действии. Узлы, созданные более года назад, скорее всего, будут более надежными и менее рискованными, чем узлы, которые начали работать неделю назад.

Автопилот

Задача открытия каналов может быть частично автоматизирована с помощью автопилота, который представляет собой программное обеспечение, открывающее каналы автоматически на основе некоторых эвристических правил. Программное обеспечение автопилота все еще относительно новое, и оно не всегда выбирает для вас лучших партнеров по каналам. Особенно вначале,

[49] См. https://1ml.com/.

возможно, было бы лучше открывать каналы вручную. Автопилоты в настоящее время существуют в трех формах:

- `lnd` включает в свой состав автопилот, который полностью интегрирован с `lnd` и постоянно работает в фоновом режиме при включении;
- `lib_autopilot.py` может предлагать вычисления автопилота для любой имплементации узла на основе эмидемических и канальных данных;
- плагин `c-lightning`, основанный на `lib_autopilot.py`, предоставляет простой в использовании интерфейс для пользователей `c-lightning`.

Имейте в виду, что автопилот `lnd` начнет работать в фоновом режиме, как только он будет включен через конфигурационный файл. И как результат он немедленно начнет открывать каналы, если в вашем кошельке `lnd` есть внутрицепные выходы. Если вы хотите иметь полный контроль над биткойновыми транзакциями, которые вы совершаете, и каналами, которые вы открываете, то обязательно отключите автопилот, прежде чем загружать свой кошелек `lnd` биткойновыми средствами. Если автопилот был включен ранее, то вам, возможно, придется перезапустить свой `lnd`, прежде чем пополнять свой кошелек внутрицепной транзакцией или перед закрытием каналов, что фактически снова даст вам внутрицепные средства. Крайне важно задать ключевые конфигурационные значения, если вы хотите работать с автопилотом. Взгляните на следующий ниже пример конфигурации:

```
[lnd-autopilot]
autopilot.active=1
autopilot.maxchannels=40
autopilot.allocation=0.70
autopilot.minchansize=500000
autopilot.maxchansize=5000000
autopilot.heuristic=top_centrality:1.0
```

Приведенный выше конфигурационный файл активирует автопилот. Он будет открывать каналы при условии, что будут соблюдаться следующие ниже два условия:

1) в настоящее время на вашем узле открыто менее 40 каналов;
2) менее 70 % ваших суммарных средств находятся вне цепи в платежных каналах.

Здесь числа 40 и 0.7 выбраны совершенно произвольно, потому что мы не можем дать никаких рекомендаций, которые будут валидны для всех, о том, сколько каналов вы должны открывать и какой процент ваших средств должен быть вне цепи. Автопилот в `lnd` не будет учитывать внутрицепные комиссионные. Другими словами, он не будет задерживать открытие каналов до периода времени, когда комиссионные будут низкими. В целях снижения комиссионных вы можете открывать каналы вручную в течение периода времени, когда комиссионные являются низкими, например в выходные дни. Автопилот будет давать рекомендации по каналу всякий раз, когда будут соблюдены условия, и немедленно попытается открыть канал, используя соответствующие текущие комиссионные. Согласно приведенному выше конфигурационному файлу, каналы будут иметь размер от 5 mBTC (минимальный размер = 500 000 сатоши) до 50 mBTC (максимальный размер = 5 000 000 сатоши). Как обычно,

суммы в конфигурационном файле указаны в сатоши. В настоящее время каналы ниже 1 mBTC не очень полезны, и мы не рекомендуем вам открывать каналы, которые слишком малы и ниже этого значения. С более широким внедрением многокомпонентных платежей меньшие каналы становятся меньшим бременем. Но на данный момент это наша рекомендация.

Плагин `c-lightning`, первоначально написанный Рене Пикхардтом (соавтором этой книги), работает совсем по-другому по сравнению с автопилотом `lnd`. Во-первых, он отличается алгоритмами, используемыми для выработки рекомендаций. Мы не будем обсуждать его здесь. Во-вторых, он отличается своим пользовательским интерфейсом. Вам нужно будет скачать плагин автопилота из репозитория плагинов `c-lightning` и его активировать.

Для активации плагина в `c-lightning` поместите его в каталог *~/.lightning/plugins*, убедитесь, что он является исполняемым (например, `chmod +x ~/.lightning/plugins/autopilot.py`), затем перезапустите `lightningd`.

В качестве альтернативы, если вы не хотите, чтобы плагин активировался автоматически при запуске `lightningd`, можно поместить его в другой каталог и активировать вручную с помощью аргумента `plugin` для `lightningd`:

```
lightningd --plugin=~/lightning-plugins/autopilot.py
```

Автопилот в `c-lightning` управляется с помощью трех конфигурационных значений, которые можно задать в конфигурационном файле или в качестве аргументов командной строки при запуске `lightningd`:

```
[c-lightning-autopilot]
autopilot-percent=75
autopilot-num-channels=10
autopilot-min-channel-size-msat=100000000msat
```

Эти значения являются фактической дефолтной конфигурацией, и вам вообще не нужно их задавать.

Указанный автопилот не будет запускаться автоматически в фоновом режиме, как в lnd. Вместо этого вы должны запустить прогон специально с помощью команды lightning-cli autopilot-run-once, если хотите, чтобы автопилот открывал рекомендуемые каналы. Но если вы хотите, чтобы он просто предоставлял вам рекомендации, из которых вы можете выбирать узлы вручную, то можете добавить опциональный аргумент dryrun.

Ключевое различие между автопилотами lnd и c-lightning заключается в том, что автопилот c-lightning также будет давать рекомендации по размеру канала. Например, если автопилот рекомендует открыть канал с малым узлом, который имеет только небольшие каналы, то он не будет рекомендовать открыть большой канал. Однако если он открывает канал с узлом, имеющим хорошую связность, который также имеет много больших каналов, то он, вероятно, порекомендует бóльший размер канала.

Как вы видите, автопилот c-lightning не такой автоматический, как в lnd, но он дает немного больше контроля. Эти различия отражают личные предпочтения и фактически могут стать решающим фактором для вас при выборе одной имплементации перед другой.

Имейте в виду, что современные автопилоты в основном будут использовать публичную информацию из эпидемического протокола о текущей топологии сети Lightning. Очевидно, что ваши личные требования к каналам могут быть отражены только в определенной степени. Более продвинутые автопилоты будут использовать историческую информацию и информацию об используемости, собранную вашим узлом при запуске в прошлом, включая информацию об успехах маршрутизации, о том, кому вы платили в прошлом и кто платил вам. В будущем такие усовершенствованные автопилоты могут также использовать эти собранные данные для выработки рекомендаций по закрытию каналов и перераспределению средств.

В целом на момент написания этой главы с автопилотами следует проявлять осторожность, чтобы не зависеть или не полагаться на них слишком сильно.

Получение входящей ликвидности

В текущей конструкции сети Lightning пользователи, как правило, получают исходящую ликвидность до получения входящей ликвидности. Они делают это, открывая канал с другим узлом, и чаще всего они могут тратить биткойн до того, как смогут его получить. Существует три типичных способа получения входящей ликвидности:

- открывать канал с исходящей ликвидностью, а затем тратить часть этих средств. Теперь остаток находится на другом конце канала, а это значит, что вы можете получать платежи;
- попросить кого-нибудь открыть канал для вашего узла. Предложить ответить взаимностью, чтобы оба ваших узла стали более связанными и сбалансированными;
- использовать подводный своп (например, петлю ввода – Loop In) для обмена внутрицепным BTC на входящий канал в ваш узел;
- оплачивать стороннюю службу, чтобы открывать канал с вами. Существует несколько таких служб. Некоторые взимают комиссионные за предоставление ликвидности, некоторые бесплатны.

Вот список имеющихся в настоящее время поставщиков ликвидности, которые будут открывать канал для вашего узла за определенную плату:

- служба Thor от Bitrefill[50];
- Lightning для меня[51];
- LNBig[52];
- Lightning Conductor (Громоотвод)[53].

Задача создания входящей ликвидности является довольно сложной как с практической точки зрения, так и с точки зрения пользовательского опыта. Входящая ликвидность не происходит автоматически, поэтому вам нужно найти способы ее создания для вашего узла. Эта асимметрия платежных каналов также не является интуитивной. В большинстве других платежных систем

[50] См. https://www.bitrefill.com/thor-lightning-network-channels.

[51] См. https://lightningto.me/.

[52] См. https://lnbig.com/.

[53] См. https://lightningconductor.net/channels.

вам сначала платят (входящий трафик), прежде чем вы платите другим (исходящий трафик).

Проблема создания входящей ликвидности наиболее заметна, если вы являетесь продавцом или продаете свои услуги за платежи Lightning. В этом случае вам необходимо проявлять бдительность, поддерживая достаточный объем входящей ликвидности, чтобы иметь возможность продолжать получать платежи. Что делать, если в вашем магазине наблюдается приток покупателей, но на самом деле они не могут вам заплатить, потому что больше нет входящей емкости?

В будущем эти проблемы могут быть частично смягчены за счет внедрения каналов двойного финансирования, которые финансируются с обеих сторон и обеспечивают сбалансированный входящий и исходящий трафик. Это бремя также можно было бы снизить с помощью более сложного автопилотного программного обеспечения, которое могло бы запрашивать и оплачивать входящую пропускную способность по мере необходимости.

В конечном счете пользователи Lightning должны проявлять стратегическую и активную позицию в отношении управления каналами, чтобы обеспечивать достаточную емкость для удовлетворения своих потребностей.

Закрытие каналов

Как обсуждалось ранее в книге, взаимное закрытие является предпочтительным способом закрытия канала. Однако бывают случаи, когда требуется принудительное закрытие.

Несколько примеров:

- ваш партнер по каналу находится в офлайновом режиме, и с ним нельзя связаться, чтобы инициировать взаимное закрытие;
- ваш партнер по каналу находится в онлайновом режиме, но не отвечает на запросы инициировать взаимное закрытие;
- ваш партнер по каналу находится в онлайновом режиме, и ваши узлы ведут переговоры о взаимном закрытии, но они застревают и не могут прийти к решению.

Перебалансировка каналов

В процессе совершения транзакций и маршрутизирования платежей в Lightning комбинация входящих и исходящих емкостей может стать несбалансированной.

Например, если один из ваших партнеров по каналу часто маршрутизирует платежи через ваш узел, то вы исчерпаете входящую емкость на этом канале, одновременно исчерпав исходящую емкость на исходящих каналах. Как только это произойдет, вы больше не сможете маршрутизировать платежи по этому маршруту.

Существует ряд способов перебалансировки каналов, каждый из которых имеет свои преимущества и недостатки. Один из способов – использовать подводный своп (например, петлю вывода Loop Out), как описано ранее в этой главе. Еще один способ перебалансировки – просто дождаться маршрутизированных платежей, которые идут в противоположном направлении. Если ваш узел имеет хорошую связность, когда определенный маршрут исчерпывается

в одном направлении, то тот же маршрут становится доступным в противоположном направлении. Другие узлы могут «обнаружить» этот маршрут в противоположном направлении и начать использовать его как часть своего платежного пути, тем самым снова перераспределяя средства.

Третий способ перебалансировки каналов заключается в целенаправленном создании *кругового маршрута*, который отправляет платеж с вашего узла обратно на ваш узел через сеть Lightning. Отправляя платеж по каналу с большой локальной емкостью и организуя путь таким образом, чтобы он возвращался на ваш узел по каналу с большой дистанционной емкостью, оба этих канала станут более сбалансированными. Пример стратегии перебалансировки на основе кругового маршрута можно увидеть на рис. 5-4.

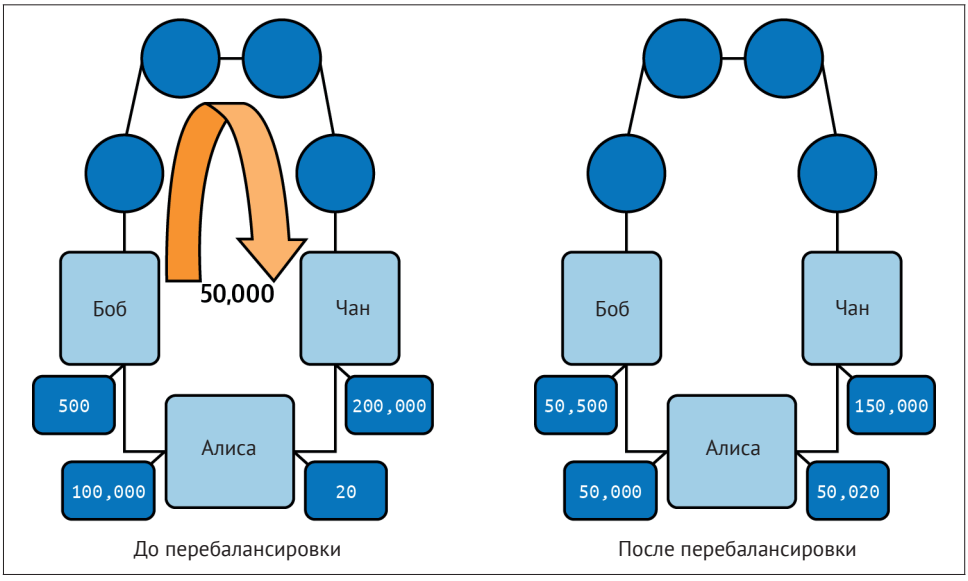


Рис. 5-4. Перебалансировка на основе кругового маршрута

Круговая перебалансировка поддерживается большинством имплементаций узлов Lightning и может выполняться в командной строке или через один из веб-интерфейсов управления, таких как Ride The Lightning (см. «Ride The Lightning» на стр. 164).

Перебалансировка каналов – это сложный вопрос, и он до сих пор является предметом активных исследований

Комиссионные за маршрутизацию

Оперирование узлом Lightning позволяет вам получать комиссионные за маршрутизацию платежей по вашим каналам. Комиссионные за маршрутизацию, как правило, не являются значительным источником дохода и незначительны по сравнению со стоимостью эксплуатации узла. Например, на относительно загруженном узле, который отправляет дюжину платежей в день, комиссионные составляют не более 2000 сатоши.

Узлы конкурируют за маршрутизационные комиссионные, устанавливая желаемый коммиссионный тариф для каждого канала. Комиссионные за маршрутизацию устанавливаются двумя параметрами для каждого канала: фиксированной базовой комиссией, которая взимается за любой платеж, и дополнительным переменным комиссионным тарифом, пропорциональным сумме платежа.

При отправке платежа Lightning узел выберет путь таким образом, чтобы минимизировать комиссионные, минимизировать переходы или и то, и другое. В результате этих взаимодействий возникает рынок комиссионных за маршрутизацию. В настоящее время существует много узлов, которые взимают очень низкие комиссионные за маршрутизацию или вообще не взимают ее, что создает понижательное давление на рынок комиссионных за маршрутизацию.

Если вы не сделаете никакого выбора, то ваш узел Lightning установит базовую комиссию по умолчанию и комиссионный тариф для каждого нового канала. Дефолтные значения зависят от используемой вами имплементации узла. Базовая комиссия устанавливается в единицах миллисатоши (тысячные доли сатоши). Пропорциональный комиссионный тариф устанавливается в миллионных долях и применяется к сумме платежа. Единица измерения в миллионных долях часто сокращается до ppm (от англ. parts per million, частей на миллион). Например, базовая комиссия в размере 1000 (миллисатоши) и комиссионный тариф в размере 1000 ppm (миллионных долей) приведут к следующим комиссионным за платеж в размере 100 000 сатоши:

$P = 100\,000$ сатоши;
$F_{база} = 1\,000$ миллисатоши = 1 сатоши;
$F_{тариф} = 1\,000$ ppm = 1 000/1 000 000 = 1/1 000 = 0.001 = 0.1 %;
$F_{итого} = F_{база} + (P * F_{тариф})$
$\rightarrow F_{итого} = 1$ сатоши + (100 000/1 000) сатоши
$\rightarrow F_{итого} = 1$ сатоши + 100 сатоши = 101 сатоши.

Вообще говоря, вы можете принять один из двух подходов к маршрутизационным комиссионным. Вы можете маршрутизировать большое число платежей с низкими комиссиями, компенсируя низкие комиссии бóльшим объемом. В качестве альтернативы вы можете выбрать более высокую комиссию. Если вы решите установить более высокие комиссионные, то ваш узел будет выбираться только в том случае, если других более дешевых маршрутов не существует. Таким образом, вы будете прокладывать маршруты реже, но будете зарабатывать больше за успешный маршрут.

Для большинства узлов обычно лучше всего использовать дефолтные значения комиссии за маршрутизацию. Благодаря этому ваш узел конкурирует в основном на равных условиях с другими узлами, которые используют дефолтные значения.

Вы также можете использовать настройки маршрутизационной комиссии для перебалансировки каналов. Если на большинстве ваших каналов установлены дефолтные комиссионные, но вы хотите перебалансировать определенный канал, то просто уменьшите комиссионные на этом конкретном канале до нуля или до очень низких тарифов. Затем сядьте поудобнее и подождите, пока кто-нибудь направит платеж по вашему «дешевому» маршруту и перебалансирует ваши каналы для вас в качестве побочного эффекта.

Управление узлом

Очевидно, что управлять вашим узлом Lightning из командной строки непросто. Это дает вам полную гибкость API узла и возможность писать свои собственные конкретно-прикладные скрипты в соответствии с вашими личными требованиями. Но если вы не хотите иметь дело со сложностью командной строки и вам нужны только некоторые базовые возможности управления узлом, то вам следует рассмотреть возможность инсталляции пользовательского веб-интерфейса, который упрощает управление узлами.

Имеется ряд конкурирующих проектов, предлагающих веб-управление узлами Lightning. Некоторые из наиболее популярных из них описаны в следующем разделе.

Ride The Lightning

Ride The Lightning (RTL – Оседлай молнию) – это графический пользовательский веб-интерфейс, помогающий пользователям управлять операциями узла Lightning трех главных имплементаций узла Lightning (LND, `c-lightning` и Eclair). RTL – это проект с открытым исходным кодом, разработанный Шаханой Фаруки (Shahana Farooqui) и многими другими участниками. Программное обеспечение RTL можно найти на GitHub[54].

На рис. 5-5 показан пример скриншота веб-интерфейса RTL, предоставленного в репозитории проекта.

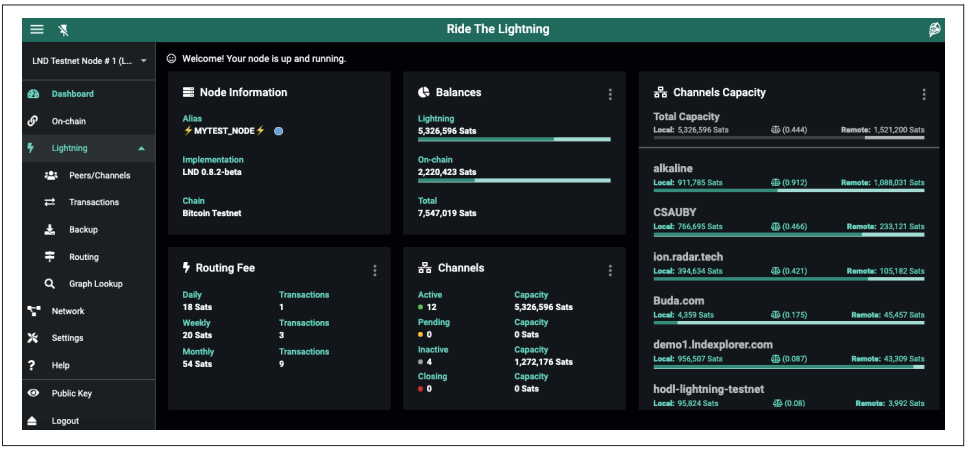


Рис. 5-5. Пример веб-интерфейса RTL

lndmon

Lightning Labs, создатели LND, предлагают графический пользовательский веб-интерфейс под названием `lndmon` для мониторинга различных показателей Lightning-узла LND. `lndmon` работает только с узлами LND. Указанный интерфейс для мониторинга работает в режиме только для чтения и как таковой

[54] См. https://github.com/Ride-The-Lightning/RTL.

не позволяет вам активно управлять узлом. Он не может открывать каналы или осуществлять платежи. lndmon находится на GitHub[55].

ThunderHub

ThunderHub[56] – это очень эстетичный графический пользовательский веб-интерфейс, аналогичный RTL, но предназначенный исключительно для LND. Его можно использовать для осуществления платежей, перебалансировки каналов и управления узлом посредством различных функциональных возможностей.

Вывод

По мере того как вы будете поддерживать свой узел и набираться опыта, вы узнаете много нового о сети Lightning. Быть оператором узла – сложная, но полезная задача. Овладение этими навыками позволит вам внести свой вклад в рост и развитие этой технологии и самой сети Lightning. Кроме того, вы получите возможность отправлять и получать платежи Lightning с максимальной степенью контроля и простоты. Вы будете играть центральную роль в инфраструктуре сети, а не просто оставаться участником на периферии.

[55] См. https://github.com/lightninglabs/lndmon.

[56] См. https://thunderhub.io/.

Часть II

Сеть Lightning в деталях

Подробное объяснение всех компонентов сети Lightning и того, как они работают. Эта часть – сугубо техническая и ожидает, что читатель имеет некоторый опыт программирования и информатики.

Глава 6

Архитектура сети Lightning

В первой части этой книги мы познакомились с главными концепциями сети Lightning и рассмотрели подробный пример маршрутизирования платежа и настройки инструментов, которые можно использовать для дальнейшего разведывательного анализа. Во второй части книги мы рассмотрим сеть Lightning гораздо более подробно с технической точки зрения, анализируя каждый из строительных блоков.

В этом разделе мы более подробно опишем компоненты сети Lightning и представим «общую картину», которая поможет вам в следующих главах.

Комплект протоколов сети Lightning

Сеть Lightning состоит из многосложного набора протоколов, которые работают поверх интернета. Указанные протоколы можно в целом разнести на пять разных слоев, составляющих стек протоколов, где каждый слой основывается на протоколах нижележащего слоя и их использует. Кроме того, каждый протокольный слой абстрагирует нижележащие слои и «скрывает» часть сложности.

Схема архитектуры, показанная на рис. 6-1, предоставляет общий вид этих слоев и их составных протоколов.

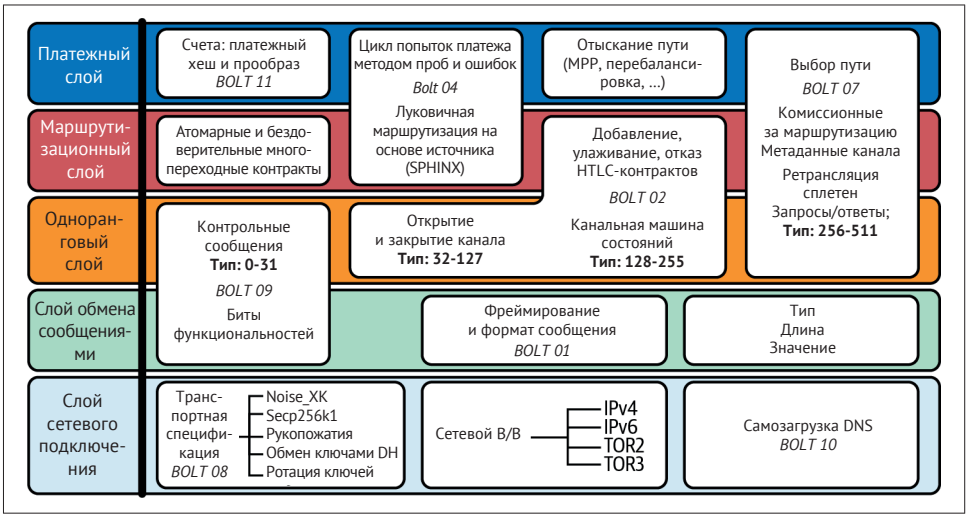


Рис. 6-1. Комплект протоколов сети Lightning

Пять слоев сети Lightning, снизу вверх, таковы:

Слой сетевого подключения

Он содержит протоколы, которые взаимодействуют напрямую со стержневыми протоколами интернета (TCP/IP), протоколами наложения (Tor v2/v3) и интернет-службами (DNS). Этот слой также содержит криптографические транспортные протоколы, которые защищают сообщения Lightning.

Слой обмена сообщениями

Этот слой содержит протоколы, которые узлы используют для согласования функциональных возможностей, форматирования сообщений и кодирования полей сообщений.

Одноранговый слой (P2P)

Этот слой является первичным протокольным слоем для связи между узлами Lightning и содержит все те разные сообщения, которыми узлы обмениваются между собой.

Маршрутизационный слой

Этот слой содержит протоколы, используемые для маршрутизирования платежей между узлами, сквозных и атомарных. Он содержит стержневую функциональность сети Lightning: маршрутизируемые платежи.

Платежный слой

Самый высокий слой сети, который представляет надежный платежный интерфейс для приложений.

Lightning в деталях

В следующих 10 главах мы детально проанализируем комплект протоколов и подробно рассмотрим каждый компонент сети Lightning.

Мы потратили довольно много времени, пытаясь определить наилучший порядок представления этой детали. Это непростой выбор, потому что между разными компонентами существует столь много взаимозависимостей: когда вы начинаете объяснять один из них, вы обнаруживаете, что он включает в себя довольно много других компонентов. Вместо подхода «сверху вниз» или «снизу вверх» мы в итоге выбрали более извилистый путь, который начинается с самых основополагающих строительных блоков, уникальных для сети Lightning, – платежных каналов – и оттуда продвигается дальше. Но поскольку этот путь неочевиден, мы будем использовать комплект протоколов Lightning, показанный на рис. 6-1, в качестве карты. В каждой главе будет уделено внимание одному или нескольким связанным компонентам, и вы увидите, что они выделены в рамках комплекта протоколов. Что-то вроде маркера на карте, говорящего: «ты – здесь!»

Вот что мы рассмотрим:

Глава 7 «Платежные каналы»

В этой главе мы разберем, как работают платежные каналы, значительно более подробно, чем мы видели в предыдущих частях книги. Мы рассмотрим структуру и Bitcoin Script финансовых и фиксационных транзакций, а также процесс, используемый узлами для согласования каждого шага протокола.

Глава 8 «Маршрутизирование в сети платежных каналов»

Далее мы объединим несколько платежных каналов в сеть и направим платеж с одного конца на другой. В этом процессе мы погрузимся в умный контракт с привязкой к хешу и времени (HTLC) и Bitcoin Script, который используем для его сборки.

Глава 9 «Работа канала и пересылка платежей»

Объединив концепции простого платежного канала и маршрутизируемого платежа с использованием HTLC-контрактов, мы теперь рассмотрим, как HTLC-контракты являются частью транзакции каждого канала. Мы также разберем протокол для добавления, урегулирования, отказа и удаления HTLC-контрактов из фиксаций.

Глава 10 «Луковичная маршрутизация»

Далее мы рассмотрим, как информация HTLC-контракта распространяется по сети внутри протокола луковичной маршрутизации. Мы разберем механизм многослойного шифрования и дешифрования, который придает сети Lightning некоторые характеристики конфиденциальности.

Глава 11 «Сплетни и граф каналов»

В этой главе мы рассмотрим, как узлы Lightning находят друг друга, и узнаем об опубликованных каналах, чтобы построить граф каналов, который они могут использовать для отыскания путей в сети.

Глава 12 «Отыскание пути и доставка платежей»

Далее мы увидим, как информация из эпидемического протокола используется каждым узлом для построения «карты» всей сети, которую он может использовать для отыскания пути из одной точки в другую для маршрутизирования платежей. Мы также рассмотрим свежие инновации в отыскании пути, такие как многокомпонентные платежи.

Глава 13 «Проводной протокол: фреймирование и расширяемость»

В основе сети Lightning лежит одноранговый протокол, который узлы используют для обмена сообщениями о сети и о своих каналах. В этой главе мы разберем, как эти сообщения конструируются и возможности расширения, встроенные в сообщения с использованием битов функциональностей и кодировки тип–длина–значение (TLV).

Глава 14 «Транспортировка зашифрованных сообщений Lightning»

Переходя к более низкоуровневой части сети, мы рассмотрим лежащую в основе зашифрованную транспортную систему, которая обеспечивает секретность и целостность всех коммуникаций между узлами.

Глава 15 «Платежные запросы Lightning»

Ключевой частью сети Lightning являются платежные запросы, также именуемые счетами Lightning. В этой главе мы познакомимся со структурой и кодировкой счета.

Давайте нырнем внутрь!

Глава 7

Платежные каналы

В этой главе мы углубимся в платежные каналы и посмотрим, как они устроены. Мы начнем с узла Алисы, открывающего канал к узлу Боба, основываясь на примерах, представленных в начале нашей книги.

Сообщения, которыми обмениваются узлы Алисы и Боба, определены в спецификации «BOLT #2: одноранговый протокол для управления каналами»[57]. Создаваемые узлами Алисы и Боба транзакции определены в спецификации «BOLT #3: форматы транзакций и скриптов»[58]. В этой главе мы сосредоточимся на частях «Открытие и закрытие каналов» и «Канальная машина состояний» архитектуры протоколов сети Lightning, выделенных контуром в центре (одноранговый слой) на рис. 7-1.

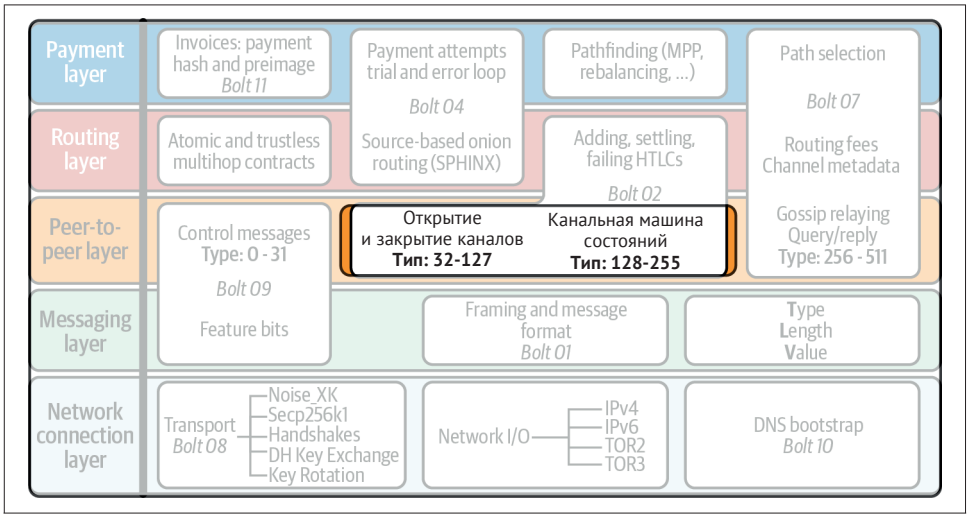


Рис. 7-1. Платежные каналы в рамках комплекта протоколов Lightning

[57] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md.

[58] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md.

Другой способ использования системы Bitcoin

Сеть Lightning часто описывается как «второслойный протокол Bitcoin», что отличает ее от собственно системы Bitcoin. Еще один вариант описания сети Lightning таков: это «более разумный способ использования системы Bitcoin» или просто «приложение поверх системы Bitcoin». Давайте разведаем этот вопрос.

Исторически сложилось так, что Bitcoin-транзакции передаются широковещательно всем и регистрируются в блочной цепи Bitcoin, чтобы считаться валидными. Однако, как мы увидим, если у кого-то есть подписанная Bitcoin-транзакция, которая тратит мультиподписной выход «2 из 2», давая ему исключительную возможность тратить этот биткойн, он фактически владеет этим биткойном, даже если и не выполняет широковещательную передачу данной транзакции.

О подписанной Bitcoin-транзакции можно думать как о чеке с отсрочкой платежа (или чеке), который можно обналичить в любое время. Однако, в отличие от традиционной банковской системы, эта транзакция не является «обещанием» платежа (также именуемым долговой распиской), а поддающимся проверке документом на предъявителя, который эквивалентен наличным деньгам. До тех пор, пока упомянутый в транзакции биткойн еще не был потрачен во время погашения (или в момент, когда вы пытаетесь «обналичить» чек), система Bitcoin гарантирует, что эта подписанная заранее транзакция может быть передана в широковещательном режиме и зарегистрирована в любое время. Разумеется, это верно только в том случае, если данная транзакция является единственной подписанной пользователем. В сети Lightning одновременно существуют две или более таких предварительно подписанных транзакций; поэтому нам нужен более сложный механизм, чтобы по-прежнему иметь функциональность такого проверяемого инструмента на предъявителя, о чем вы также узнаете в этой главе.

Сеть Lightning – это просто другой и творческий способ использования системы Bitcoin. В сети Lightning комбинация зарегистрированных (внутри цепи) и подписанных, но удерживаемых (вне цепи) транзакций формирует «слой» платежей, который является более быстрым, дешевым и приватным способом использования системы Bitcoin. Эту взаимосвязь между внутрицепными и внецепными Bitcoin-транзакциями можно увидеть на рис. 7-2.

Lightning – это Bitcoin. Это просто другой способ использования системы Bitcoin.

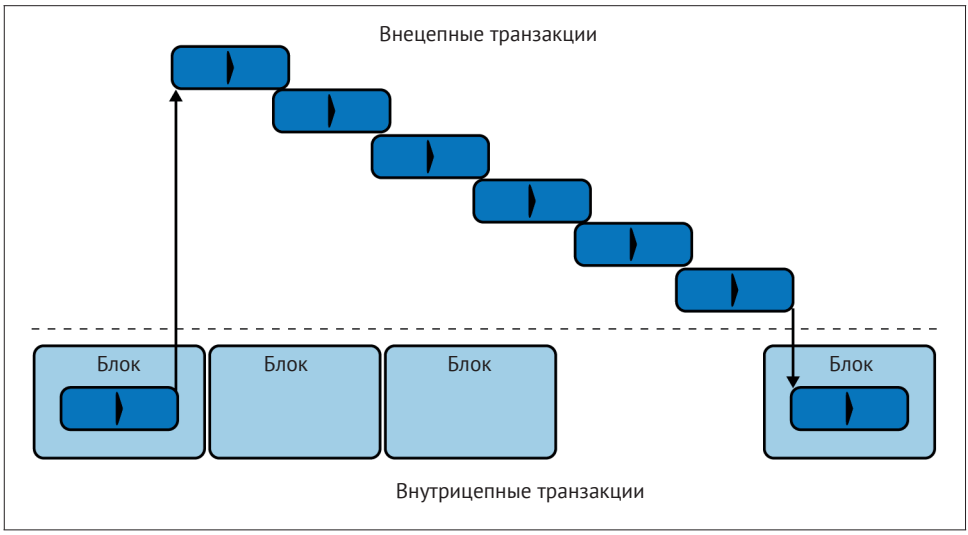


Рис. 7-2. Платежный канал Lightning, состоящий из внутрицепных и внецепных транзакций

Владение биткойном и контроль над ним

Прежде чем разобраться в платежных каналах, мы должны сделать небольшой шаг назад и понять, как владение и контроль работают в системе Bitcoin.

Когда кто-то говорит, что он «владеет» биткойном, он обычно имеет в виду, что знает приватный ключ Bitcoin-адреса, который имеет некоторые неизрасходованные транзакционные выходы (см. приложение А). Приватный ключ позволяет ему подписать транзакцию, чтобы потратить этот биткойн, переведя его на другой адрес. В системе Bitcoin «владение» биткойном можно определить как *способность тратить* этот биткойн.

Имейте в виду, что термин «владение», используемый в системе Bitcoin, отличается от термина «владение», используемого в юридическом смысле. Вор, у которого есть секретные ключи и который может потратить биткойн, является фактическим владельцем этого биткойна, даже если он не является законным владельцем.

Владение биткойнами – это только контроль над ключами и возможность тратить биткойны с помощью этих ключей. Как гласит популярная биткойновая поговорка: «Ваши ключи, ваши монеты – не ваши ключи, не ваши монеты».

Разнообразие форм (независимого) владения и мультиподпись

Владение приватными ключами и контроль над ними не всегда находятся в руках одного человека. Вот тут-то все и становится интересным и сложным. Мы знаем, что несколько человек могут узнать один и тот же приватный ключ либо в результате кражи, либо потому, что первоначальный владелец ключа делает копию и передает ее кому-то другому. Являются ли все эти люди владельцами? В практическом смысле так оно и есть, потому что любой из людей, знающих приватный ключ, может потратить биткойн без одобрения кого-либо другого.

Система Bitcoin также имеет мультиподписные адреса, где перед расходованием необходимо подписать несколько приватных ключей (см. «Мультиподписные скрипты» на стр. 392). С практической точки зрения владение мультиподписным адресом зависит от кворума (K) и общего числа (N), определенных в схеме «K из N». Мультиподписная схема «1 из 10» позволяет любому 1 (K) из 10 (N) подписантов потратить сумму биткойнов, привязанную к этому адресу. Это похоже на сценарий, когда у 10 человек есть копия одного и того же приватного ключа и любой из них может независимо его потратить.

Совместное владение без независимого контроля

Существует также сценарий, когда ни у кого нет кворума. В схеме «2 из 2», подобной той, что используется в сети Lightning, ни один подписант не может потратить биткойн без получения подписи от другой стороны. Кому в таком случае принадлежит биткойн? Никто на самом деле не владеет, потому что никто не контролирует. Каждый из них владеет долей, эквивалентной доле голоса при принятии решения, но необходимы оба голоса. Ключевая проблема (преднамеренный каламбур) со схемой «2 из 2» как в системе Bitcoin, так и в законодательстве заключается в том, что происходит, если одна из сторон недоступна или если голосование зашло в тупик и любая из сторон отказывается сотрудничать.

Предотвращение «привязанности» и нерасходуемости биткойна

Если один из двух подписантов мультиподписи «2 из 2» не может или не хочет подписывать, то средства становятся нерасходуемыми. Этот сценарий может произойти не только случайно (потеря ключей), но и может быть использован как форма шантажа любой из сторон: «Я не подпишу, пока вы не заплатите мне часть средств».

Платежные каналы в Lightning основаны на мультиподписном адресе «2 из 2», при этом два партнера по каналу являются подписантами в мультиподписи. В настоящее время каналы финансируются только одним из двух партнеров канала: когда вы решаете «открыть» канал, вы вносите средства на мультиподписной адрес «2 из 2» с помощью транзакции. После того как эта транзакция будет добыта и средства будут находиться в мультиподписи, вы не сможете вернуть их без сотрудничества от вашего партнера по каналу, потому что вам нужна его подпись, (также) чтобы потратить биткойн.

В следующем далее разделе, когда мы рассмотрим, как открывать (создавать) канал Lightning, мы увидим, как можно предотвращать потерю средств или любой сценарий шантажа между двумя партнерами путем внедрения протокола справедливости с целью строительства канала с помощью предварительно подписанных транзакций, которые тратят мультиподписные выходы таким образом, который дает одноранговым узлам в канале исключительную возможность тратить один из выходов, кодирующих сумму биткойнов, которыми они владеют в канале.

Строительство платежного канала

В разделе «Что такое платежный канал?» на стр. 66 мы описали платежные каналы как финансовую взаимосвязь между двумя узлами Lightning, которые устанавливаются путем финансирования мультиподписного адреса «2 из 2» двумя партнерами по каналу.

Давайте допустим, что Алиса хочет создать платежный канал, позволяющий ей подключаться к магазину Боба напрямую. Прежде всего два узла (Алиса и Боб) должны установить интернет-соединение друг с другом, чтобы они могли договориться о платежном канале.

Приватный и публичный ключи узла

Каждый узел в сети Lightning идентифицируется публичным ключом узла. Публичный ключ однозначно идентифицирует конкретный узел и обычно представляется в виде шестнадцатеричной кодировки. Например, Рене Пикхардт в настоящее время управляет узлом Lightning (`ln.rene-pickhardt.de`), который идентифицируется следующим ниже публичным ключом узла:

```
02a1cebfacb2674143b5ad0df3c22c609e935f7bc0ebe801f37b8e9023d45ea7b8
```

Каждый узел генерирует корневой приватный ключ при первоначальной инициализации. Приватный ключ всегда остается приватным (никогда не передается) и надежно хранится в кошельке узла. Из этого приватного ключа узел извлекает публичный ключ, который является идентификатором узла и предоставляется сети в коллективное пользование. Поскольку пространство ключей огромно, до тех пор, пока каждый узел генерирует приватный ключ случайным образом, он будет иметь уникальный публичный ключ, который, следовательно, может однозначно идентифицировать его в сети.

Сетевой адрес узла

Вдобавок каждый узел также рекламирует сетевой адрес, по которому к нему можно подсоединиться, в одном из нескольких возможных форматов:

TCP/IP

Адрес IPv4 или IPv6 и номер TCP-порта.

TCP/Tor

«Луковичный» адрес Tor и номер TCP-порта.

Идентификатор сетевого адреса записывается как Адрес:Порт, что соответствует международным стандартам сетевых идентификаторов, используемым, например, в интернете.

Например, узел Рене с узловым публичным ключом 02a1ceb...45ea7b8 в настоящее время рекламирует свой сетевой адрес как TCP/IP-адрес:

```
172.16.235.20:9735
```

Дефолтным TCP-портом сети Lightning является 9735, но узел может выбрать прослушивание любого TCP-порта.

Идентификаторы узлов

Вместе публичный ключ узла и сетевой адрес записываются в следующем ниже формате, разделенные знаком `@`:

ИдентификаторУзла@Адрес:Порт.

Таким образом, полный идентификатор узла Рене будет:

```
02a1cebfacb2674143b5ad0df3c22c609e935f7bc0ebe801f37b8e9023d45ea7b8
@172.16.235.20:9735
```

Псевдонимом узла Рене является `ln.rene-pickhardt.de`; однако это имя существует только для лучшей читаемости. Каждый оператор узла может объявить любой псевдоним, который он захочет, и нет никакого механизма, который мешал бы операторам узлов выбирать псевдоним, который уже используется. Таким образом, для ссылки на узел необходимо использовать схему `ИдентификаторУзла@Адрес:Порт`.

Приведенный выше идентификатор часто кодируется в QR-коде, что облегчает пользователям сканирование, если они хотят подсоединить свой собственный узел к конкретному узлу, определяемому этим адресом.

Подобно узлам Bitcoin, узлы Lightning рекламируют свое присутствие в сети Lightning, «сплетничая» о своем узловом публичном ключе и сетевом адресе. Таким образом, другие узлы могут их находить и вести учет (базу данных) всех известных узлов, к которым они могут подключаться, и обмениваться сообщениями, определенными в протоколе P2P-сообщений сети Lightning.

Соединение узлов в качестве прямых одноранговых участников сети

Для того чтобы узел Алисы мог подсоединиться к узлу Боба, ей понадобится публичный ключ узла Боба или полный адрес, содержащий публичный ключ, IP-адрес или Tor-адрес и порт. Поскольку Боб управляет магазином, адрес узла Боба можно получить из счета или платежной страницы магазина в интернете. Алиса может отсканировать содержащий адрес QR-код и дать указание своему узлу подключиться к узлу Боба.

После того как Алиса подсоединилась к узлу Боба, их узлы теперь являются напрямую подсоединенными одноранговыми узлами.

Для того чтобы открыть платежный канал, два узла сначала должны быть подсоединены как прямые одноранговые узлы, открыв соединение через интернет (или Tor).

Строительство канала

Теперь, когда узлы Lightning Алисы и Боба подсоединены, они могут начать процесс строительства платежного канала. В этом разделе мы рассмотрим взаимодействие между их узлами, именуемое одноранговым протоколом Lightning по управлению каналами, и криптографическим протоколом, который они используют для сборки Bitcoin-транзакций.

В этом сценарии мы описываем два разных протокола. Во-первых, существует протокол сообщений, который устанавливает то, как узлы Lightning взаимодействуют через интернет и какими сообщениями они обмениваются друг с другом. Во-вторых, существует криптографический протокол, который определяет то, как два узла строят и подписывают Bitcoin-транзакции.

Одноранговый протокол для управления каналами

Одноранговый протокол Lightning для управления каналами определен в спецификации «BOLT #2: одноранговый протокол для управления каналами»[59]. В этой главе мы рассмотрим разделы «Установление канала» и «Закрытие канала» спецификации BOLT #2 более подробно.

Поток сообщений об установлении канала

Установление канала достигается путем обмена шестью сообщениями между узлами Алисы и Боба (по три от каждого узла): `open_channel` (открыть канал), `accept_channel` (принять канал), `funding_created` (финансирование создано), `funding_signed` (финансирование подписано), `funding_locked` (финансирование привязано) и `funding_locked`. Указанные шесть сообщений показаны в виде диаграммы временной последовательности на рис. 7-3.

На рис. 7-3 узлы Алисы и Боба представлены вертикальными линиями «A» и «B» по обе стороны диаграммы. Подобного рода диаграмма временной последовательности показывает, как время течет вниз, а сообщения передаются с одной стороны на другую между двумя одноранговыми узлами связи. Строки наклонены вниз, чтобы отобразить время, необходимое для передачи каждого сообщения, а направление сообщения показано стрелкой в конце каждой строки.

[59] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md.

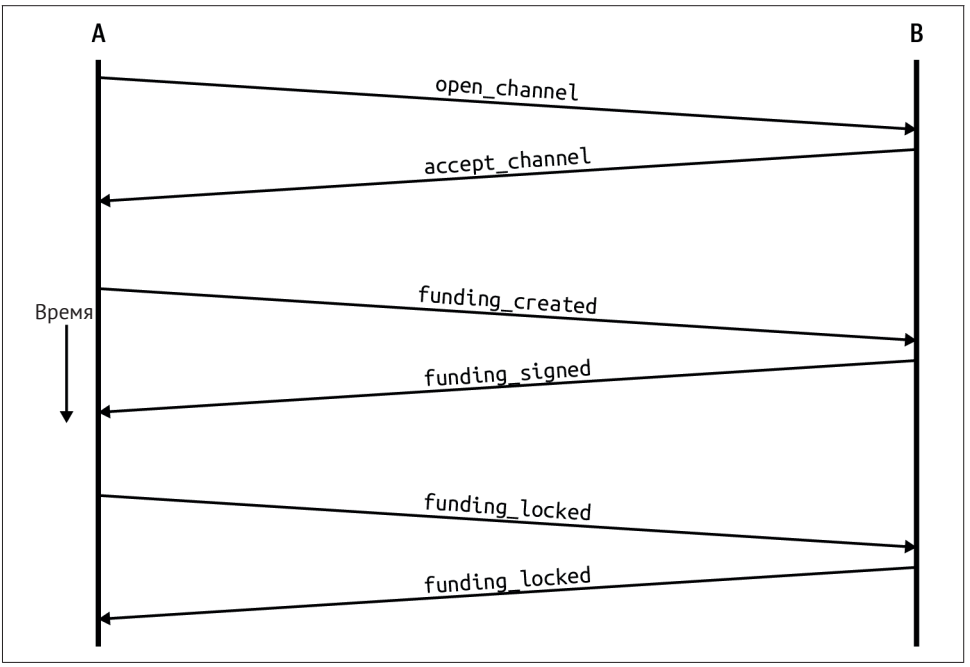


Рис. 7-3. Поток сообщений об установлении канала

Установление канала состоит из трех частей. Во-первых, два одноранговых узла сообщают о своих возможностях и ожиданиях, при этом Алиса инициирует запрос посредством `open_channel`, а Боб принимает запрос канала посредством `accept_channel`.

Во-вторых, Алиса строит финансовую и возвратную транзакции (как мы увидим позже в этом разделе) и отправляет `funding_created` Бобу. Другое название «возвратной» транзакции – «фиксационная» транзакция, поскольку она привязана к текущему распределению остатков в канале. Боб отвечает, отправляя обратно необходимые подписи посредством `funding_signed`. Это взаимодействие является основой криптографического протокола для защиты канала и предотвращения кражи. Теперь Алиса выполнит широковещательную передачу финансовой транзакции (внутри цепи), чтобы установить и закрепить платежный канал. Транзакцию необходимо будет подтвердить в блочной цепи Bitcoin.

Название сообщения `funding_signed` может немного дезориентировать. Это сообщение не содержит подписи под финансовой транзакцией, а наоборот – содержит подпись Боба под возвратной транзакцией, которая позволяет Алисе вернуть свой биткойн из мультиподписи.

После того как транзакция получит достаточное число подтверждений (как определено полем `minimum_depth` в сообщении `accept_channel`), Алиса и Боб обмениваются сообщениями `funding_locked`, и канал переходит в обычный режим работы.

Сообщение open_channel

Узел Алисы запрашивает платежный канал у узла Боба, отправляя сообщение `open_channel`. Указанное сообщение содержит информацию об ожиданиях Алисы относительно настройки канала, которую Боб может принять либо отклонить.

Структура сообщения open_channel (взятого из спецификации BOLT #2) показана в примере 7-1.

Пример 7-1. Сообщение open_channel

```
[chain_hash:chain_hash]
[32*byte:temporary_channel_id]
[u64:funding_satoshis]
[u64:push_msat]
[u64:dust_limit_satoshis]
[u64:max_htlc_value_in_flight_msat]
[u64:channel_reserve_satoshis]
[u64:htlc_minimum_msat]
[u32:feerate_per_kw]
[u16:to_self_delay]
[u16:max_accepted_htlcs]
[point:funding_pubkey]
[point:revocation_basepoint]
[point:payment_basepoint]
[point:delayed_payment_basepoint]
[point:htlc_basepoint]
[point:first_per_commitment_point]
[byte:channel_flags]
[open_channel_tlvs:tlvs]
```

Содержащиеся в этом сообщении поля определяют параметры канала, которые требуются Алисе, а также различные конфигурации настройки из узла Алисы, которые отражают ожидания безопасности для работы канала.

Некоторые параметры строительства канала перечислены ниже:

`chain_hash`

Определяет, какая блочная цепь (например, главная сеть mainnet системы Bitcoin) будет использоваться для этого канала. Обычно это хеш первичного блока этой блочной цепи.

`funding_satoshis`

Сумма, которую Алиса будет использовать для финансирования канала, то есть общая емкость канала.

`channel_reserve_satoshis`

Минимальный остаток в сатоши, зарезервированный на каждой стороне канала. Мы вернемся к этому, когда будем говорить о штрафах.

`push_msat`

Опциональная сумма, которую Алиса немедленно «закинет» Бобу в качестве оплаты при финансировании канала. *Установка этого значения равным чему угодно, кроме 0, практически означает пожертвование денег вашему партнеру по каналу, и его следует использовать с осторожностью*.

`to_self_delay`

Очень важный для протокола параметр безопасности. Значение в сообщении `open_channel` используется в фиксационной транзакции ответчика, а `accept_channel` – в транзакции инициатора. Эта асимметрия существует для того, чтобы каждая сторона могла указать, как долго другой стороне необходимо ждать, чтобы в одностороннем порядке истребовать средства в рамках фиксационной транзакции. Если Боб в любое время в одностороннем порядке закроет канал против воли Алисы, то он обязуется не получать доступ к своим собственным средствам в течение определенной здесь задержки. Чем выше это значение, тем большей безопасностью обладает Алиса, но тем дольше Боб может держать свои средства привязанными.

`funding_pubkey`

Публичный ключ, который Алиса внесет в мультиподпись «2 из 2», который заякоревает этот канал.

`X_basepoint`

Главные ключи, используемые для выведения дочерних ключей для различных частей фиксационной, отзывной, маршрутизационно-платежной (HTLC-контракты) и закрывающей транзакций. Они будут использованы и объяснены в последующих главах.

Если вы хотите разобраться в других полях и сообщениях одноранговой протокола Lightning, которые мы не обсуждаем в этой книге, мы предлагаем вам ознакомиться с ними в спецификациях BOLT. Эти сообщения и поля важны, но их невозможно рассмотреть довольно подробно в рамках нашей книги. Мы хотим, чтобы вы достаточно хорошо поняли основополагающие принципы, чтобы вы могли заполнить все пробелы, прочитав фактическую спецификацию протокола (спецификаций BOLT).

Сообщение accept_channel

В ответ на сообщение `open_channel` от Алисы Боб отправляет обратно сообщение `accept_channel`, показанное в примере 7-2.

Пример 7-2. Сообщение accept_channel

```
[32*byte:temporary_channel_id]
[u64:dust_limit_satoshis]
[u64:max_htlc_value_in_flight_msat]
[u64:channel_reserve_satoshis]
[u64:htlc_minimum_msat]
[u32:minimum_depth]
[u16:to_self_delay]
[u16:max_accepted_htlcs]
[point:funding_pubkey]
[point:revocation_basepoint]
[point:payment_basepoint]
[point:delayed_payment_basepoint]
[point:htlc_basepoint]
[point:first_per_commitment_point]
[accept_channel_tlvs:tlvs]
```

Как вы видите, оно похоже на сообщение `open_channel` и содержит ожидания узла Боба и конфигурационные значения.

Два наиболее важных поля в `accept_channel`, которые Алиса будет использовать для строительства платежного канала, таковы:

`funding_pubkey`

Узел Боба с публичным ключом вносит свой вклад в мультиподписной адрес «2 из 2», который заякоревает канал.

`minimum_depth`

Число подтверждений, которые узел Боба ожидает получить для финансовой транзакции, прежде чем он сочтет канал «открытым» и готовым к использованию.

Финансовая транзакция

После того как узел Алисы получает сообщение `accept_channel` от Боба, у него есть информация, необходимая для сборки *финансовой транзакции*, которая заякоревает канал с блочной цепью Bitcoin. Как мы обсуждали в предыдущих главах, платежный канал Lightning поддерживается мультиподписным адресом «2 из 2». Прежде всего необходимо сгенерировать этот мультиподписной адрес, чтобы иметь возможность собрать финансовую транзакцию (и возвратную транзакцию, как описано ниже).

Генерирование мультиподписного адреса

Финансовая транзакция отправляет некоторую сумму биткойна (`funding_satoshis` из сообщения `open_channel`) на мультиподписной выход «2 из 2», который строится из публичных ключей `funding_pubkey` Алисы и Боба.

Узел Алисы строит мультиподписной скрипт, как показано ниже:

```
2 <Alice_funding_pubkey> <Bob_funding_pubkey> 2 CHECKMULTISIG
```

Обратите внимание, что на практике финансовые ключи сортируются детерминированно (с использованием лексикографического порядка сериализованной сжатой формы публичных ключей) перед размещением в свидетельском скрипте. Соглашаясь с этим отсортированным порядком заранее, мы обеспечиваем, чтобы обе стороны строили идентичный выход финансовой транзакции, который подписывается подписью обменянной фиксационной транзакции.

Этот скрипт кодируется как Bitcoin-адрес Pay-to-Witness-Script-Hash (P2WSH, оплата по хешу свидетельского скрипта), который выглядит примерно так:

```
bc1q89ju02heg32yrqdrnqghe6132wek25p6sv6e564znvrvez7tq5zqt4dn02
```

Сборка финансовой транзакции

Узел Алисы теперь может собрать финансовую транзакцию, отправив согласованную с Бобом сумму (`funding_satoshis`) на мультиподписной адрес «2 из 2». Давайте допустим, что сумма `funding_satoshis` составила 140 000, а Алиса тратит 200 000 сатоши и создает сдачу в размере 60 000 сатоши. Транзакция будет выглядеть примерно так, как показано на рис. 7-4.

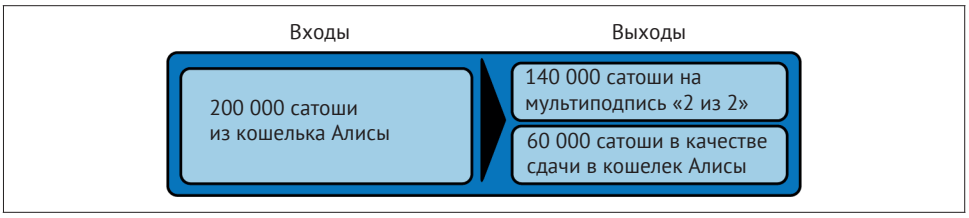


Рис. 7-4. Алиса строит финансовую транзакцию

Алиса *не выполняет широковещательную передачу* этой транзакции, потому что это поставило бы под угрозу ее 140 000 сатоши. После того как Алиса потратила деньги на мультиподпись «2 из 2», у нее нет возможности вернуть свои деньги без подписи Боба.

> **Платежные каналы с двойным финансированием**
>
> В текущей имплементации Lightning каналы финансируются только тем узлом, который инициирует канал (Алисой в нашем примере). Каналы с двойным финансированием были предложены, но еще не имплементированы. В канале двойного финансирования и Алиса, и Боб будут вносить свой вклад в финансовую транзакцию. Каналы с двойным финансированием требуют немного более сложного потока сообщений и криптографического протокола, поэтому они еще не имплементированы, но запланированы для будущего обновления спецификаций BOLT сети Lightning. Имплементация c-lightning содержит экспериментальную версию варианта каналов с двойным финансированием.

Удерживание подписанных транзакций без широковещательной передачи

Важной функциональной особенностью системы Bitcoin, которая делает возможным сеть Lightning, является возможность строить и подписывать транзакции, но не передавать их широковещательно. Транзакция валидна во всех отношениях, но до тех пор, пока она не будет передана широковещательно и подтверждена в блочной цепи Bitcoin, она не распознается, и ее выходы не подлежат расходованию, поскольку они не были созданы в блочной цепи. В сети Lightning мы будем использовать эту возможность много раз, и узел Алисы использует эту возможность при сборке финансовой транзакции: удерживает ее и пока не выполняет ее широковещательную передачу.

Возврат средств до финансирования

Во избежание потери средств Алиса не может поместить свой биткойн в «2 из 2», пока у нее не будет способа получить его назад, если что-то пойдет не так. По сути, она должна спланировать свой «выход» из канала, прежде чем она вступит в эту договоренность.

Рассмотрим юридическую конструкцию брачного соглашения, т. н. «брачного контракта». Когда два человека вступают в брак, их деньги связаны законом

(в зависимости от юрисдикции). До вступления в брак они могут подписать соглашение, в котором указывается, как делить свои активы, если они расторгнут свой брак путем развода.

Мы можем создать аналогичное соглашение в системе Bitcoin. Например, мы можем создать возвратную транзакцию, которая функционирует как брачный контракт, позволяя сторонам решать, как будут делиться средства в их канале, до того, как их средства будут фактически привязаны к мультиподписному адресу финансирования.

Сборка предварительно подписанной возвратной транзакции

Алиса соберет возвратную транзакцию сразу после сборки (но не широковещательной передачи) финансовой транзакции. Возвратная транзакция расходует мультиподпись «2 из 2» обратно в кошелек Алисы. Мы называем эту возвратную транзакцию *фиксационной транзакцией*, т. к. она фиксирует согласие обоих партнеров по каналу распределять остаток канала справедливо. Поскольку Алиса сама финансировала канал, она получает весь остаток, и Алиса, и Боб обязуются возместить Алисе эту транзакцию.

Как мы увидим в последующих главах, на практике это немного сложнее, но пока давайте не будем усложнять и допустим, что это выглядит как на рис. 7-5.

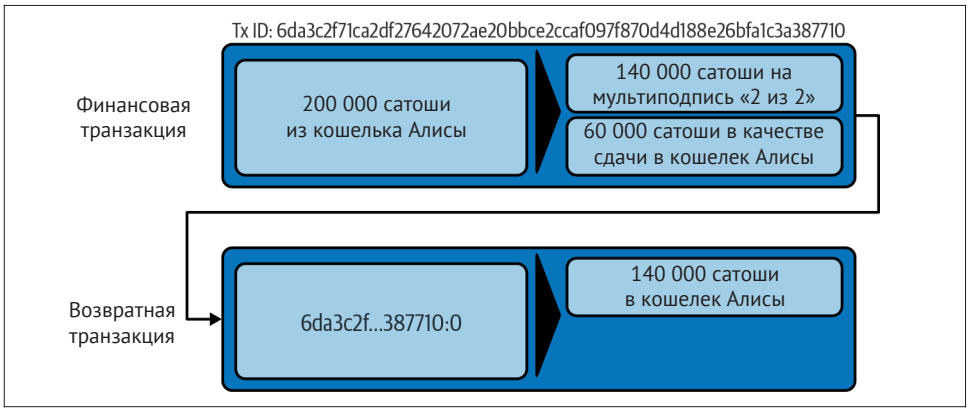


Рис. 7-5. Алиса также строит возвратную транзакцию

Позже в этой главе мы увидим, как совершать больше фиксационных транзакций с целью распределения остатка канала в разных суммах.

Выстраивание транзакций в цепь без широковещательной передачи

Итак, теперь Алиса собрала две транзакции, показанные на рис. 7-5. Но вам может быть интересно, как это возможно. Алиса не выполняла широковещательную передачу финансовой транзакции в блочной цепи Bitcoin. Что касается всех участников сети, то этой транзакции не существует. Возвратная транзакция собрана таким образом, чтобы потратить один из выходов финансовой

транзакции, даже если этот выход еще не существует. И как же можно потратить выход, который не был подтвержден в блочной цепи Bitcoin?

Возвратная транзакция еще не является валидной транзакцией. Для того чтобы она стала валидной транзакцией, должны произойти два события:

- финансовая транзакция должна быть передана широковещательно в сеть биткойнов (В целях обеспечения безопасности сети Lightning мы также потребуем, чтобы она была подтверждена блочной цепью Bitcoin, хотя это не является строго необходимым для выстраивания транзакций в цепь.);
- вход в возвратную транзакцию требует подписей Алисы и Боба.

Но даже несмотря на то, что эти два события не произошли, и даже несмотря на то, что узел Алисы не выполнил широковещательную передачу финансовой транзакции, она все равно может собрать возвратную транзакцию. Она может сделать это, потому что может вычислить хеш финансовой транзакции и ссылаться на него в качестве входа в возвратную транзакцию.

Обратите внимание, как Алиса вычислила `6da3c2...387710` в качестве хеша финансовой транзакции. Если и когда финансовая транзакция будет транслироваться широковещательно, этот хеш будет зарегистрирован как ИД финансовой транзакции. Следовательно, выход 0 финансовой транзакции (выход в виде адреса «2 из 2») будет затем указан как ИД выхода `6da3c2... 387710:0`. Возвратная транзакция может быть собрана таким образом, чтобы потратить выход этой финансовой транзакции, даже если она еще не существует, потому что Алиса знает, каким будет ее идентификатор после подтверждения.

Это означает, что Алиса может создать цепную транзакцию, сославшись на выход, который еще не существуют, зная, что ссылка будет валидной, если финансовая транзакция будет подтверждена, что также сделает возвратную транзакцию валидной. Как мы увидим в следующем разделе, этот «трюк» с выстраиванием транзакций в цепь до их широковещательной передачи в сеть требует очень важной функциональности системы Bitcoin, которая была введена в августе 2017 года: функциональность сегрегированного свидетеля.

Решение проблемы деформируемости (сегрегированный свидетель)

Алиса должна была зависеть от известности ИД финансовой транзакции до подтверждения. Но до введения сегрегированного свидетеля (Segregated Witness, аббр. SegWit) в августе 2017 года этого было недостаточно для защиты Алисы. Из-за способа сборки транзакций с подписями (свидетелями), включаемыми в ИД транзакции, третья сторона (например, Боб) могла выполнить широковещательную передачу альтернативной версии транзакции с *деформированным* (модифицированным) ИД транзакции. Эта проблема известна как *деформируемость* транзакций, и до появления SegWit она затрудняла безопасную имплементацию неограниченных пожизненных платежных каналов.

Если бы Боб мог изменить финансовую транзакцию Алисы до того, как она была подтверждена, и произвести реплику с другим ИД транзакции, то Боб мог бы сделать возвратную транзакцию Алисы невалидной и похитить ее биткойн. Алиса была бы во власти Боба в получении подписи для освобождения своих

средств, и ее легко можно было бы шантажировать. Боб не мог украсть средства, но он мог бы помешать Алисе вернуть их обратно.

Внедрение SegWit сделало неподтвержденные ИД транзакций немутируемыми с точки зрения третьих сторон, а это означает, что Алиса может быть уверена в том, что ИД финансовой транзакции не изменится. Как следствие Алиса может быть уверена в том, что если она получит подпись Боба на возвратной транзакции, то у нее будет способ вернуть свои деньги. Теперь у нее есть способ имплементировать Bitcoin-эквивалент «брачного контракта», прежде чем привязывать свои средства в мультиподписи.

Возможно, вы задались вопросом, как Боб может изменить (деформировать) транзакцию, созданную и подписанную Алисой. У Боба, конечно же, нет приватных ключей Алисы. Однако подписи ECDSA под сообщением не являются уникальными. Знание подписи (которая включена в валидную транзакцию) позволяет генерировать много подписей разного вида, которые по-прежнему валидны. До того, как SegWit удалил подписи из алгоритма дайджеста транзакций, Боб мог заменить подпись эквивалентной валидной подписью, которая создавала другой ИД транзакции, разрывая цепь между финансовой и возвратной транзакциями.

Сообщение funding_created

Теперь, когда Алиса построила необходимые транзакции, поток сообщений о строительстве канала продолжается. Алиса передает Бобу сообщение `funding_created`. Вы можете ознакомиться с содержанием этого сообщения ниже:

```
[32*byte:temporary_channel_id]
[sha256:funding_txid]
[u16:funding_output_index]
[signature:signature]
```

С помощью этого сообщения Алиса дает Бобу важную информацию о финансовой транзакции, которая заякоревает платежный канал:

`funding_txid`

Это ИД финансовой транзакции (TxID), который используется для создания ИД канала после установления канала.

`funding_output_index`

Это индекс выхода, поэтому Боб знает, какой выход транзакции (например, выход 0) является мультиподписным выходом «2 из 2», финансируемым Алисой. Индекс выхода также используется для формирования ИД канала.

Наконец, Алиса также отправляет подпись (`signature`), соответствующую `funding_pubkey` Алисы и используемую для расходования из мультиподписи «2 из 2». Это необходимо Бобу, потому что ему также нужно будет создать свою собственную версию фиксационной транзакции. Для этой фиксационной транзакции требуется подпись Алисы, которую она ему предоставляет. Обратите внимание, что фиксационные транзакции Алисы и Боба выглядят немного по-разному, поэтому подписи будут разными. Знание того, как выглядит фиксационная транзакция другой стороны, имеет решающее значение и является частью протокола, чтобы предоставлять валидную подпись.

В протоколе Lightning мы часто видим, как узлы отправляют подписи вместо подписанных транзакций целиком. Это обусловлено тем, что любая сторона может реконструировать ту же самую транзакцию, и поэтому для ее подтверждения требуется только подпись. Отправка только подписи, а не всей транзакции целиком, значительно экономит пропускную способность сети.

Сообщение funding_signed

После получения сообщения `funding_created` от Алисы Боб теперь знает ИД финансовой транзакции и индекс выхода. ИД канала определяется побитовым «исключающим или» (XOR) ИД финансовой транзакции и индекса выхода:

```
channel_id = funding_txid XOR funding_output_index
```

Точнее, `channel_id`, то есть 32-байтовое представление UTXO-выхода финансовой транзакции, генерируется путем применения операции XOR к младшим 2 байтам финансового TxID и индексу выхода.

Боб также должен будет отправить Алисе свою подпись под возвратной транзакцией, основываясь на публичном ключе `funding_pubkey` Боба, который сформировал мультиподпись «2 из 2». Хотя у Боба уже есть локальная возвратная транзакция, это позволит Алисе завершить возвратную транзакцию со всеми необходимыми подписями и быть уверенной, что ее деньги будут возвращены в случае, если что-то пойдет не так.

Боб создает сообщение `funding_signed` и отправляет его Алисе. Ниже мы видим содержание этого сообщения:

```
[channel_id:channel_id]
[signature:signature]
```

Широковещательная передача финансовой транзакции

После получения сообщения `funding_signed` от Боба у Алисы теперь есть обе подписи, необходимые для подписания возвратной транзакции. Ее «план выхода» теперь защищен, и поэтому она может выполнить широковещательную передачу финансовой транзакции, не опасаясь привязывания своих средств. Если что-то пойдет не так, то Алиса сможет просто передать возвратную транзакцию в сеть и вернуть свои деньги без какой-либо дополнительной помощи со стороны Боба.

Теперь Алиса отправляет финансовую транзакцию в сеть Bitcoin, чтобы ее можно было добыть в блочной цепи. И Алиса, и Боб будут следить за этой транзакцией и ждать minimum_depth подтверждений (например, шести подтверждений) в блочной цепи Bitcoin.

Разумеется, Алиса будет использовать протокол Bitcoin, чтобы убедиться, что подпись, которую отправил ей Боб, действительно валидна. Этот шаг очень важен. Если бы по какой-то причине Боб отправлял Алисе неверные данные, то ее «план выхода» был бы сорван.

Сообщение funding_locked

Как только финансовая транзакция наберет необходимое число подтверждений, Алиса и Боб отправят друг другу сообщение `funding_locked`, и канал будет готов к использованию.

Отправка платежей по каналу

Канал настроен, но в его начальном состоянии вся емкость (140 000 сатоши) находится на стороне Алисы. Это означает, что Алиса может отправлять платежи Бобу по каналу, но у Боба пока нет средств для отправки Алисе.

В следующих нескольких разделах мы покажем, как осуществляются платежи по платежному каналу и как обновляется состояние канала.

Давайте допустим, что Алиса хочет отправить Бобу 70 000 сатоши, чтобы оплатить свой счет в кофейне Боба.

Разделение остатка

В принципе, отправка платежа от Алисы Бобу – это просто вопрос перераспределения остатка канала. Перед отправкой платежа у Алисы есть 140 000 сатоши, а у Боба их нет. После отправки платежа в размере 70 000 сатоши у Алисы остается 70 000 сатоши, а у Боба будет 70 000 сатоши.

Следовательно, Алисе и Бобу лишь нужно создать и подписать транзакцию, которая расходует мультиподпись «2 из 2» на два выхода, выплачивая Алисе и Бобу их соответствующие остатки. Мы называем эту обновленную транзакцию *фиксационной транзакцией*.

Алиса и Боб оперируют платежным каналом, *продвигая состояние канала вперед* с помощью ряда фиксаций (commitments). Каждая фиксация обновляет остатки, чтобы отразить платежи, которые прошли по каналу. И Алиса, и Боб могут инициировать новую фиксацию, чтобы обновить канал.

На рис. 7-6 мы видим несколько фиксационных транзакций.

Первая фиксационная транзакция, показанная на рис. 7-6, – это возвратная транзакция, которую Алиса собрала перед финансированием канала. На диаграмме это фиксация #0. После того как Алиса заплатит Бобу 70 000 сатоши, новая фиксационная транзакция (фиксация #1) имеет два выхода, выплачивающих Алисе и Бобу их соответствующие остатки. Мы включили две последующие фиксационные транзакции (фиксация #2 и фиксация #3), которые представляют Алису, соответственно выплачивающую Бобу дополнительные 10 000 сатоши, а затем 20 000 сатоши.

Каждая подписанная и валидная фиксационная транзакция может быть использована любым партнером канала в любое время для закрытия канала путем ее широковещательной передачи в сеть Bitcoin. Поскольку у них обоих есть самая последняя фиксационная транзакция и они могут использовать ее в любое время, они также могут ее просто удерживать и не передавать в сеть. Это их гарантия справедливого выхода из канала.

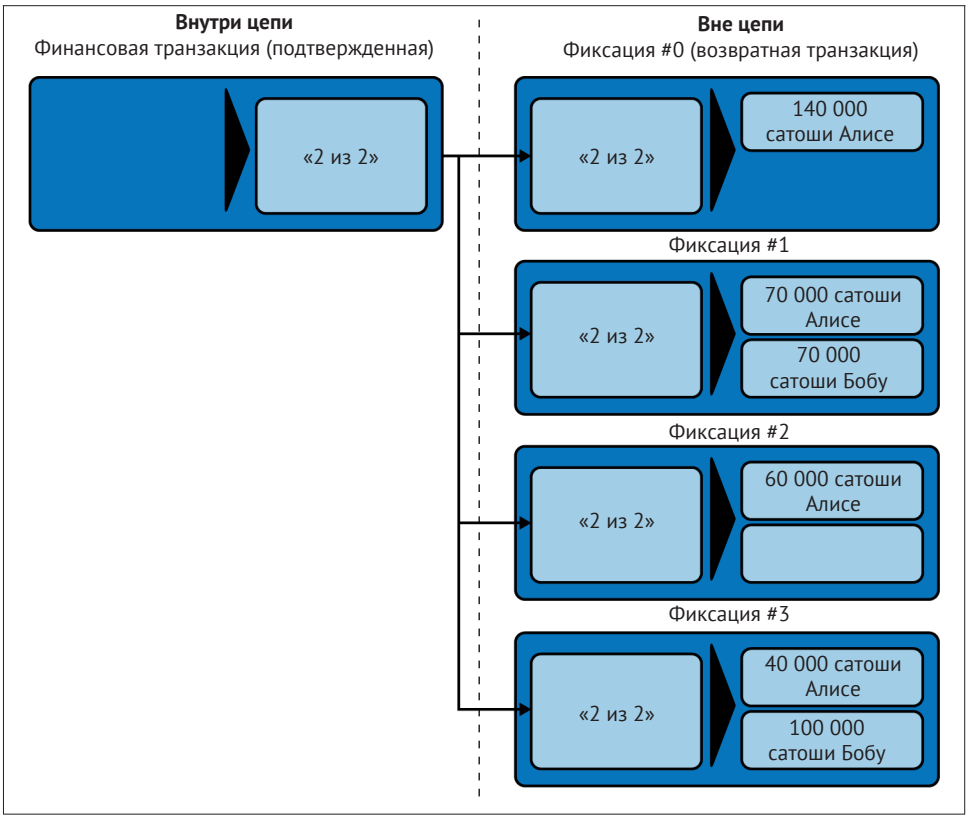


Рис. 7-6. Несколько фиксационных транзакций

Конкурирующие фиксации

Возможно, вам интересно, как Алиса и Боб могут иметь несколько фиксационных транзакций и при этом пытаться потратить один и тот же выход «2 из 2» из финансовой транзакции. Разве эти фиксационные транзакции не противоречат друг другу? Разве это не «двойное расходование», которое система Bitcoin призвана предотвращать?

Это действительно так! Фактически мы полагаемся на способность Bitcoin *предотвращать* двойное расходование, чтобы наладить работу Lightning. Независимо от того, сколько фиксационных транзакций Алиса и Боб строят и подписывают, на самом деле только одна из них может быть подтверждена.

До тех пор, пока Алиса и Боб удерживают эти транзакции и не выполнят их широковещательную передачу, финансовый выход остается неизрасходованным. Но если фиксационная транзакция будет передана в сеть и подтверждена, то она израсходует финансовый выход. Если Алиса или Боб попытаются выполнить широковещательную передачу более одной фиксационной транзакции, то только одна из них будет подтверждена, а остальные будут отклонены как (безуспешная) попытка двойного расходования.

Если в сеть передается более одной фиксационной транзакции, то существует множество факторов, которые будут определять, какая из них будет подтверждена первой: сумма включенных комиссионных, скорость распространения этих конкурирующих транзакций, топология сети и т. д. По сути, все это становится гонкой без предсказуемого результата и выглядит не очень надежно. Это звучит так, как будто кто-то может обмануть.

Обман со старыми фиксационными транзакциями

Давайте повнимательнее рассмотрим фиксационные транзакции на рис. 7-6. Все четыре фиксационные транзакции подписаны и валидны. Но только последняя из них точно отражает самые последние остатки каналов. В этом конкретном сценарии у Алисы есть возможность обмануть, передав более старую фиксацию и подтвердив ее в блочной цепи Bitcoin. Допустим, Алиса передает фиксацию #0 и получает ее подтверждение: она фактически закроет канал и заберет все 140 000 сатоши себе. Фактически в этом конкретном примере любая фиксация, кроме фиксации #3, улучшает положение Алисы и позволяет ей «отменить» по меньшей мере часть платежей, отраженных в канале.

В следующем разделе мы увидим, как сеть Lightning решает эту проблему – предотвращает использование партнерами канала старых фиксационных транзакций с помощью механизма отзыва и штрафных санкций. Существуют и другие способы предотвращения передачи старых фиксационных транзакций, такие как каналы eltoo, но для них требуется модернизация в Bitcoin Script под названием «перепривязка входа» (см. «Инновации в Bitcoin-протоколе и в Bitcoin Script» на стр. 189).

Отзыв старых фиксационных транзакций

Bitcoin-транзакции не истекают, и их невозможно «отменить». Их также нельзя остановить или цензурировать после того, как они были широковещательно переданы в сеть. Тогда как можно «отозвать» транзакцию, которой владеет другой человек и которая уже была подписана?

Используемое в Lightning решение является еще одним примером протокола справедливости. Вместо того чтобы пытаться контролировать возможность широковещательной передачи, существует встроенный механизм штрафных санкций, который обеспечивает, чтобы широковещательная передача старой фиксационной транзакции не отвечала наилучшим интересам потенциального обманщика. Он всегда может передать ее в сеть, но, скорее всего, потеряет деньги, если это сделает.

Слово «отозвать» является неправильным, поскольку оно подразумевает, что более старые фиксации каким-то образом становятся невалидными и не могут быть широковещательно переданы в сеть и подтверждены. Но это не так, поскольку валидные транзакции в системе Bitcoin нельзя отозвать. Вместо этого протокол Lightning использует механизм штрафных санкций для штрафования партнера по каналу, который передает в сеть старую фиксацию.

Механизм отзыва и штрафных санкций в рамках протокола Lightning составляют три элемента:

Асимметричные фиксационные транзакции

Фиксационные транзакции Алисы немного отличаются от транзакций Боба.

Отложенное расходование

Платеж стороне, удерживающей фиксационную транзакцию, задерживается (привязан ко времени), в то время как платеж другой стороне может быть истребован немедленно.

Отзывные ключи

Используются для отвязывания штрафной опции за старые фиксации.

Давайте рассмотрим эти три элемента по очереди.

Асимметричные фиксационные транзакции

Алиса и Боб удерживают слегка разные фиксационные транзакции. Давайте конкретно рассмотрим фиксацию #2 из рис. 7-6 более подробно на рис. 7-7.

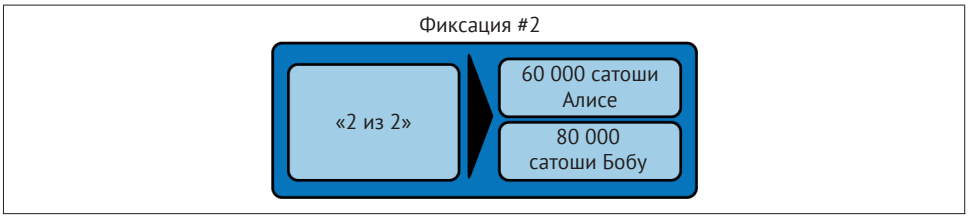


Рис. 7-7. Фиксационная транзакция #2

Как показано на рис. 7-8, Алиса и Боб придерживаются двух разных вариантов этой транзакции.

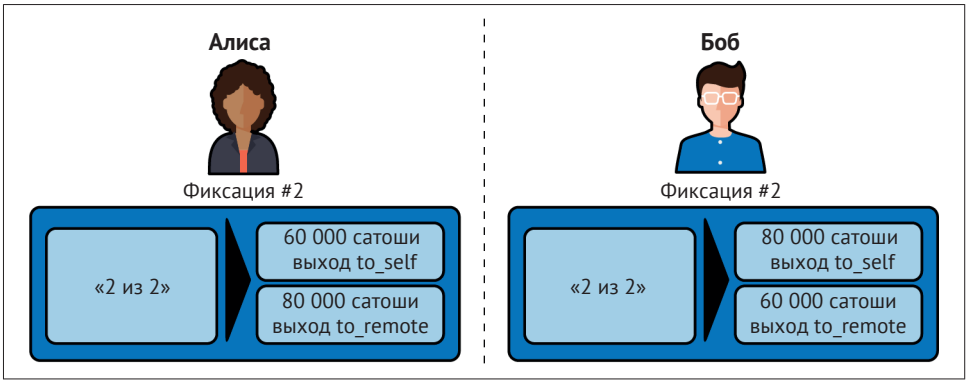


Рис. 7-8. Асимметричные фиксационные транзакции

По традиции, в рамках протокола Lightning мы называем двух партнеров по каналу локальным `self` (или `local`) и дистанцированным `remote`, в зависимо-

сти от того, с какой стороны мы смотрим. Выходы, которые оплачиваются каждому партнеру по каналу, называются соответственно `to_local` и `to_remote`.

На рис. 7-8 мы видим, что Алиса удерживает транзакцию, которая выплачивает 60 000 сатоши выходу `to_self` (который может быть потрачен ключами Алисы), и 80 000 сатоши дистанцированно выходу `to_remote` (которые могут быть потрачены ключами Боба).

Боб хранит зеркальное отражение этой транзакции, где первый выход составляет 80 000 сатоши выходу `to_self` (которые могут быть потрачены ключами Боба) и 60 000 сатоши выходу `to_remote` (который может быть потрачен ключами Алисы).

Задержанное (привязанное ко времени) расходование выхода to_self

Использование асимметричных транзакций позволяет протоколу легко возлагать *вину* на обманывающую сторону. Инвариант, заключающийся в том, что *широковещающая* сторона всегда должна ждать, обеспечивает, чтобы у «честной» стороны было время опровергнуть истребование и отозвать свои средства. Эта асимметрия проявляется в форме отличающихся выходов для каждой стороны: выход `to_local` всегда привязан ко времени и не может быть потрачен немедленно, тогда как выход `to_remote` не привязан ко времени и может быть потрачен немедленно.

Например, в фиксационной транзакции, которую удерживает Алиса, выход to_local, который платит ей, привязан ко времени на 432 блока, тогда как выход to_remote, который платит Бобу, может быть потрачен немедленно (см. рис. 7-9). Фиксационная транзакция Боба для фиксации #2 является зеркальным отражением: его собственный выход (to_local) привязан ко времени, а выход to_remote Алисы может быть потрачен немедленно.

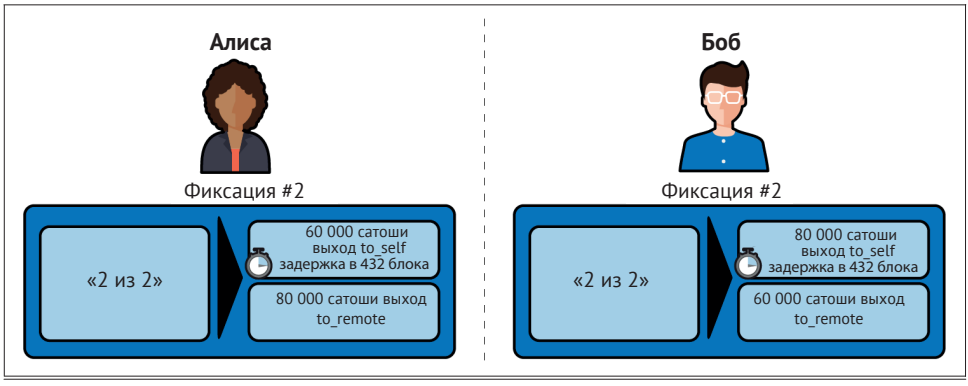


Рис. 7-9. Асимметричная и отложенная фиксационные транзакции

Это означает, что если Алиса закроет канал, выполнив широковещательную передачу и подтвердив фиксационную транзакцию, которую она удерживает, то она не сможет потратить свой остаток в течение 432 блоков, но Боб может немедленно истребовать свой остаток. Если же Боб закроет канал, используя фик-

сационную транзакцию, которую он удерживает, то он не сможет потратить свой выход в течение 432 блоков, тогда как Алиса может немедленно потратить свой.

Задержка существует по одной причине: чтобы позволить *дистанцированной* стороне воспользоваться штрафной опцией, если старая (отозванная) фиксация должна быть передана широковещательно другим партнером по каналу. Далее давайте рассмотрим отзывные ключи и штрафную опцию.

Задержка согласовывается Алисой и Бобом во время потока сообщений начального строительства канала в виде поля `to_self_delay`. В целях обеспечения безопасности канала задержка масштабируется в соответствии с емкостью канала – это означает, что канал с большей суммой средств имеет более длительные задержки в выходах `to_self` в фиксациях. Узел Алисы включает желаемую задержку `to_self_delay` в сообщении `open_channel`. Если Боб сочтет ее приемлемой, то его узел включит то же значение для `to_self_delay` в сообщение `accept_channel`. Если он не согласен, то канал отклоняется (см. «Сообщение shutdown» на стр. 201).

Отзывные ключи

Как мы обсуждали ранее, слово «отзыв» чуть-чуть дезориентирует, поскольку подразумевает, что «отозванная» транзакция не может быть использована.

На самом деле отозванная транзакция может быть использована, но если она используется и была отозвана, то один из партнеров канала может забрать все средства канала, создав штрафную транзакцию.

То, как это работает, заключается в том, что выход to_local не только привязан ко времени, но также имеет два условия расходования в скрипте: он может быть потрачен *локально* после задержки привязки ко времени *либо* может быть потрачен *дистанцированно* и немедленно с помощью отзывного ключа для этой фиксации.

Итак, в нашем примере каждая сторона держит фиксационную транзакцию, которая имеет отзывную опцию в выходе to_local, как показано на рис. 7-10.

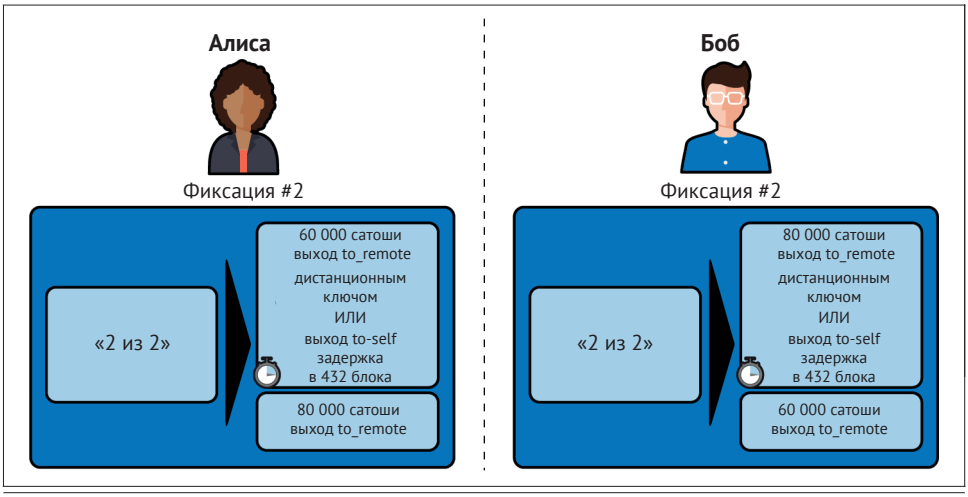


Рис. 7-10. Асимметричные, задержанные и отзываемые фиксации

Фиксационная транзакция

Теперь, когда мы понимаем структуру фиксационных транзакций и причину, почему нам нужны асимметричные, задержанные, отзываемые фиксации, давайте обратимся к исходному коду Bitcoin Script, который это имплементирует.

Первый (to_local) выход фиксационной транзакции определен в спецификации «BOLT #3: Фиксационная транзакция, выход to_local»[60] следующим образом:

```
OP_IF
    # Штрафная транзакция
    <revocationpubkey>
OP_ELSE
    <to_self_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

Это условный скрипт (см. «Скрипты с несколькими условиями» на стр. 394), что означает, что выход может быть израсходован, если удовлетворено одно из двух условий. Первое условие позволяет использовать выход любому, кто может подписаться под `<revocationpubkey>`. Второе привязано ко времени `<to_self_delay>` блоками и может быть использовано только после этого числа блоков любым, кто может подписаться под `<local_delayedpubkey>`. В нашем примере мы установили привязку ко времени `<to_self_delay>` равной 432 блокам, но эта задержка конфигурируема и согласовывается двумя партнерами по каналу. Длительность `to_self_delay` привязки ко времени обычно выбирается пропорционально емкости канала, и, стало быть, каналы с большей емкостью (больше средств) имеют более длительные (`to_self_delay`) привязки ко времени для защиты сторон.

Первое условие позволяет расходовать выход любому, кто может подписаться под `<revocationpubkey>`. Критически важным требованием к безопасности этого скрипта является то, что дистанцированная сторона не может в одностороннем порядке подписываться ключом `revocationpubkey`. В целях понимания причины, почему это важно, рассмотрим сценарий, в котором дистанцированная сторона нарушает ранее отозванную фиксацию. Если она может подписаться этим ключом, то она может просто *сама* воспользоваться пунктом об отзыве и украсть все средства в канале. Вместо этого мы выводим отзывной ключ `revocationpubkey` для *каждого* состояния, основываясь на информации как от локальной (self), так и от дистанцированной (remote) стороны. Разумное использование симметричной и асимметричной криптографии используется для того, чтобы обе стороны могли вычислять публичный ключ `revocationpubkey`, но только честная сторона могла вычислять приватный ключ, имея свою секретную информацию, как подробно описано во вставке «Выведение отзывного и фиксационного секретов» на стр. 194.

[60] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#to_local-output.

Выведение отзывного и фиксационного секретов

Каждая сторона отправляет точку revocation_basepoint во время начальных сообщений о согласовании канала, а также точку first_per_commitment_point. Точка revocation_basepoint является статической в течение всего срока канала, тогда как каждое новое состояние канала будет основываться на новой точке first_per_commitment_point.

Учитывая эту информацию, отзывной ключ revocationpubkey для каждого состояния канала выводится с помощью следующей ниже серии операций на эллиптической кривой и хеширования:

```
revocationpubkey = revocation_basepoint * sha256(revocation_basepoint ||
per_commitment_point) + per_commitment_point * sha256(per_commitment_point
|| revocation_basepoint)
```

Из-за коммутативного свойства абелевых групп, над которыми определены эллиптические кривые, после того как дистанцированная сторона раскрывает секрет `per_commitment_secret` (приватный ключ для точки `per_commitment_point`), `self` может вывести приватный ключ для отзывного ключа `revocationpubkey` с помощью следующей ниже операции:

```
revocation_priv = (revocationbase_priv * sha256(revocation_basepoint
|| per_commitment_point)) + (per_commitment_secret * sha256(per_commitment_
point || revocation_basepoint)) mod N
```

В целях понимания причины, почему это работает на практике, обратите внимание, что мы можем переупорядочить (переставить местами) и разложить вычисление изначальной формулы публичного ключа в отношении отзывного ключа `revocationpubkey`:

```
revocationpubkey = G*(revocationbase_priv * sha256(revocation_basepoint
|| per_commitment_point) + G*(per_commitment_secret * sha256(per_commitment_
point || revocation_basepoint))
                 = revocation_basepoint * sha256(revocation_basepoint ||
per_commitment_point) + per_commitment_point * sha256(per_commitment_point
|| revocation_basepoint))
```

Другими словами, `revocationbase_priv` может быть выведен (и использован для подписи под отзывным ключом `revocationpubkey`) только той стороной, которая знает как `revocationbase_priv`, так и `per_commitment_secret`. Этот маленький трюк – именно то, что делает используемую в сети Lightning систему отзыва на основе публичного ключа безопасной.

Привязка ко времени, используемая в фиксационной транзакции с `CHECKSEQUENCEVERIFY`, является привязкой к относительному времени. Она подсчитывает блоки, прошедшие с момента подтверждения этого выхода. Это означает, что она не будет расходоваться до тех пор, пока не наступит `to_self_delay`-й блок *после* того, как эта фиксационная транзакция была широковещательно передана и подтверждена.

Второй выход (to_remote) фиксационной транзакции определен в спецификации «BOLT # 3: фиксационная транзакция, выход to_remote»[61] и в простейшей форме является оплатой по хешу публичного ключа свидетеля (Pay-to-Witness-Public-Key-Hash, аббр. P2WPKH) для <remote_pubkey>, что означает, что он просто платит владельцу, который может подписаться под <remote_pubkey>.

Теперь, когда мы дали подробное определение фиксационным транзакциям, давайте посмотрим, как Алиса и Боб продвигают вперед состояние канала, создают и подписывают новые фиксационные транзакции и отзывают старые фиксационные транзакции.

Продвижение состояния канала вперед

В целях продвижения состояния канала вперед Алиса и Боб обмениваются двумя сообщениями: сообщениями commitment_signed о подписании фиксации и revoke_and_ack об отзыве и возврате. Сообщение committement_signed может быть отправлено любым партнером по каналу, когда у него есть обновление состояния канала. Затем другой партнер по каналу может ответить с помощью revoke_and_ack, чтобы отозвать старую фиксацию и подтвердить новую фиксацию.

На рис. 7-11 мы видим, как Алиса и Боб обмениваются двумя парами сообщений committement_signed и revoke_and_ack. Первый поток показывает обновление состояния, инициированное Алисой (слева направо committement_signed), на которое отвечает Боб (справа налево revoke_and_ack). Второй поток показывает обновление состояния, инициированное Бобом и на которое ответила Алиса.

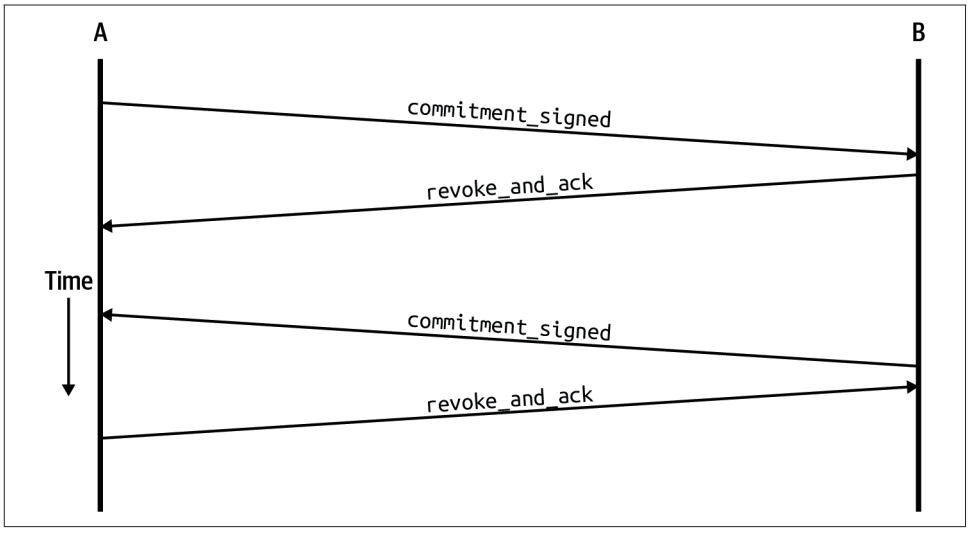


Рис. 7-11. Поток фиксационных и отзывных сообщений

[61] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#to_remote-output.

Сообщение committement_signed

Структура сообщения `committement_signed` определена в спецификации «BOLT #2: одноранговый протокол, сообщение `committement_signed`»[62] и показана здесь:

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

`channel_id`

Это идентификатор канала.

`signature`

Подпись под новой дистанцированной фиксацией.

`num_htlcs`

Число обновленных HTLC-контрактов в этой фиксации.

`htlc_signature`

Подписи под обновлениями.

Применение HTLC-контрактов для фиксации обновлений будет подробно объяснено в разделе «Контракты с привязкой к хешу и времени» на стр. 216 и в главе 9.

Сообщение `committement_signed` от Алисы дает Бобу необходимую подпись (часть «2 из 2» Алисы) для новой фиксационной транзакции.

Сообщение об отзыве и возврате

Теперь, когда у Боба есть новая фиксационная транзакция, он может отменить предыдущую фиксацию, предоставив Алисе отзывной ключ, и собрать новую фиксацию с подписью Алисы.

Сообщение revoke_and_ack определено в спецификации «BOLT #2: одноранговый протокол, сообщение revoke_and_ack»[63] и показано ниже:

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

`channel_id`

Это идентификатор канала.

`per_commitment_secret`

Используется для генерирования отзывного ключа для предыдущей (старой) фиксации, эффективно ее отзывающего.

[62] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#committing- updates-so-far-commitment_signed.

[63] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#completing-the-transition-to-the-updated-state-revoke_and_ack.

`next_per_commitment_point`

Используется для создания `revocation_pubkey` для новой фиксации, чтобы впоследствии ее можно было отозвать.

Отзыв и рефиксация

Давайте рассмотрим это взаимодействие между Алисой и Бобом повнимательнее.

Алиса дает Бобу средства для создания новой фиксации. В свою очередь, Боб отзывает старую фиксацию, чтобы заверить Алису, что он не будет ей пользоваться. Алиса может доверять новой фиксации только в том случае, если у нее есть отзывной ключ, чтобы оштрафовать Боба за публикацию старой фиксации. С точки зрения Боба, он может спокойно отозвать старую фиксацию, дав Алисе ключи, чтобы оштрафовать его, потому что у него есть подпись под новой фиксацией.

Когда Боб отвечает с помощью `revoke_and_ack`, он дает Алисе секрет `per_commitment_secret`. Этот секрет может быть использован для сборки ключа подписания отзыва под старой фиксацией, который позволяет Алисе конфисковать все средства канала, применив штраф.

После того как Боб выдаст этот секрет Алисе, он никогда не должен выполнять широковещательную передачу этой старой фиксации. Если он это сделает, то он даст Алисе возможность его оштрафовать и забрать средства. По сути, Боб дает Алисе возможность привлечь его к ответственности за широковещательную передачу старой фиксации; в сущности, он отозвал свою способность использовать эту старую фиксацию.

После того как Алиса получит `revoke_and_ack` от Боба, она может быть уверена, что Боб не сможет передать старую фиксацию без штрафа. Теперь у нее есть ключи, необходимые для создания штрафной транзакции, если Боб выполнит широковещательную передачу старой фиксации.

Обман и наказание на практике

На практике и Алисе, и Бобу приходится выполнять мониторинг обмана. Они отслеживают блочную цепь Bitcoin на предмет любых фиксационных транзакций, связанных с любым из каналов, которыми они оперируют. Если они увидят фиксационную транзакцию, подтвержденную внутри цепи, то они проверят, является ли она самой последней фиксацией. Если это «старая» фиксация, то они должны немедленно собрать и широковещательно передать штрафную транзакцию. Штрафная транзакция расходует как выход `to_local`, так и выход `to_remote`, закрывая канал и отправляя оба остатка обманутому партнеру по каналу.

Для того чтобы обеим сторонам было проще отслеживать номера переданных отзывных фиксаций, каждая фиксация фактически *кодирует* свой номер внутри полей времени привязки и последовательности при переходе. В рамках протокола эта специальная кодировка называется *подсказками состояния*. Исходя из того, что сторона знает текущий номер фиксации, она может использовать подсказки состояния, чтобы легко распознавать, была ли переданная фиксация отозвана, и если да, то фиксация под каким номером была нарушена, поскольку этот номер используется для простого поиска отзывного секрета, который следует использовать в дереве отзывных секретов (shachain).

Вместо того чтобы кодировать подсказку состояния у всех на виду, используется *запутанная* подсказка состояния. Это запутывание (обфускация) достигается сначала путем применения операции XOR к номеру текущей фиксации и набору случайных байтов, сгенерированных детерминированным образом с использованием финансовых публичных ключей обеих сторон канала. Для кодирования подсказки состояния в фиксационной транзакции используются в общей сложности 6 байт времени привязки и последовательности (24 бита времени привязки и 24 бита последовательности), поэтому для использования XOR требуется 6 случайных байт. Для того чтобы получить эти 6 байт, обе стороны получают SHA-256-хеш финансового ключа инициатора, конкатенированный с финансовым ключом ответчика. Перед кодированием высоты текущей фиксации к целому числу и этому обфускатору подсказки состояния применяется операция XOR, а затем кодируется в нижних 24 битах времени привязки и верхних 64 битах последовательности.

Давайте рассмотрим наш канал между Алисой и Бобом и покажем конкретный пример штрафной транзакции. На рис. 7-12 мы видим четыре фиксации на канале Алисы и Боба. Алиса сделала три платежа Бобу:

- 70 000 сатоши выплачены и переданы Бобу с фиксацией #1;
- 10 000 сатоши выплачены и переданы Бобу с фиксацией #2;
- 20 000 сатоши выплачены и переданы Бобу с фиксацией #3.

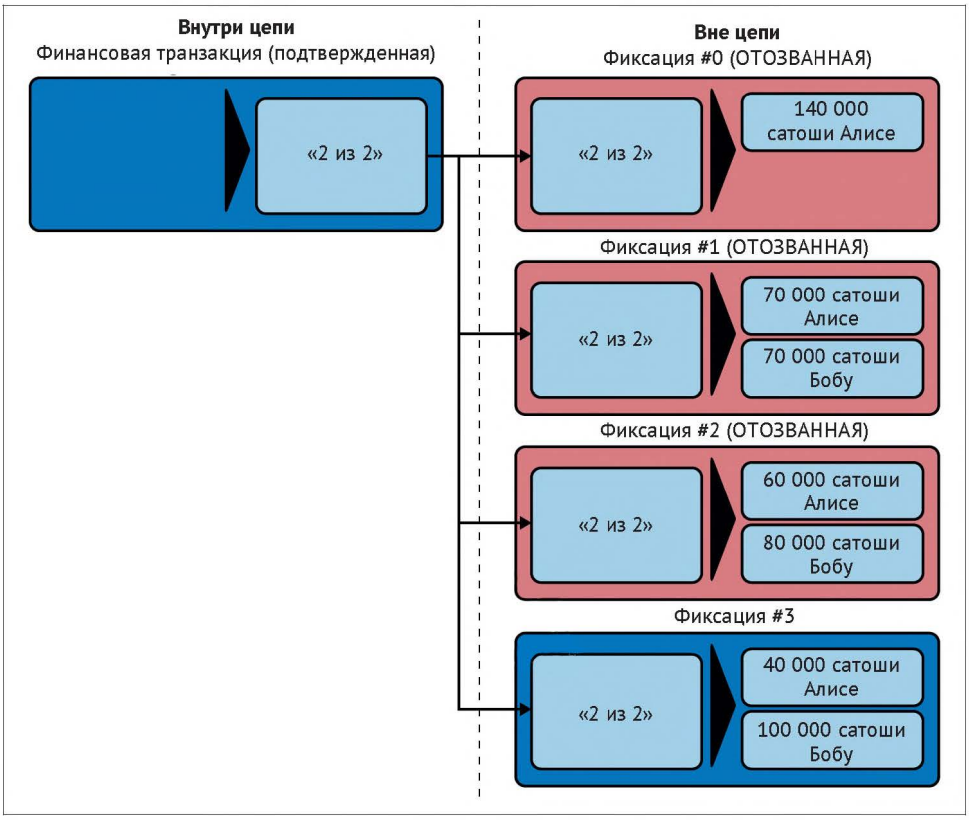


Рис. 7-12. Отозванные и текущие фиксации

С каждой фиксацией Алиса отзывала предыдущую (более старую) фиксацию. Текущее состояние канала и правильный остаток представлены фиксацией #3. Все предыдущие фиксации были отозваны, и у Боба есть ключи, необходимые для выдачи штрафных транзакций против них, на случай если Алиса попытается выполнить широковещательную передачу одной из них.

У Алисы может быть стимул обманывать, потому что все предыдущие фиксационные транзакции дали бы ей более высокую долю остатка канала, чем она имеет право. Допустим, например, что Алиса попыталась передать фиксацию #1. Эта фиксационная транзакция принесет Алисе 70 000 сатоши и Бобу 70 000 сатоши. Если бы Алиса смогла широковещательно передать и израсходовать свой выход `to_local`, то она фактически украла бы 30 000 сатоши у Боба, откатив свои последние два платежа Бобу.

Алиса решает пойти на огромный риск и выполнить широковещательную передачу отозванной фиксации #1, чтобы украсть 30 000 сатоши у Боба. На рис. 7-13 мы видим старую фиксацию Алисы, которую она передает в блочную цепь Bitcoin.

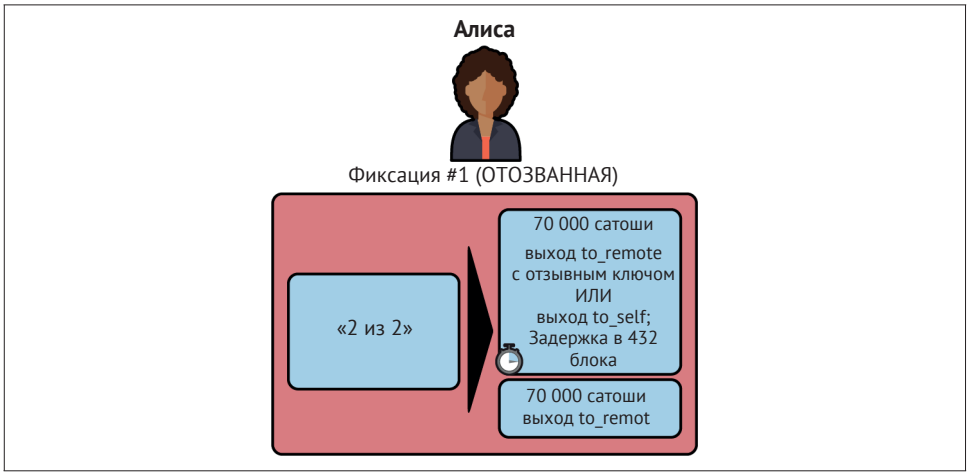


Рис. 7-13. Алиса обманывает

Как вы видите, старая фиксация Алисы имеет два выхода: один платит 70 000 сатоши себе (выход `to_local`), а другой платит 70 000 сатоши Бобу. Алиса еще не может потратить свой выход `to_local` в размере 70 000, потому что у нее есть привязка ко времени продолжительностью 432 блока (3 дня). Теперь она надеется, что Боб не заметит этого в течение трех дней.

К несчастью для Алисы, узел Боба тщательно отслеживает блочную цепь Bitcoin и видит, что старая фиксационная транзакция вещается в сеть и (в конечном итоге) подтверждается внутри цепи.

Узел Боба немедленно выполнит широковещательную передачу штрафной транзакции. Поскольку эта старая фиксация была отозвана Алисой, у Боба есть секрет `per_commitment_secret`, который ему отправила Алиса. Он использует этот секрет для сборки подписи под `revocation_pubkey`. В то время как Алисе приходится ждать 432 блока, Боб может израсходовать оба выхода сразу.

Он может потратить выход `to_remote` с помощью своих приватных ключей, потому что он все равно должен был ему заплатить. Он также может потратить выход, предназначенный для Алисы, с помощью подписи из отзывного ключа. Его узел выполняет широковещательную передачу штрафной транзакции, показанной на рис. 7-14.

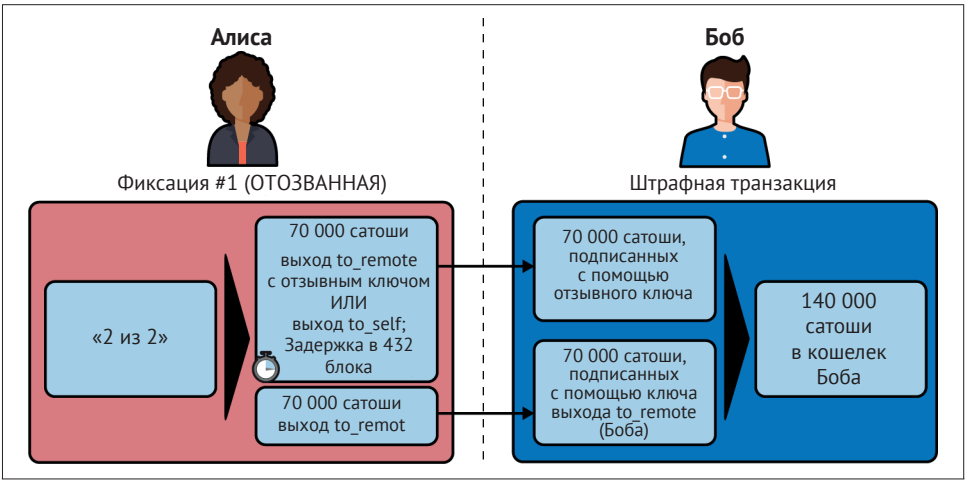


Рис. 7-14. Обман и наказание

Штрафная транзакция Боба выплачивает 140 000 сатоши на его собственный кошелек, занимая всю емкость канала. Алисе не только не удалось обмануть, но она потеряла в этой попытке абсолютно все!

Резерв канала: обеспечение личной заинтересованности

Возможно, вы заметили, что существует особая ситуация, в которой необходимо разобраться. Если бы Алиса могла продолжать тратить свой остаток до тех пор, пока он не станет нулевым, она была бы в состоянии закрыть канал, сделав широковещательную передачу старой фиксационной транзакции, не рискуя получить штраф: либо отозванная фиксационная транзакция завершится успешно после задержки, либо обманщик будет пойман, но никаких последствий не будет, потому что штраф равен нулю. С точки зрения теории игр, попытка обмануть в этой ситуации – это бесплатные деньги. Вот почему в игре задействован резерв канала, поэтому потенциальный обманщик всегда рискует получить штраф.

Закрытие канала (кооперативное закрытие)

До сих пор мы рассматривали фиксационные транзакции как один из возможных способов закрыть канал в одностороннем порядке. Этот тип закрытия канала не идеален, поскольку он навязывает привязку ко времени канальному партнеру, который его использует.

Более оптимальным способом закрытия канала является кооперативное закрытие. При кооперативном закрытии два партнера по каналу согласовывают окончательную фиксационную транзакцию, именуемую *закрывающей транз-*

акцией, которая немедленно переводит остаток каждой стороны на кошелек назначения по их выбору. Затем партнер, инициировавший поток закрытия канала, выполнит широковещательную передачу закрывающей транзакции.

Поток закрывающих сообщений определен в спецификации «BOLT #2: одноранговый протокол, закрытие канала» и показан на рис. 7-15.

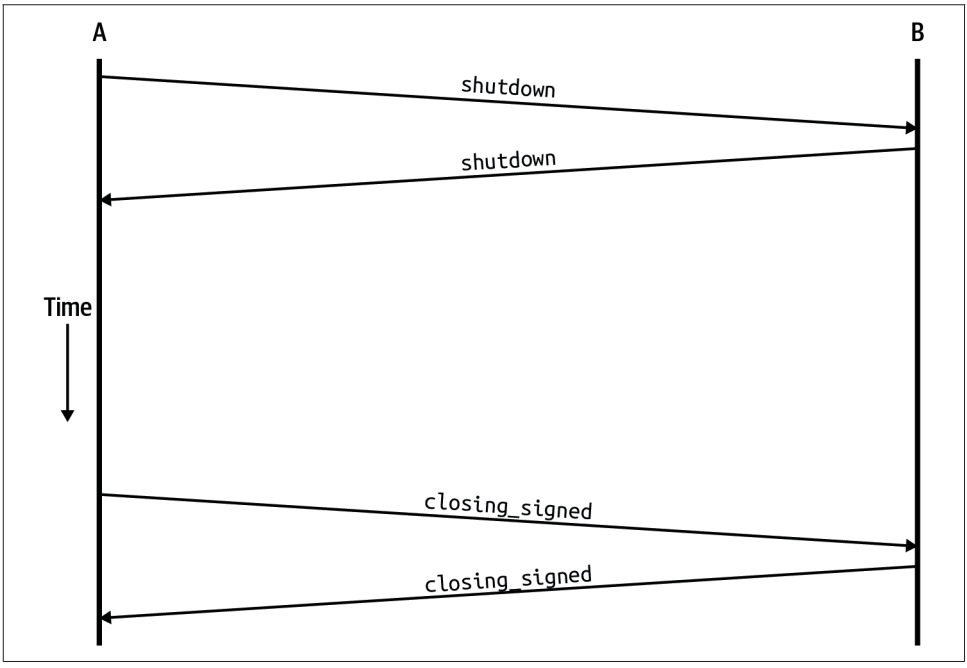


Рис. 7-15. Поток сообщений о закрытии канала

Сообщение shutdown

Закрытие канала начинается с того, что один из двух партнеров канала отправляет сообщение `shutdown` о завершении работы. Содержание этого сообщения показано ниже:

```
[channel_id:channel_id]
[u16:len]
[len*byte:scriptpubkey]
```

`channel_id`

Идентификатор канала, который мы хотим закрыть.

`len`

Длина скрипта целевого кошелька, на который этот партнер по каналу хочет получить свой остаток.

`scriptpubkey`

Скрипт Bitcoin Script целевого кошелька в одном из «стандартных» форматов Bitcoin-адресов (P2PKH, P2SH, P2WPKH, P2WSH и т. д.; см. глоссарий).

Допустим, Алиса отправляет Бобу сообщение `shutdown`, чтобы закрыть их канал. Алиса укажет скрипт Bitcoin Script, соответствующий Bitcoin-адресу ее кошелька. Она говорит Бобу: давайте совершим заключительную транзакцию, которая переведет мой остаток на этот кошелек.

Боб ответит своим собственным сообщением `shutdown`, указывающим на то, что он согласен кооперативно закрыть канал. Его сообщение `shutdown` содержит скрипт для адреса его кошелька.

Теперь и у Алисы, и у Боба есть адрес предпочтительного кошелька друг друга, и они могут создать идентичные закрывающие транзакции для расчета остатка канала.

Сообщение closing_signed

Исходя из того, что у канала нет неисполненных фиксаций или обновлений и партнеры канала обменялись сообщениями `shutdown`, показанными в предыдущем разделе, теперь они могут завершить это кооперативное закрытие.

Финансист канала (в нашем примере Алиса) начинает с отправки сообщения closing_signed Бобу. В этом сообщении предлагаются транзакционные комиссионные за внутрицепную транзакцию и подпись Алисы (мультиподпись «2 из 2») под закрывающей транзакцией. Сообщение closing_signed показано ниже:

```
[channel_id:channel_id]
[u64:fee_satoshis]
[signature:signature]
```

`channel_id`

Идентификатор канала.

`fee_satoshis`

Предлагаемые комиссионные за внутрицепную транзакцию в сатоши.

`signature`

Подпись отправителя под закрывающей транзакцией.

Когда Боб его получит, он сможет ответить своим собственным сообщением `closing_signed`. Если он согласен с комиссионными, то он просто возвращает те же предложенные комиссионные и свою собственную подпись. Если он не согласен, то он должен предложить другие комиссионные `fee_satoshis`.

Это согласование может продолжаться с обратными сообщениями `closing_signed` до тех пор, пока два партнера по каналу не договорятся о комиссионных.

После того как Алиса получит сообщение `closing_signed` с теми же комиссионными, что и те, которые она предложила в своем последнем сообщении, согласование будет завершено. Алиса подписывает и выполняет широковещательную передачу закрывающей транзакции, и канал закрывается.

Транзакция кооперативного закрытия

Транзакция кооперативного закрытия выглядит аналогично последней фиксационной транзакции, о которой договорились Алиса и Боб. Однако, в отличие от последней фиксационной транзакции, в выходе нет привязок ко времени

или штрафных отзывных ключей. Поскольку в сборке этой транзакции сотрудничают обе стороны и они не будут делать никаких дальнейших фиксаций, эта транзакция не нуждается ни в каких асимметричных, задержанных и отзываемых элементах.

В типичной ситуации адреса, используемые в этой транзакции кооперативного закрытия, генерируются заново для каждого закрываемого канала. Тем не менее обе стороны также могут привязываться к адресу «доставки», который будет использоваться для отправки их кооперативно улаженных средств. В пространстве имен TLV сообщений open_channel и accept_channel обе стороны могут свободно указывать «скрипт предварительного завершения работы». Обычно этот адрес является производным от ключей, которые находятся в холодном хранилище. Эта практика служит повышению безопасности каналов: если партнер по каналу каким-то образом взломан, то хакер не сможет кооперативно закрыть канал, используя адрес, который он контролирует. Вместо этого нескомпрометированный честный партнер по каналу откажется сотрудничать при закрытии канала, если указанный адрес предварительного отключения не используется. Эта функциональная возможность эффективно создает «замкнутый цикл», ограничивая поток средств из данного канала.

Алиса выполняет широковещательную передачу транзакции, показанной на рис. 7-16, чтобы закрыть канал.

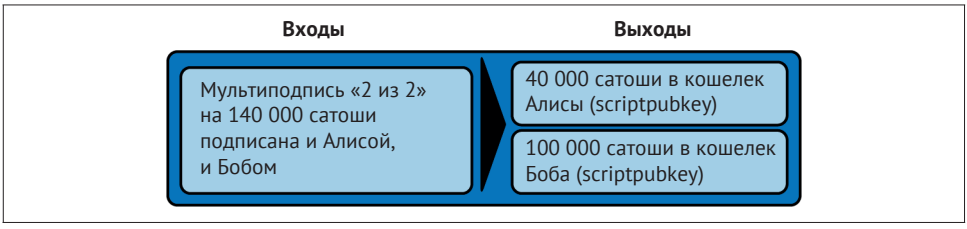


Рис. 7-16. Транзакция кооперативного закрытия

После подтверждения этой закрывающей транзакции в блочной цепи Bitcoin канал закрывается. Теперь Алиса и Боб могут расходовать свои выходы так, как им заблагорассудится.

Вывод

В этом разделе мы рассмотрели платежные каналы гораздо подробнее. Мы проинспектировали три потока сообщений, используемых Алисой и Бобом для согласования финансирования, фиксаций и закрытия канала. Мы также показали структуру финансирования, фиксаций и закрытия транзакций, а еще рассмотрели механизмы отзыва и штрафных санкций.

Как мы увидим в следующих нескольких главах, HTLC-контракты используются даже для локальных платежей между партнерами по каналу. В них нет необходимости, но протокол намного проще, если локальные (одноканальные) и маршрутизируемые (многоканальные) платежи осуществляются одним и тем же способом.

В одном платежном канале число платежей в секунду зависит только от пропускной способности сети между Алисой и Бобом. До тех пор, пока партнеры по каналу могут отправлять несколько байтов данных туда и обратно, чтобы согласиться на новый остаток канала, они фактически произвели платеж. Вот почему мы можем достичь гораздо большей емкости платежей в сети Lightning (вне цепи), чем емкость транзакций, которая может обрабатываться блочной цепью Bitcoin (внутри цепи).

В следующих нескольких главах мы обсудим маршрутизацию, HTLC-контракты и их использование в канальных операциях.

Глава 8

Маршрутизация в сети платежных каналов

В этой главе мы наконец полностью распакуем тему о том, как можно подсоединять платежные каналы для формирования сети платежных каналов с помощью процесса, именуемого *маршрутизацией*. В частности, мы рассмотрим первую часть маршрутизационного слоя, протокол «атомарных и бездоверительных многопереходных контрактов». Это подчеркивается схемой в комплекте протоколов, показанной на рис. 8-1.

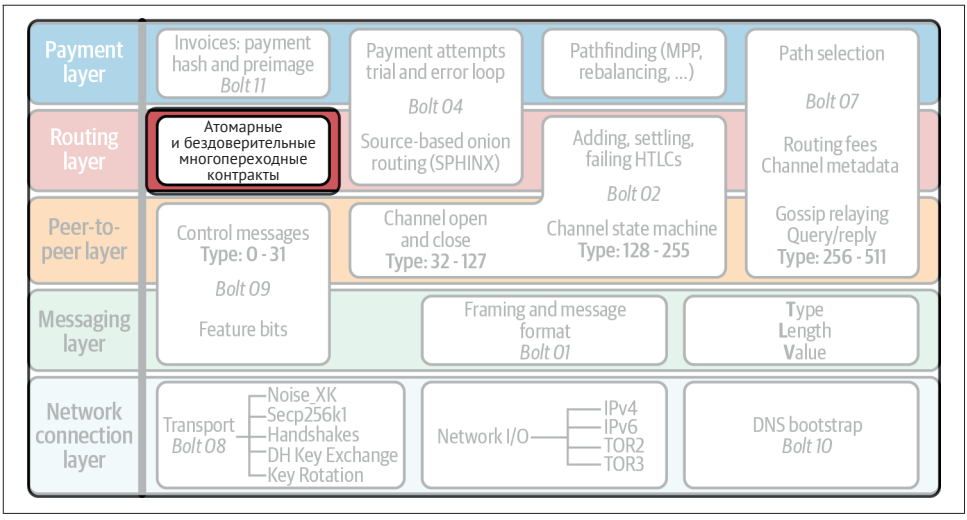


Рис. 8-1. Атомарное маршрутизирование платежей в рамках комплекта протоколов Lightning

Маршрутизирование платежа

В этом разделе мы рассмотрим маршрутизацию с точки зрения Дины, геймера, которая получает донаты от своих поклонников во время потоковой трансляции своих игровых сеансов.

Инновация маршрутизируемых платежных каналов позволяет Дине получать донаты, не поддерживая отдельный канал с каждым из ее поклонников, которые хотят пожертвовать ей донаты. До тех пор, пока существует путь хорошо финансируемых каналов от этого зрителя к Дине, она сможет получать оплату от этого поклонника.

На рис. 8-2 мы видим возможную схему сети, созданную различными платежными каналами между узлами Lightning. Каждый на этой диаграмме может отправить Дине платеж, построив путь. Представьте, что фанат 4 хочет отправить Дине платеж. Видите ли вы путь, который мог бы позволить этому произойти? Фанат 4 мог направить платеж Дине через фаната 3, Боба и Чана. Точно так же Алиса могла бы направить платеж Дине через Боба и Чана.

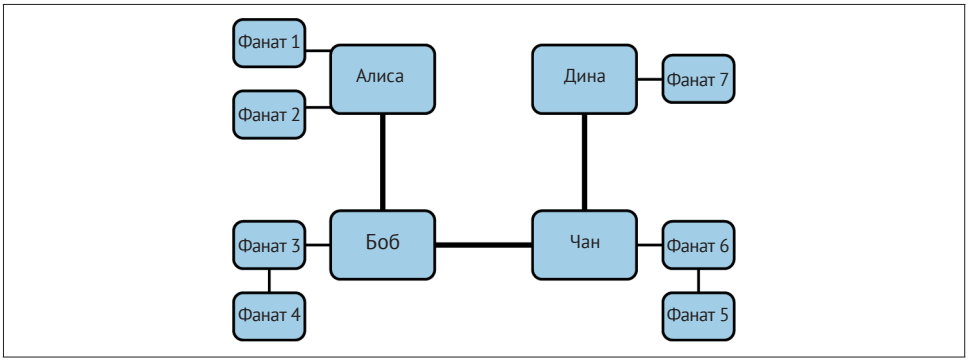


Рис. 8-2. Фанаты, подсоединенные косвенно (напрямую) к Дине в сети Lightning

Узлы на пути от фаната к Дине в контексте маршрутизирования платежа являются посредниками, именуемыми маршрутизационными узлами. Нет никакой функциональной разницы между маршрутизационными узлами и узлами, оперируемыми фанатами Дины. Любой узел Lightning способен маршрутизировать платежи по своим платежным каналам.

Важно отметить, что маршрутизационные узлы не могут красть средства во время маршрутизирования платежа от фаната к Дине. Кроме того, маршрутизационные узлы не могут терять деньги, участвуя в процессе маршрутизации. Маршрутизационные узлы могут взимать маршрутизационные комиссионные за посредничество, хотя в этом нет необходимости, и они могут выбрать бесплатное маршрутизирование платежей.

Еще одна важная деталь заключается в том, что из-за использования луковичной маршрутизации промежуточные узлы явно знают только об одном узле, предшествующем им, и об одном узле, следующем за ними по маршруту. Они не обязательно будут знать, кто является отправителем и получателем платежа. В силу этого фанаты получают возможность использовать промежуточные узлы для оплаты Дине без утечки личной информации и без риска кражи.

Этот процесс соединения ряда платежных каналов сквозной безопасностью и структурой стимулов для узлов, чтобы делать *пересылку* платежей, является одним из ключевых нововведений сети Lightning.

В этой главе мы углубимся в механизм маршрутизации в сети Lightning, подробно описав точный способ прохождения платежей по сети. Сначала мы

проясним концепцию маршрутизации и сравним ее с концепцией отыскания пути, поскольку их часто путают и используют как взаимозаменяемые. Далее мы создадим протокол справедливости: атомарный, бездоверительный многопереходной протокол, используемый для маршрутизирования платежей. В целях демонстрации того, как работает этот протокол справедливости, мы будем использовать физический эквивалент передачи золотых монет между четырьмя людьми. Наконец, мы рассмотрим имплементацию атомарного, бездоверительного многопереходного протокола, используемую в настоящее время в сети Lightning, которая называется контрактом с привязкой к хешу и времени (HTLC).

Маршрутизация против отыскания пути

Важно отметить, что мы отделяем концепцию маршрутизации от концепции отыскания пути. Эти два понятия часто путают, и термин «маршрутизация» нередко используется для описания обоих понятий. Давайте устраним двусмысленность, прежде чем двигаться дальше.

Отыскание пути (pathfinding), которое рассматривается в главе 12, представляет собой процесс отыскания и выбора непрерывного пути, состоящего из платежных каналов, который соединяет отправителя A с получателем B. Отправитель платежа отыскивает путь, инспектируя граф канала, который он собрал из канальных объявлений, переданных другими узлами.

Маршрутизация относится к серии взаимодействий по всей сети, которые пытаются переадресовать платеж из некоторой точки A в другую точку B по пути, ранее выбранному в процессе отыскания пути. Маршрутизация – это активный процесс отправки платежа по пути, который предусматривает сотрудничество всех промежуточных узлов вдоль этого пути.

Важное эмпирическое правило заключается в том, что между Алисой и Бобом может существовать путь (возможно, даже более одного), однако активного маршрута для отправки платежа может и не быть. Одним из примеров является сценарий, в котором все узлы, соединяющие Алису и Боба, в настоящее время находятся в офлайновом режиме. В этом примере можно разведать граф каналов и подсоединить ряд платежных каналов от Алисы к Бобу, следовательно, *путь* существует. Однако, поскольку промежуточные узлы находятся в офлайне, платеж отправить нельзя, и поэтому *маршрут* не существует.

Создание сети платежных каналов

Прежде чем мы углубимся в концепцию атомарного бездоверительного многопереходного платежа, давайте рассмотрим пример. Вернемся к Алисе, которая в предыдущих главах покупала кофе у Боба, с которым у нее есть открытый канал. Сейчас Алиса смотрит прямой эфир от Дины, геймера, и хочет отправить Дине донат в размере 50 000 сатоши через сеть Lightning. Но у Алисы нет прямого канала с Диной. Что может сделать Алиса?

Алиса могла бы открыть прямой канал с Диной; однако для этого потребовались бы ликвидность и внутрицепные комиссионные, которые могли бы

превысить стоимость самого доната. Вместо этого Алиса может использовать свои существующие открытые каналы, чтобы отправить донат Дине без необходимости открывать канал напрямую с Диной. Это возможно до тех пор, пока существует некий путь из каналов от Алисы к Дине с достаточной емкостью, чтобы промаршрутизировать донат.

Как вы видите на рис. 8-3, у Алисы есть открытый канал с Бобом, владельцем кофейни. У Боба, в свою очередь, есть открытый канал с разработчиком программного обеспечения Чаном, который помогает ему с системой торговых точек, которую он использует в своей кофейне. Чан также является владельцем крупной софтверной компании, которая разрабатывает игру, в которую играет Дина, и у них уже есть открытый канал, который Дина использует для оплаты лицензии на игру и внутриигровых предметов.

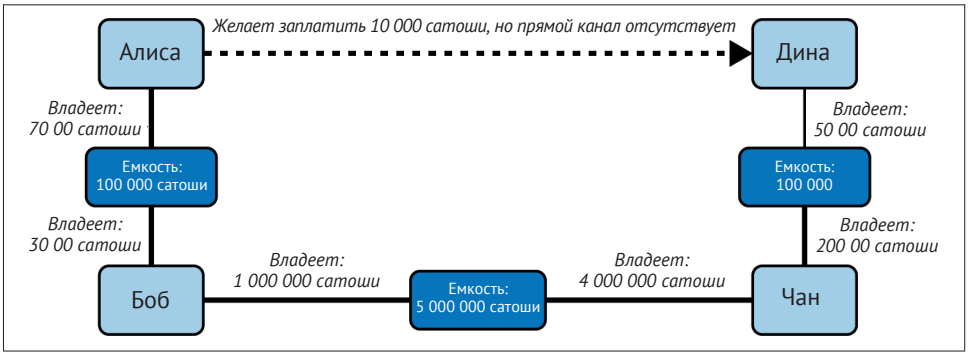


Рис. 8-3. Сеть платежных каналов между Алисой и Диной

Можно проследить путь от Алисы до Дины, в котором Боб и Чан используются в качестве промежуточных маршрутизационных узлов. Затем Алиса может проложить маршрут по этому намеченному пути и использовать его для отправки доната в размере нескольких тысяч сатоши Дине, при этом оплата будет перенаправлена Бобом и Чаном. По сути, Алиса заплатит Бобу, который заплатит Чану, который заплатит Дине. Прямой канал между Алисой и Диной не требуется.

Главная задача состоит в том, чтобы сделать это таким образом, чтобы Боб и Чан не смогли украсть деньги, которые Алиса хочет передать Дине.

Физический пример «маршрутизирования»

В целях понимания того, как сеть Lightning защищает платеж во время маршрутизации, мы можем сравнить это с примером маршрутизирования физических платежей золотыми монетами в реальном мире.

Допустим, Алиса хочет отдать 10 золотых монет Дине, но не имеет прямого доступа к Дине. Однако Алиса знает Боба, который знает Чана, который знает Дину, поэтому она решает попросить Боба и Чана о помощи. Это показано на рис. 8-4.

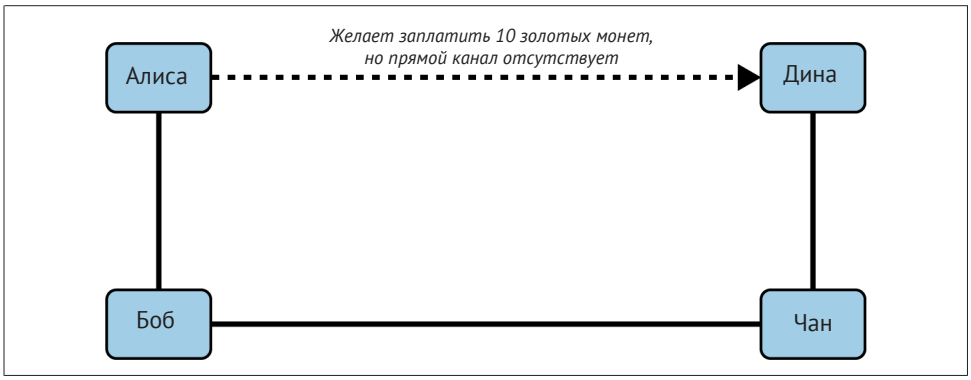


Рис. 8-4. Алиса желает заплатить Дине 10 золотых монет

Алиса может заплатить Бобу, чтобы тот заплатил Чану, чтобы тот заплатил Дине, но как она может быть уверена в том, что Боб или Чан не убегут с монетами после их получения? В физическом мире для безопасного проведения серии платежей можно было бы воспользоваться контрактами.

Алиса могла бы заключить контракт с Бобом, который гласит:

> *Я, Алиса, дам тебе, Боб, 10 золотых монет, если ты передашь их Чану.*

Хотя абстрактно этот контракт хорош, в реальном мире Алиса рискует, что Боб нарушит контракт и будет надеяться, что его не поймают. Даже если Боба поймают и привлекут к ответственности, Алиса рискует, что он обанкротится и не сможет вернуть ей 10 золотых монет. Если даже допустить, что эти проблемы волшебным образом были решены, все еще не ясно, как использовать такой контракт для достижения желаемого результата: доставки монет Дине.

Давайте улучшим наш контракт, чтобы учесть эти соображения:

> *Я, Алиса, возмещу тебе, Боб, твои 10 золотых монет, если ты сможешь доказать мне (например, с помощью квитанции), что ты доставил 10 золотых монет Чану.*

Вы можете спросить себя, почему Боб должен подписывать такой контракт. Он должен заплатить Чану, но в конечном счете ничего не получает от обмена, и он рискует, что Алиса не возместит ему ущерб. Боб мог бы предложить Чану аналогичный контракт, чтобы заплатить Дине, но точно так же у Чана тоже нет причин его принимать.

Даже если отбросить риск, у Боба и Чана уже должно быть 10 золотых монет для отправки; в противном случае они не смогли бы участвовать в контракте.

Таким образом, Боб и Чан сталкиваются как с риском, так и с альтернативными издержками, соглашаясь на этот контракт, и им потребуется компенсация, чтобы его принять.

Тогда Алиса может сделать его привлекательным как для Боба, так и для Чана, предложив им вознаграждение в размере одной золотой монеты каждому, если они передадут ее платеж Дине.

И тогда контракт будет гласить:

Я, Алиса, возмещу тебе, Боб, 12 золотых монет, если ты сможешь доказать мне (например, с помощью квитанции), что ты доставил 11 золотых монет Чану.

Алиса теперь обещает Бобу 12 золотых монет. Есть 10, которые нужно доставить Дине, и 2 для оплаты комиссионных. Она обещает 12 Бобу, если он сможет доказать, что переслал 11 Чану. Разница в одну золотую монету – это комиссионные, которые Боб заработает за помощь с этим конкретным платежом. На рис. 8-5 мы видим, как при таком раскладе Дина получит 10 золотых монет через Боба и Чана.

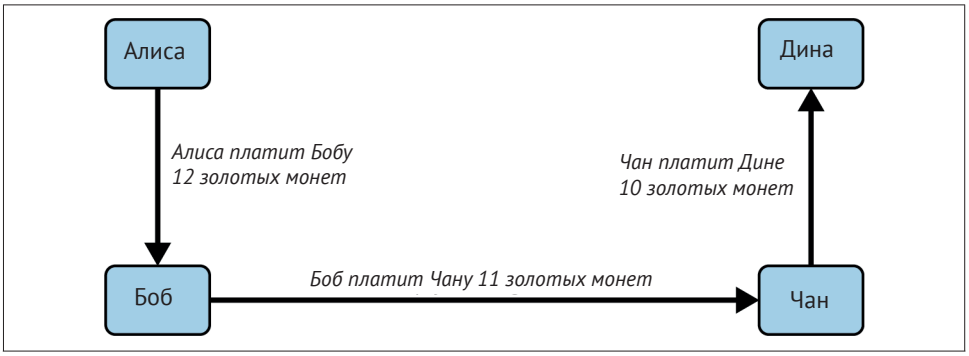


Рис. 8-5. Алиса платит Бобу, Боб платит Чану, Чан платит Дине

Поскольку все еще существует проблема доверия и риск того, что Алиса или Боб не выполнят контракт, все стороны решают воспользоваться службой условного депонирования. В начале обмена Алиса может «запереть» эти 12 золотых монет на условном депонировании, которые будут выплачены Бобу только после того, как он докажет, что заплатил 11 золотых монет Чану.

Указанная служба условного депонирования является идеализированной, не создающей других рисков (например, риска контрагента). Позже мы увидим, как заменить условное депонирование умным Bitcoin-контрактом. Давайте пока допустим, что этой службе условного депонирования все доверяют.

В сети Lightning квитанция (подтверждение платежа) может иметь форму секрета, который знает только Дина. На практике этот секрет будет случайным числом, которое достаточно велико, чтобы другие не могли его угадать (обычно это очень, очень большое число, закодированное с использованием 256 бит!).

Дина генерирует это секретное значение `R` с помощью генератора случайных чисел.

Затем секрет может быть передан контракту путем включения хеша SHA-256 секрета в сам контракт следующим образом:

H = SHA-256(R)

Мы называем этот хеш секрета платежа *платежным хешем*. Секрет, который «отопрет» платеж, называется *платежным секретом*.

На данный момент мы сохраняем простоту и исходим из того, что секрет Дины – это просто текстовая строка: секрет Дины. Это секретное сообщение называется *платежным секретом*, или *платежным прообразом*.

Для того чтобы «зафиксировать» этот секрет, Дина вычисляет хеш SHA-256, который при кодировании в шестнадцатеричном формате при выводе на экран может иметь следующий вид:

```
0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
```

Для того чтобы облегчить Алисе оплату, Дина создаст платежный секрет и платежный хеш и отправит Алисе платежный хеш. На рис. 8-6 мы видим, что Дина отправляет платежный хеш Алисе по какому-либо внешнему каналу (пунктирная линия), например по электронной почте или текстовым сообщением.

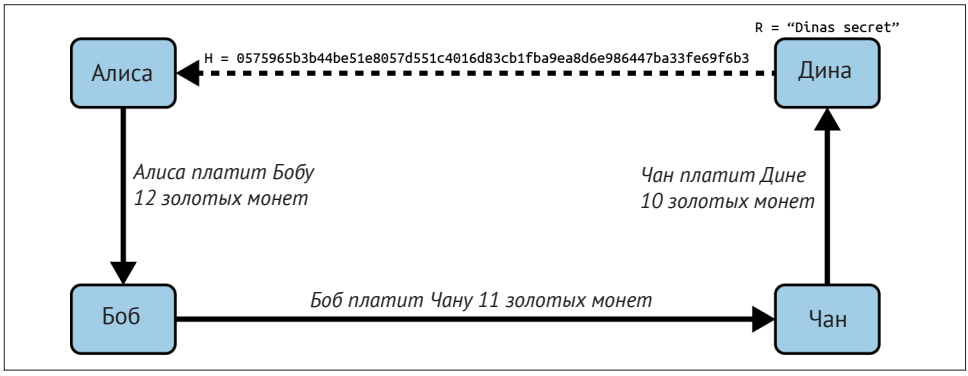


Рис. 8-6. Дина отправляет хешированный секрет Алисе

Алиса не знает секрета, но она может переписать свой контракт, чтобы использовать хеш секрета в качестве доказательства оплаты:

> *Я, Алиса, возмещу тебе, Боб, 12 золотых монет, если ты сможешь показать мне валидное сообщение, которое хешируется в:* `057596...`. *Ты можешь получить это сообщение, заключив аналогичный контракт с Чаном, который должен заключить аналогичный контракт с Диной. Для того чтобы заверить тебя в том, что тебе возместят убытки, я передам 12 золотых монет на доверенное депонирование до того, как ты заключишь свой следующий контракт.*

Этот новый контракт теперь защищает Алису от того, что Боб не перешлет деньги Чану, защищает Боба от того, что Алиса не возместит ему ущерб, и гарантирует, что будут доказательства того, что в конечном счете Дине заплатили посредством хеша секрета Дины.

После того как Боб и Алиса согласятся с контрактом и Боб получит сообщение от депозитария о том, что Алиса внесла 12 золотых монет, Боб теперь может заключить аналогичный контракт с Чаном.

Обратите внимание, что поскольку Боб берет комиссионные за услугу в размере 1 монеты, он перешлет Чану 11 золотых монет только после того, как Чан предъявит доказательства того, что он заплатил Дине. Точно так же Чан тоже потребует оплату и рассчитывает получить 11 золотых монет, как только он докажет, что заплатил Дине обещанные 10 золотых монет.

Контракт Боба с Чаном будет гласить:

> *Я, Боб, возмещу тебе, Чан, 11 золотых монет, если ты сможешь показать мне валидное сообщение, которое хешируется в:* `057596...` *Ты можешь получить это сообщение, заключив аналогичный контракт с Диной. Для того чтобы заверить тебя в том, что тебе возместят убытки, я передам 11 золотых монет на доверенное депонирование до того, как ты заключишь свой следующий контракт.*

После того как Чан получает сообщение от депозитария о том, что Боб внес 11 золотых монет, Чан заключает аналогичный контракт с Диной:

> *Я, Чан, возмещу тебе, Дина, 10 золотых монет, если ты сможешь показать мне валидное сообщение, которое хешируется в:* `057596...` *Для того чтобы заверить тебя в том, что ты получишь возмещение после раскрытия секрета, я передам 10 золотых монет на доверенное депонирование.*

Теперь все на своих местах. Алиса заключила контракт с Бобом и поместила 12 золотых монет на условное депонирование. У Боба есть контракт с Чаном, и он поместил 11 золотых монет на условное депонирование. У Чана есть контракт с Диной, и он поместил 10 золотых монет на условное депонирование. Теперь Дина должна раскрыть секрет, который является прообразом хеша, установленный ею в качестве доказательства оплаты.

Теперь Дина передает `секрет Дины` Чану.

Чан проверяет, что `секрет Дины` хешируется в `057596...`. У Чана теперь есть подтверждение оплаты, и поэтому он поручает службе условного депонирования передать 10 золотых монет Дине.

Теперь Чан раскрывает секрет Бобу. Боб его проверяет и дает указание службе условного депонирования передать 11 золотых монет Чану.

Боб теперь раскрывает секрет Алисе. Алиса его проверяет и дает указание условному депонированию выдать Бобу 12 золотых монет.

Все контракты уже заключены. Алиса заплатила в общей сложности 12 золотых монет, 1 из которых был получен Бобом, 1 из которых был получен Чаном и 10 из которых были получены Диной. Имея такую цепочку контрактов, Боб и Чан не смогли бы сбежать с деньгами, потому что сначала они депонировали их на условное депонирование.

Однако один вопрос все еще остается нерешенным. Если бы Дина отказалась обнародовать свой секретный прообраз, то у Чана, Боба и Алисы все их монеты застряли бы на условном депонировании, но не были бы возвращены. И точно так же, если бы кто-то еще по цепочке не смог передать секрет, то произошло бы то же самое. Таким образом, хотя никто не может украсть деньги у Алисы, у всех все равно их деньги застрянут на условном депонировании навсегда.

К счастью, это можно решить, добавив в контракт крайний срок.

Мы могли бы внести изменения в контракт таким образом, что если он не был бы выполнен к определенному сроку, то срок действия контракта истекал, и служба условного депонирования возвращала бы деньги лицу, внесшему первоначальный депозит. Мы называем этот крайний срок *привязкой ко времени*.

Депозит запирается службой условного депонирования на определенный период времени и в конечном итоге освобождается, даже если не было предоставлено никаких доказательств оплаты.

Для того чтобы это учесть, контракт между Алисой и Бобом снова дополнен новым пунктом:

> *У Боба есть 24 часа, чтобы раскрыть секрет после подписания контракта. Если Боб не предоставит секрет к этому времени, то депозит Алисы будет возвращен службой условного депонирования, и контракт станет недействительным.*

Боб, конечно, теперь должен обеспечить, чтобы он получил подтверждение оплаты в течение 24 часов. Даже если он успешно оплатит Чану, если он получит подтверждение оплаты позже, чем через 24 часа, ему не будут возвращены его средства. В целях устранения этого риска Боб должен дать Чану еще более короткий срок.

В свою очередь, Боб изменит свой контракт с Чаном следующим образом:

> *У Чана есть 22 часа, чтобы раскрыть секрет после подписания контракта. Если он не предоставит секрет к этому времени, то депозит Боба будет возвращен службой условного депонирования, и контракт станет недействительным.*

Как вы уже могли догадаться, Чан тоже изменит свой контракт с Диной:

> *У Дины есть 20 часов, чтобы раскрыть секрет после подписания контракта. Если она не предоставит секрет к этому времени, то депозит Чана будет возвращен службой условного депонирования, и контракт станет недействительным.*

С помощью такой цепочки контрактов мы можем обеспечить, чтобы через 24 часа платеж успешно перешел от Алисы к Бобу, от Чана к Дине, либо он завершится безуспешно, и все получат свои средства назад. Контракт завершится либо безуспешно, либо успешно, середины нет.

В контексте сети Lightning мы называем это свойство *атомарностью* «все или ничего».

До тех пор, пока условное депонирование заслуживает доверия и добросовестно выполняет свои обязанности, ни у одной из сторон в этом процессе их монеты украдены не будут.

Предпосылкой для того, чтобы этот маршрут вообще работал, является то, что у всех участников пути достаточно денег, чтобы выполнить требуемую серию депозитов.

Хотя это кажется незначительной деталью, позже в этой главе мы увидим, что данное требование на самом деле является одной из наиболее сложных проблем для узлов LN. Оно становится все труднее по мере увеличения размера платежа. Кроме того, стороны не могут использовать свои деньги, пока они заперты на условном депонировании.

Таким образом, пользователи, которые пересылают платежи, сталкиваются с альтернативными издержками связывания денег, которые в конечном итоге возмещаются за счет комиссионных за маршрутизацию, как мы видели в предыдущем примере.

Теперь, когда мы рассмотрели пример маршрутизирования физических платежей, мы увидим, как это может быть имплементировано в блочной цепи Bitcoin без какой-либо необходимости в стороннем условном депонировании.

Для этого мы будем настраивать контракты между участниками с помощью Bitcoin Script. Мы заменяем стороннее условное депонирование умными контрактами, которые имплементируют протокол справедливости. Давайте разберем эту концепцию и имплементируем ее!

Протокол справедливости

Как мы видели в первой главе этой книги, инновация системы Bitcoin заключается в способности использовать криптографические примитивы для имплементирования протокола справедливости, который заменяет доверие к третьим сторонам (посредникам) доверенным протоколом.

В нашем примере с золотой монетой нам понадобилась служба условного депонирования, чтобы ни одна из сторон не нарушила свои обязательства. Инновация протоколов криптографической справедливости позволяет нам заменить службу условного депонирования протоколом.

Свойства протокола справедливости, который мы хотим создать, таковы:

Бездоверительная работа

Участникам в маршрутизированном платеже не нужно доверять друг другу, или какому-либо посреднику, или третьей стороне. Вместо этого они доверяют протоколу, который защищает их от обмана.

Атомарность

Либо платеж будет полностью выполнен, либо он завершится безуспешно, и все вернут свои средства назад. Посредник не может забрать маршрутизированный платеж и не перенаправить его на следующий переходный узел. Таким образом, посредники не могут обманывать или красть.

Многопереходность

Безопасность системы распространяется от начала до конца для платежей, проходящих по нескольким платежным каналам, равно как и для платежей между двумя концами одного платежного канала.

Опциональным дополнительным свойством является возможность разделения платежей на несколько частей при сохранении атомарности всего платежа. Они называются многокомпонентными платежами (MPP) и подробно рассматриваются в разделе «Многокомпонентные платежи» на стр. 314.

Имплементирование атомарных бездоверительных многопереходных платежей

Bitcoin Script достаточно гибок, чтобы существовали десятки способов имплементирования протокола справедливости, обладающего свойствами атомарности, бездоверительной работы и многопереходной безопасности. Выбор конкретной имплементации зависит от определенных компромиссов между конфиденциальностью, эффективностью и сложностью.

Протокол справедливости для маршрутизирования, используемый сегодня в сети Lightning, называется *контрактом с привязкой к хешу и времени* (HTLC). В HTLC-контрактах в качестве секрета используется хеш-прообраз, который от-

пирает платеж, как мы видели в примере с золотой монетой в этой главе. Получатель платежа генерирует случайное секретное число и вычисляет его хеш. Хеш становится условием оплаты, и как только секрет будет раскрыт, все участники смогут погасить свои входящие платежи. HTLC-контракты обеспечивают атомарность, бездоверительную работу и многопереходную безопасность.

Еще одним предлагаемым механизмом имплементирования маршрутизации является *контракт с точечной привязкой ко времени* (Point Time-Locked Contract, аббр. PTLC). PTLC-контракты также обеспечивают атомарность, бездоверительную работу и многопереходную безопасность, но делают это с повышенной эффективностью и лучшей конфиденциальностью. Эффективная имплементация PTLC-контрактов зависит от нового алгоритма цифровой подписи, именуемого *подписями Шнорра*, который был запланирован к принятию в конце 2021 года.

Возвращаясь к примеру с донатами

Давайте вернемся к нашему примеру из первой части этой главы. Алиса хочет дать Дине донат в виде платежа Lightning. Допустим, Алиса хочет послать Дине 50 000 сатоши в качестве доната.

В целях выполнения Алисой платежа Дине Алисе понадобится узел Дины, чтобы сгенерировать счет Lightning. Мы обсудим это подробнее в главе 15. На данный момент давайте допустим, что у Дины есть веб-сайт, который может выставлять счет Lightning для донатов.

Платежи Lightning можно отправлять без счета, используя функциональную возможность под названием keysend, которую мы подробнее обсудим в разделе «Спонтанные платежи keysend» на стр. 279. На данный момент мы объясним более простой процесс оплаты с использованием счета.

Алиса посещает веб-сайт Дины, вводит сумму в размере 50 000 сатоши в форму, и в ответ узел Lightning Дины генерирует платежный запрос на сумму 50 000 сатоши в виде счета Lightning. Это взаимодействие происходит через интернет и за пределами сети Lightning, как показано на рис. 8-7.

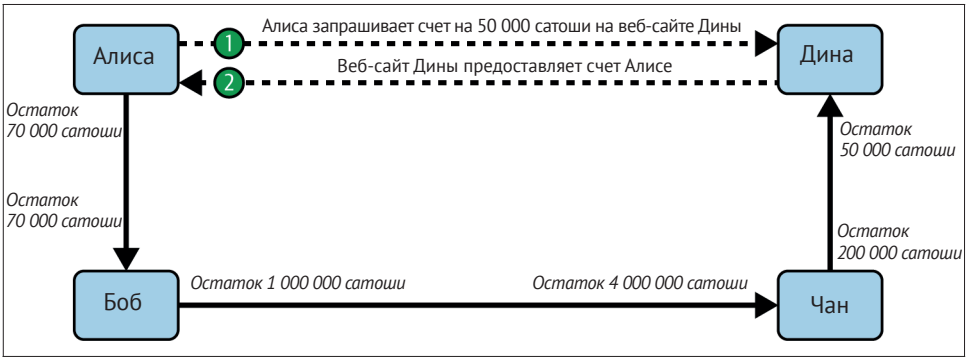


Рис. 8-7. Алиса запрашивает счет на веб-сайте Дины

Как мы видели в предыдущих примерах, мы исходим из того, что у Алисы нет прямого платежного канала с Диной. Вместо этого у Алисы есть канал с Бобом, у Боба есть канал с Чаном и у Чана есть канал с Диной. Для того чтобы заплатить Дине, Алиса должна найти путь, который соединит ее с Диной. Мы обсудим этот шаг подробнее в главе 12. На данный момент давайте допустим, что Алиса способна собирать информацию о доступных каналах и видит, что есть путь от нее к Дине через Боба и Чана.

Помните, как Боб и Чан ожидали небольшой компенсации за маршрутизацию платежа через свои узлы? Алиса хочет заплатить Дине 50 000 сатоши, но, как вы увидите в следующих разделах, она отправит Бобу 50 200 сатоши. Дополнительные 200 сатоши уйдут Бобу и Чану по 100 сатоши каждому в качестве комиссионных за маршрутизацию.

Теперь узел Алисы может создать платеж Lightning. В следующих нескольких разделах мы увидим, как узел Алисы создает HTLC-контракт для оплаты Дине и как этот HTLC-контракт пересылается по пути от Алисы к Дине.

Внутрицепное и внецепное улаживание HTLC-контрактов

Предназначение сети Lightning состоит в том, чтобы активировать внецепные транзакции, которым доверяют точно так же, как внутрицепным транзакциям, потому что никто не может обмануть. Причина, по которой никто не может обмануть, заключается в том, что в любой момент любой из участников может перенести свои внецепные транзакции во внутрицепную среду. Каждая внецепная транзакция готова к отправке в блочную цепь Bitcoin в любое время. Таким образом, блочная цепь Bitcoin действует как механизм разрешения споров и окончательного урегулирования, если это необходимо.

Сам факт того, что любая транзакция может быть перенесена во внутрицепную среду в любое время, как раз и является причиной того, что все эти транзакции могут храниться вне цепи. Если вы знаете, что у вас есть возможность обратиться за помощью, то вы можете продолжать сотрудничать с другими участниками и избегать необходимости внутрицепных расчетов и дополнительных комиссионных.

Во всех последующих примерах мы будем исходить из того, что любая из этих транзакций может быть совершена внутри цепи в любое время. Участники предпочтут оставлять их вне цепи, но в функциональности системы нет никакой разницы, кроме более высоких комиссионных и задержек, связанных с внутрицепной добычей транзакций. Пример работает одинаково, если все транзакции являются внутрицепными или внецепными.

Контракты с привязкой к хешу и времени

В этом разделе мы объясняем, как работают HTLC-контракты.

Первая часть HTLC-контракта – это хеш. Он относится к использованию криптографического алгоритма хеширования для привязки к случайно сгенерированному секрету. Знание секрета позволяет погасить платеж. Криптографическая хеш-функция гарантирует, что хотя никто не сможет угадать секрет-

ный прообраз, любому легко проверить хеш, и есть только один возможный прообраз, который выносит решение по условию оплаты.

На рис. 8-8 мы видим, как Алиса получает счет Lightning от Дины. Внутри этого счета Дина закодировала платежный хеш, который является криптографическим хешем секрета, произведенного узлом Дины. Секрет Дины называется *платежным прообразом*. Платежный хеш действует как идентификатор, который может использоваться для маршрутизирования платежа к Дине. Платежный прообраз действует как квитанция и подтверждение платежа после завершения платежа.

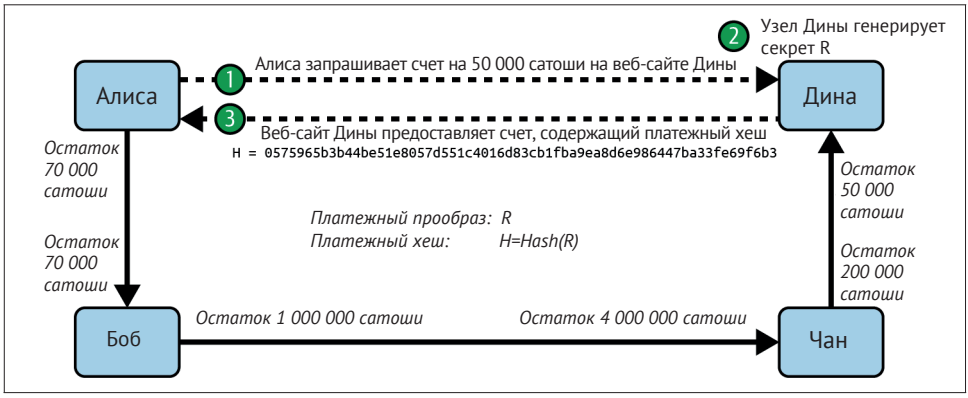


Рис. 8-8. Алиса получает платежный хеш от Дины

В сети Lightning платежным прообразом Дины будет не фраза типа «секрет Дины», а случайное число, сгенерированное узлом Дины. Давайте назовем его случайным числом R.

Узел Дины вычислит криптографический хеш R, такой что

$$H = \text{SHA-256}(R)$$

В этом уравнении H – это хеш, или платежный хеш, а R – секрет, или платежный прообраз.

Использование криптографической хеш-функции является одним из элементов, гарантирующих бездоверительную работу. Платежным посредникам не нужно доверять друг другу, потому что они знают, что никто не сможет разгадать секрет или его подделать.

HTLC-контракты на Bitcoin Script

В нашем примере с золотой монетой у Алисы был контракт, заключенный с помощью условного депонирования следующим образом:

> *Алиса возместит Бобу 12 золотых монет, если ты сможешь показать валидное сообщение, которое хеширует в:* `0575...f6b3`. *У Боба есть 24 часа, чтобы раскрыть секрет после подписания контракта. Если Боб не предоставит секрет к этому времени, то депозит Алисы будет возвращен службой условного депонирования, и контракт станет недействительным.*

Давайте посмотрим, как мы могли бы имплементировать его как HTLC-контракт на языке Bitcoin Script. В примере 8-1 мы видим Bitcoin Script для HTLC-контракта. Указанный скрипт в настоящее время используется в сети Lightning. Его определение можно найти в спецификации «BOLT #3: транзакции»[64].

Пример 8-1. HTLC-контракт, имплементированный на Bitcoin Script (BOLT #3)

```
# На дистанционный узел с отзывным ключом
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
    OP_CHECKSIG
OP_ELSE
    <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_IF
        # На локальный узел посредством HTLC-успешной транзакции.
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # На дистанционный узел после тайм-аута.
        OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

Да уж, выглядит хитроумно! Однако не волнуйтесь, мы будем идти пошагово и упрощать.

Язык Bitcoin Script, используемый в настоящее время в сети Lightning, довольно сложен, поскольку он оптимизирован для повышения эффективности использования сетевого пространства, что делает его очень компактным, но трудным для чтения.

В следующих далее разделах мы сосредоточимся на главных элементах скрипта и представим упрощенные скрипты, которые немного отличаются от того, что фактически используется в Lightning.

Основная часть HTLC-контракта находится в строке 10 примера 8-1. Давайте соберем его с нуля!

Платежный прообраз и верификация хеша

Ядром HTLC-контракта является хеш, по которому может быть произведена оплата, если получатель знает платежный прообраз. Алиса привязывает платеж к определенному платежному хешу, и Боб должен представить платежный прообраз, чтобы получить средства. Система Bitcoin может верифицировать правильность платежного прообраза Боба, хешируя его и сравнивая результат с платежным хешем, который Алиса использовала для привязки средств.

Эта часть HTLC-контракта может быть имплементирована на Bitcoin Script следующим образом:

```
OP_SHA256 <H> OP_EQUAL
```

[64] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs.

Алиса может создать выход транзакции, который выплачивает 50 200 сатоши с помощью приведенного выше привязывающего скрипта, заменив <H> хеш-значением 0575...f6b3, предоставленным Диной. Затем Алиса может подписать эту транзакцию и предложить ее Бобу:

```
OP_SHA256 0575...f6b3 OP_EQUAL
```

Боб не может потратить этот HTLC-контракт до тех пор, пока он не узнает секрет Дины, поэтому расходование HTLC-контракта зависит от выполнения Бобом платежа на всем пути до Дины.

После того как Боб получит секрет Дины, Боб сможет использовать этот выход с помощью отвязывающего скрипта, содержащего секретное значение прообраза R.

Отвязывающий скрипт в сочетании с привязывающим скриптом к:

```
<R> OP_SHA256 <H> OP_EQUAL
```

Механизм Bitcoin Script будет оценивать этот скрипт следующим образом:

1. R помещается в стек.
2. Оператор OP_SHA256 извлекает значение R из стека и хеширует его, помещая результат H_R в стек.
3. H помещается в стек.
4. Оператор OP_EQUAL сравнивает H и H_R. Если они равны, то результат равен TRUE, скрипт завершен, платеж подтвержден.

Распространение HTLC-контрактов от Алисы к Дины

Теперь Алиса распространит HTLC-контракт по сети так, чтобы он достиг Дины.

На рис. 8-9 мы видим, как HTLC-контракт распространяется по сети от Алисы к Дине. Алиса дала Бобу HTLC-контракт на 50 200 сатоши. Боб теперь может создать HTLC-контракт на 50 100 сатоши и передать его Чану.

Боб знает, что Чан не может погасить HTLC-контракт Боба без широковещательной передачи секрета, после чего Боб также может использовать секрет, чтобы погасить HTLC-контракт Алисы. Это действительно важный момент, потому что он обеспечивает сквозную атомарность HTLC-контракта. Для того чтобы потратить HTLC-контракт, нужно раскрыть секрет, который затем позволит другим также потратить и свой HTLC-контракт. Либо все HTLC-контракты могут быть израсходованы, либо ни один из HTLC-контрактов не может быть израсходован: атомарность!

Поскольку HTLC-контракт Алисы на 100 сатоши больше, чем HTLC-контракт, который Боб дал Чану, Боб заработает 100 сатоши в качестве комиссионных за маршрутизацию, если этот платеж завершится.

Боб не рискует и не доверяет ни Алисе, ни Чану. Вместо этого Боб верит, что подписанная транзакция вместе с секретом будет доступна для обмена в блочной цепи Bitcoin.

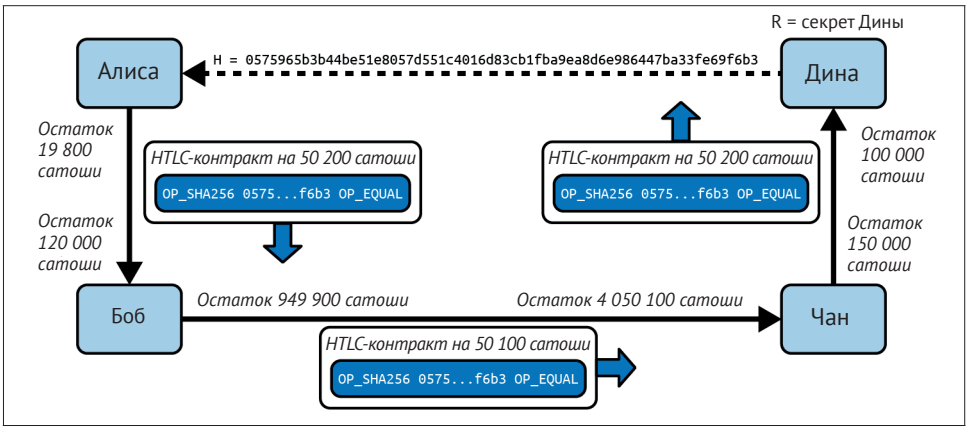


Рис. 8-9. Распространение HTLC-контракта по сети

Аналогичным образом Чан может предоставить HTLC-контракт на 50 000 Дине. Он ничем не рискует и не доверяет ни Бобу, ни Дине. Для того чтобы погасить HTLC-контракт, Дине пришлось бы передать секрет, который Чан мог бы использовать для погашения HTLC-контракта Боба. Чан также заработал бы 100 сатоши в качестве комиссионных за маршрутизацию.

Обратное распространение секрета

После того как Дина получит HTLC-контракт на 50 000 от Чана, она сможет получить деньги. Дина может просто передать этот HTLC-контракт внутрь цепи и потратить его, раскрыв секрет в транзакции расходования. Либо вместо этого Дина может обновить остаток канала с Чаном, открыв ему секрет. Нет никаких причин взимать комиссию за транзакцию и переходить внутрь цепи. Таким образом, вместо этого Дина отправляет секрет Чану, и они соглашаются обновить остаток своих каналов, чтобы отразить платеж Lightning в размере 50 000 сатоши Дине. На рис. 8-10 мы видим, как Дина передает секрет Чану, тем самым выполняя HTLC-контракт.

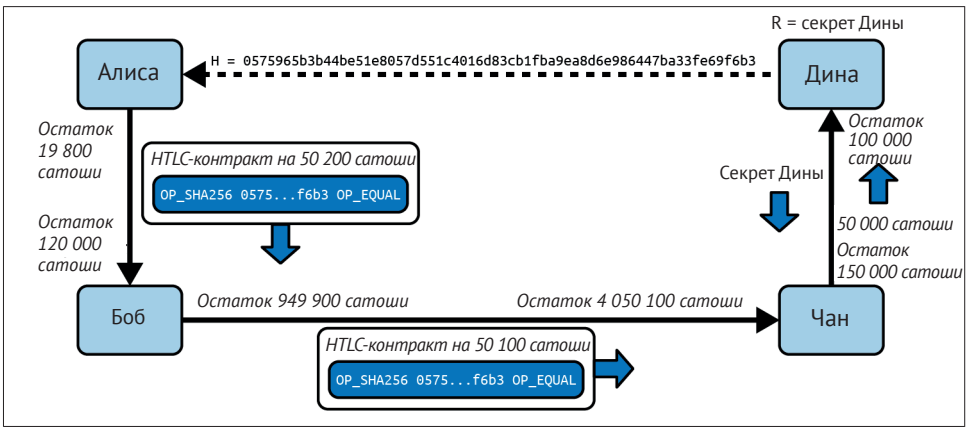


Рис. 8-10. Дина улаживает HTLC-контракт Чана вне цепи

Обратите внимание, что остаток канала Дины увеличивается с 50 000 сатоши до 100 000 сатоши. Остаток канала Чана уменьшается с 200 000 сатоши до 150 000 сатоши. Емкость канала не изменилась, но 50 000 перешли со стороны канала Чана на сторону канала Дины.

Теперь Чан владеет секретом и заплатил Дине 50 000 сатоши. Он может сделать это без какого-либо риска, потому что секрет позволяет Чану погасить HTLC-контракт на 50 100 с Бобом. У Чана есть возможность зафиксировать этот HTLC-контракт внутри цепи и потратить его, раскрыв секрет в блочной цепи Bitcoin. Но, как и Дина, он предпочел бы избежать комиссионных за транзакции. И вместо этого он отправляет секрет Бобу, чтобы они могли обновить остаток своих каналов и отразить платеж Lightning в размере 50 100 сатоши от Боба к Чану. На рис. 8-11 мы видим, как Чан отправляет секрет Бобу и взамен получает платеж.

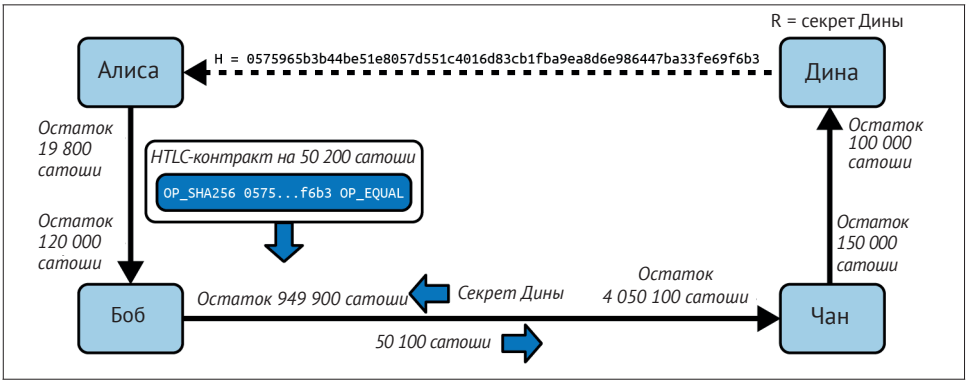


Рис. 8-11. Чан улаживает HTLC Боба вне цепи

Чан заплатил Дине 50 000 сатоши и получил 50 100 сатоши от Боба. Таким образом, у Чана на остатке канала на 100 сатоши больше, которые он заработал в качестве комиссионных за маршрутизацию.

У Боба теперь тоже есть секрет. Он может им воспользоваться, чтобы потратить HTLC-контракт Алисы внутри цепи. Либо он может избежать комиссионных за транзакции, уладив HTLC-контракт в канале с Алисой. На рис. 8-12 мы видим, что Боб отправляет секрет Алисе, и они обновляют остаток канала, чтобы отразить платеж Lightning в размере 50 200 сатоши от Алисы Бобу.

Боб получил 50 200 сатоши от Алисы и заплатил 50 100 сатоши Чану, в результате чего у него есть дополнительные 100 сатоши на остатке канала от комиссионных за маршрутизацию.

Алиса получает секрет и уладила HTLC-контракт в размере 50 200 сатоши. Секрет может быть использован в качестве квитанции, чтобы доказать, что Дине заплатили за этот конкретный платежный хеш.

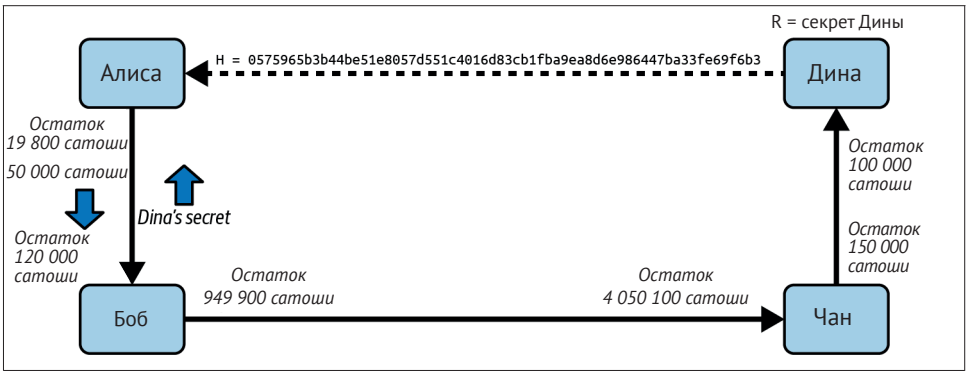


Рис. 8-12. Боб улаживает HTLC-контракт Алисы вне цепи

Итоговые остатки каналов отражают платеж Алисы Дине и комиссионные за маршрутизацию, уплаченную при каждом переходе, как показано на рис. 8-13.

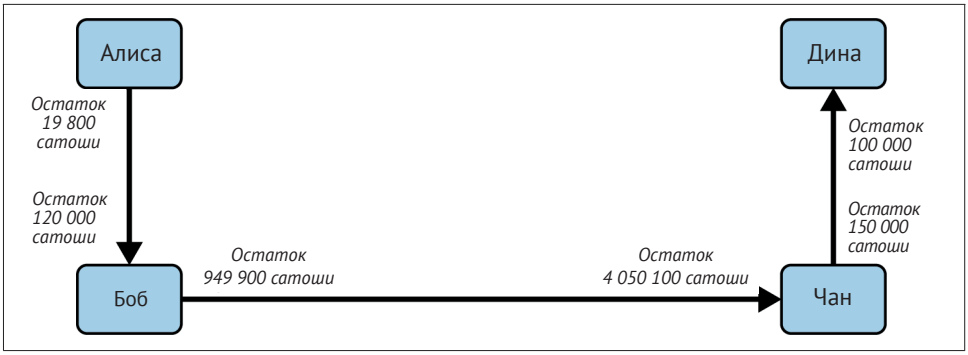


Рис. 8-13. Остаток канала после оплаты

Привязка подписи: предотвращение кражи HTLC-контрактов

Тут есть одна загвоздка. Вы ее заметили?

Если Алиса, Боб и Чан создадут HTLC-контракты, как показано на рис. 8-13, то они столкнутся с маленьким, но не незначительным риском потери. Любой из этих HTLC-контрактов может быть погашен (потрачен) любым, кто знает секрет. Сначала только Дина знает секрет. Предполагается, что Дина будет тратить HTLC-контракт только у Чана. Но Дина могла потратить все три HTLC-контракта одновременно или даже за одну транзакцию расходования! В конце концов, Дина знает секрет раньше всех остальных. Точно так же, после того как Чан узнает секрет, он должен потратить только тот HTLC-контракт, который был предложен Бобом. Но что, если Чан тоже потратит предложенный Алисой HTLC-контракт?

Все это имеет бездоверительный характер! И приводит к сбою самой важной особенности безопасности. Нам нужно это исправить.

Скрипт HTLC-контракта должен иметь дополнительное условие, которое привязывает каждый HTLC-контракт к определенному получателю. Мы делаем это, требуя цифровую подпись, соответствующую публичному ключу каж-

дого получателя, тем самым не позволяя никому другому тратить этот HTLC-контракт. Поскольку только назначенный получатель имеет возможность создать цифровую подпись, соответствующую этому публичному ключу, только назначенный получатель сможет потратить этот HTLC-контракт.

Давайте еще раз посмотрим на скрипты с учетом этой модификации. HTLC-контракт Алисы для Боба изменен, чтобы включить публичный ключ Боба и оператор OP_CHECKSIG.

Вот модифицированный скрипт HTLC-контракта:

```
OP_SHA256 <H> OP_EQUALVERIFY <Публ ключ Боба> OP_CHECKSIG
```

Обратите внимание, что мы также поменяли OP_EQUAL на OP_EQUALVERIFY. Когда оператор имеет суффикс VERIFY, он не возвращает TRUE или FALSE в стеке. Вместо этого он останавливает исполнение и завершает скрипт с ошибкой, если результат ложен, и продолжается без какого-либо выхода из стека, если он истинен.

В целях погашения этого HTLC-контракта Боб должен представить отвязывающий скрипт, который включает в свой состав подпись из приватного ключа Боба, а также секретный платежный прообраз, например:

```
<Подпись Боба> <R>
```

Отвязывающий и привязывающий скрипты комбинируются и оцениваются скриптовым механизмом следующим образом:

```
<Подпись Боба> <R> OP_SHA256 <H> OP_EQUALVERIFY <Публ ключ Боба> OP_CHECKSIG
```

1. <Подпись Боба> помещается в стек.
2. R помещается в стек.
3. OP_SHA256 выталкивает и хеширует R из вершины стека и помещает H_R в стек.
4. H помещается в стек.
5. OP_EQUALVERIFY проверяет значения H и H_R и их сравнивает. Если они не совпадают, то исполнение останавливается. В противном случае мы продолжим без выхода из стека.
6. Ключ <Публ ключ Боба> помещается в стек.
7. OP_CHECKSIG выводит <Подпись Боба> и <Публ ключ Боба> и верифицирует подпись. Результат (TRUE/FALSE) помещается в стек.

Как вы видите, это немного сложнее, но теперь мы исправили HTLC-контракт и обеспечили, чтобы только намеченный получатель мог его расходовать.

Оптимизация хеша

Давайте посмотрим на первую часть скрипта HTLC-контракта, который мы имеем на данный момент:

```
OP_SHA256 <H> OP_EQUALVERIFY
```

Если мы посмотрим на нее в предыдущем символическом представлении, то, похоже, операторы OP_ занимают больше всего места. Но это не так. Bitcoin Script кодируется в двоичном формате, причем каждый оператор представля-

ет собой один байт. Между тем значение <H>, которое мы используем в качестве местозаполнителя для платежного хеша, представляет собой 32-байтовое (256-битовое) значение. Список всех операторов Bitcoin Script и их двоичную и шестнадцатеричную кодировку можно найти в wiki-учебнике по Bitcoin Script[65] или в приложении D «Операторы, константы и символы транзакционного скриптового языка» в книге «Освоение системы Bitcoin»[66].

Представленный в шестнадцатеричном формате, наш скрипт HTLC-контракта будет выглядеть следующим образом:

```
a8 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3 88
```

В шестнадцатеричной кодировке OP_SHA256 равно a8, а OP_EQUALVERIFY – 88. Общая длина этого скрипта составляет 34 байта, из которых 32 байта являются хешем.

Как мы уже упоминали ранее, любой участник сети Lightning должен иметь возможность брать удерживаемую им транзакцию вне цепи и включать ее внутрь цепи, если ему необходимо обеспечить исполнение своих требований к средствам. Для перенесения транзакции внутрь цепи им пришлось бы платить майнерам комиссионные за транзакцию, и эти комиссионные пропорциональны размеру транзакции в байтах.

Поэтому мы хотим найти способы минимизировать «вес» транзакций внутри цепи, максимально оптимизировав скрипт. Один из способов состоит в том, чтобы добавить еще одну хеш-функцию поверх алгоритма SHA-256, которая создает меньшие хеши. Язык Bitcoin Script предоставляет оператор OP_HASH160, который «хеширует» прообраз дважды: сначала прообраз хешируется с помощью SHA-256, а затем результирующий хеш снова хешируется с помощью алгоритма хеширования RIPEMD160. Хеш, полученный в результате работы RIPEMD160, составляет 160 бит или 20 байт – гораздо компактнее. В Bitcoin Script это очень распространенная оптимизация, которая используется во многих распространенных форматах адресов.

Итак, давайте вместо этого воспользуемся указанной оптимизацией. Наш хеш SHA-256 равен 057596...69f6b3. Пропустив его через еще один раунд хеширования с помощью RIPEMD160, мы получим результат:

```
R = "Секрет Дины"
H256 = SHA256(R)
H256 = 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
H160 = RIPEMD160(H256)
H160 = 9e017f6767971ed7cea17f98528d5f5c0ccb2c71
```

Алиса может вычислить хеш RIPEMD160 платежного хеша, который предоставляет Дина, и использовать более короткий хеш в своем HTLC-контракте, как и Боб и Чан!

«Оптимизированный» скрипт HTLC-контракта будет выглядеть следующим образом:

```
OP_HASH160 <H160> OP_EQUALVERIFY
```

[65] См. https://en.bitcoin.it/wiki/Script.

[66] См. https://github.com/bitcoinbook/bitcoinbook/blob/develop/appdx-scriptops.asciidoc.

Закодированный в шестнадцатеричном формате, он составит:

```
a9 9e017f6767971ed7cea17f98528d5f5c0ccb2c71 88
```

где OP_HASH160 равен a9, а OP_EQUALVERIFY равен 88. Этот скрипт имеет длину всего 22 байта! Мы сэкономили 12 байт от каждой транзакции, которая погашает HTLC-контракт внутри цепи.

Благодаря этой оптимизации вы теперь видите, как мы приходим к скрипту HTLC-контракта, показанному в строке 10 примера 8-1:

```
...
    # На локальный узел посредством HTLC-успешной транзакции.
    OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY...
```

Кооперативный отказ и отказ тайм-аута по HTLC-контракту

До сих пор мы рассматривали «хеш-часть» HTLC-контракта и то, как она будет работать, если все будут сотрудничать и будут находиться онлайн во время оплаты.

Что произойдет, если кто-то выйдет в офлайн или откажется сотрудничать? Что произойдет, если платеж не удастся выполнить?

Нам нужно обеспечить способ «изящного отказа», потому что случайные отказы маршрутизации неизбежны. Есть два способа испытать отказ: совместно и с привязанным ко времени возвратом средств.

Совместный отказ относительно прост: HTLC-контракт разматывается каждым участником маршрута, удаляя выход HTLC-контракта из своих фиксационных транзакций без изменения остатка. Мы подробно рассмотрим принцип его работы в главе 9.

Давайте посмотрим, как обратить HTLC-контракт без сотрудничества одного или нескольких участников. Нам нужно обеспечить, чтобы, если один из участников не будет сотрудничать, средства не были просто заблокированы в HTLC-контракте *навсегда*. Это дало бы кому-то возможность получить выкуп за средства другого участника: «Я оставлю ваши средства связанными навсегда, если вы не заплатите мне выкуп».

Во избежание этой ситуации каждый скрипт HTLC-контракта включает в состав условие возврата средств, которое имеет отношение к привязке ко времени. Помните наш первоначальный договор условного депонирования? «У Боба есть 24 часа, чтобы раскрыть секрет после подписания контракта. Если Боб не предоставит секрет к этому времени, то депозит Алисы будет возвращен».

Привязанный ко времени возврат средств является важной частью скрипта, который обеспечивает атомарность, вследствие чего весь сквозной платеж либо завершается успешно, либо завершается с ошибкой. Нет никакого состояния «оплаченности наполовину», о котором стоило бы беспокоиться. В случае отказа каждый участник может либо размотать HTLC-контракт совместно со своим партнером по каналу, либо вложить транзакцию привязанного ко времени возврата средств внутрь цепи в одностороннем порядке, чтобы вернуть свои деньги.

В целях имплементирования этого возврата на Bitcoin Script мы используем специальный оператор OP_CHECKLOCKTIMEVERIFY, также сокращенно обозначаемый как OP_CLTV. Вот скрипт, как продемонстрировано ранее в строке 13 примера 8-1:

```
...
    OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
    OP_CHECKSIG
...
```

Оператор OP_CLTV принимает время истечения срока, определенное как высота блока, после которого эта транзакция становится валидной. Если транзакционная привязка ко времени не установлена так же, как <cltv_expiry>, то оценивание скрипта завершается ошибкой, и транзакция является невалидной. В противном случае скрипт продолжается без какого-либо выхода из стека. Помните, что суффикс VERIFY означает, что этот оператор не выводит TRUE или FALSE, а вместо этого либо останавливается/завершается с ошибкой, либо продолжается без выхода из стека.

По сути, OP_CLTV действует как «привратник», предотвращающий дальнейшее исполнение скрипта, если высота блока <cltv_expiry> не была достигнута в блочной цепи Bitcoin.

Оператор OP_DROP просто отбрасывает самый верхний элемент в скриптовом стеке. Это необходимо в начале из-за наличия «оставшегося» элемента из предыдущих строк скрипта. После OP_CLTV необходимо удалить элемент <cltv_expiry> из вершины стека, потому что в нем больше нет необходимости.

Наконец, после очистки стека должны остаться публичный ключ и сигнатура, которые OP_CHECKSIG может проверить. Как мы видели в разделе «Привязка подписи: предотвращение кражи HTLC-контрактов» на стр. 222, это необходимо для обеспечения того, чтобы только законный владелец средств мог на них претендовать, привязывая этот выход к своему публичному ключу и требуя подписи.

Декрементирование привязок ко времени

По мере того как HTLC-контракты распространяются от Алисы до Дины, условие о привязанном ко времени возврате средств в каждом HTLC-контракте имеет разное значение cltv_expiry. Мы рассмотрим это подробнее в главе 10. Но сейчас достаточно сказать, что для обеспечения упорядоченного разматывания безуспешного платежа каждый переход должен ждать возврата средств немного меньше. Разница между привязками ко времени для каждого перехода называется cltv_expiry_delta и устанавливается каждым узлом и рекламируется им в сети, как мы увидим в главе 11.

Например, Алиса настраивает привязку возврата средств ко времени в первом HTLC-контракте на высоту блока, равную текущему + 500 блоков («текущий» – это текущая высота блока). Затем Боб настроил бы привязку ко времени cltv_expiry в HTLC-контракте с Чаном на текущий + 450 блоков. Чан настроил бы привязку ко времени на текущий + 400 блоков от текущей высоты блока. Таким образом, Чан может получить назад свои средства по HTLC-контракту, который он предложил Дине, *до* того, как Боб получит назад свои средства по HTLC-контракту, который он предложил Чану. Боб может получить назад свои средства по HTLC-контракту, который он предложил Чану, прежде чем Алиса сможет получить назад свои средства по HTLC-контракту, который она предложила Бобу. Декрементирующаяся (уменьшающаяся) привязка ко времени предотвращает гоночные условия и обеспечивает, чтобы цепочка HTLC-контрактов разматывалась назад, от пункта назначения к источнику.

Вывод

В этой главе мы увидели, как Алиса может делать платежи Дине, даже если у нее нет прямого платежного канала. Алиса может найти путь, соединяющий ее с Диной, и направить платеж по нескольким платежным каналам, чтобы он дошел до Дины.

В целях обеспечения атомарности и бездоверительности платежа по многочисленным переходам Алиса должна имплементировать протокол справедливости в сотрудничестве со всеми промежуточными узлами в пути. Протокол справедливости в настоящее время имплементирован как HTLC-контракт, который привязывает средства к платежному хешу, выводимому из секретного платежного прообраза.

Каждый участник платежного маршрута может распространить HTLC-контракт на следующего участника, не беспокоясь о краже или застрявших средствах. HTLC-контракт можно погасить, раскрыв секретный платежный прообраз. После того как HTLC-контракт достигает Дины, она раскрывает прообраз, который течет в обратном направлении, урегулируя все предложенные HTLC-контракты.

Наконец, мы увидели, как условие о привязанном ко времени возврате средств завершает HTLC-контракт, обеспечивая, чтобы каждый участник смог получить назад свои средства в случае отказа платежа, но по какой-либо причине один из участников не сотрудничает в разматывании HTLC-контрактов. Всегда имея возможность зайти внутрь цепи для возврата средств, HTLC-контракт достигает цели справедливости – атомарности и бездоверительной работы.

Глава 9

Работа канала и пересылка платежей

В этой главе мы объединим платежные каналы и контракты с привязкой к хешу и времени (HTLC). В главе 7 мы объяснили, как Алиса и Боб создают платежный канал между своими двумя узлами. Мы также рассмотрели фиксационный и штрафной механизмы, которые защищают платежный канал. В главе 8 мы рассмотрели HTLC-контракты и то, как их можно использовать для маршрутизирования платежа по пути, состоящему из нескольких платежных каналов. В этой главе мы объединяем две концепции, рассмотрев, как HTLC-контракты управляются в каждом платежном канале, как HTLC-контракты привязаны к состоянию канала и как они улаживаются с целью обновления остатков каналов.

В частности, мы будем обсуждать темы «Добавление, улаживание, отказ HTLC-контрактов» и «Канальная машина состояний», которые формируют наложение между одноранговым слоем и маршрутизационным слоем, как показано на рис. 9-1.

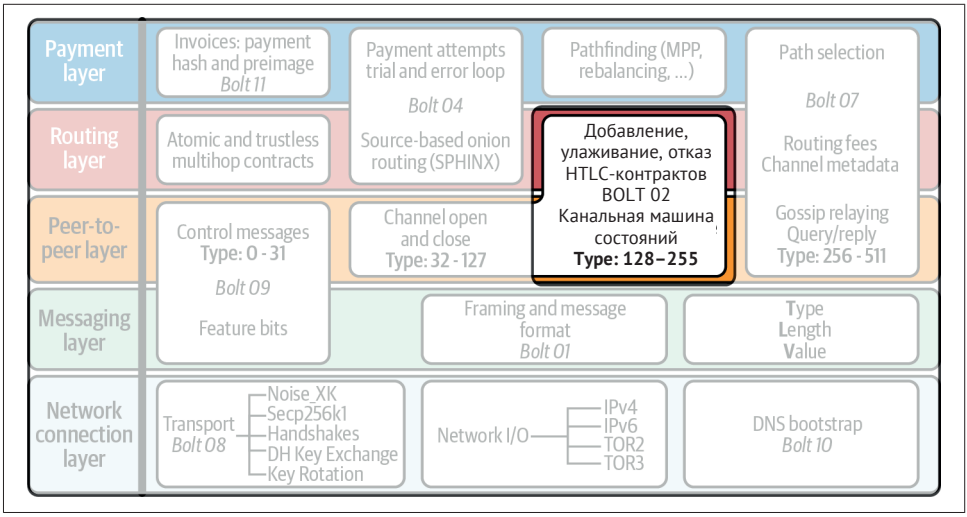


Рис. 9-1. Управление каналами и пересылка платежей в рамках комплекта протоколов Lightning

Локальный (один) канал против маршрутизируемых (многочисленных) каналов

Несмотря на то что можно отправлять платежи по платежному каналу, просто обновляя остатки каналов и создавая новые фиксационные транзакции, протокол Lightning использует HTLC-контракты даже для «локальных» платежей по платежному каналу. Причина этого кроется в поддержании одинаковой конструкции протокола независимо от того, находится ли платеж лишь в одном переходе (через один платежный канал) или в нескольких переходах (маршрутизируется по нескольким платежным каналам).

Поддерживая одинаковую абстракцию как для локальных, так и для дистанционных устройств, мы не только упрощаем разработку протокола, но и повышаем конфиденциальность. Для получателя платежа нет заметной разницы между платежом, произведенным непосредственно его партнером по каналу, и платежом, отправленным его партнером по каналу от имени кого-либо другого.

Пересылка платежей и обновление фиксаций с помощью HTLC-контрактов

Мы вернемся к нашему примеру из главы 8, чтобы продемонстрировать, как HTLC-контракты от Алисы до Дины привязываются к каждому платежному каналу. Как вы помните, в нашем примере Алиса платит Дине 50 000 сатоши, маршрутизируя HTLC-контракт через Боба и Чана. Сеть показана на рис. 9-2.

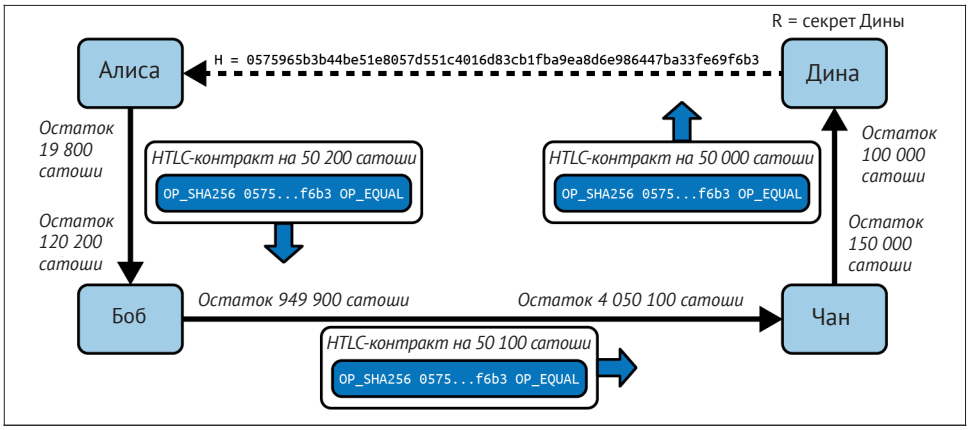


Рис. 9-2. Алиса платит Дине с помощью HTLC-контракта, маршрутизируемого через Боба и Чана

Мы сосредоточимся на платежном канале между Алисой и Бобом и рассмотрим сообщения и транзакции, которые они используют для обработки этого HTLC-контракта.

HTLC-контракт и поток фиксационных сообщений

Поток сообщений между Алисой и Бобом (а также между любой парой партнеров по каналу) показан на рис. 9-3.

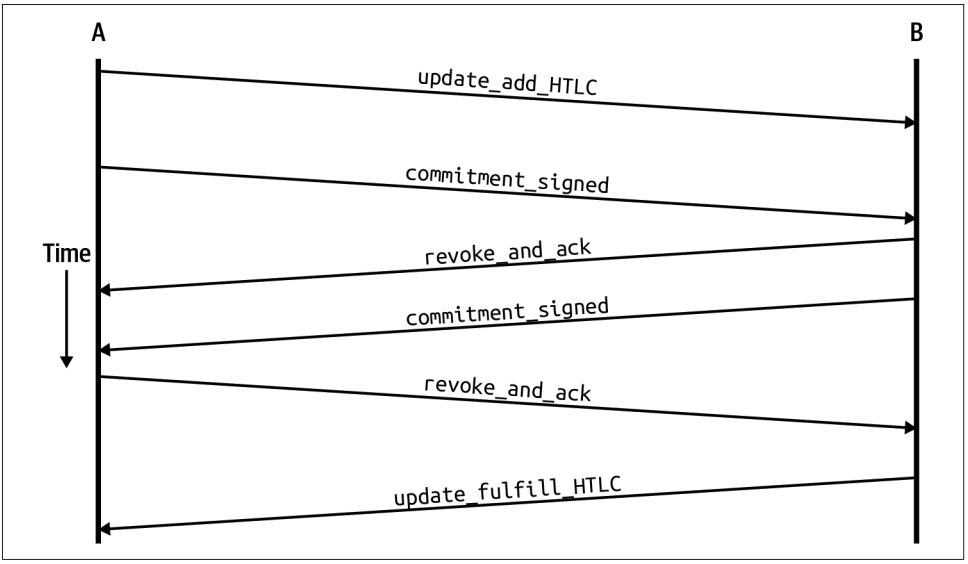


Рис. 9-3. Поток сообщений в отношении фиксаций в HTLC-контрактах между партнерами по каналу

Мы уже встречали сообщения `committement_signed` и `revoke_and_ack` в главе 7. Теперь посмотрим, как HTLC-контракты вписываются в схему фиксаций. Два новых сообщения – `update_add_htlc`, которое Алиса использует, чтобы попросить Боба добавить HTLC-контракт, и `update_ful fill_htlc`, которое Боб использует для погашения HTLC-контракта, как только он получит платежный секрет (секрет Дины).

Пересылка платежей с помощью HTLC-контрактов

Алиса и Боб начинают с платежного канала, остаток которого составляет 70 000 сатоши с каждой стороны.

Как мы видели в главе 7, это означает, что Алиса и Боб провели переговоры и каждый из них удерживает фиксационные транзакции. Эти фиксационные транзакции являются асимметричными, задержанными и отзываемыми и выглядят как пример на рис. 9-4.

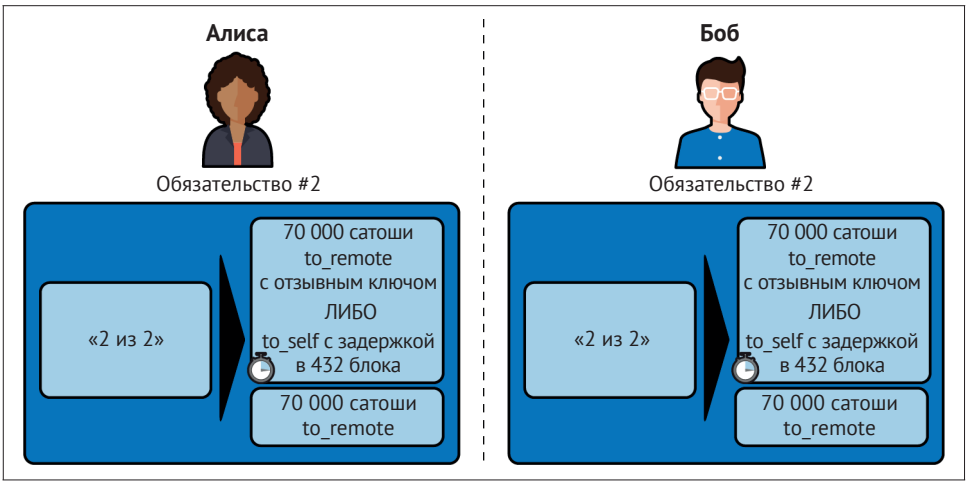


Рис. 9-4. Первоначальные фиксационные транзакции Алисы и Боба

Добавление HTLC-контракта

Алиса хочет, чтобы Боб принял HTLC-контракт на сумму 50 200 сатоши для пересылки Дине. Для этого Алиса должна отправить Бобу подробную информацию об этом HTLC-контракте, включая хеш и сумму платежа. Бобу также нужно будет знать, куда его переслать, что мы подробно обсудим в главе 10.

В целях добавления HTLC-контракта Алиса запускает поток, который мы видели на рис. 9-3, отправляя Бобу сообщение update_add_htlc.

Сообщение update_add_HTLC

Алиса отправляет Бобу сообщение Lightning `update_add_HTLC`. Это сообщение определено в спецификации «BOLT #2: одноранговый протокол, update_add_HTLC» и показано в примере 9-1.

Пример 9-1. Сообщение update_add_HTLC

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

`channel_id`

Это канал, который есть у Алисы с Бобом, куда она хочет добавить HTLC-контракт. Вспомните, что у Алисы и Боба может быть несколько каналов друг с другом.

`id`

Это счетчик HTLC-контрактов, который начинается с 0 для первого HTLC-контракта, предложенного Бобу Алисой, и увеличивается для каждого последующего предлагаемого HTLC-контракта.

`amount_msat`

Это сумма HTLC-контракта в миллисатошах. В нашем примере это 50 200 000 миллисатоши (т. е. 50 200 сатоши).

`payment_hash`

Это платежный хеш, рассчитанный по счету Дины, т. е. H = RIPEMD160(SHA-256(R)), где R – это секрет Дины, который известен только Дине и будет раскрыт, если Дине заплатят.

`cltv_expiry`

Это время истечения срока этого HTLC-контракта, которое будет закодировано как привязанный ко времени возврат в случае, если HTLC-контракт не сможет добраться до Дины за это время.

`onion_routing_packet`

Это луковично-зашифрованный маршрут, который сообщает Бобу, куда переслать этот HTLC-контракт дальше (к Чану). Луковичная маршрутизация подробно описана в главе 10.

Напомним, что учет в сети Lightning ведется в единицах миллисатоши (тысячные доли сатоши), тогда как учет в системе Bitcoin ведется в сатоши. Любые суммы в HTLC-контрактах представляют миллисатоши, которые затем округляются до ближайшего сатоши в транзакциях в системе Bitcoin.

HTLC-контракт в фиксационных транзакциях

Полученной информации для Боба достаточно, чтобы создать новую фиксационную транзакцию. Новая фиксационная транзакция имеет те же два выхода to_self и to_remote для остатка Алисы и Боба, а также новый выход, представляющий HTLC-контракт, предлагаемый Алисой.

Мы уже видели базовую структуру HTLC-контракта в главе 8. Полный скрипт предлагаемого HTLC-контракта определен в спецификации «BOLT #3: транзакции, выход из предлагаемого HTLC-контракта»[67] и показан в примере 9-2.

Пример 9-2. Скрипт выхода из предлагаемого HTLC-контракта

```
1 # Отзыв ❶
2 OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
3 OP_IF
4     OP_CHECKSIG
5 OP_ELSE
6     <remote_HTLCpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
```

[67] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs.

```
 7     OP_IF
 8         # Погашение ❷
 9         OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
10         2 OP_SWAP <local_HTLCpubkey> 2 OP_CHECKMULTISIG
11     OP_ELSE
12         # Возврат средств ❸
13         OP_DROP <cltv_expiry> OP_CHECKLOCKTIMEVERIFY OP_DROP
14         OP_CHECKSIG
15     OP_ENDIF
16 OP_ENDIF
```

❶ Первый компонент условия `OP_IF` может быть погашен Алисой с помощью отзывного ключа. Если эта фиксация позже будет отозвана, то у Алисы будет отзывной ключ, чтобы истребовать этот выход в штрафной транзакции, взяв весь остаток канала.

❷ Второй компонент может быть погашен прообразом (платежным секретом или, в нашем примере, секретом Дины), если он будет раскрыт. Это позволяет Бобу истребовать этот выход, если у него есть секрет от Дины, а стало быть, он успешно доставил платеж Дине.

❸ Третьим и последним компонентом является возврат средств по HTLC-контракту Алисе, если срок действия HTLC-контракта истекает, не добравшись до Дины. Он привязан ко времени истечением срока `cltv_expiry`. За счет этого обеспечивается, что остаток Алисы не «застрянет» в HTLC-контракте, который не может быть маршрутизирован к Дине.

Этот выход можно истребовать тремя способами. Попробуйте прочитать скрипт и посмотреть, сможете ли вы в нем разобраться (помните, что это язык, основанный на стеке, поэтому все выглядит «задом наперед»).

Новая фиксация с выходом из HTLC-контракта

Теперь у Боба есть необходимая информация, чтобы добавить этот HTLC-скрипт в качестве дополнительного выхода и создать новую фиксационную транзакцию. Новая фиксация Боба будет иметь 50 200 сатоши в выходе из HTLC-контракта. Эта сумма будет взята с остатка канала Алисы, поэтому новый остаток Алисы составит 19 800 сатоши (70 000 – 50 200 = 19 800). Боб строит эту фиксацию как предварительную «фиксацию #3», показанную на рис. 9-5.

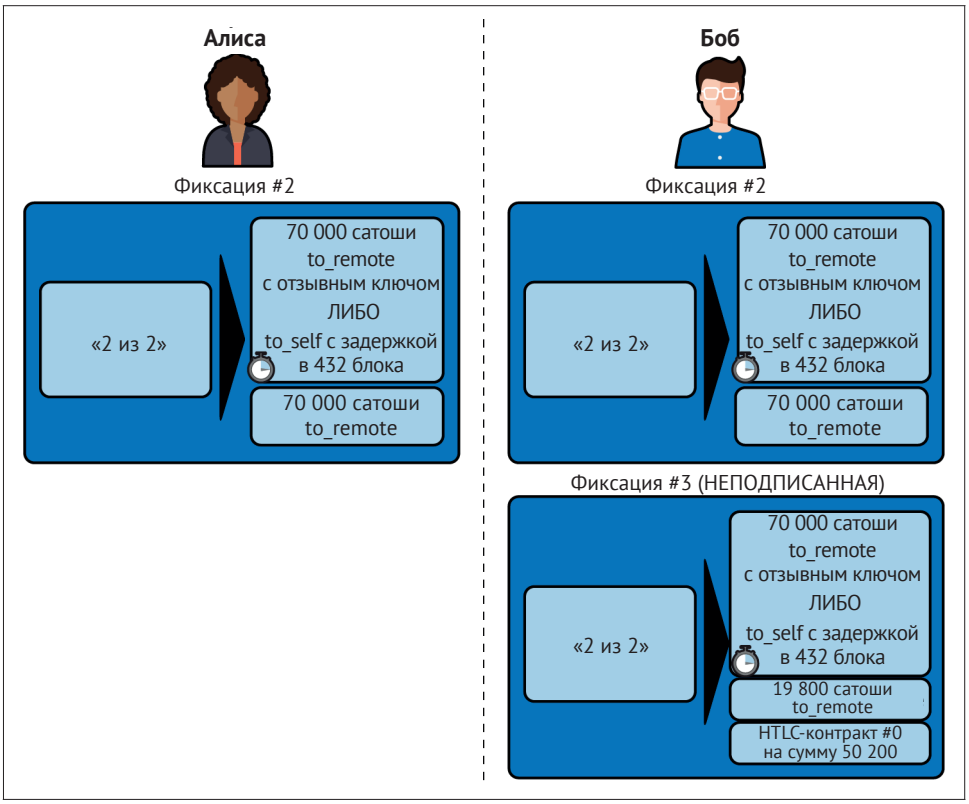


Рис. 9-5. Новая фиксация Боба с выходом HTLC-контракта

Алиса фиксирует

Вскоре после отправки сообщения `update_add_htlc` она зафиксируется на новом состоянии канала, чтобы Боб мог безопасно добавить HTLC-контракт. Боб располагает информацией HTLC-контракта и создал новую фиксацию, но эта новая фиксация еще не подписана Алисой.

Алиса отправляет `committement_signed` Бобу с подписью под новой фиксацией и под HTLC-контрактом внутри. Мы видели сообщение `committement_signed` в главе 7, но теперь можем понять остальные поля. Как напоминание оно показано в примере 9-3.

Пример 9-3. Сообщение committement_signed

```
[channel_id:channel_id]
[signature:signature]
[u16:num_htlcs]
[num_htlcs*signature:htlc_signature]
```

Поля `num_htlcs` и `htlc_signature` теперь имеют больше смысла:

`num_htlcs`

Это число HTLC-контрактов, которые не завершены в фиксационной транзакции. В нашем примере только один HTLC-контракт, тот, который предложила Алиса.

`htlc_signature`

Это массив подписей (длиной `num_htlcs`), содержащий подписи для выходов HTLC-контракта.

Алиса может отправить эти подписи без колебаний: она всегда может вернуть свои средства, если срок действия HTLC-контракта истечет, не будучи маршрутизированным Дине.

Теперь у Боба есть новая подписанная фиксационная транзакция, как показано на рис. 9-6.

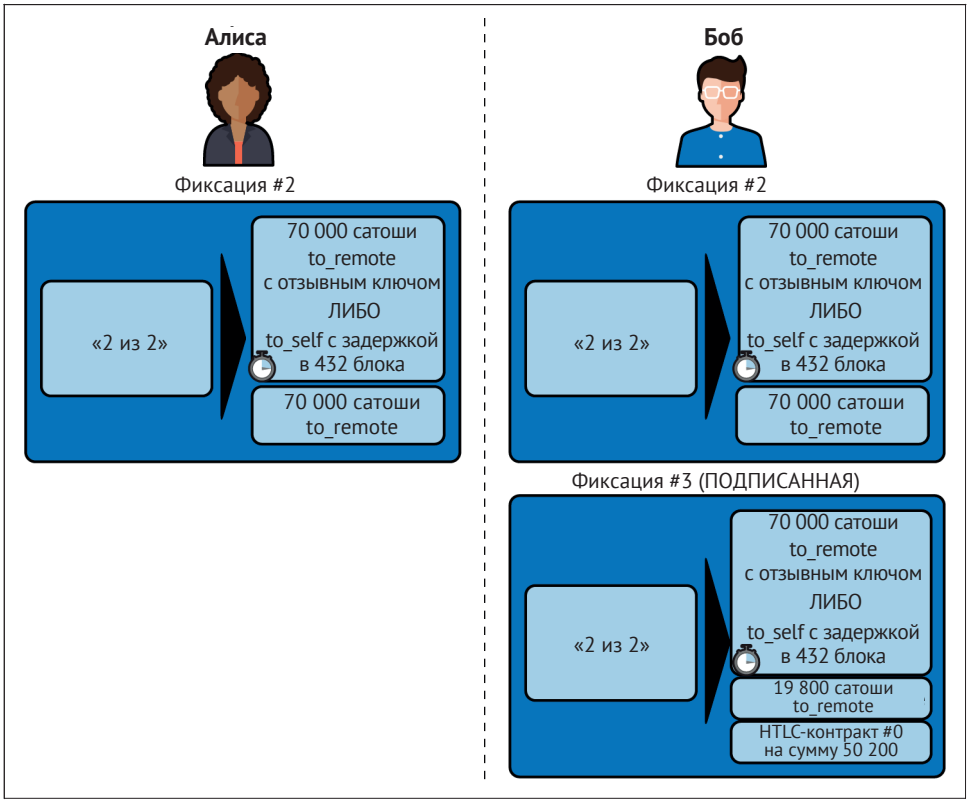


Рис. 9-6. У Боба есть новая подписанная фиксация

Боб признает новую фиксацию и отзывает старую

Теперь, когда у Боба есть новая подписанная фиксация, ему нужно ее подтвердить и отозвать старую фиксацию. Он делает это, отправляя сообщение `revoke_and_ack` об отзыве и возврате, как мы видели в главе 7 ранее. Как напоминание это сообщение показано в примере 9-4.

Пример 9-4. Сообщение revoke_and_ack

```
[channel_id:channel_id]
[32*byte:per_commitment_secret]
[point:next_per_commitment_point]
```

Боб отправляет `per_commitment_secret`, которое позволяет Алисе собрать отзывной ключ для сборки штрафной транзакции, расходующей старую фиксацию Боба. После того как Боб его отправит, он никогда не сможет опубликовать «фиксацию #2», не рискуя получить штрафную транзакцию и потерять все свои деньги. Таким образом, старая фиксация фактически отзывается.

Боб эффективно переместил состояние канала дальше, как показано на рис. 9-7.

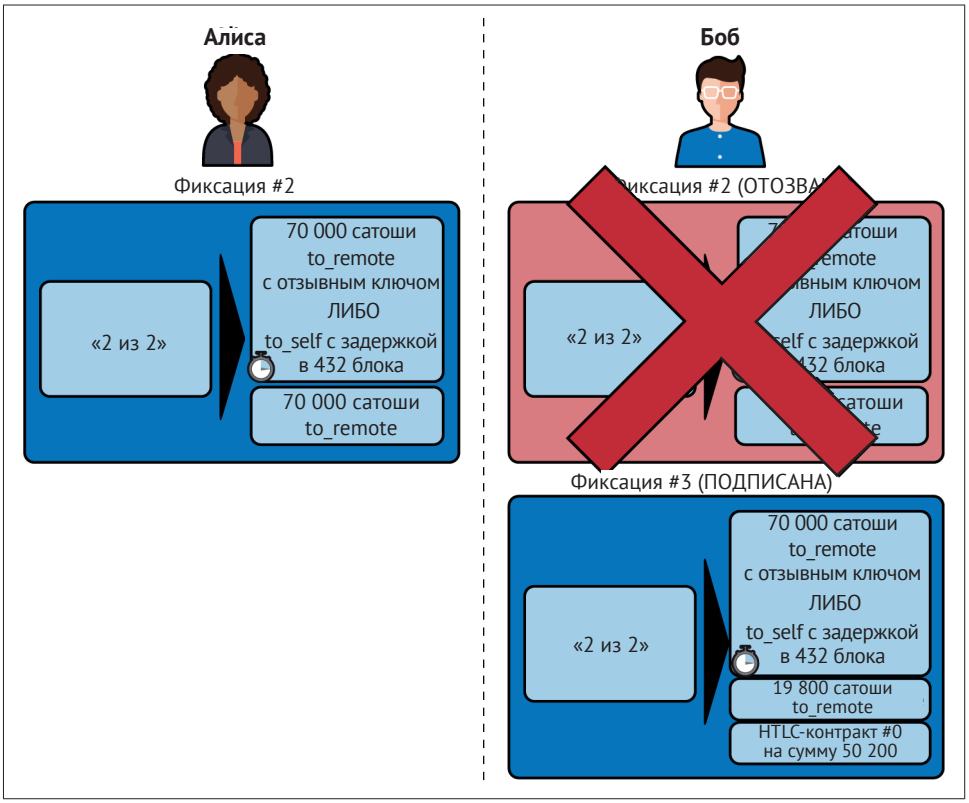


Рис. 9-7. Боб отозвал старую фиксацию

Несмотря на то что у Боба есть новая (подписанная) фиксационная транзакция и выход HTLC-контракта внутри, он не может считать, что его HTLC-контракт настроен успешно.

Сначала ему нужно, чтобы Алиса отозвала свою старую фиксацию, потому что в противном случае Алиса может откатить свой остаток до 70 000 сатоши. Боб должен убедиться, что у Алисы тоже есть фиксационная транзакция, содержащая HTLC-контракт, и она отозвала старую фиксацию.

Вот почему, если Боб не является конечным получателем средств HTLC-контракта, ему пока не следует переадресовывать HTLC-контракт, предлагая HTLC-контракт на следующем канале с Чаном.

Алиса создала зеркальную новую фиксационную транзакцию, содержащую новый HTLC-контракт, но Боб еще ее не подписал. Это видно на рис. 9-8.

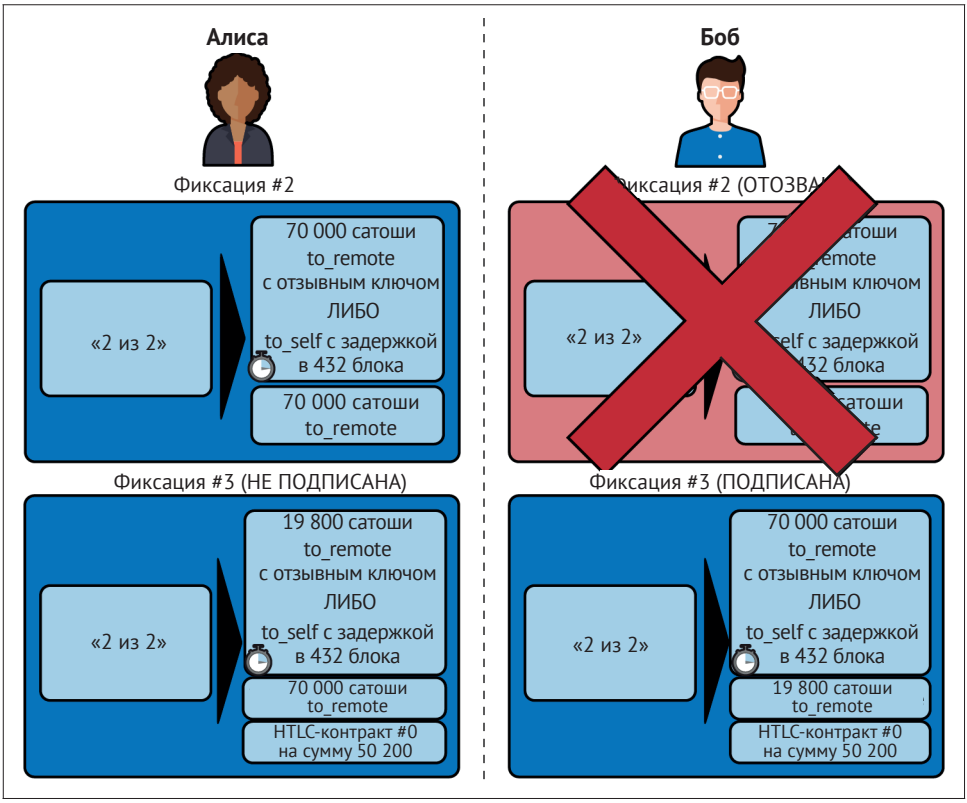


Рис. 9-8. Новая фиксация Алисы с выходом HTLC-контракта

Как мы описали в главе 7, фиксация Алисы зеркальна фиксации Боба, поскольку она содержит асимметричный, задержанный, отзываемый конструкт для отзывных и штрафных мер исполнения старых фиксаций. Остаток Алисы в размере 19 800 сатоши (после вычета значения HTLC-контракта) задерживается и может быть отозван. Остаток Боба в размере 70 000 сатоши подлежит немедленному погашению.

Далее поток сообщений для `committement_signed` и `revoke_and_ack` теперь повторяется, но в противоположном направлении. Боб отправляет `committement_signed`, чтобы подписать новую фиксацию Алисы, и Алиса отвечает, отменяя свою старую фиксацию.

Для полноты картины давайте проведем быстрый осмотр фиксационных транзакций, по мере того как происходит этот раунд фиксации/отзыва.

Боб фиксирует

Боб теперь отправляет `committement_signed` обратно Алисе со своими подписями под новой фиксационной транзакцией Алисы, включая добавленный ею выход HTLC-контракта.

Теперь у Алисы есть подпись под новой фиксационной транзакцией. Состояние канала показано на рис. 9-9.

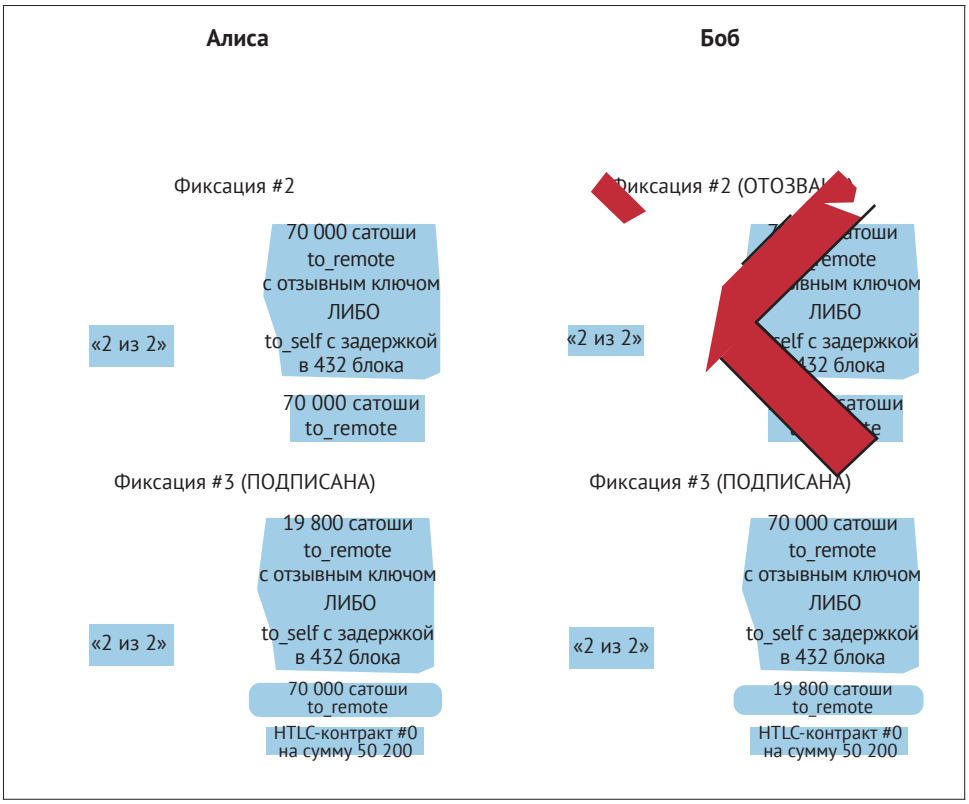


Рис. 9-9. У Алисы есть новая подписанная фиксация

Теперь Алиса может подтвердить новую фиксацию, отозвав старую. Алиса отправляет сообщение `revoke_and_ack`, содержащее необходимое `per_commitment_point`, которое позволит Бобу создать отзывной ключ и штрафную транзакцию. Таким образом, Алиса отменяет свою старую фиксацию.

Состояние канала показано на рис. 9-10.

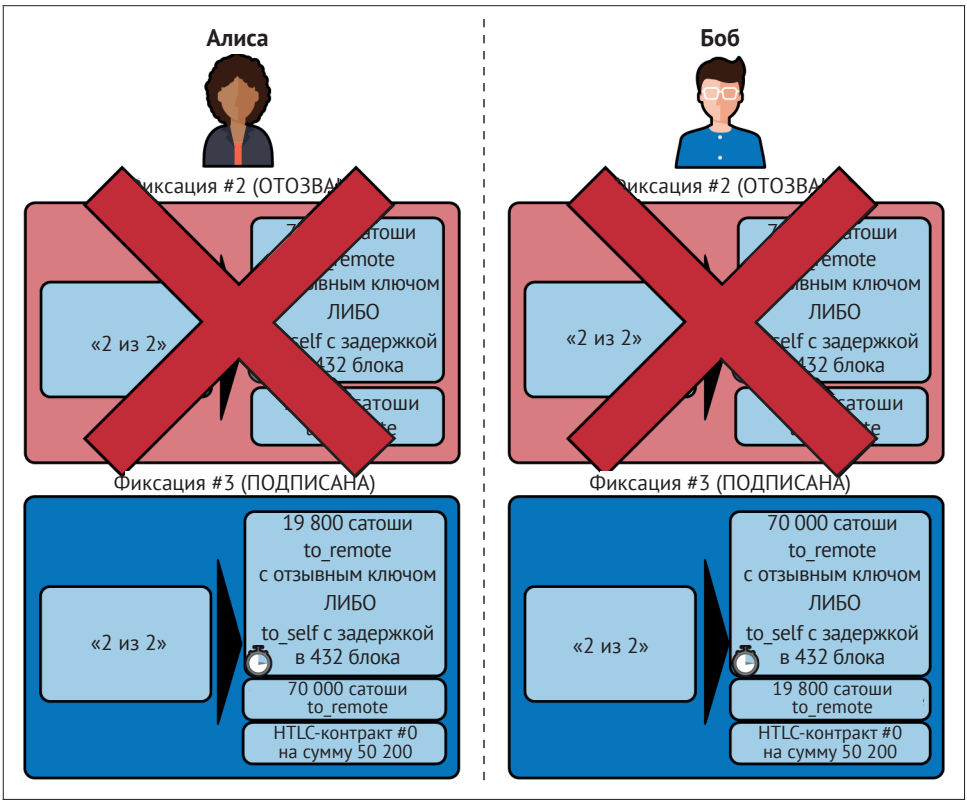


Рис. 9-10. Алиса отозвала старую фиксацию

Несколько HTLC-контрактов

В любой момент времени у Алисы и Боба могут быть десятки или даже сотни HTLC-контрактов на одном канале. Каждый HTLC-контракт предлагается и добавляется в фиксационную транзакцию в качестве дополнительного выхода. Таким образом, фиксационная транзакция всегда имеет два выхода для остатков партнеров канала и любое число выходов HTLC-контракта, по одному на HTLC-контракт.

Как мы видели в сообщении committement_signed, существует массив для подписей HTLC-контракта, чтобы одновременно можно было передавать несколько HTLC-фиксаций.

Текущее максимальное число разрешенных на канале HTLC-контрактов составляет 483. Указанное число учитывает максимальный размер Bitcoin-транзакции и обеспечивает, чтобы фиксационные транзакции оставались валидными Bitcoin-транзакциями.

Как мы увидим в следующем разделе, максимум предназначен только для ожидающих своей очереди HTLC-контрактов, потому что как только HTLC-контракт выполняется (или отказывает из-за тайм-аута/ошибки), он удаляется из фиксационной транзакции.

Исполнение HTLC-контракта

Теперь у Боба и Алисы есть новая фиксационная транзакция с дополнительным HTLC-выходом, и мы сделали важный шаг к обновлению платежного канала.

Новый остаток Алисы и Боба еще не отражает того, что Алиса успешно отправила Бобу 50 200 сатоши.

Однако в настоящее время HTLC-контракты настроены таким образом, что будет возможен безопасный взаиморасчет в обмен на подтверждение платежа.

Распространение HTLC-контракта

Давайте предположим, что Боб продолжает цепочку и устанавливает HTLC-контракт с Чаном на 50 100 сатоши. Процесс будет точно таким же, как мы только что видели между Алисой и Бобом. Боб отправит `update_add_htlc` Чану, потом они обменяются сообщениями `commit ment_signed` и `revoke_and_ack` в два раунда, продвигая свой канал к следующему состоянию.

Затем Чан сделает то же самое с Диной: предложит HTLC-контракт на 50 000 сатоши, зафиксирует и отзовет и т. д. Однако Дина является конечным получателем HTLC-контракта. Дина – единственная, кто знает платежный секрет (прообраз платежного хеша). Таким образом, Дина может немедленно исполнить HTLC-контракт с Чаном!

Дина исполняет HTLC-контракт с Чаном

Дина может урегулировать HTLC-контракт, отправив Чану сообщение `update_fulfill_htlc`. Сообщение `update_fulfill_htlc` определено в спецификации «№BOLT #2: одноранговый протокол, update_fulfill_htlc№»[68] и показано ниже:

```
[channel_id:channel_id]
[u64:id]
[32*byte:payment_preimage]
```

Это действительно простое сообщение:

`channel_id`

Идентификатор канала, на котором фиксируется HTLC-контракт.

`id`

Идентификатор HTLC-контракта (мы начали с 0 и наращивали номер каждого HTLC-контракта на канале).

`payment_preimage`

Секрет, который доказывает, что платеж был произведен, и погашает HTLC-контракт. Это значение R, которое было хешировано Диной для генерирования платежного хеша в счете Алисе.

Когда Чан получит это сообщение, он немедленно проверит, генерирует ли `payment_preimage` (назовем его *R*) платежный хеш (назовем его *H*) в HTLC-контракте, который он предложил Дине. Он хеширует это так:

[68] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#removing-an-htlc-update_fulfill_htlc-update_fail_htlc-and-update_fail_malformed_htlc.

$$H = \text{RIPEMD160}(\text{SHA-256}(R))$$

Если результат H соответствует платежному хешу в HTLC-контракте, то Чан может устроить небольшой праздничный танец. Этот долгожданный секрет может быть использован для выкупа HTLC-контракта и будет передан обратно по цепочке платежных каналов вплоть до Алисы, улаживая каждый HTLC-контракт, который был частью этого платежа Дине.

Давайте вернемся к каналу Алисы и Боба и посмотрим, как они разматывают HTLC-контракт. Для того чтобы туда добраться, допустим, что Дина отправила `update_fulfill_htlc` Чану, Чан отправил `update_fulfill_htlc` Бобу, а Боб отправил `update_fulfill_htlc` Алисе. Платежный прообраз распространился вплоть до Алисы.

Боб улаживает HTLC-контракт с Алисой

Когда Боб отправит `update_fulfill_htlc` Алисе, он будет содержать тот же платежный прообраз `payment_preimage`, который Дина выбрала для своего счета. Этот `payment_preimage` переместился назад по пути оплаты. На каждом шаге идентификатор канала `channel_id` будет отличаться, а идентификатор `id` (идентификатор HTLC-контракта) может отличаться. Но прообраз тот же самый!

Алиса также проверит `payment_preimage`, полученный от Боба. Она сравнит его хеш с платежным хешем HTLC-контракта в HTLC-контракте, который она предложила Бобу. Она также обнаружит, что этот прообраз соответствует хешу в счете Дины. Это доказательство того, что Дине заплатили.

Поток сообщений между Алисой и Бобом показан на рис. 9-11.

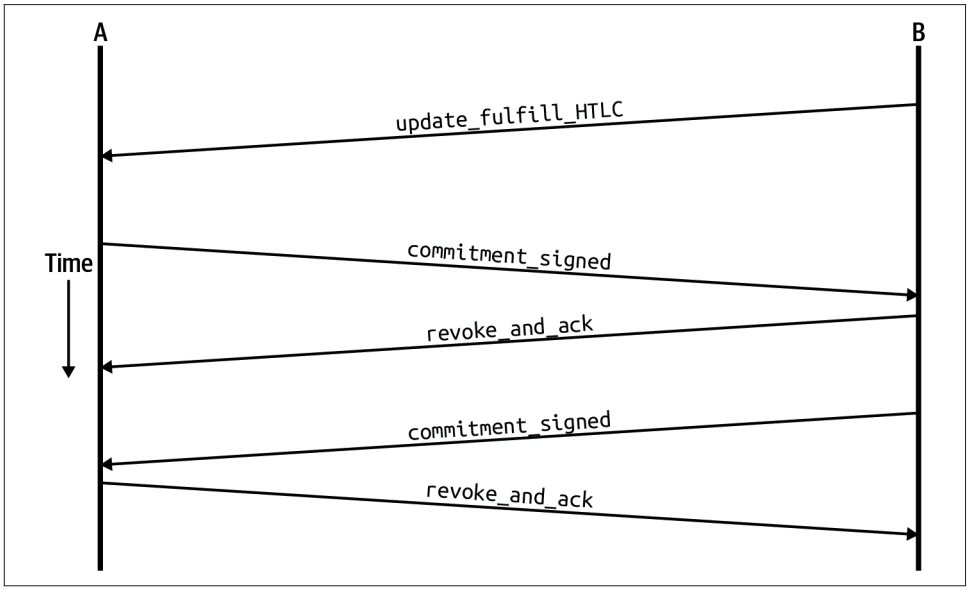


Рис. 9-11. Поток сообщений о выполнении HTLC-контракта

И Алиса, и Боб теперь могут удалить HTLC-контракт из фиксационных транзакций и обновить остатки своих каналов.

Они создают новые фиксации (фиксация № 4), как показано на рис. 9-12.

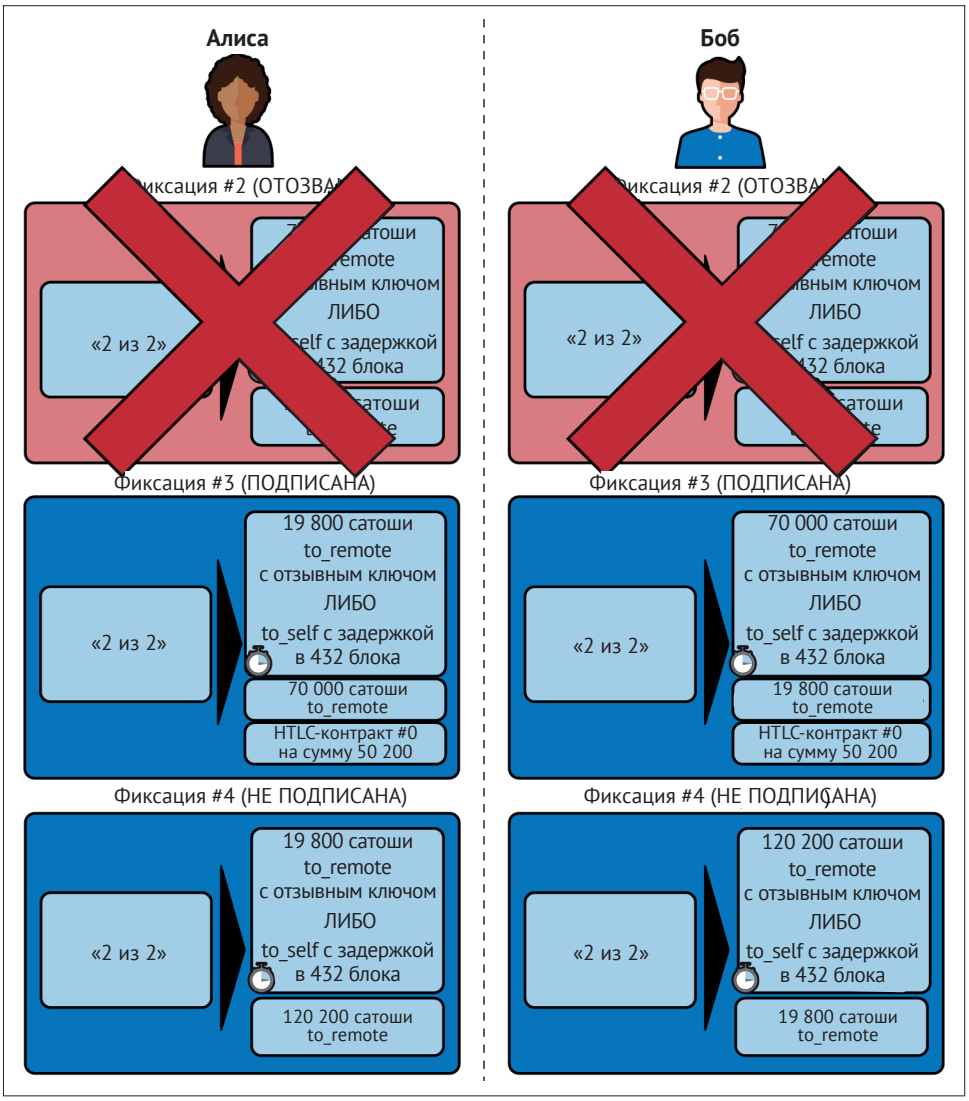


Рис. 9-12. HTLC-контракт удаляется, а остатки обновляются в новых фиксациях

Затем они завершают два раунда фиксации и отзыва. Сначала Алиса отправляет `committement_signed`, чтобы подписать новую фиксационную транзакцию Боба. Боб отвечает сообщением `revoke_and_ack`, чтобы отозвать свою старую фиксацию. Как только Боб переместил состояние канала вперед, фиксация выглядит так, как мы видим на рис. 9-13.

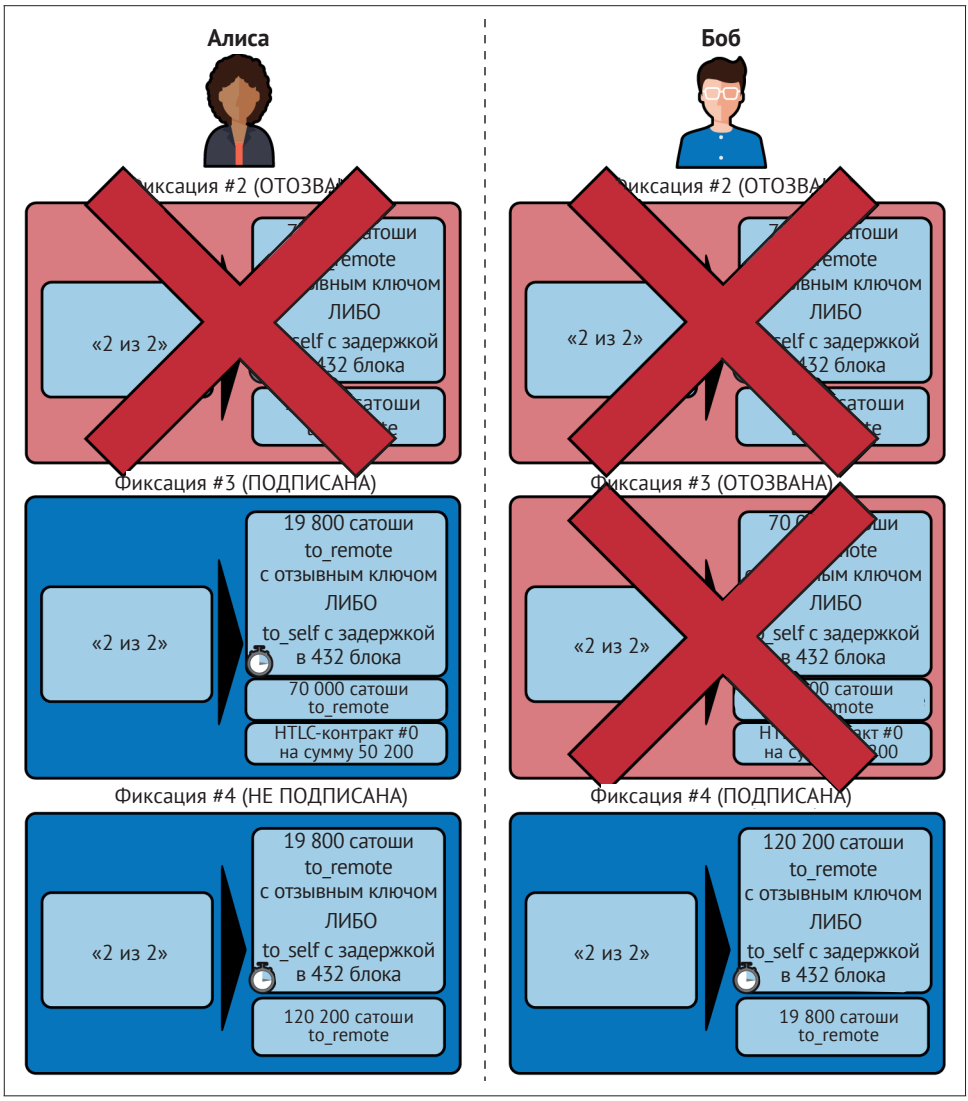


Рис. 9-13. Алиса подписывает новую фиксацию Боба, а Боб отозвал старую

Наконец, Боб подписывает фиксацию Алисы, отправляя Алисе сообщение `committement_signed`. Затем Алиса подтверждает и отзывает свою старую фиксацию, отправив `revoke_and_ack` Бобу. Конечным результатом является то, что и Алиса, и Боб переместили состояние своего канала в фиксацию #4, удалили HTLC-контракт и обновили свои остатки. Их текущее канальное состояние представлено фиксационными транзакциями, которые показаны на рис. 9-14.

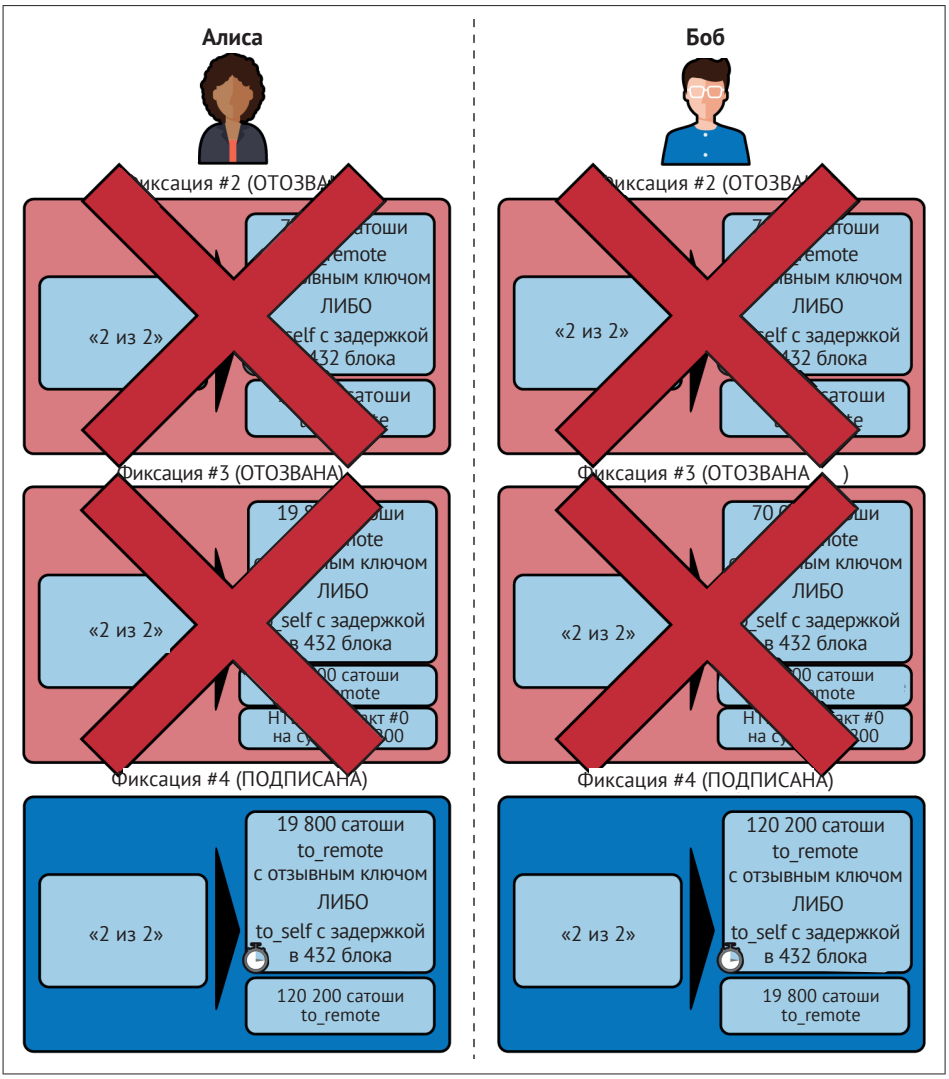


Рис. 9-14. Алиса и Боб улаживают HTLC-контракт и обновляют остатки

Удаление HTLC-контракта из-за ошибки или истечения срока

Если HTLC-контракт не может быть исполнен, его можно удалить из фиксации канала, используя тот же процесс фиксации и отзыва.

Вместо сообщения update_fulfill_htlc Боб отправил бы сообщение update_fail_htlc или update_fail_malformed_htlc. Эти два сообщения определены в спецификации «BOLT #2: одноранговый протокол, удаление HTLC-контракта»[69].

[69] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#removing-an-htlc-update_fulfill_htlc-update_fail_htlc-and-update_fail_malformed_htlc.

Сообщение update_fail_htlc выглядит следующим образом:

```
[channel_id:channel_id]
[u64:id]
[u16:len]
[len*byte:reason]
```

Оно довольно понятно само по себе. Многобайтовое поле причины `reason` определено в спецификации «BOLT #4: луковичная маршрутизация», которую мы опишем в главе 10.

Если бы Алиса получила `update_fail_htlc` от Боба, то процесс разворачивался бы во многом таким же образом: два партнера по каналу удалили бы HTLC-контракт, создали обновленные фиксационные транзакции и прошли два раунда фиксации/отзыва, чтобы переместить состояние канала вперед к следующей фиксации. Единственное отличие: конечные остатки вернутся к тому, чем они были без HTLC-контракта, по сути возвращая Алисе стоимость HTLC-контракта.

Осуществление локального платежа

На этом этапе вы легко поймете, почему HTLC-контракты используются как для дистанционных, так и для локальных платежей. Когда Алиса платит Бобу за кофе, она не просто обновляет остаток канала и переходит в новое состояние. Вместо этого оплата производится с помощью HTLC-контракта, точно так же, как Алиса заплатила Дине. Тот факт, что существует только один переход по каналу, не имеет никакого значения. Это будет работать следующим образом:

1. Алиса заказывает кофе со страницы магазина Боба.
2. Магазин Боба отправляет счет с платежным хешем.
3. Алиса создает HTLC-контракт из этого платежного хеша.
4. Алиса предлагает HTLC-контракт Бобу с помощью `update_add_htlc`.
5. Алиса и Боб обмениваются фиксациями и отзывами, добавляя HTLC-контракт в свои фиксационные транзакции.
6. Боб отправляет `update_fulfill_htlc` Алисе с платежным прообразом.
7. Алиса и Боб обмениваются фиксациями и отзывами, удаляя HTLC-контракт и обновляя остатки каналов.

Независимо от того, пересылается ли HTLC-контракт по многочисленным каналам или просто исполняется в одном канальном «прыжке», процесс точно такой же.

Вывод

В этой главе мы рассмотрели, как фиксационные транзакции (из главы 7) и HTLC-контракты (из главы 8) работают вместе. Мы увидели, как HTLC-контракт добавляется в фиксационную транзакцию и как он исполняется. Мы увидели, как асимметричная, задержанная, отзываемая система обеспечения состояния канала распространяется на HTLC-контракты.

Мы также увидели, как локальный платеж и многопереходный маршрутизируемый платеж обрабатываются одинаково: с использованием HTLC-контрактов.

В следующей главе мы рассмотрим систему маршрутизации зашифрованных сообщений, именуемую луковичной маршрутизацией.

Глава 10

Луковичная маршрутизация

В этой главе мы опишем механизм луковичной маршрутизации сети Lightning. Изобретение луковичной маршрутизации опережает сеть Lightning на 25 лет! Луковичная маршрутизация была изобретена исследователями ВМС США в качестве протокола безопасной связи. Луковичная маршрутизация наиболее широко используется Tor, луковично-маршрутизируемым интернет-оверлеем, который позволяет исследователям, активистам, агентам внутренней и внешней разведки и всем остальным пользоваться интернетом конфиденциально и анонимно.

В этой главе мы сосредоточимся на той части архитектуры протокола Lightning, которая называется «Луковичная маршрутизация на основе источника (SPHINX)» и выделена контуром в центре (маршрутизационный слой) на рис. 10-1.

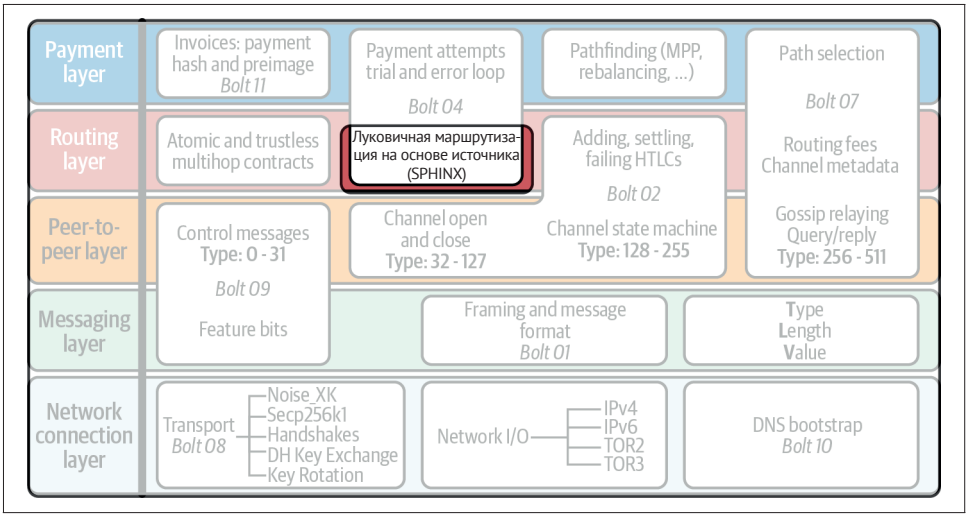


Рис. 10-1. Луковичная маршрутизация в рамках комплекта протоколов Lightning

Луковичная маршрутизация описывает метод шифрованной связи, при котором отправитель сообщения строит последовательные *вложенные слои шифрования*, которые «отслаиваются» каждым промежуточным узлом до тех пор, пока самый внутренний слой не будет доставлен намеченному получателю. Название «луковичная маршрутизация» описывает это использование много-

слойного шифрования, которое отслаивается по одному слою за раз, как чешуйка кожуры луковицы.

Каждый промежуточный узел может «отслаивать» только один слой и просматривать, кто следующий на пути связи. Луковичная маршрутизация обеспечивает, чтобы никто, кроме отправителя, не знал назначения или длины коммуникационного пути. Каждый посредник знает только предыдущий и следующий переходы.

В сети Lightning используется имплементация протокола луковичной маршрутизации на основе Sphinx[70], разработанного в 2009 году Джорджем Данезисом и Яном Голдбергом.

Имплементация луковичной маршрутизации в сети Lightning определена в спецификации «BOLT #4: протокол луковичной маршрутизации»[71].

Физический пример, иллюстрирующий луковичную маршрутизацию

Луковичная маршрутизация описывается по-разному, но один из самых простых приемов – использовать физический эквивалент запечатанных конвертов. Конверт представляет слой шифрования, позволяющий только указанному получателю его открыть и прочитать содержимое.

Допустим, Алиса хочет отправить секретное письмо Дине косвенно через каких-то посредников.

Выбор пути

В сети Lightning используется *источниковая маршрутизация*, то есть платежный путь выбирается и указывается отправителем, и только отправителем. В данном примере секретное письмо Алисы Дине будет эквивалентом платежа. В целях обеспечения доставки письма Дине Алиса создаст путь от нее к Дине, используя Боба и Чана в качестве посредников.

Алиса может добраться до Дины многими путями. Мы объясним процесс выбора оптимального пути в главе 12. На данный момент мы будем исходить из того, что в выбранном Алисой пути Боб и Чан используются в качестве посредников, чтобы добраться до Дины.

В качестве напоминания выбранный Алисой путь показан на рис. 10-2.

[70] См. *Джордж Данезис и Ян Голдберг*. Sphinx: компактный и доказуемо безопасный формат Mix (George Danezis, Ian Goldberg, Sphinx: A Compact and Provably Secure Mix Format, in IEEE Symposium on Security and Privacy (New York: IEEE, 2009), 269–282.

[71] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md.

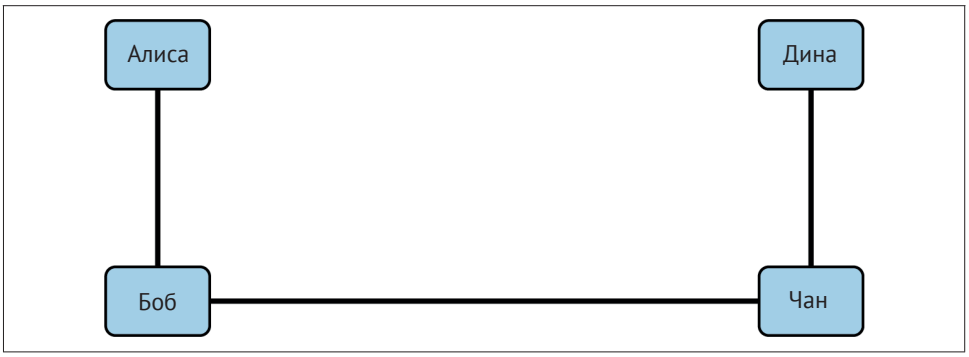


Рис. 10-2. Путь: от Алисы до Боба до Чана до Дины

Давайте посмотрим, как Алиса будет использовать этот путь, не раскрывая информацию посредникам Бобу и Чану.

> **Маршрутизация на основе источника**
>
> Маршрутизация на основе источника – это не то, как пакеты обычно маршрутизируются в интернете сегодня, хотя на первых порах маршрутизация на основе источника была возможна. Интернет-маршрутизация основана на коммутации пакетов на каждом промежуточном узле маршрутизации. Пакет IPv4, например, включает IP-адреса отправителя и получателя, и все последующие узлы IP-маршрутизации решают, как пересылать каждый пакет к месту назначения. Однако отсутствие конфиденциальности в таком механизме маршрутизации, где каждый промежуточный узел видит отправителя и получателя, делает его негодным вариантом выбора для использования в платежной сети.

Сборка слоев

Алиса начинает с того, что пишет секретное письмо Дине. Затем она запечатывает письмо в конверт и пишет «Дине» снаружи (см. рис. 10-3). Конверт представляет собой шифрование с использованием публичного ключа Дины, в результате чего только Дина может открыть конверт и прочитать письмо.

Письмо Дине будет доставлено Дине Чаном, который находится непосредственно перед Диной на «пути». Итак, Алиса кладет Динин конверт в конверт, адресованный Чану (см. рис. 10-4). Единственная часть, которую Чан может прочитать, – это пункт назначения (инструкции по маршруту): «Дине». Запечатывание этого сообщения внутрь конверта, адресованного Чану, представляет собой его шифрование с помощью публичного ключа Чана, чтобы только Чан мог прочитать адрес конверта. Чан по-прежнему не может открыть Динин конверт. Он видит лишь инструкции снаружи (адрес).

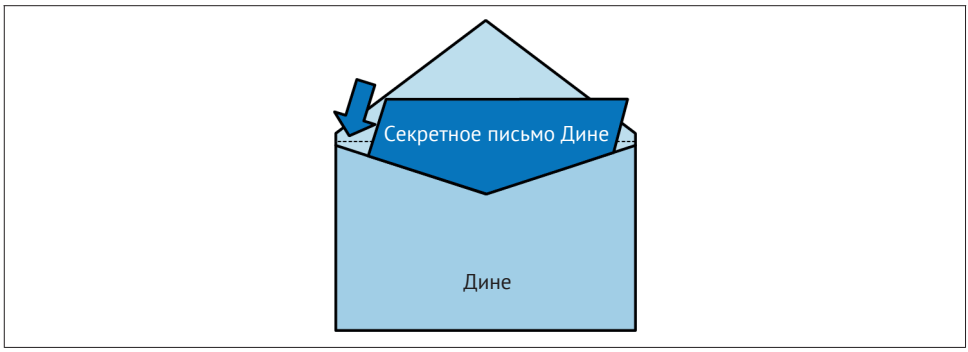


Рис. 10-3. Секретное письмо Дины, запечатанное в конверт

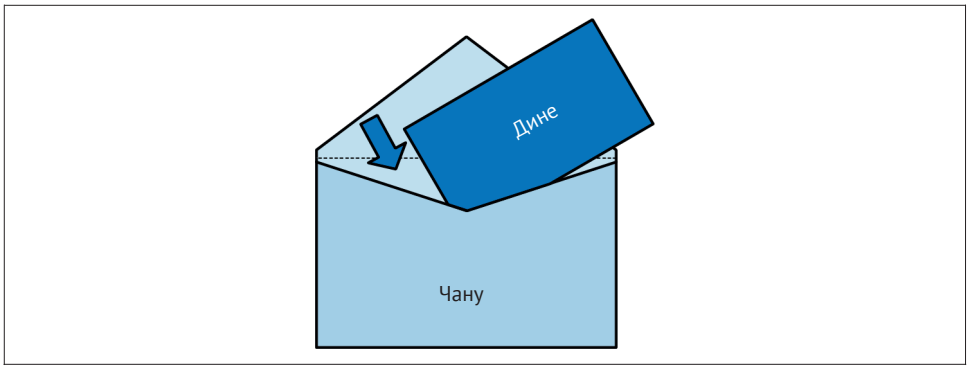


Рис. 10-4. Конверт Чана, содержащий запечатанный Динин конверт

Далее это письмо будет доставлено Чану Бобом. Поэтому Алиса кладет его в конверт, адресованный Бобу (см. рис. 10-5). Как и прежде, конверт представляет собой зашифрованное сообщение для Боба, которое может прочитать только Боб. Боб может прочитать только внешнюю сторону конверта Чана (адрес), поэтому он знает, что его нужно отправить Чану.

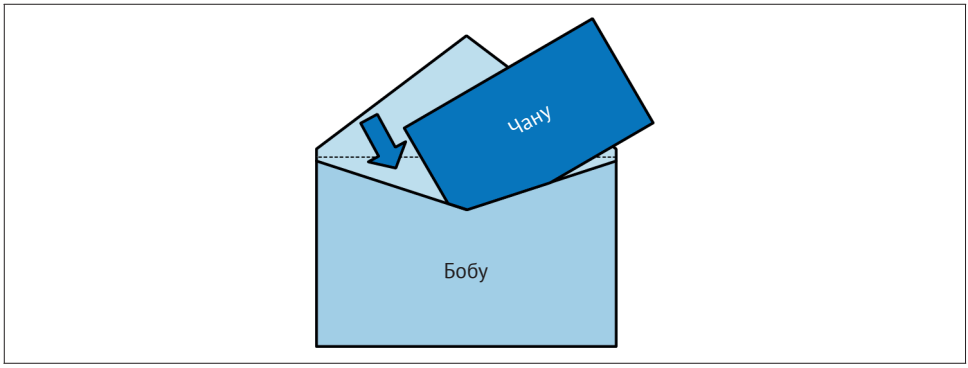


Рис. 10-5. Конверт Боба, содержащий запечатанный конверт Чана

Теперь, если бы мы могли просмотреть конверты (с помощью рентгеновских лучей!), то мы бы увидели конверты, вложенные один в другой, как показано на рис. 10-6.

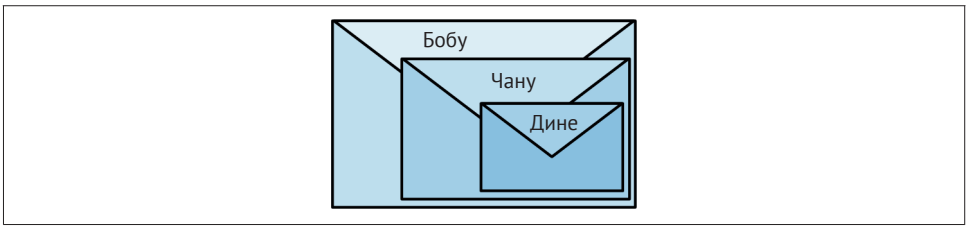


Рис. 10-6. Вложенные конверты

Отслаивание слоев

У Алисы есть конверт с надписью «Бобу» снаружи. Он представляет собой зашифрованное сообщение, которое может раскрыть (дешифровать) только Боб. Теперь Алиса начнет процесс, отправив его Бобу. Весь процесс показан на рис. 10-7.

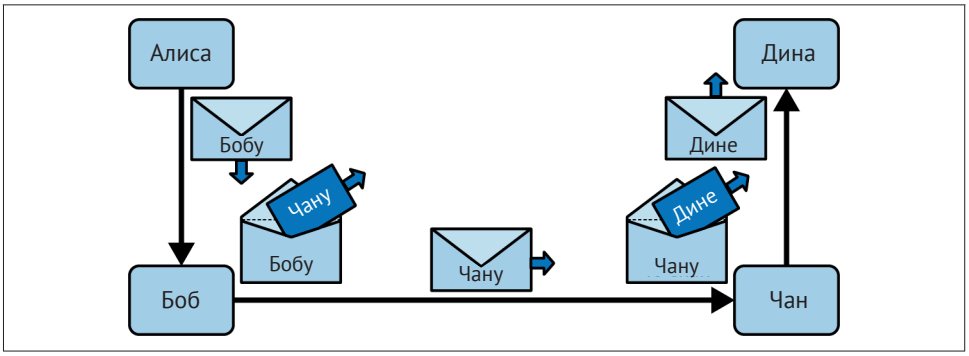


Рис. 10-7. Отправка конвертов

Как вы видите, Боб получает конверт от Алисы. Он знает, что письмо пришло от Алисы, но не знает, является ли Алиса первоначальным отправителем или просто кем-то, кто пересылает конверты. Он открывает его и обнаруживает внутри конверт с надписью «Чану». Поскольку письмо адресовано Чану, Боб не может его открыть. Он не знает, что там внутри, и не знает, получает ли Чан письмо или другой конверт для пересылки. Боб не знает, является Чан конечным получателем или нет. Боб передает конверт Чану.

Чан получает конверт от Боба. Он не знает, что он пришел от Алисы. Он не знает, является Боб посредником либо отправителем письма. Чан открывает конверт и находит внутри другой конверт, адресованный «Дине», который он не может открыть. Чан пересылает его Дине, не зная, является ли Дина конечным получателем.

Дина получает конверт от Чана. Открыв его, она обнаруживает внутри письмо, так что теперь она знает, что является намеченным получателем этого сообщения. Она читает письмо, зная, что никто из посредников не знает, откуда оно пришло, и никто другой не читал ее секретное письмо!

В этом суть луковичной маршрутизации. Отправитель обертывает сообщение слоями, точно указывая, как оно будет маршрутизироваться, и не позволяя никому из посредников получать какую-либо информацию о пути или полезном грузе. Каждый посредник отслаивает один слой, видит только адрес пересылки и не знает ничего, кроме предыдущего и следующего переходов по пути.

Теперь давайте рассмотрим детали имплементации луковичной маршрутизации в сети Lightning.

Введение в луковичную маршрутизацию на основе HTLC-контрактов

Луковичная маршрутизация в сети Lightning на первый взгляд кажется сложной, но как только вы поймете основную концепцию, на самом деле она окажется довольно простой.

С практической точки зрения, Алиса сообщает каждому промежуточному узлу о том, какой HTLC-контракт следует настроить для следующего узла в пути.

Первый узел, который в нашем примере является отправителем платежа, или Алисой, называется *исходным узлом*. Последний узел, который в нашем примере является получателем платежа, или Диной, называется *заключительным узлом*.

Каждый промежуточный узел, или Боб и Чан в нашем примере, называется переходом (или переходным узлом, hop). Каждый переход должен настраивать исходящий HTLC-контракт для следующего перехода. Информация, передаваемая Алисой каждому переходу, называется переходным полезным грузом, или данными перехода. Сообщение, которое направляется от Алисы к Дине, называется луковицей и состоит из зашифрованного переходного полезного груза или сообщений в данных перехода, зашифрованных для каждого перехода.

Теперь, когда мы знаем терминологию, используемую в луковичной маршрутизации Lightning, давайте переформулируем задачу Алисы: Алиса должна сконструировать луковицу с данными перехода, сообщая каждому переходу, как строить исходящий HTLC-контракт для отправки платежа заключительному узлу (Дине).

Алиса выбирает путь

Из главы 8 мы знаем, что Алиса отправит Дине платеж в размере 50 000 сатоши через Боба и Чана. Этот платеж передается через серию HTLC-контрактов, как показано на рис. 10-8.

Как мы увидим в главе 11, Алиса может построить этот путь к Дине, потому что узлы Lightning объявляют о своих каналах всей сети Lightning, используя эпидемический протокол Lightning. После первоначального объявления канала Боб и Чан отправили дополнительное сообщение `channel_update` с указанием комиссионных за маршрутизацию и ожидаемых значений привязок ко времени для маршрутизирования платежей.

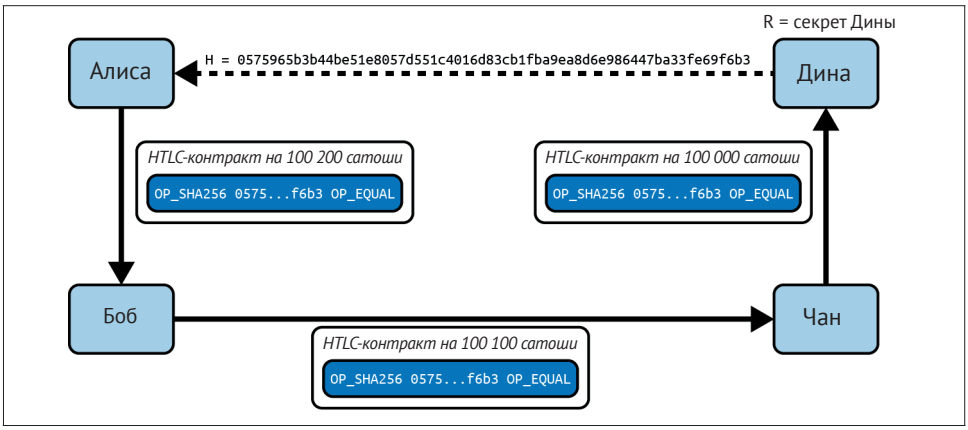


Рис. 10-8. Путь платежа с использованием HTLC-контрактов от Алисы к Дине

Из объявлений и обновлений Алиса знает следующую ниже информацию о каналах между Бобом, Чаном и Диной:

- `short_channel_id` (короткий ИД) для каждого канала, который Алиса может использовать для ссылки на канал при построении пути;
- `cltv_expiry_delta` (дельта привязки ко времени), которую Алиса может добавить ко времени истечения срока действия по каждому HTLC-контракту;
- `fee_base_msat` и `fee_proportional_millionths`, которые Алиса может использовать для расчета суммарных комиссионных за маршрутизацию, ожидаемых этим узлом за ретрансляцию по этому каналу.

На практике также осуществляется обмен другой информацией, такой как наибольшие (`htlc_maximum_msat`) и наименьшие (`htlc_minimum_msat`) HTLC-контракты, которые канал будет передавать, но они не используются столь непосредственно при построении луковичного маршрута, как предыдущие поля.

Эта информация используется Алисой для определения узлов, каналов, комиссионных и привязок ко времени для следующего ниже подробного пути, показанного на рис. 10-9.

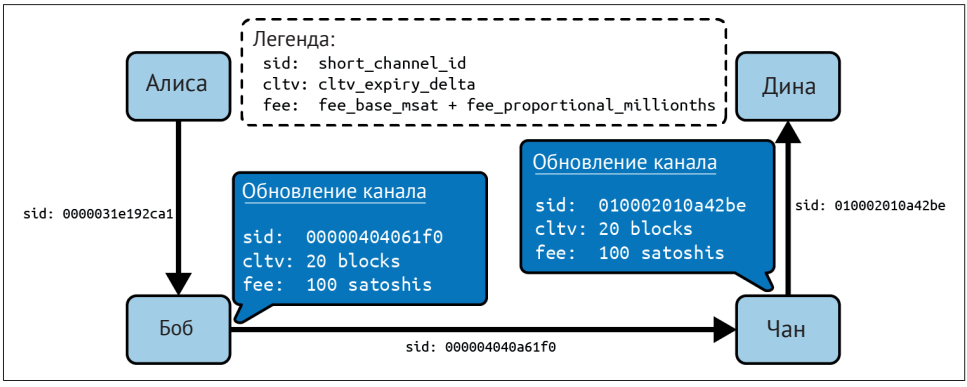


Рис. 10-9. Подробный путь, построенный из передаваемой по эпидемическому протоколу информации о канале и узле

Алиса уже знает свой собственный канал до Боба и поэтому не нуждается в этой информации для построения пути. Обратите также внимание, что Алисе не требовалось обновление канала от Дины, потому что у нее есть обновление от Чана для этого последнего канала в пути.

Алиса конструирует полезные грузы

Алиса может использовать два возможных формата для информации, передаваемой каждому переходу: устаревший формат фиксированной длины, именуемый данными перехода, и более гибкий формат, основанный на типе–длине–значении (Type-Length-Value, аббр. TLV), именуемый переходным полезным грузом. TLV-формат сообщения подробнее объясняется в разделе «Формат “тип–длина–значение”» на стр. 323. Он обеспечивает гибкость, позволяя добавлять поля в протокол по желанию.

Оба формата указаны в спецификации «BOLT #4: протокол луковичной маршрутизации, структура пакета»[72].

Алиса начнет собирать данные перехода с конца пути в обратном направлении: Дина, Чан, затем Боб.

Полезный груз для Дины в заключительном узле

Алиса сначала конструирует полезный груз, который будет доставлен Дине. Дина не будет создавать «исходящий HTLC-контракт», потому что она является заключительным узлом и получателем платежа. По этой причине полезный груз для Дины отличается от всех остальных (в нем для `short_channel_id` используются одни нули), но только Дина будет его знать, потому что он будет зашифрован в самом внутреннем слое луковицы. По сути, это «секретное письмо Дине», которое мы видели в нашем примере с физическим конвертом.

Переходный полезный груз для Дины должен соответствовать информации в счете, сгенерированном Диной для Алисы, и будет содержать (как минимум) следующие ниже поля в формате TLV:

`amt_to_forward`

Сумма этого платежа в миллисатошах. Если это только одна часть платежа, состоящего из нескольких частей, то эта сумма меньше общей суммы. В противном случае это единый полный платеж, и он равен сумме счета и значению `total_msat`.

`outgoing_cltv_value`

Привязка ко времени истечения срока платежа установлена равной значению `min_final_cltv_expiry` в счете.

[72] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-structure.

`payment_secret`

Специальное 256-битовое секретное значение из счета, позволяющее Дине распознать этот входящий платеж. Оно также предотвращает класс прощупывания, который ранее делал небезопасными счета с нулевой стоимостью. Прощупывание со стороны промежуточных узлов смягчается, поскольку это значение зашифровано только получателю, то есть он не может реконструировать заключительный пакет, который «выглядит» законным.

`total_msat`

Общая сумма, соответствующая счету. Она может быть опущена, если имеется только одна часть, и в этом случае предполагается, что она соответствует `am_to_forward` и должна равняться сумме счета.

В счете, который Алиса получила от Дины, указывалась сумма в 50 000 сатоши, что составляет 50 000 000 миллисатоши. Дина указала минимальный срок платежа min_final_cltv_expiry в 18 блоков (3 часа, учитывая в среднем 10-минутные Bitcoin-блоки). Допустим, что когда Алиса пыталась произвести платеж, блочная цепь Bitcoin зарегистрировала 700 000 блоков. Значит, Алиса должна установить значение outgoing_cltv_value равным минимальной высоте блока 700 018.

Алиса конструирует переходный полезный груз для Дины следующим образом:

```
amt_to_forward : 50,000,000
outgoing_cltv_value: 700,018
payment_secret: fb53d94b7b65580f75b98f10...03521bdab6d519143cd521d1b3826
total_msat: 50,000,000
```

Алиса сериализует его в формате TLV, как (упрощенно) показано на рис. 10-10.

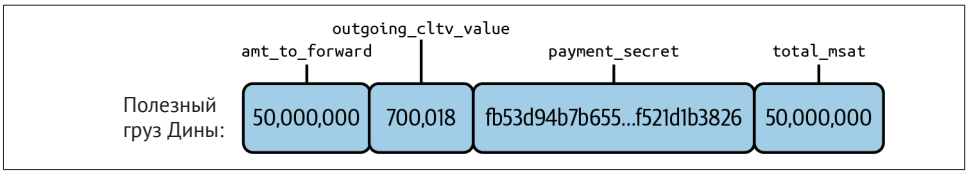


Рис. 10-10. Полезный груз Дины сконструирован Алисой

Переходный полезный груз для Чана

Затем Алиса конструирует переходный полезный груз для Чана. Он подскажет Чану, как настроить исходящий HTLC-контракт для Дины.

Переходный полезный груз для Чана содержит три поля: `short_channel_id`, `am_to_forward` и `outgoing_cltv_value`:

```
short_channel_id: 010002010a42be
amt_to_forward: 50,000,000
outgoing_cltv_value: 700,018
```

Алиса сериализует этот полезный груз в формате TLV, как (упрощенно) показано на рис. 10-11.

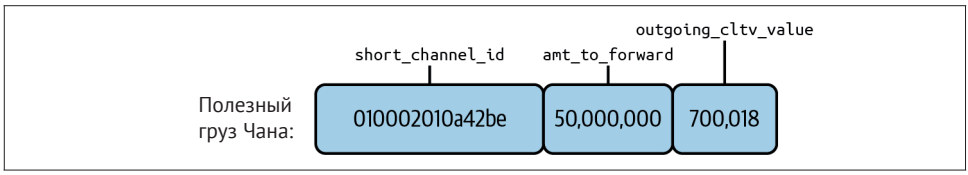


Рис. 10-11. Полезный груз Чана сконструирован Алисой

Переходный полезный груз для Боба

Наконец, Алиса конструирует переходный полезный груз для Боба, который также содержит те же три поля, что и переходный полезный груз для Чана, но с другими значениями:

```
short_channel_id: 000004040a61f0
amt_to_forward: 50,100,000
outgoing_cltv_value: 700,038
```

Как вы видите, поле `am_to_forward` равно 50 100 000 миллисатоши, или 50 100 сатоши. Это связано с тем, что Чан ожидает комиссионные в размере 100 сатоши за маршрутизирование платежа Дине. Для того чтобы Чан «заработал» эти комиссионные за маршрутизацию, входящий HTLC-контракт Чана должен быть на 100 сатоши больше, чем его исходящий HTLC-контракт. Поскольку входящий HTLC-контракт Чана – это исходящий HTLC-контракт Боба, то инструкции Бобу отражают плату, которую зарабатывает Чан. Проще говоря, Бобу нужно сказать, чтобы он отправил 50 100 сатоши Чану, чтобы Чан мог отправить 50 000 сатоши и оставить себе 100 сатоши.

Аналогичным образом Чан ожидает дельту привязки ко времени продолжительностью 20 блоков. Таким образом, входящий HTLC-контракт Чана должен истекать на 20 блоков позже, чем его исходящий HTLC-контракт. Для этого Алиса говорит Бобу, чтобы его исходящий HTLC-контракт для Чана истек на высоте блока 700 038 – на 20 блоков позже, чем HTLC-контракт Чана для Дины.

Ожидания в отношении дельты комиссионных и привязки ко времени для канала определяются разницей между входящим и исходящим HTLC-контрактами. Поскольку входящий HTLC-контракт создается предыдущим узлом, дельта комиссионных и привязки ко времени устанавливается в луковичном полезном грузе для этого предыдущего узла. Бобу сообщается, как создать HTLC-контракт, который соответствовал бы ожиданиям Чана в отношении комиссионных и привязки ко времени.

Алиса сериализует этот полезный груз в формате TLV, как (упрощенно) показано на рис. 10-12.

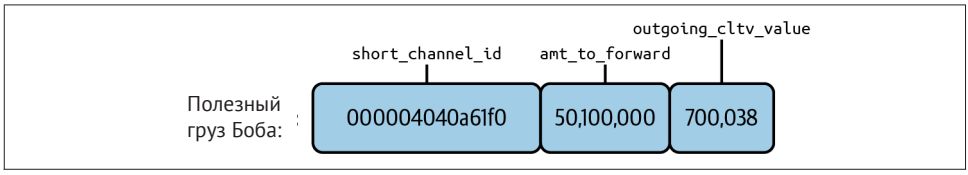


Рис. 10-12. Полезный груз Боба сконструирован Алисой

Окончательные полезные грузы переходов

Сейчас Алиса сконструировала полезные грузы для трех переходов, и эти полезные грузы будут завернуты в луковицу. Упрощенный вид полезных грузов показан на рис. 10-13.

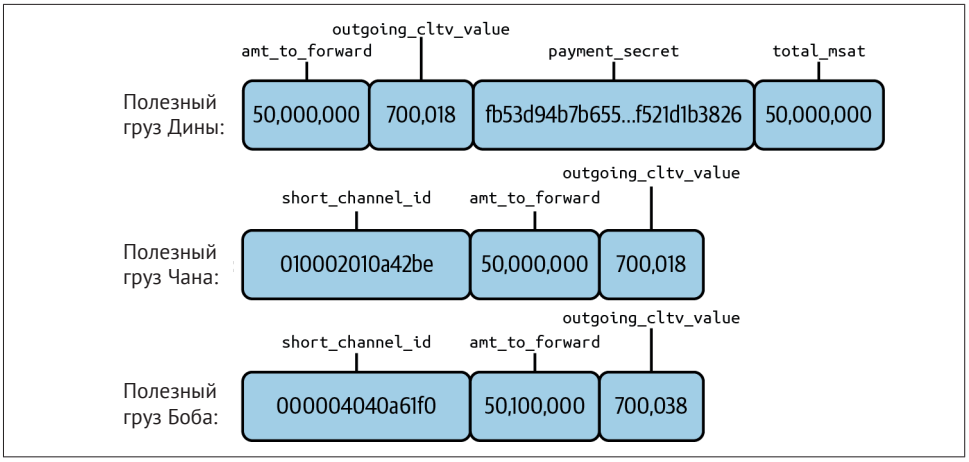


Рис. 10-13. Полезный груз для всех переходов

Генерация ключей

Теперь Алиса должна сгенерировать несколько ключей, которые будут использоваться для шифрования разных слоев в луковице.

С помощью этих ключей Алиса может обеспечить высокую степень конфиденциальности и целостности:

- Алиса может зашифровать каждый слой луковицы так, чтобы только намеченный получатель мог его прочитать;
- каждый посредник может проверить, что сообщение не изменено;
- никто на пути не будет знать, кто послал эту луковицу или куда она направляется. Алиса не раскрывает свою личность как отправителя платежа или личность Дины как его получателя;
- каждый переход узнает только о предыдущем и следующем переходах;
- никто не может знать длину пути или место, в котором он находится на этом пути.

Подобно нарезанному луку, следующие далее технические детали могут вызвать у вас на глазах слезы. Можете смело перейти к следующему разделу, если запутались. Вернитесь сюда и прочитайте спецификацию «BOLT #4: луковичная маршрутизация, конструирование пакетов»[73], если хотите узнать больше.

Основой всех используемых в луковице ключей является *совместный секрет*, который Алиса и Боб могут генерировать независимо друг от друга, ис-

[73] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction.

пользуя алгоритм Диффи–Хеллмана на эллиптической кривой (Elliptic Curve Diffie–Hellman, аббр. ECDH). Из совместного секрета (ss) они могут независимо генерировать четыре дополнительных ключа с именами `rho`, `mu`, `um` и `pad`:

`rho`

Используется для генерирования потока случайных байтов из потокового шифра (применяется алгоритм CSPRNG). Эти байты применяются для шифрования/дешифрования тела сообщения, а также как заполнительные нулевые байты во время обработки пакетов Sphinx.

`mu`

Используется в хеш-ориентированном коде аутентификации сообщений (hash-based message authentication code, аббр. HMAC) для верификации целостности/подлинности.

`um`

Применяется в отчетах об ошибках.

`pad`

Используется для генерирования заполнительных байтов для заполнения луковицы до фиксированной длины.

Взаимосвязь между разными ключами и тем, как они генерируются, показана на рис. 10-14.

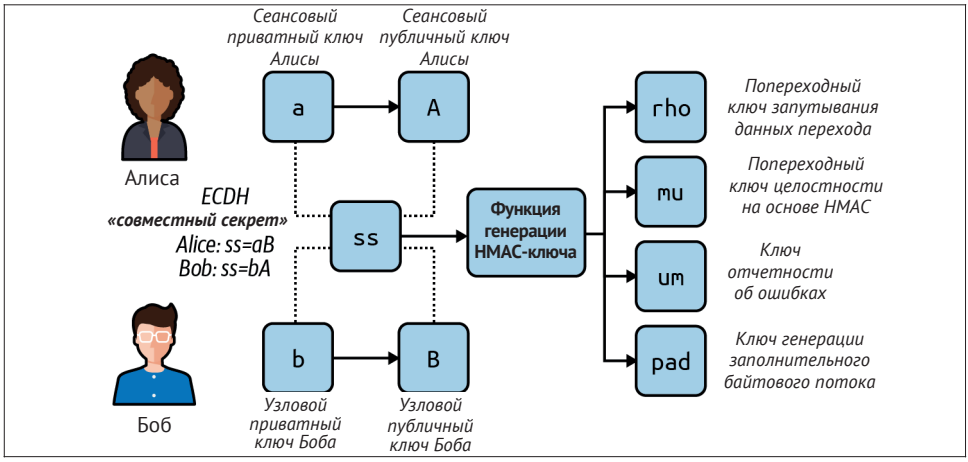


Рис. 10-14. Генерация луковичного ключа

Сеансовый ключ Алисы

Во избежание раскрытия своей личности при сборке луковицы Алиса не использует публичный ключ своего собственного узла. Вместо этого Алиса создает временный 32-байтовый (256-битовый) ключ, именуемый сеансовым приватным ключом, и соответствующий сеансовый публичный ключ. Это служит временной «идентификацией» и ключом только для этой луковицы. Из указанного сеансового ключа Алиса собирает все остальные ключи, которые будут использоваться в этой луковице.

Детали генерации ключей

Генерация ключей, генерация случайных байтов, эфемерные ключи и то, как они используются при конструировании пакетов, указано в трех разделах спецификации BOLT #4:

- генерация ключей[74];
- поток случайных байтов[75];
- конструирование пакетов[76].

Для простоты и во избежание излишних технических подробностей мы не включили эти детали в книгу. Смотрите приведенные выше ссылки, если хотите увидеть внутреннее устройство.

Генерация совместных секретов

Одна важная деталь выглядит почти волшебной, и это способность Алисы создавать совместный секрет с еще одним узлом, просто зная их публичные ключи. Это основано на изобретении 1970-х годов – обмене ключами Диффи–Хеллмана (DH), которое произвело революцию в криптографии. В луковой маршрутизации Lightning используется метод эллиптической кривой Диффи–Хеллмана (ECDH) на кривой `secp256k1` системы Bitcoin. Это настолько классный трюк, что мы попытаемся объяснить его простыми словами во вставке «Объяснение эллиптической кривой Диффи–Хеллмана» ниже.

Объяснение эллиптической кривой Диффи–Хеллмана

Допустим, что приватным ключом Алисы является a, а приватным ключом Боба является b. Используя эллиптическую кривую, Алиса и Боб умножают каждый свой приватный ключ на генераторную точку G, чтобы произвести свои публичные ключи соответственно A и B:

$$A = aG$$
$$B = bG$$

Теперь Алиса и Боб могут использовать обмен ключами на основе эллиптической кривой Диффи–Хеллмана, чтобы создать совместный секрет ss – значение, которое они оба могут вычислять независимо, без обмена какой-либо информацией.

Совместный секрет ss вычисляется каждым из них путем умножения их собственного приватного ключа на публичный ключ другого, так что:

$$ss = aB = bA$$

Но почему эти два произведения приводят к одному и тому же значению ss? Мотайте на ус, пока мы демонстрируем математику, которая доказывает, что это возможно:

[74] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#key-generation.

[75] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#pseudo-random-byte-stream.

[76] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction.

ss вычисляется Алисой, которая знает как *a* (ее приватный ключ), так и *B* (публичный ключ Боба):

ss

$= aB$

далее мы подставляем, поскольку мы знаем, что $B = bG$:

$= a(bG)$

далее мы можем переместить круглые скобки из-за ассоциативности:

$= (ab)G$

поскольку $xy = yx$ (кривая является абелевой группой):

$= (ba)G$

далее мы можем переместить круглые скобки из-за ассоциативности:

$= b(aG)$

и, наконец, мы можем подставить A вместо aG:

$= bA$

Результат bA может быть рассчитан независимо Бобом, который знает b (свой приватный ключ) и A (публичный ключ Алисы).

Таким образом, мы показали, что:

$ss = aB$ (Алиса может это рассчитать);

$ss = bA$ (Боб может это рассчитать).

Следовательно, каждый из них может независимо вычислять *ss*, который они могут использовать в качестве совместного ключа для симметричного шифрования секретов между ними двумя без передачи совместного секрета.

Уникальная особенность Sphinx как формата пакетов mix-net заключается в том, что, вместо того чтобы включать отдельный сеансовый ключ для каждого перехода по маршруту, что значительно увеличило бы размер пакета mix-net, используется хитроумная схема *ослепления*[77] для детерминированной рандомизации сеансового ключа при каждом переходе.

На практике этот маленький трюк позволяет держать луковичный пакет как можно более компактным, сохраняя при этом желаемые свойства безопасности.

Сеансовый ключ для перехода `i` выводится с использованием узлового публичного ключа и производного совместного секрета перехода `i - 1`:

```
session_key_i = session_key_{i-1} * SHA-256(node_pubkey_{i-1} ||
shared_secret_{i-1})
```

Другими словами, мы берем сеансовый ключ предыдущего перехода и умножаем его на значение, выведенное из публичного ключа и производного совместного секрета для этого перехода.

[77] Ослеплением (blinding) в Sphinx называется метод выведения сеансового ключа, основанный на многократно модифицируемом случайном элементе циклической группы простого порядка. – *Прим. перев.*

Поскольку умножение эллиптической кривой может выполняться с публичным ключом без знания приватного ключа, каждый переход может рерандомизировать сеансовый ключ для следующего перехода детерминированным способом.

Создатель луковичного пакета знает все совместные секреты (поскольку он зашифровал пакет уникально для каждого перехода) и, таким образом, может вывести все коэффициенты ослепления.

Эти знания позволяют выводить все сеансовые ключи, используемые заранее во время генерации пакетов.

Обратите внимание, что при самом первом переходе применяется исходный сгенерированный сеансовый ключ, поскольку этот ключ используется для запуска ослепления сеансового ключа при каждом последующем переходе.

Обертывание луковичных слоев

Процесс обертывания луковицы подробно описан в спецификации «BOLT #4: луковичная маршрутизация, конструирование пакета»[78].

В этом разделе мы опишем этот процесс на высоком и несколько упрощенном уровне, опустив некоторые детали.

Луковицы фиксированной длины

Мы уже упоминали тот факт, что ни один из «переходных» узлов (переходов) не знает ни длину пути, ни место, где они находятся на пути. Как это возможно?

Если у вас есть набор направлений, даже если они зашифрованы, разве нельзя определить, как далеко вы находитесь от начала или конца, просто посмотрев, где в списке направлений вы находитесь?

Используемая в луковичной маршрутизации хитрость заключается в том, чтобы всегда делать путь (список направлений) одинаковой длины для каждого узла. Это достигается за счет поддержания одинаковой длины луковичного пакета на каждом шаге.

При каждом переходе переходный полезный груз появляется в начале луковичного полезного груза, за которым, по-видимому, следует еще 19 переходов и их полезных грузов. Каждый переход воспринимается как первый из 20 переходов.

Луковичный полезный груз составляет 1300 байт. Полезный груз каждого перехода составляет 65 байт или меньше (дополняется до 65 байт, если меньше). Таким образом, суммарный луковичный полезный груз может вместить 20 переходных полезных грузов (1300 = 20 × 65). Следовательно, максимальный луковичный маршрутизируемый путь составляет 20 переходов.

По мере того как каждый слой «отслаивается», в конец полезного груза луковицы добавляется больше заполнительных данных (по сути, мусор), поэтому следующий переход получает луковицу того же размера и снова является «первым переходом» в луковице.

[78] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-construction.

Размер луковицы составляет 1366 байт, структурированный, как показано на рис. 10-15:

1 байт

Байт версии

33 байта

Сжатый сеансовый публичный ключ (см. раздел «Сеансовый ключ Алисы» на стр. 257), из которого можно генерировать попереходный совместный секрет (см. раздел «Генерация совместных секретов» на стр. 258), не раскрывая личности Алисы.

1300 байт

Фактический луковичный полезный груз, содержащий инструкции для каждого перехода.

32 байта

Контрольная сумма целостности на основе HMAC-кода.

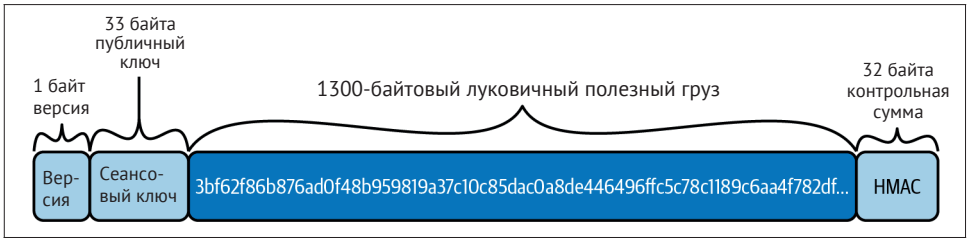


Рис. 10-15. Луковичный пакет

Уникальная особенность Sphinx как формата пакетов mix-net заключается в том, что, вместо того чтобы включать отдельный сеансовый ключ для каждого перехода по маршруту, что значительно увеличивало бы размер пакета mix-net, используется продуманная схема ослепления с целью детерминированной рандомизации сеансового ключа при каждом переходе.

На практике этот маленький трюк позволяет держать луковичный пакет как можно более компактным, поддерживая при этом желаемые свойства безопасности.

Обертывание луковицы (в общих чертах)

Вот описанный далее процесс обертывания луковицы. Вернитесь к этому перечню, когда мы рассмотрим каждый шаг на нашем примере из реального мира.

На каждом переходе отправитель (Алиса) повторяет один и тот же процесс:

1. Алиса генерирует совместный секрет и ключи `rho`, `mu` и `pad` для каждого перехода.
2. Алиса генерирует 1300 байт заполнителя и заполняет этим заполнителем 1300-байтовое поле полезного груза луковицы.
3. Алиса вычисляет HMAC-код для переходного полезного груза (нули для заключительного перехода).

4. Алиса вычисляет длину переходного полезного груза + HMAC-код + пробел для сохранения самой длины.
5. Алиса *сдвигает* луковичный полезный груз *вправо* на вычисленное пространство, необходимое для размещения переходного полезного груза. Крайние правые «заполнительные» данные отбрасываются, оставляя достаточно места слева для полезного груза.
6. Алиса вставляет длину + полезный груз + переход + HMAC-код в начале поля полезного груза в пространство, образованное в результате сдвига заполнителя.
7. Алиса использует ключ `rho` для генерирования одноразового заполнения размером 1300 байт.
8. Алиса запутывает всю полезную нагрузку луковицы путем применения к ней операции XOR с байтами, сгенерированными из `rho`.
9. Алиса вычисляет HMAC-код полезного груза луковицы, используя ключ `mu`.
10. Алиса добавляет сеансовый публичный ключ (чтобы пользователь мог вычислить совместный секрет).
11. Алиса добавляет номер версии.
12. Алиса детерминистически повторно ослепляет сеансовый ключ, используя значение, выведенное путем хеширования совместного секрета и публичного ключа предыдущего перехода.

Затем Алиса повторяет процесс. Вычисляются новые ключи, луковичный полезный груз сдвигается (отбрасывая больше мусора), новый переходный полезный груз добавляется спереди, и весь луковичный полезный груз шифруется потоком байтов `rho` для следующего перехода.

Для заключительного перехода HMAC-код, включенный в шаг № 3 поверх текстовых инструкций, фактически равен нулю. Заключительный переход использует этот сигнал, чтобы определить, что это действительно последний переход в маршруте. В качестве альтернативы также можно использовать тот факт, что идентификатор `short_chan_id`, включенный в полезный груз для обозначения «следующего перехода», равен нулю.

Обратите внимание, что в каждой фазе ключ `mu` используется для генерирования HMAC-кода поверх зашифрованного (с точки зрения узла, обрабатывающего полезный груз) луковичного пакета, а также поверх содержимого пакета с удаленным одним уровнем шифрования. Этот внешний HMAC-код позволяет обрабатывающему пакет узлу проверять целостность луковичного пакета (байты не изменены). Затем внутренний HMAC-код раскрывается во время выполнения процедуры, обратной описанной ранее процедуре «сдвига и шифровки», которая служит *внешним* HMAC-кодом для следующего перехода.

Обертывание переходного полезного груза Дины

Как напоминание луковица обертывается, начиная с конца пути от Дины, заключительного узла или получателя. Затем путь строится в обратном порядке вплоть до отправителя, Алисы.

Алиса начинает с пустого 1300-байтового поля, луковичного полезного груза фиксированной длины. Затем она заполняет луковичный полезный груз «заполнителем» из потока псевдослучайных байтов, который генерируется с помощью ключа pad.

Это показано на рис. 10-16.

В генерации потока случайных байтов используется алгоритм ChaCha20, который выступает в качестве криптографически защищенного генератора псевдослучайных чисел (CSPRNG). Такой алгоритм будет генерировать детерминированный, длинный, неповторяющийся поток кажущихся случайными байтов из начального числа. Подробности указаны в спецификации «BOLT #4: луковичая маршрутизация, поток псевдослучайных байтов»[79].

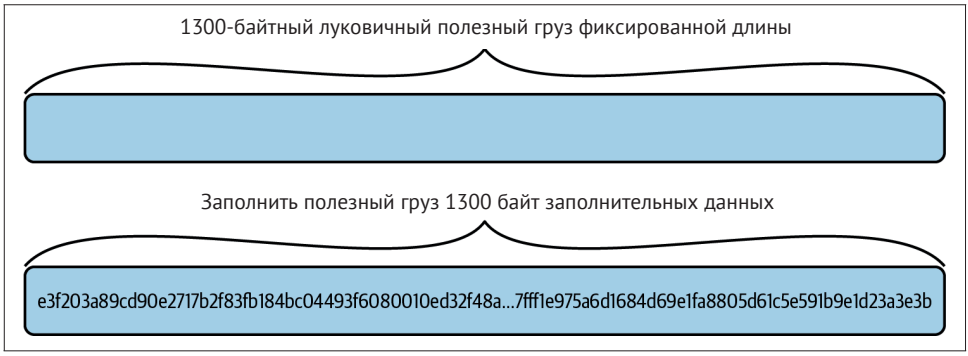


Рис. 10-16. Заполнение полезного груза луковицы потоком случайных байтов

Теперь Алиса вставит переходный полезный груз Дины в левую часть 1300-байтового массива, сдвигая заполнитель вправо и отбрасывая все лишнее. Это наглядно показано на рис. 10-17.

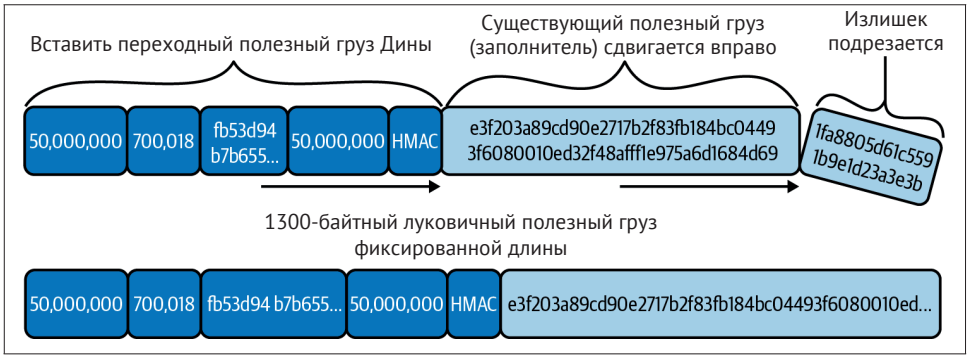


Рис. 10-17. Добавление переходного полезного груза Дины

На это можно взглянуть и по-другому, как на то, что Алиса измеряет длину переходного полезного груза Дины, сдвигает заполнитель вправо, чтобы создать равное пространство в левой части луковичного полезного груза, и вставляет полезный груз Дины в это пространство.

[79] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#pseudo-random-byte-stream.

Строкой ниже мы видим результат: 1300-байтовый луковичный полезный груз содержит переходный полезный груз Дины, а затем поток заполнительных байтов заполняет остальное пространство.

Затем Алиса запутывает весь луковичный полезный груз, чтобы только Дина смогла его прочитать.

Для этого Алиса генерирует поток байтов, применяя ключ `rho` (который также знает Дина). Алиса использует побитовое исключающее ИЛИ (XOR) между битами луковичного полезного груза и потоком байтов, созданным из `rho`. Результат выглядит как случайный (или зашифрованный) поток байтов длиной 1300 байт. Этот шаг показан на рис. 10-18.

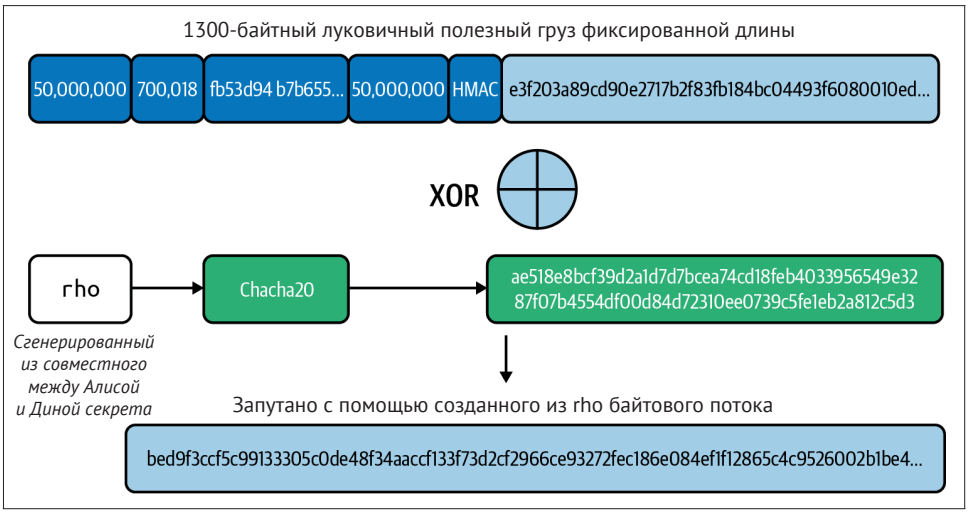


Рис. 10-18. Запутывание луковичного полезного груза

Одним из свойств операции XOR является то, что если выполнить ее дважды, то можно вернуться к изначальным данным. Как мы увидим в разделе «Боб распутывает свой переходный полезный груз» на стр. 271, если Дина применит ту же операцию XOR к байтовому потоку, сгенерированному из `rho`, то в результате получит изначальный луковичный полезный груз.

XOR – это инволюторная функция, которая означает, что если она применяется дважды, то она себя отменяет. В частности, XOR(XOR(a, b), b) = a. Это свойство широко используется в криптографии с симметричным ключом.

Поскольку только у Алисы и Дины есть ключ `rho` (выведенный из совместного секрета Алисы и Дины), лишь они могут это сделать. По сути, за счет этого шифруется луковичный полезный груз только для глаз Дины.

Наконец, Алиса вычисляет хеш-код аутентификации сообщения (HMAC) для полезного груза Дины, в котором ключ `mu` используется в качестве ключа инициализации. Это показано на рис. 10-19.

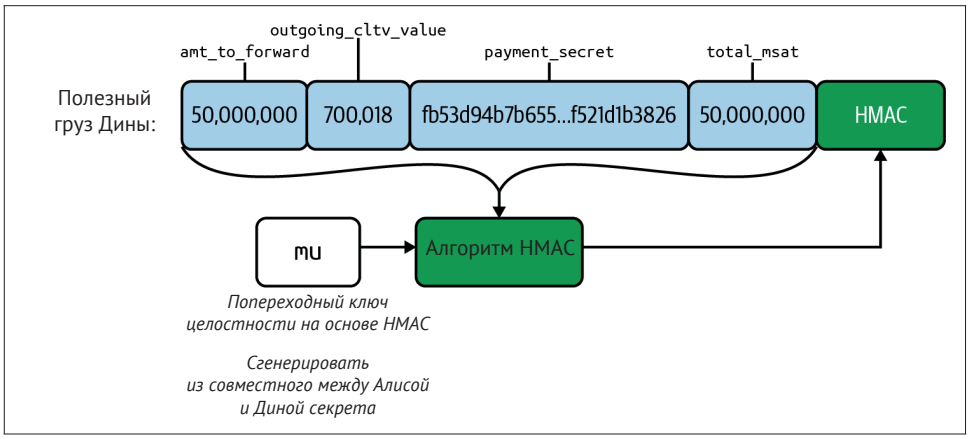


Рис. 10-19. Добавление контрольной суммы целостности на основе HMAC-кода в переходный полезный груз Дины

Луковично-маршрутизационная защита от повторного воспроизведения и его обнаружение

HMAC-код действует как безопасная контрольная сумма и помогает Дине верифицировать целостность переходного полезного груза. 32-байтовый HMAC-код добавляется в переходный полезный груз Дины. Обратите внимание, что мы вычисляем HMAC-код по шифроданным, а не по обычным текстовым данным. Это называется «зашифровать и затем аутентифицировать» (encrypt-then-mac) и является рекомендуемым способом использования MAC, поскольку обеспечивает целостность как обычного текста, так и шифротекста.

Современное аутентифицируемое шифрование позволяет использовать дополнительный набор байтов обычного текста, также подлежащих аутентификации, именуемый *ассоциированными данными*. На практике это обычно что-то вроде заголовка пакета в виде обычного текста или другой вспомогательной информации. Включив эти ассоциированные данные в полезную нагрузку, подлежащую аутентификации (MAC'ed), верификатор MAC обеспечивает, чтобы эти ассоциированные данные не были изменены (например, замена заголовка из обычного текста на зашифрованном пакете).

В контексте сети Lightning указанные ассоциированные данные используются для усиления защиты от повторного воспроизведения этой схемы. Как мы узнаем далее, защита от воспроизведения обеспечивает невозможность ретрансляции (воспроизведения) пакета в сеть и прослеживания его результирующего пути со стороны злоумышленника. Вместо этого промежуточные узлы могут использовать определенные меры защиты от воспроизведения с целью обнаружения и отклонения воспроизведенного пакета. Базовый формат пакета Sphinx использует журнал из всех эфемерных секретных ключей, применяемых для обнаружения воспроизведений. Если секретный ключ когда-либо будет использован снова, то узел сможет его обнаружить и отклонить пакет.

Природа HTLC-контрактов в сети Lightning позволяет еще больше усиливать защиту от воспроизведения, добавив дополнительный *экономический* стимул.

Напомним, что платежный хеш HTLC-контракта можно безопасно использовать (для полного платежа) только один раз. Если платежный хеш используется снова и пересекает узел, который уже видел платежный секрет для этого хеша, тогда можно просто вытянуть средства и забрать всю сумму платежа без пересылки! Этим фактом можно воспользоваться для укрепления защиты от воспроизведения, требуя, чтобы платежный хеш был включен в наши HMAC-вычисления в качестве ассоциированных данных. За счет этого дополнительного шага попытка воспроизведения луковичного пакета также требует, чтобы отправитель был привязан к использованию того же платежного хеша. Как следствие, помимо обычной защиты от воспроизведения, злоумышленник также может потерять всю сумму воспроизводимого HTLC-контракта.

Одно из соображений, связанных с постоянно растущим набором сеансовых ключей, хранящихся для защиты от воспроизведения, заключается в следующем: могут ли узлы истребовать это пространство назад? В контексте сети Lightning ответ таков: да! Еще раз, благодаря уникальным свойствам конструкции HTLC-контракта, мы можем вносить дополнительные улучшения по сравнению с базовым протоколом Sphinx. Учитывая, что HTLC-контракты являются контрактами с привязкой ко времени, основанными на абсолютной высоте блока, по истечении срока действия HTLC-контракта он фактически закрывается навсегда. Как следствие узлы могут использовать эту основанную на CLTV (операторе CHECKLOCKTIMEVERIFY) высоту истечения срока в качестве индикатора, чтобы знать, когда запись в журнале защиты от воспроизведения можно безопасно освободить (с последующей ее обработкой сборщиком мусора).

Обертывание переходного полезного груза Чана

На рис. 10-20 мы видим шаги, применяемые для обертывания переходного полезного груза Чана. Это те же самые шаги, которые Алиса использовала для обертывания переходного полезного груза Дины.

Алиса начинает с 1300-байтового луковичного полезного груза, созданного для Дины. Его первые 65 (или меньше) байт являются запутанным полезным грузом Дины, а остальное – заполнителем. Алиса должна соблюдать осторожность, чтобы не перезаписать полезный груз Дины.

Затем Алисе нужно найти эфемерный публичный ключ (который был сгенерирован в самом начале для каждого перехода), который будет добавлен в начало маршрутизационного пакета в этом переходе.

Напомним, что вместо того чтобы включать уникальный эфемерный публичный ключ (который отправитель и промежуточный узел используют в операции ECDH для генерирования совместного секрета), в Sphinx применяется единый эфемерный публичный ключ, который детерминированно рандомизируется в каждом переходе.

При обработке пакета Дина будет использовать свой совместный секрет и публичный ключ, чтобы вывести значение ослепления (`b_dina`) и использовать его для рерандомизации эфемерного публичного ключа в операции, идентичной той, которую Алиса выполняет во время первоначального конструирования пакета.

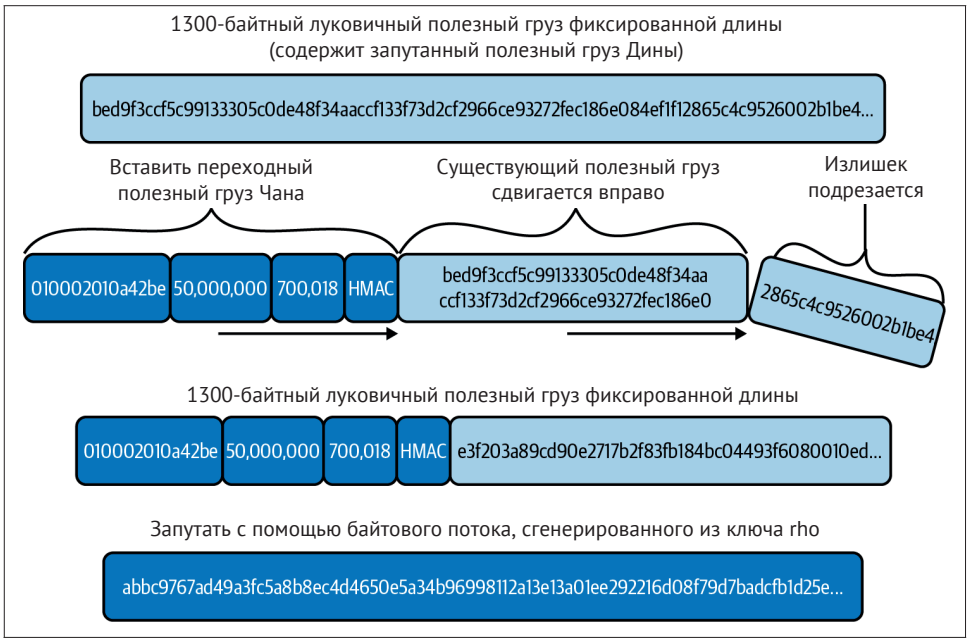


Рис. 10-20. Обертывание луковицы для Чана

Алиса добавляет внутреннюю контрольную сумму HMAC в полезный груз Чана и вставляет его в «начало» (в левую сторону) луковичного полезного груза, сдвигая существующий полезный груз вправо на равную величину. Напомним, что в этой схеме фактически используются два HMAC-кода: внешний HMAC и внутренний HMAC. В данном случае внутренний HMAC-код Чана на самом деле является внешним HMAC-кодом Дины.

Теперь полезный груз Чана находится перед луковицей. Когда Чан его видит, он понятия не имеет, сколько полезных грузов было до или после. Это переход всегда выглядит как первый из 20 переходов!

Затем Алиса запутывает весь полезный груз с помощью операции XOR с привлечением байтового потока, сгенерированного из ключа `rho` Алисы–Чана. Только Алиса и Чан имеют этот ключ `rho`, и только они могут производить поток байтов для запутывания и распутывания луковицы. Наконец, как мы делали на предыдущем шаге, мы вычисляем внешний HMAC-код Чана, который она будет использовать для проверки целостности зашифрованного луковичного пакета.

Обертывание переходного полезного груза Боба

На рис. 10-21 мы видим шаги, используемые для обертывания переходного полезного груза Боба в луковице.

Ладно, теперь это уже просто!

Начать с луковичного (запутанного) полезного груза, содержащего переходные полезные грузы Чана и Дины.

Получить сеансовый ключ для этого перехода, выведенный из коэффициента ослепления, сгенерированного предыдущим переходом. Включить внешний HMAC-код предыдущего перехода в качестве внутреннего HMAC-кода этого перехода. Вставить переходный полезный груз Боба в начало и сдвинуть все остальное вправо, отбросив фрагмент размером с переходный полезный груз Боба с конца (в любом случае это был заполнитель).

Запутать все это операцией XOR с ключом `rho` из совместного секрета Алисы и Боба, чтобы только Боб мог это развернуть.

Рассчитать внешний HMAC-код и прикрепить его к концу переходного полезного груза Боба.

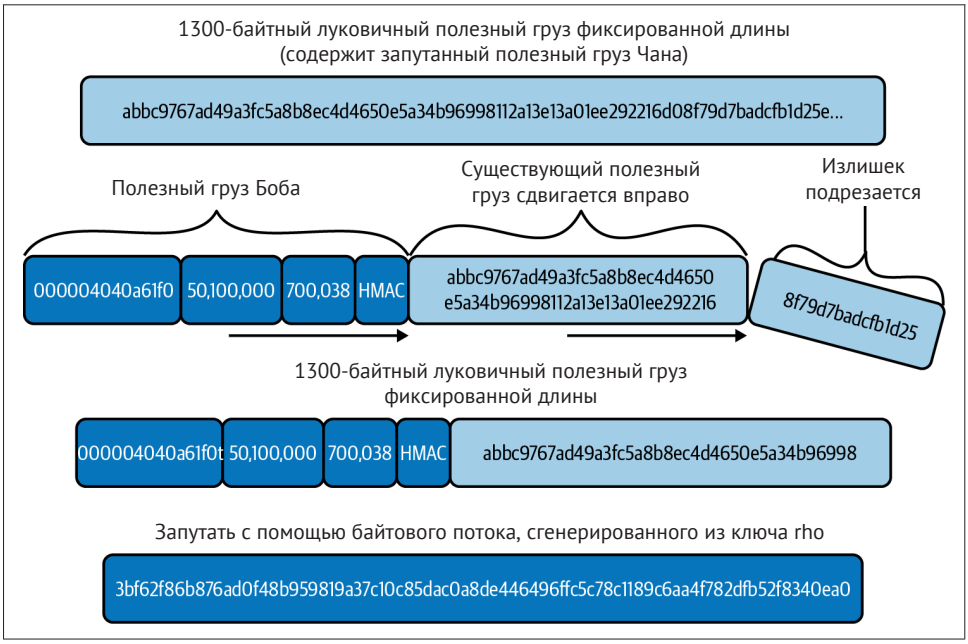


Рис. 10-21. Обертывание луковицы для Боба

Заключительный луковичный пакет

Заключительный луковичный полезный груз готов к отправке Бобу. Алисе больше не нужно добавлять никаких переходных полезных грузов.

Алиса вычисляет HMAC-код для луковичного полезного груза, чтобы криптографически защитить его контрольной суммой, которую Боб может верифицировать.

Алиса добавляет 33-байтовый публичный сеансовый ключ, который будет использоваться при каждом переходе для генерирования совместного секрета, а также ключей rho, mu и pad.

Наконец, Алиса помещает номер версии луковицы (в настоящее время 0) спереди. Это позволит в будущем обновить формат луковичных пакетов.

Результат можно увидеть на рис. 10-22.

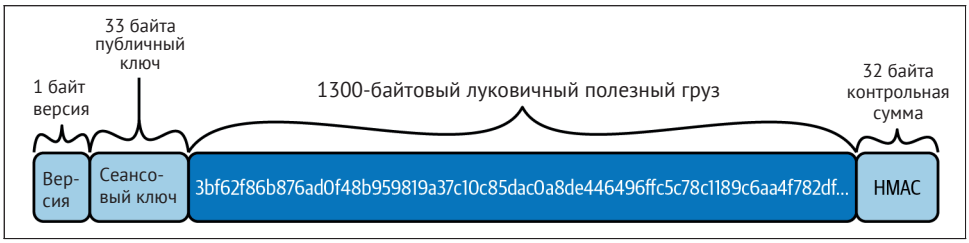


Рис. 10-22. Луковичный пакет

Отправка луковицы

В этом разделе мы рассмотрим принцип работы пересылки луковичного пакета и то, как HTLC-контракты развертываются по пути.

Сообщение update_add_htlc

Луковичные пакеты отправляются как часть сообщения `update_add_htlc`. Если вы помните из раздела «Сообщение update_add_HTLC» на стр. 231, в главе 9, мы увидели, что содержимое сообщения `update_add_htlc` выглядит следующим образом:

```
[channel_id:channel_id]
[u64:id]
[u64:amount_msat]
[sha256:payment_hash]
[u32:cltv_expiry]
[1366*byte:onion_routing_packet]
```

Вы помните, что это сообщение отправляется одним партнером по каналу, чтобы попросить другого партнера по каналу добавить HTLC-контракт. Вот как Алиса попросит Боба добавить HTLC-контракт, чтобы заплатить Дине. Теперь вы понимаете предназначение последнего поля, `onion_routing_packet`, длина которого составляет 1366 байт. Это полностью обернутый луковичный пакет, который мы только что сконструировали!

Алиса отправляет луковицу Бобу

Алиса отправит Бобу сообщение `update_add_htlc`. Давайте посмотрим, что это сообщение будет содержать:

`channel_id`

Это поле содержит ИД канала Алисы–Боба, который в нашем примере равен `0000031e192ca1` (см. рис. 10-9).

`id`

ИД этого HTLC-контракта в этом канале, начинающийся с 0.

`amount_msat`

Сумма HTLC-контракта: 50 200 000 миллисатоши.

`payment_hash`

Платежный хеш RIPEMD160(SHA-256):

`9e017f6767971ed7cea17f98528d5f5c0ccb2c71`.

`cltv_expiry`

Привязка ко времени истечения срока для HTLC-контракта составит 700 058. Алиса добавляет 20 блоков к сроку действия, установленному в полезном грузе Боба, в соответствии с согласованной Бобом дельтой `cltv_expiry_delta`.

`onion_routing_packet`

Окончательный луковичный пакет, сконструированный Алисой со всеми переходными полезными грузами!

Боб проверяет луковицу

Как мы увидели в главе 9, Боб добавит HTLC-контракт к фиксационным транзакциям и обновит состояние канала с Алисой.

Боб развернет луковицу, которую он получил от Алисы, следующим образом:

1. Боб берет сеансовый ключ из луковичного пакета и извлекает совместный секрет Алисы и Боба.
2. Боб генерирует ключ `mu` из совместного секрета и использует для верификации контрольной суммы HMAC-код луковичного пакета.

Теперь, когда Боб сгенерировал совместный ключ и подтвердил HMAC-код, он может начать разворачивать 1300-байтовый луковичный полезный груз внутри луковичного пакета. Цель Боба – извлечь свой собственный переходный полезный груз, а затем переслать оставшуюся луковицу в следующий переход.

Если Боб извлечет и удалит свой переходный полезный груз, то оставшаяся луковица не будет составлять 1300 байт, она будет короче! И поэтому следующий переход будет знать, что это не первый переход, и сможет определить длину пути. Во избежание этого Бобу нужно добавить больше заполнителя, чтобы заполнить луковицу.

Боб генерирует заполнитель

Боб генерирует заполнитель несколько иным способом, чем Алиса, но следуя тому же общему принципу.

Во-первых, Боб расширяет луковичный полезный груз на 1300 байт и заполняет их значениями 0. Теперь длина луковичного пакета составляет 2600 байт, причем первая половина содержит данные, отправленные Алисой, а следующая половина содержит нули. Эта операция показана на рис. 10-23.

Это пустое пространство станет запутанным и превратится в «заполнитель» с помощью того же процесса, который Боб использует для распутывания своего собственного переходного полезного груза. Давайте посмотрим, как это работает.

Рис. 10-23. Боб расширяет луковичный полезный груз на 1300 байт (нулями)

Боб распутывает свой переходный полезный груз

Затем Боб сгенерирует ключ `rho` из совместного между Алисой и Бобом ключа. Он будет использовать его для генерирования потока из 2600 байт, используя алгоритм ChaCha20.

Первые 1300 байт байтового потока, сгенерированного Бобом, точно такие же, как и те, которые сгенерированы Алисой с использованием ключа `rho`.

Затем Боб применяет 2600 байт байтового потока `rho` к 2600-байтовому луковичному полезному грузу с помощью побитовой операции XOR.

Эта операция XOR распутает первые 1300 байт, потому что это та же самая операция, которую применила Алиса, а XOR – инволюторная операция. Таким образом, Боб раскроет свой переходный полезный груз, за которым следуют некоторые данные, которые кажутся зашифрованными.

В то же время применение байтового потока `rho` к 1300 нулям, которые были добавлены в луковичный полезный груз, превратит их в, казалось бы, случайные заполнительные данные. Эта операция показана на рис. 10-24.

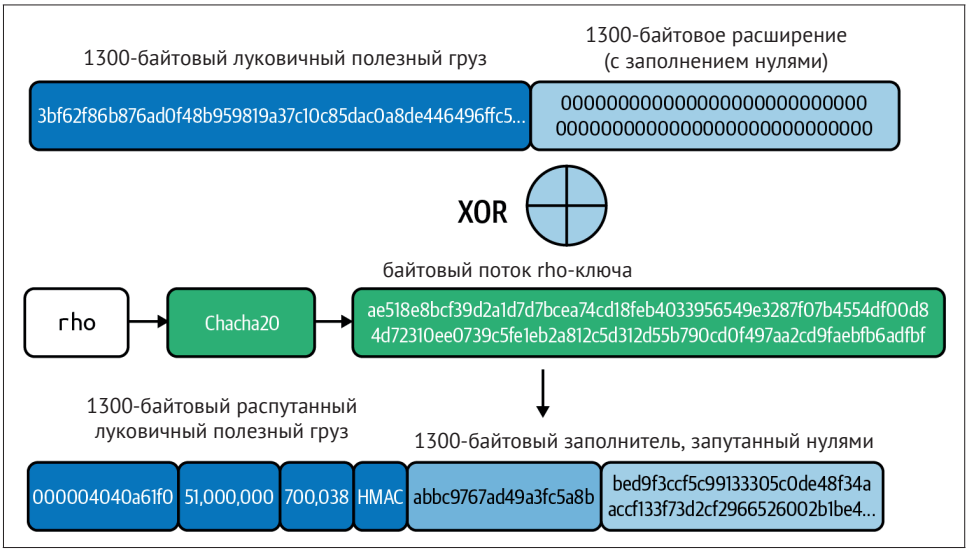


Рис. 10-24. Боб распутывает луковицу, запутывает заполнитель

Боб извлекает внешний HMAC для следующего перехода

Напомним, что в состав каждого перехода включается внутренний HMAC-код, который затем станет внешним HMAC-кодом для следующего перехода. В этом случае Боб извлекает внутренний HMAC-код (он уже проверил целостность зашифрованного пакета с помощью внешнего HMAC-кода) и откладывает его в сторону, потому что он добавит его к распутанному пакету, чтобы дать возможность Чану проверить HMAC-код своего зашифрованного пакета.

Боб удаляет свой полезный груз и сдвигает луковицу влево

Теперь Боб может удалить свой переходный полезный груз спереди луковицы и сдвинуть оставшиеся данные влево. Объем данных, равный переходному полезному грузу Боба из второй половины 1300 байт заполнителя, уже переместится в пространство луковичного полезного груза. Это показано на рис. 10-25.

Теперь Боб может оставить первую половину 1300 байт и отбросить расширенные (заполнительные) 1300 байт.

У Боба уже есть 1300-байтовый луковичный пакет для отправки на следующий переход. Он почти идентичен луковичному полезному грузу, который Алиса сконструировала для Чана, за исключением того, что последние 65 или около того байт заполнителя были помещены туда Бобом и будут отличаться.

Никто не может отличить заполнитель, положенный туда Алисой, от заполнителя, положенного туда Бобом. Заполнитель и в Африке заполнитель! В любом случае, это все случайные байты. Обратите внимание, что если Боб (или один из других псевдонимов Боба) присутствует в маршруте в двух разных местах, то они могут заметить разницу, потому что в базовом протоколе всегда используется один и тот же платежный хеш по всему маршруту. Атомарные многопутевые платежи (Atomic multipath payments, аббр. AMP) и контракты

с точечной привязкой ко времени (PTLC) устраняют вектор корреляции путем рандомизации платежного идентификатора по каждому маршруту/переходу.

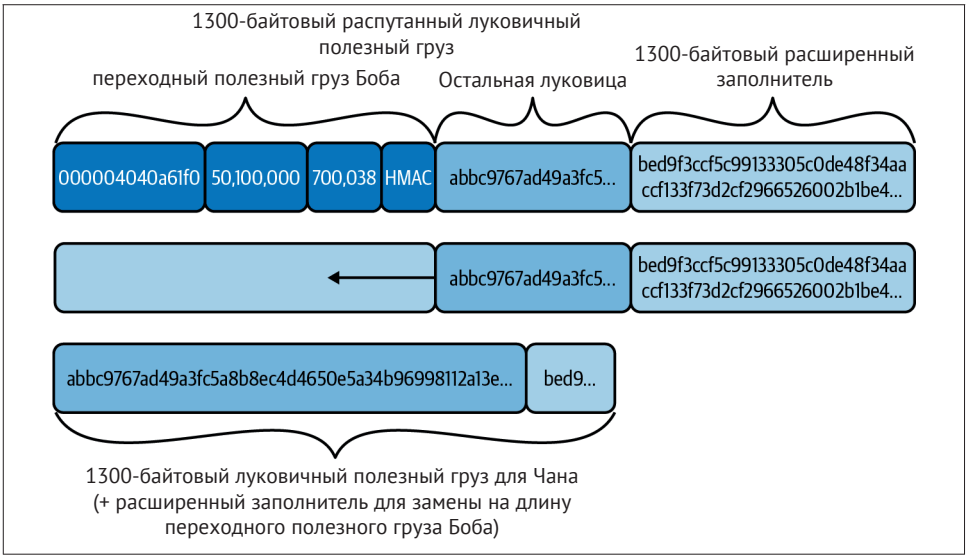


Рис. 10-25. Боб удаляет переходный полезный груз и сдвигает остальное влево, заполняя пробел новым заполнителем

Боб конструирует новый луковичный пакет

Теперь Боб копирует луковичный полезный груз в луковичный пакет, добавляет внешний HMAC-код для Чана, рерандомизирует сеансовый ключ (так же, как это делает отправитель Алиса) с помощью операции умножения на эллиптическую кривую и добавляет байт новой версии.

Для того чтобы рерандомизировать сеансовый ключ, Боб сначала вычисляет коэффициент ослепления для своего перехода, используя публичный ключ своего узла и выведенный им совместный секрет:

```
b_bob = SHA-256(P_bob || shared_secret_bob)
```

Сгенерировав его, Боб теперь рерандомизирует сеансовый ключ, выполняя умножение на эллиптическую кривую с использованием своего сеансового ключа и коэффициента ослепления:

```
session_key_chan = session_key_bob * b_bob
```

Публичный ключ `session_key_chan` затем будет добавлен в начало луковичного пакета для обработки Чаном.

Боб верифицирует детали HTLC-контракта

Переходный полезный груз Боба содержит инструкции, необходимые для создания HTLC-контракта для Чана.

В переходном полезном грузе Боб находит `short_channel_id`, `am_to_forward` и `cltv_expiry`.

Сначала Боб проверяет, есть ли у него канал с таким коротким идентификатором. Он обнаруживает, что у него есть такой канал с Чаном.

Затем Боб подтверждает, что исходящая сумма (50 100 сатоши) меньше входящей суммы (50 200 сатоши), и, следовательно, ожидания Боба в отношении комиссионных оправдались.

Точно так же Боб проверяет, что исходящий `cltv_expiry` меньше, чем входящий `cltv_expiry`, давая Бобу достаточно времени, чтобы истребовать входящий HTLC-контракт, если есть нарушение.

Боб отправляет update_add_htlc Чану

Боб теперь конструирует HTLC-контракт для отправки Чану следующим образом:

`channel_id`

Это поле содержит идентификатор канала Боба–Чана, который в нашем примере равен `000004040a61f0` (см. рис. 10-9).

`id`

Идентификатор этого HTLC-контракта в этом канале, начинающийся с 0.

`amount_msat`

Сумма HTLC-контракта: 50 100 000 миллисатоши.

`payment_hash`

Платежный хеш RIPEMD160(SHA-256):

`9e017f6767971ed7cea17f98528d5f5c0ccb2c71`.

Он точно такой же, что и платежный хеш из HTLC-контракта Алисы.

`cltv_expiry`

Привязка ко времени истечения срока для HTLC-контракта составит 700 038.

`onion_routing_packet`

Луковичный пакет, который Боб реконструировал после удаления своего переходного полезного груза.

Чан пересылает луковицу

Чан повторяет тот же самый процесс, что и Боб:

1. Чан получает `update_add_htlc` и обрабатывает запрос на HTLC-контракт, добавляя его в фиксационные транзакции.
2. Чан генерирует совместный ключ Алисы–Чана и подключ `mu`.
3. Чан проверяет HMAC-код луковичного пакета, затем извлекает 1300-байтовый луковичный полезный груз.
4. Чан расширяет луковичный полезный груз на 1300 дополнительных байт, заполняя его нулями.
5. Чан использует ключ `rho`, чтобы сгенерировать 2600 байт.
6. Чан использует сгенерированный байтовый поток, чтобы применить операцию XOR и распутать луковичный полезный груз. Одновременно операция XOR запутывает дополнительные 1300 нулей, превращая их в заполнитель.

7. Чан извлекает внутренний HMAC-код из полезного груза, который станет внешним HMAC-кодом для Дины.
8. Чан удаляет свой переходный полезный груз и сдвигает влево луковичный полезный груз на ту же величину. Часть заполнителя, сгенерированного в 1300 расширенных байтах, перемещается в первую половину 1300 байт, становясь частью луковичного полезного груза.
9. Чан конструирует луковичный пакет для Дины с помощью этого луковичного полезного груза.
10. Чан собирает сообщение `update_add_htlc` для Дины и вставляет в него луковичный пакет.
11. Чан отправляет `update_add_htlc` Дине.
12. Чан рерандомизирует сеансовый ключ, как это сделал Боб в предыдущем переходе для Дины.

Дина получает заключительный полезный груз

Когда Дина получает сообщение `update_add_htlc` от Чана, она знает из `payment_hash`, что это платеж для нее. Она знает, что она – последний переход в луковице.

Дина следует тому же процессу, что и Боб и Чан, чтобы верифицировать и развернуть луковицу, за исключением того, что она не конструирует новый заполнитель и ничего не пересылает. Вместо этого Дина отвечает Чану с помощью `update_fulfill_htlc`, чтобы погасить HTLC-контракт. Сообщение `update_fulfill_htlc` будет течь в обратном направлении пути до тех пор, пока не достигнет Алисы. Все HTLC-контракты погашаются, а остатки каналов обновляются. Оплата завершена!

Возвращение ошибок

До сих пор мы рассматривали прямое распространение луковицы, устанавливающее HTLC-контракты, и обратное распространение платежного секрета, разматывающее HTLC-контракты после успешного выполнения платежа.

Существует еще одна очень важная функция луковичной маршрутизации: возврат ошибки. Если есть проблема с платежом, луковицей или переходами, то мы должны распространить ошибку в обратном направлении, чтобы сообщить всем узлам о сбое и размотать все HTLC-контракты.

Ошибки обычно делятся на три категории: сбои луковицы, сбои узла и сбои канала. Кроме того, они могут быть разделены на постоянные и преходящие ошибки. Наконец, некоторые ошибки содержат обновления канала, которые помогут при будущих попытках доставки платежей.

В отличие от сообщений в одноранговом протоколе (P2P) (определенном в спецификации «BOLT #2: одноранговый протокол для управления каналами»[80]), ошибки не отправляются как сообщения P2P, а заключаются в луковичные пакеты возврата и следуют обратному луковичному пути (распространяясь в обратную сторону).

[80] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md.

Возврат ошибки определен в спецификации «BOLT #4: луковичная маршрутизация, возвращение ошибок»[81].

Ошибки кодируются возвращающим узлом (тем, который обнаружил ошибку) в пакете возврата следующим образом:

```
[32*byte:hmac]
[u16:failure_len]
[failure_len*byte:failuremsg]
[u16:pad_len]
[pad_len*byte:pad]
```

Контрольная сумма целостности на основе HMAC-кода пакета возврата вычисляется с помощью ключа `um`, сгенерированного из совместного секрета, установленного луковицей.

Имя ключа `um` является обратным имени `mu`, указывая на то же использование, но в противоположном направлении (обратное распространение).

Затем возвращающий узел генерирует ключ `ammag` (обратный слову «gamma») и запутывает пакет возврата, используя операцию XOR с байтовым потоком, сгенерированным из `ammag`.

Наконец, возвращающий узел отправляет пакет возврата переходному узлу, из которого он получил изначальную луковицу.

Каждый получающий ошибку переход генерирует ключ `ammag` и снова запутывает пакет возврата, используя операцию XOR с байтовым потоком из `ammag`.

В конечном счете отправитель (исходный узел) получает пакет возврата. Затем он генерирует ключи `ammag` и `um` для каждого перехода, а XOR будет итеративно распутывать возвращаемую ошибку до тех пор, пока он не раскроет пакет возврата.

Сообщения о сбоях

Сообщение `failuremsg` определено в спецификации «BOLT #4: луковичная маршрутизация, сообщения о сбоях»[82].

Сообщение о сбое состоит из двухбайтового кода сбоя, за которым следуют данные, применимые к этому типу сбоя.

Верхний байт кода failure_code – это набор двоичных флагов, которые можно комбинировать (двоичной операцией OR):

`0x8000 (BADONION)`

Неразбираемая луковица, зашифрованная отправляющим одноранговым узлом.

[81] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#returning-errors.

[82] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#failure-messages.

0x4000 (PERM)

Постоянный сбой (в противном случае преходящий).

0x2000 (NODE)

Сбой узла (в противном случае канала).

0x1000 (UPDATE)

Вложено новое обновление канала.

В табл. 10-1 указаны типы сбоев, которые в настоящее время определены.

Таблица 10-1. Типы ошибок и сбоев луковицы

Тип	Символическое имя	Смысл
PERM|1	invalid_realm	Байт realm не был понят обрабатывающим узлом
NODE|2	temporary_node_failure	Общий временный сбой обрабатывающего узла
PERM| NODE|2	permanent_node_failure	Общий постоянный сбой обрабатывающего узла
PERM| NODE|3	required_node_feature_missing	Обрабатывающий узел имеет требуемый признак, которого не было в этой луковице
BADONION| PERM|4	invalid_onion_version	Байт version не был понят обрабатывающим узлом
BADONION| PERM|5	invalid_onion_hmac	HMAC-код луковицы был неправильным, когда он достиг обрабатывающего узла
BADONION| PERM|6	invalid_onion_key	Эфемерный ключ не был разобран обрабатывающим узлом
UPDATE|7	temporary_channel_failure	Канал из обрабатывающего узла не смог обработать этот HTLC-контракт, но, возможно, сможет обработать его или другие позже
PERM|8	permanent_channel_failure	Канал из обрабатывающего узла не может обрабатывать какие-либо HTLC-контракты
PERM|9	required_channel_feature_ missing	Канал из обрабатывающего узла требует признаков, отсутствующих в луковице
PERM|10	unknown_next_peer	Луковица указала ИД short_channel_ id, который не соответствует ни одному ведущему из обрабатывающего узла
UPDATE|11	amount_below_minimum	Сумма HTLC-контракта была ниже htlc_minimum_msat канала из обрабатывающего узла
UPDATE|12	fee_insufficient	Сумма комиссионных была ниже той, которую требовал канал из обрабатывающего узла

Окончание табл. 10-1

Тип	Символическое имя	Смысл
UPDATE|13	incorrect_cltv_expiry	cltv_expiry не соответствует cltv_expiry_delta, требуемой каналом из обрабатывающего узла
UPDATE|14	expiry_too_soon	CLTV-срок действия слишком близок к текущей высоте блока, чтобы безопасно обрабатываться обрабатывающим узлом
PERM|15	incorrect_or_unknown_payment_details	payment_hash неизвестен заключительному узлу, payment_secret не соответствует payment_hash, сумма для этого payment_hash неверна, или CLTV-срок действия HTLC-контракта слишком близок к текущей высоте блока, чтобы безопасно обрабатываться
18	final_incorrect_cltv_expiry	CLTV-срок действия в HTLC-контракте не соответствует значению в луковице
19	final_incorrect_htlc_amount	Сумма в HTLC-контракте не соответствует значению в луковице
UPDATE|20	channel_disabled	Канал из обрабатывающего узла был деактивирован
21	expiry_too_far	CLTV-срок действия в HTLC-контракте находится слишком далеко в будущем
PERM|22	invalid_onion_payload	Дешифрованный луковичный попереходный полезный груз не был понят обрабатывающим узлом или является неполным
23	mpp_timeout	Полная сумма многокомпонентного платежа не была получена в разумный срок

Застрявшие платежи

В текущей имплементации сети Network существует вероятность того, что попытка оплаты застрянет: не будет выполнена или отменена из-за ошибки. Это может произойти из-за дефекта на промежуточном узле, узла, переходящего в режим офлайн во время обработки HTLC-контрактов, или вредоносного узла, удерживающего HTLC-контракты, не сообщая об ошибке. Во всех этих случаях HTLC-контракт не может быть урегулирован до истечения срока его действия. Установленная на каждом HTLC-контракте привязка ко времени (CLTV) помогает устранить эту ситуацию (среди других возможных сбоев канала и маршрутизации HTLC-контракта).

Однако это означает, что отправитель HTLC-контракта должен дождаться истечения срока, и средства, привязанные к этому HTLC-контракту, остаются недоступными до истечения срока HTLC-контракта. Кроме того, отправитель не может повторить тот же платеж, потому что если он это сделает, то рискует, что как первоначальный, так и повторный платеж окажется успешным – получателю будет заплачено дважды. Это связано с тем, что после отправки HTLC-контракт не может быть «отменен» отправителем – он либо должен за-

вершиться отказом/сбоем, либо истечь. Застрявшие платежи, хотя и редки, создают нежелательный пользовательский опыт, когда кошелек пользователя не может оплатить или отменить платеж.

Одно из предлагаемых решений этой проблемы называется платежами без застревания (stuckless payments) и зависит от контрактов с точечной привязкой ко времени (PTLC-контрактов), то есть платежных контрактов, в которых используется криптографический примитив, отличный от HTLC-контрактов (т. е. прибавление точек на эллиптической кривой вместо хеша и секретного прообраза). PTLC-контракты громоздки с использованием алгоритма ECDSA, но намного проще с функциональностями подписи на основе Taproot (стержневого корня) и подписи Шнорра в системе Bitcoin, которые недавно были зафиксированы для активации в ноябре 2021 года. Ожидается, что PTLC-контракты будут имплементированы в сети Lightning после того, как эти функциональности системы Bitcoin будут активированы.

Спонтанные платежи keysend

В потоке платежей, описанном ранее в этой главе, мы допустили, что Дина получила счет от Алисы «вне полосы» или получила его с помощью какого-либо механизма, не связанного с протоколом (обычно копирование/вставка или сканирование QR-кода). Эта особенность означает, что процесс оплаты всегда состоит из двух шагов: во-первых, отправитель получает счет, а во-вторых, использует платежный хеш (закодированный в счете) для успешной маршрутизации HTLC-контракта. Дополнительный пробег в оба конца, необходимый для получения счета до совершения платежа, бывает узким местом в приложениях, связанных с потоковыми микроплатежами через Lightning. Что, если бы мы могли просто «проталкивать» платеж спонтанно, без необходимости сначала получать счет от получателя? Протокол keysend – это сквозное расширение (оно известно только отправителю и получателю) протокола Lightning, которое позволяет осуществлять спонтанные push-платежи.

Конкретно-прикладные луковичные TLV-записи

В рамках современного протокола Lightning в луковице используется кодировка TLV (тип–длина–значение), шифрующая информацию, которая сообщает каждому узлу о том, куда и как пересылать платеж. Используя формат TLV, за каждой частью информации о маршруте (например, следующему узлу, на который передается HTLC-контракт) закрепляется определенный тип (или ключ), кодированный как целое число переменной длины `BigSize` (максимальный размер как 64-разрядное целое число). Эти «основные» (обратные значения ниже `65536`) типы определены в спецификации BOLT #4 вместе с остальными деталями луковичной маршрутизации. Типы луковиц со значением больше `65536` предназначены для использования кошельками и приложениями в качестве «конкретно-прикладных записей».

Конкретно-прикладные записи позволяют платежным приложениям присоединять дополнительные метаданные или контекст к платежу в виде пар ключ/значение в луковице. Поскольку конкретно-прикладные записи включены в сам луковичный полезный груз, как и все остальное содержимое перехо-

да, записи зашифрованы в сквозном порядке. Так как конкретно-прикладные записи эффективно потребляют часть луковичного пакета с фиксированным размером 1300 байт, кодирование каждого ключа и значения каждой конкретно-прикладной записи уменьшает объем доступного пространства для кодирования остальной части маршрута. На практике это означает, что чем больше пространства луковицы используется для конкретно-прикладных записей, тем короче может быть маршрут. Учитывая, что каждый HTLC-пакет имеет фиксированный размер, конкретно-прикладные записи не вносят никаких дополнительных данных в HTLC-контракт; скорее, они перераспределяют байты, которые в противном случае были бы заполнены случайными данными.

Отправка и получение платежей keysend

Платеж `keysend` инвертирует типичный поток HTLC-контракта, когда получатель раскрывает отправителю секретный прообраз. Вместо этого отправитель включает прообраз внутрь луковицы для получателя и маршрутизирует HTLC-контракт получателю. Затем получатель дешифрует луковичный полезный груз и использует включенный прообраз (который должен соответствовать платежному хешу HTLC-контракта) для улаживания платежа. Как следствие платежи `keysend` могут осуществляться без предварительного получения счета от получателя, поскольку прообраз «проталкивается» получателю. Платеж `keysend` использует конкретно-прикладной тип TLV-записи `5482373484` для кодирования 32-байтового значения прообраза.

Платеж keysend и конкретно-прикладные записи в приложениях Lightning

Во многих потоковых приложениях Lightning протокол `keysend` используется для непрерывной потоковой передачи сатоши в пункт назначения, идентифицируемый его публичным ключом в сети. В типичной ситуации в дополнение к записи `keysend` приложение также будет включать метаданные, такие как примечание о пожертвованиях/донатах или другую информацию уровня приложения.

Вывод

Протокол луковичной маршрутизации сети Lightning адаптирован на основе протокола Sphinx с целью более оптимального удовлетворения потребностей платежной сети. В силу этого он предлагает огромное улучшение конфиденциальности и контрнаблюдения по сравнению с публичной и прозрачной блочной цепью Bitcoin.

В главе 12 мы увидим, как Алиса использует комбинацию маршрутизации на основе источника и луковичной маршрутизации, чтобы найти правильный путь и промаршрутизировать платеж Дине. В целях отыскания пути Алисе сначала нужно узнать о топологии сети, которая является темой главы 11.

Глава 11

Сплетни и канальный граф

В этой главе мы опишем эпидемический протокол (протокол сплетен) сети Lightning и то, как он используется узлами для построения и поддержания канального графа. Мы также рассмотрим механизм самозагрузки DNS, используемый для отыскания одноранговых узлов, с которыми можно «сплетничать».

Раздел «Комиссионные за маршрутизацию и ретрансляция сплетен» выделен контуром, охватывающим маршрутизационный и одноранговый слои на рис. 11-1.

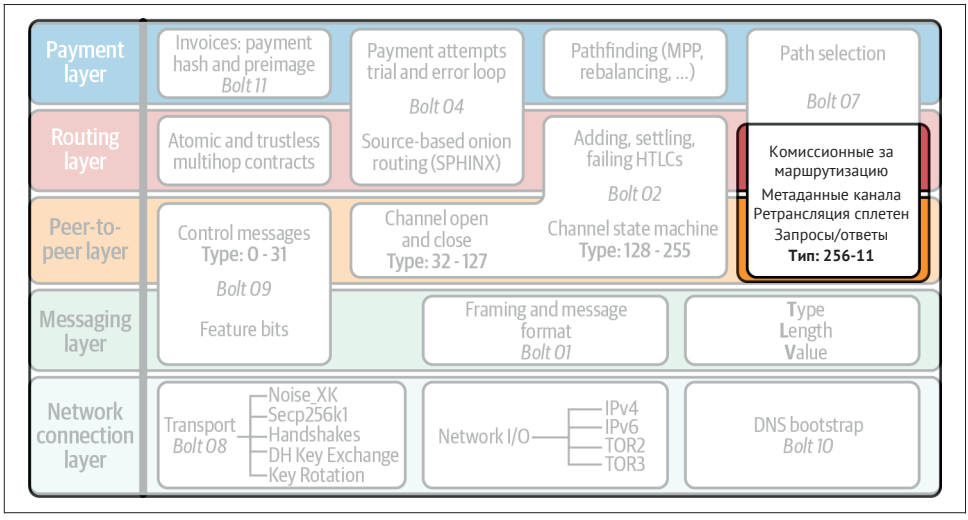


Рис. 11-1. Эпидемический протокол в рамках комплекта протоколов Lightning

Как мы уже узнали, в сети Lightning источниковый протокол луковичной маршрутизации используется для доставки платежа от отправителя получателю. Для этого отправляющий узел должен иметь возможность построить путь из платежных каналов, который соединяет его с получателем, как мы увидим в главе 12. Следовательно, отправитель должен иметь возможность картировать сеть Lightning путем построения канального графа. Канальный граф – это взаимосвязанный набор публично рекламируемых каналов и узлов, которые эти каналы связывают между собой.

Поскольку каналы поддерживаются финансовой транзакцией, которая происходит внутри цепи, можно ошибочно полагать, что узлы Lightning могут просто извлекать существующие каналы из блочной цепи Bitcoin. Однако это возможно только в определенной степени. Финансовые транзакции представляют собой адреса Pay-to-Witness-Script-Hash (аббр. P2WSH, оплата по хешу скрипта свидетеля), и характер скрипта (мультиподпись «2 из 2») будет раскрыт только после того, как будут израсходованы выходы финансовой транзакции. Даже если бы характер скрипта был известен, важно помнить, что не все скрипты мультиподписи «2 из 2» соответствуют платежным каналам.

Есть еще больше причин, по которым просмотр блочной цепи Bitcoin может оказаться бесполезным. Например, в сети Lightning используемые для подписи Bitcoin-ключи ротируются узлами для каждого канала и обновления. Отсюда даже если бы мы были способны надежно обнаруживать финансовые транзакции в блочной цепи Bitcoin, то мы бы не знали, какие два узла в сети Lightning владеют этим конкретным каналом.

Сеть Lightning решает эту проблему, внедряя эпидемический протокол, или протокол сплетен. Эпидемические протоколы типичны для одноранговых сетей (P2P-сетей) и позволяют узлам обмениваться информацией со всей сетью всего за несколько прямых соединений с одноранговыми узлами. Узлы Lightning раскрывают зашифрованные одноранговые соединения друг с другом и обмениваются информацией (сплетнями), которую они получили от других одноранговых узлов. Как только узел хочет поделиться какой-либо информацией, например о недавно созданном канале, он отправляет сообщение всем своим одноранговым узлам. Получив сообщение, узел решает, было ли полученное сообщение новым, и если да, то пересылает информацию своим одноранговым узлам. Таким образом, если одноранговая сеть имеет хорошую связность, то вся новая информация, необходимая для работы сети, в конечном итоге будет передана всем другим одноранговым узлам.

Очевидно, что если новый одноранговый узел присоединяется к сети в первый раз, то ему необходимо знать некоторые другие одноранговые узлы в сети, чтобы он мог подсоединяться к другим узлам и участвовать в сети.

В этой главе мы подробно разведаем то, как узлы Lightning обнаруживают друг друга, раскрывают и обновляют статус своего узла, а также взаимодействуют друг с другом.

Когда большинство ссылается на сетевую часть сети Lightning, они имеют в виду канальный граф, который сам по себе представляет собой уникальную аутентифицированную структуру данных, заякоренную в базовой блочной цепи Bitcoin.

Однако сеть Lightning также является одноранговой сетью узлов, которые обмениваются информацией о платежных каналах и узлах. Обычно для того, чтобы два одноранговых узла поддерживали платежный канал, им необходимо общаться друг с другом напрямую, и, стало быть, между ними будет одноранговое соединение. Это говорит о том, что канальный граф является подсетью одноранговой сети. Однако это неверно, поскольку платежные каналы могут оставаться открытыми, даже если один или оба одноранговых узла временно находятся в офлайне.

Давайте вернемся к некоторым терминам, которые мы использовали на протяжении всей книги, конкретно рассмотрев, что они означают в рамках канального графа и одноранговой сети (см. табл. 11-1).

Таблица 11-1. Терминология разных сетей

Канальный граф	Одноранговая сеть
канал	соединение
открыт	подсоединить
закрыть	отсоединить
финансовая транзакция	зашифрованное соединение по TCP/IP
отправить	передать
платеж	сообщение

Поскольку сеть Lightning является одноранговой сетью, для того чтобы одноранговые узлы могли обнаруживать друг друга, требуется некоторая самозагрузка (начальная загрузка, бутстрап). В этой главе мы рассмотрим историю первого подсоединения нового одноранговой узла к сети и каждый шаг процесса самозагрузки, от первоначального обнаружения одноранговой узла до синхронизации и валидации канального графа.

В качестве начального шага наш новый узел должен каким-то образом *обнаружить* по меньшей мере *один* одноранговый узел, который уже подсоединен к сети и имеет полный канальный граф (как мы увидим позже, канонической версии канального графа не существует). Используя один из многих протоколов самозагрузки, чтобы найти этот первый одноранговый узел, после установления соединения нашему новому одноранговому узлу теперь необходимо *скачать* и *валидировать* канальный граф. После того как канальный граф будет полностью валидирован, наш новый одноранговый узел будет готов начать открывать каналы и отправлять платежи по сети.

После начальной самозагрузки узел в сети должен продолжать поддерживать свою проекцию канального графа, обрабатывая обновления политики маршрутизации новых каналов, обнаруживая и валидируя новые каналы, удаляя каналы, которые были закрыты внутри цепи, и, наконец, обрезая каналы, которые не посылают правильное «сердцебиение» каждые два недели или около того.

По завершении этой главы вы поймете ключевой компонент одноранговой сети Lightning, а именно как одноранговые узлы обнаруживают друг друга и поддерживают локальную копию (перспективу) канального графа. Мы начнем с разведывания истории нового узла, который только что самозагрузился и которому необходимо найти другие одноранговые узлы для подсоединения к сети.

Обнаружение одноранговых узлов

В этом разделе мы начнем следить за новым узлом Lightning, который желает присоединиться к сети, выполнив три шага:

1) обнаружив набор одноранговых узлов (P2P-узлов) в рамках самозагрузки;
2) скачав и валидировав канальный граф;
3) начав процесс текущего сопровождения самого канального графа.

Самозагрузка P2P-узлов

Прежде чем делать что-либо еще, наш новый узел сначала должен обнаружить набор одноранговых узлов, которые уже являются частью сети. Мы называем этот процесс самозагрузкой одноранговых узлов, и это то, что каждая одноранговая сеть должна правильно имплементировать, чтобы обеспечивать устойчивую и работоспособную сеть.

Самозагрузка новых одноранговых узлов в существующие одноранговые сети – это очень хорошо изученная задача с несколькими известными решениями, каждое из которых имеет свои собственные отличительные компромиссы. Самое простое решение данной задачи – просто упаковать набор жестко закодированных одноранговых самозагрузочных узлов в пакетированное программное обеспечение одноранговых узла. Это просто в том смысле, что у каждого нового узла есть список одноранговых узлов для самозагрузки в программном обеспечении, которое они выполняют, но довольно хрупкое, учитывая, что если набор одноранговых узлов самозагрузки перейдет в офлайновый режим, то никакие новые узлы не смогут присоединиться к сети. Из-за этой хрупкости данная опция обычно используется в качестве запасного варианта на случай, если ни один из других механизмов самозагрузки P2P-узлов не работает должным образом.

Вместо того чтобы жестко кодировать набор одноранговых самозагрузочных узлов в самом программном обеспечении/двоичном коде, можно разрешить одноранговым узлам динамически получать свежий / новый набор одноранговых самозагрузочных узлов, которые они могут использовать для подсоединения к сети. Мы будем называть этот процесс *первоначальным обнаружением одноранговых узлов*. Как правило, для поддержания и распространения набора одноранговых самозагрузочных узлов используются существующие интернет-протоколы. Неисчерпывающий список протоколов, которые использовались в прошлом для выполнения первоначального одноранговых обнаружения, включает:

- службу доменных имен (Domain Name Service, аббр. DNS);
- интернет-ретрансляционный чат (Internet Relay Chat, аббр. IRC);
- протокол передачи гипертекста (Hypertext Transfer Protocol, аббр. HTTP).

Подобно протоколу Bitcoin, первичный механизм обнаружения одноранговых узлов, используемый в сети Lightning, осуществляется через DNS. Поскольку первоначальное обнаружение одноранговых узлов является критически важной и универсальной задачей для сети, этот процесс был стандартизирован в спецификации «BOLT #10: самозагрузка на основе DNS»[83].

Самозагрузка адресов DNS-серверов

Документ BOLT #10 описывает стандартизированный способ имплементирования обнаружения одноранговых узлов с использованием DNS. Версия Lightning для самозагрузки на основе DNS применяет до трех разных типов записей:

- записи `SRV` для обнаружения набора публичных ключей узла;
- записи `A` для соотнесения публичного ключа узла с его текущим IPv4-адресом;
- записи `AAA` для соотнесения публичного ключа узла с его текущим IPv6-адресом.

[83] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md.

Те, кто немного знаком с протоколом DNS, возможно, уже знакомы с типами записей `A` (имя по адресу IPv4) и `AAA` (имя по адресу IPv6), но не с типом `SRV`. Тип `SRV`-записи используется протоколами, построенными поверх DNS, для определения местоположения указанной службы. В нашем контексте рассматриваемая служба – это конкретный узел Lightning, а местоположение – его IP-адрес. Этот дополнительный тип записи необходимо использовать, потому что, в отличие от узлов в рамках протокола Bitcoin, для подсоединения к узлу нам нужны как публичный ключ, так и IP-адрес. Как мы увидим в главе 13, протокол транспортного шифрования, используемый в сети Lightning, требует знания публичного ключа узла перед подсоединением, чтобы имплементировать сокрытие идентификационных данных узлов в сети.

Рабочий поток самозагрузки нового одноранговой узла

Прежде чем углубиться в специфику BOLT #10, мы сначала обрисуем высокоуровневый поток нового узла, который хочет использовать спецификацию BOLT # 10 для подсоединения к сети.

Прежде всего узлу необходимо идентифицировать один DNS-сервер или набор DNS-серверов, которые понимают спецификацию BOLT #10, чтобы их можно было использовать для самозагрузки P2P-узлов.

В то время как в спецификации BOLT #10 в качестве начального сервера используется *lseed.bitcoinstats.com*, для этой цели не существует «официального» набора начальных DNS-серверов, но каждая мажорная имплементация поддерживает свои собственные начальные DNS-серверы, и они выполняют перекрестные запросы к начальным DNS-серверам друг друга в целях избыточности. В табл. 11-2 вы видите неисчерпывающий список некоторых популярных начальных DNS-серверов.

Таблица 11-2. Таблица известных начальных серверов DNS в Lightning

DNS-сервер	Сопроводитель
lseed.bitcoinstats.com	Кристиан Декер (Christian Decker)
nodes.lightning.directory	Lightning Labs (Олаулува Осунтокун)
soa.nodes.lightning.directory	Lightning Labs (Олаулува Осунтокун)
lseed.darosior.ninja	Антуан Пуансо

Начальные DNS-серверы существуют как для главной сети (mainnet) Bitcoin, так и для тестовой сети (testnet). Ради нашего примера мы допустим существование валидного в рамках спецификации BOLT #10 начального DNS-сервера по адресу *nodes.lightning.directory*.

Затем наш новый узел выдаст `SRV`-запрос, чтобы получить набор *кандидатных самозагрузочных одноранговых узлов*. Ответом на наш запрос будет серия публичных ключей в кодировке bech32. Поскольку DNS – это текстовый протокол, мы не можем отправлять сырые двоичные данные, поэтому требуется схема кодирования. В спецификации BOLT #10 кодировка bech32 определена из-за ее использования в более широкой экосистеме Bitcoin. Число возвращаемых закодированных публичных ключей зависит от возвращающего запрос

сервера, а также от всех сопоставителей (resolvers), которые стоят между клиентом и авторизованным сервером.

Используя широко доступный инструмент командной строки `dig`, упомянутую ранее *testnet*-версию начального DNS-сервера можно запросить с помощью следующей ниже команды:

```
$ dig @8.8.8.8 test.nodes.lightning.directory SRV
```

Мы применяем аргумент `@` для принудительного сопоставления через сервер имен Google (с IP-адресом 8.8.8.8), поскольку он не фильтрует крупные ответы на SRV-запросы. В конце команды указываем, что хотим возвращать только SRV-записи. Ответ выглядит примерно так, как в примере 11-1.

Пример 11-1. Запрашивание начального DNS-сервера для достижимых узлов

```
$ dig @8.8.8.8 test.nodes.lightning.directory SRV

; <<>> DiG 9.10.6 <<>> @8.8.8.8 test.nodes.lightning.directory SRV
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43610
;; flags: qr rd ra; QUERY: 1, ANSWER: 25, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;test.nodes.lightning.directory. IN SRV

;; ANSWER SECTION:
test.nodes.lightning.directory. 59 IN SRV 10 10 9735 ❶
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.
lightning.
directory. ❷
test.nodes.lightning.directory. 59 IN SRV 10 10 15735
ln1qtgsl3efj8verd4z27k44xu0a59kncvsarxatahm334exgnuvwhnz8dkhx8.test.nodes.
lightning.
directory.

 [...]

;; Query time: 89 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Dec 31 16:41:07 PST 2020
```

❶ Номер TCP-порта, через который можно связаться с узлом LN.
❷ Публичный ключ узла (ИД), закодированный как виртуальное доменное имя.

Мы сократили ответ для краткости и показываем только два возвращенных ответа. Ответы содержат «виртуальное» доменное имя целевого узла, затем слева у нас TCP-порт, через который можно связаться с этим узлом. В первом ответе используется стандартный TCP-порт сети Lightning: 9735. Во втором ответе используется конкретно-прикладной порт, который разрешен протоколом.

Далее попытаемся получить другую информацию, необходимую для подсоединения к узлу: его IP-адрес. Однако, прежде чем мы сможем его запросить, сначала *декодируем* кодировку bech32 публичного ключа из виртуального доменного имени:

```
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7
```

Декодируя эту строку bech32, мы получаем следующий ниже валидный публичный ключ secp256k1:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3
```

Теперь, когда у нас есть сырой публичный ключ, мы попросим DNS-сервер *сопоставить* указанный виртуальный хост, чтобы иметь возможность получить информацию об IP (запись A) для узла, как показано в примере 11-2.

Пример 11-2. Получение самого последнего IP-адреса для узла

```
$
dig ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.
lightning.
directory A

; <<>> DiG 9.10.6 <<>>
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.
lightning.
directory A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41934
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.
lightning.
directory. IN A

;; ANSWER SECTION:
ln1qfkxfad87fxx7lcwr4hvsalj8vhkwta539nuy4zlyf7hqcmrjh40xx5frs7.test.nodes.
lightning.
directory. 60 IN A X.X.X.X (1)

;; Query time: 83 msec
;; SERVER: 2600:1700:6971:6dd0::1#53(2600:1700:6971:6dd0::1)
;; WHEN: Thu Dec 31 16:59:22 PST 2020
;; MSG SIZE rcvd: 138
```

(1) DNS-сервер возвращает IP-адрес X.X.X.X. Здесь мы заменили его на X в тексте, чтобы не показывать реальный IP-адрес.

В приведенной выше команде мы запросили сервер, чтобы получить IPv4-адрес (запись A) для нашего целевого узла (заменен на X.X.X.X в вышеприведенном примере). Теперь, когда у нас есть сырой публичный ключ, IP-адрес и TCP-порт, можно подключиться к транспортному протоколу узла по адресу:

```
026c64f5a7f24c6f7f0e1d6ec877f23b2f672fb48967c2545f227d70636395eaf3@X.X.X.X:9735
```

Запрос текущей DNS-записи для данного узла также может использоваться для поиска самого последнего набора адресов. Такие запросы можно использовать для более быстрой синхронизации самой последней адресной инфор-

мации для узла по сравнению с ожиданием в отношении обновлений адресов в эпидемической сети (см. раздел «Сообщение node_announcement» на стр. 291).

В этом месте нашего путешествия наш новый узел Lightning нашел свой первый одноранговый узел и установил свое первое соединение! Теперь можно приступить ко второй фазе самозагрузки нового одноранговой узла: синхронизации и валидации канального графа.

Сначала мы подробнее разведаем тонкости самой спецификации BOLT #10, чтобы глубже разобраться в том, как все работает под капотом.

Опции SRV-запроса

Стандарт BOLT #10[84] обладает высокой расширяемостью благодаря использованию вложенных поддоменов в качестве коммуникационного слоя для дополнительных опций запроса. Протокол самозагрузки позволяет клиентам дополнительно указывать тип узлов, которые они пытаются запросить, в отличие от получения по умолчанию случайного подмножества узлов в ответах на запрос.

В поддоменной схеме опции запроса используется серия пар ключ–значение, где сам ключ – это одна буква, а оставшийся набор текста – само значение. В текущей версии документа стандартов спецификации BOLT #10 существуют следующие типы запросов:

`r`

Байт области (realm byte), который используется для определения того, для какого запроса цепи либо области должны быть возвращены ответы. В настоящий момент единственное значение для этого ключа равно 0, которое означает «Bitcoin».

`a`

Позволяет клиентам отфильтровывать возвращаемые узлы на основе типов рекламируемых ими адресов. В качестве примера его можно использовать для получения только тех узлов, которые рекламируют валидный IPv6-адрес. Значение, следующее за этим типом, основано на битовом поле, которое индексируется в набор указанных типов адресов, определенных в спецификации BOLT #7[85]. Дефолтное значение для этого поля равно 6, которое представляет как IPv4, так и IPv6 (биты 1 и 2 установлены).

`l`

Валидный публичный ключ узла, сериализованный в сжатом формате. Позволяет клиенту запрашивать указанный узел, а не получать набор случайных узлов.

`n`

Число возвращаемых записей. Дефолтное значение для этого поля равно 25.

Пример запроса с дополнительными опциями запроса выглядит примерно следующим образом:

---

[84] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/10-dns-bootstrap.md.

[85] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md.

```
r0.a2.n10.nodes.lightning.directory
```

Разбив запрос по одной паре ключ–значение за раз, мы получаем следующую ниже информацию:

`r0`

Запрос нацелен на область Bitcoin.

`a2`

Запрос требует возврата только IPv4-адресов.

`n10`

Число записей запроса.

Попробуйте несколько комбинаций различных флагов, используя DNS-инструмент командной строки `dig` самостоятельно:

```
dig @8.8.8.8 r0.a6.nodes.lightning.directory SRV
```

Канальный граф

Теперь, когда наш новый узел может использовать протокол самозагрузки DNS для подсоединения к своему самому первому одноранговому узлу, он может начать синхронизировать канальный граф! Однако, прежде чем выполнить синхронизацию канального графа, необходимо точно узнать, что мы подразумеваем под канальным графом. В этом разделе мы разведаем точную структуру канального графа и рассмотрим уникальные аспекты канального графа по сравнению с типичной абстрактной структурой данных «граф», которая хорошо известна/используется в информатике.

Ориентированный граф

Граф в информатике – это специальная структура данных, состоящая из вершин (обычно именуемых узлами) и ребер (также именуемых связями). Два узла могут быть соединены одним или несколькими ребрами. Канальный граф также ориентирован, учитывая, что платеж может проходить в любом направлении по заданному ребру (каналу). Пример ориентированного графа показан на рис. 11-2.

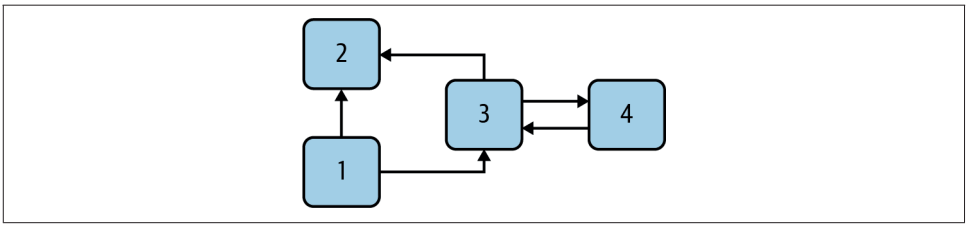


Рис. 11-2. Ориентированный граф

В контексте сети Lightning вершины сами являются узлами Lightning, а ребра – платежными каналами, соединяющими эти узлы. Поскольку мы занима-

емся маршрутизацией платежей, в нашей модели узел без ребер (без платежных каналов) не считается частью графа, т. к. он бесполезен.

Поскольку сами каналы являются UTXO-выходами (финансируемыми мультиподписными адресами «2 из 2»), канальный граф можно рассматривать как специальное подмножество набора UTXO-выходов Bitcoin, поверх которого можно добавлять некоторую дополнительную информацию (узлы и т. д.), в целях получения окончательной оверлейной структуры, то есть канального графа. Это заякоревание основополагающих компонентов канального графа в базовой блочной цепи Bitcoin означает, что отсутствует возможность подделывать валидный канальный граф, который обладает полезными свойствами, когда дело доходит до предотвращения спама, как мы увидим позже.

Сообщения эпидемического протокола

Информация о канальном графе распространяется по одноранговой сети (P2P-сети) Lightning в виде трех сообщений, которые описаны в спецификации BOLT #7[86]:

`node_announcement`

: Вершина в нашем графе, которая передает публичный ключ узла, а также способ доступа к узлу через интернет и некоторые дополнительные метаданные, описывающие набор функциональных признаков, поддерживаемых узлом.

`channel_announcement`

: Заякоренное в блочной цепи доказательство существования канала между двумя отдельными узлами. Любая третья сторона может верифицировать это доказательство, чтобы убедиться, что рекламируется реальный канал. Подобно сообщению `node_announcement`, это сообщение также содержит информацию, описывающую *функциональные признаки* канала, что полезно при попытке маршрутизировать платеж.

`channel_update`

: *Пара* структур, описывающих набор политик маршрутизации в данном канале. Сообщения `channel_update` идут в *паре*, поскольку канал является направленным ребром, поэтому каждая сторона канала может указывать свою собственную конкретно-прикладную политику маршрутизации.

Важно отметить, что каждый компонент канального графа проходит проверку подлинности, что позволяет третьей стороне обеспечивать, чтобы владелец канала/обновления/узла на самом деле был тем, кто отправляет обновление. Это эффективно превращает канальный граф в уникальный тип аутентифицированной структуры данных, которую невозможно подделать. Для аутентификации мы используем цифровую ECDSA-подпись secp256k1 (или их серию) поверх сериализованного дайджеста самого сообщения. В этой главе мы не будем вдаваться в специфику фреймирования/сериализации сообщений, используемых в сети Lightning, поскольку рассмотрим эту информацию в главе 13.

[86] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md.

После изложения высокоуровневой структуры канального графа мы теперь углубимся в точную структуру каждого из трех сообщений, используемых для распространения сплетен в канальном графе. Мы также объясним, как верифицировать каждое сообщение и компонент канального графа.

Сообщение node_announcement

В первую очередь у нас есть сообщение `node_announcement`, которое служит двум первичным целям:

1) рекламировать информацию о соединении, чтобы другие узлы могли подсоединяться к узлу либо для самозагрузки в сеть, либо для попытки установить новый платежный канал с этим узлом;
2) для передачи набора функциональных признаков (возможностей) уровня протокола, которые узел понимает/поддерживает. Согласование функциональных признаков между узлами позволяет разработчикам добавлять новые функциональные признаки независимо и поддерживать их с помощью любого другого узла на согласованной основе.

В отличие от объявлений канала, объявления узла не заякорены в базовой блочной цепи. Следовательно, объявления узла считаются валидными только в том случае, если они были распространены с помощью соответствующего объявления канала. Другими словами, мы всегда отклоняем узлы без платежных каналов в целях обеспечения невозможности наводнять сеть поддельными узлами, которые не являются частью канального графа, со стороны вредоносного однорангового узла.

Структура сообщения node_announcement

`node_announcement` состоит из следующих полей:

`signature`

Валидная ECDSA-подпись, которая охватывает сериализованный дайджест всех полей, перечисленных ниже. Эта подпись должна соответствовать публичному ключу рекламируемого узла.

`features`

Битовый вектор, описывающий набор функциональностей протокола, которые этот узел понимает. Мы рассмотрим это поле подробнее в разделе «Биты функциональностей и расширяемость протокола» на стр. 325, посвященном расширяемости протокола Lightning. На высоком уровне это поле содержит множество битов, которые представляют понятные узлу функциональности. В качестве примера узел может сигнализировать о том, что он понимает новейший тип канала.

`timestamp`

Временная метка, закодированная в эпохах Unix. Она позволяет клиентам применять частичное упорядочение обновлений к объявлению узла.

`node_id`

Публичный ключ `secp256k1`, которому принадлежит это объявление узла. В любой момент времени для данного узла в канальном графе может быть

только одно объявление node_announcement. В результате node_announcement может заменять предыдущее node_announcement для того же узла, если оно содержит более высокую (более позднюю) временную метку.

rgb_color

Поле, которое позволяет узлу указывать цвет RGB, который будет с ним связан, часто используется в визуализациях канального графа и каталогах узлов.

alias

Строка в формате UTF-8, которая будет использоваться в качестве псевдонима для данного узла. Обратите внимание, что эти псевдонимы не обязательно должны быть глобально-уникальными, и они никоим образом не верифицируются. Как следствие на них не следует полагаться как на форму идентификации – их можно легко подделать.

addresses

Набор публичных достижимых в интернете адресов, которые должны быть ассоциированы с данным узлом. В текущей версии протокола поддерживаются четыре типа адресов: IPv4 (тип: 1), IPv6 (тип: 2), Tor v2 (тип: 3) и Tor v3 (тип: 4). В сообщении node_announcement каждый из этих типов адресов обозначается целочисленным типом, который включается в круглые скобки после типа адреса.

Валидация объявлений узла

Валидация входящего сообщения node_announcement проста. При инспектировании объявления узла следует придерживаться следующих проверочных утверждений:

- если существующее сообщение node_announcement для этого узла уже известно, то поле timestamp нового входящего сообщения node_announcement должно быть больше предыдущего;
- с помощью этого ограничения мы обеспечиваем соблюдение уровня «свежести»;
- если для данного узла не существует сообщения node_announcement, то существующее сообщение channel_announcement, которое ссылается на данный узел (подробнее об этом позже), должно уже быть в локальном канальном графе;
- включенная signature должна быть валидной ECDSA-подписью, верифицированной с использованием включенного публичного ключа node_id и дайджеста двойного-SHA-256 кодировки сырого сообщения (за вычетом подписи и фреймового заголовка) в качестве сообщения;
- все включенные адреса должны быть отсортированы в возрастающем порядке на основе их идентификатора;
- включенные байты псевдонима должны быть допустимым литералом UTF-8.

Сообщение channel_announcement

Далее у нас есть сообщение channel_announcement, которое используется для объявления нового публичного канала в более широкой сети. Обратите внима-

ние, что объявлять канал необязательно. Канал необходимо объявлять только в том случае, если он предназначен для маршрутизирования сетью Lightning. Активные маршрутизационные узлы могут пожелать объявить все свои каналы. Однако некоторые узлы, такие как мобильные узлы, скорее всего, не имеют времени безотказной работы или желания быть активным маршрутизационным узлом. Как следствие данные мобильные узлы (в которых для подсоединения к P2P-сети Bitcoin обычно используются облегченные клиенты) вместо этого могут иметь чисто *необъявленные* (приватные) каналы.

Необъявленные (приватные) каналы

Необъявленный канал не является частью известного графа публичных каналов, но все равно может использоваться для отправки/получения платежей. Проницательный читатель, возможно, теперь задается вопросом, как канал, который не является частью графа публичных каналов, может получать платежи. Решением этой проблемы является набор «помощников по отысканию пути», который мы называем маршрутизационными подсказками. Как мы увидим в главе 15, счета, созданные узлами с нерекламированными каналами, будут содержать информацию, которая поможет отправителю маршрутизировать к ним, исходя из допущения, что узел имеет по меньшей мере один канал с существующим публичным маршрутизационным узлом.

Из-за существования нерекламированных каналов истинный размер канального графа (как из публичных, так и из приватных компонентов) неизвестен.

Локализация канала в блочной цепи Bitcoin

Как упоминалось ранее, канальный граф аутентифицируется благодаря использованию криптографии с публичным ключом, а также блочной цепи Bitcoin в качестве системы предотвращения спама. Для того чтобы узел принял новое сообщение `channel_announcement`, рекламирование должно *доказать*, что канал действительно существует в блочной цепи Bitcoin. Эта система доказательства добавляет первоначальные затраты на добавление новой записи в канальный граф (внутрицепные комиссионные, которые необходимо заплатить за создание UTXO-выхода канала). Как следствие мы уменьшаем спам и обеспечиваем, что нечестный узел в сети не сможет бесплатно заполнять память честного узла поддельными каналами.

Учитывая, что нам нужно построить доказательство существования канала, возникает естественный вопрос: как мы «указываем» или ссылаемся на данный канал для верификатора? Помня, что платежный канал заякорен к неизрасходованному транзакционному выходу (см. «Входы и выходы» на стр. 383), первоначальная мысль может заключаться в том, чтобы сначала попытаться рекламировать полную выходную точку (txid:index) канала. Учитывая, что выходная точка является глобально-уникальной и подтвержденной в цепи, это звучит как неплохая идея; однако у нее есть недостаток: верификатор должен поддерживать полную копию набора UTXO-выходов для верифицирования каналов. Это прекрасно работает для полных узлов Bitcoin, но клиенты, которые полагаются на облегченную верификацию, обычно не поддерживают полный набор UTXO-выходов. Поскольку мы хотим обеспечить способность поддерживать мобильные узлы в сети Lightning, то вынуждены найти другое решение.

Что, если вместо того, чтобы ссылаться на канал по его UTXO-выходу, мы будем ссылаться на него на основе его «местоположения» в цепи? Для этого нам понадобится схема, которая позволяет ссылаться на данный блок, затем на транзакцию внутри этого блока и, наконец, на конкретный выход, созданный этой транзакцией. Такой идентификатор описан в спецификации BOLT #7[87] и называется коротким идентификатором канала, или scid. Короткий ИД канала scid используется в сообщении channel_announcement (и channel_update), а также в луковично-зашифрованном маршрутизационном пакете, включаемом в HTLC-контракты, как мы узнали в главе 10.

Короткий ИД канала

Основываясь на предыдущей информации, у нас есть три фрагмента информации, которые необходимо закодировать, чтобы однозначно ссылаться на данный канал. Поскольку нам нужно компактное представление, мы попытаемся закодировать информацию в одно целое число. Выбранный нами целочисленный формат представляет собой беззнаковое 64-разрядное целое число, состоящее из 8 байт.

Сначала высота блока. Используя 3 байта (24 бита), можно закодировать 16 777 216 блоков. Это оставляет нам 5 байт для кодирования соответственно индекса транзакции и индекса выхода. Мы будем использовать следующие 3 байта для кодирования индекса транзакции внутри блока. Этого более чем достаточно, учитывая, что исправлять десятки тысяч транзакций в блоке можно только при текущих размерах блока. Это оставляет нам 2 байта для кодирования выходного индекса канала в транзакции.

Окончательный формат scid выглядит так:

```
высота_блока (3 байта) || индекс_транзакции (3 байта) || индекс_выхода (2 байта)
```

Используя технические приемы упаковки битов, мы сначала кодируем наиболее значимые 3 байта в качестве высоты блока, следующие 3 байта в качестве индекса транзакции и наименее значимые 2 байта в качестве выходного индекса, создаваемого канальным UTXO-выходом.

Короткий ИД канала может быть представлен в виде одного целого числа (`695313561322258433`) или в виде более удобного для пользователя литерала: `632384x1568x1`. Здесь мы видим, что канал был добыт в блоке 632384, был 1568-й транзакцией в блоке, с канальным выходом в качестве второго произведенного транзакцией (UTXO-выходы с нулевым индексом) выхода.

Теперь, когда можно кратко указать на данный выход финансирования канала в цепи, можно проинспектировать полную структуру сообщения `channel_announcement`, а также посмотреть, как верифицировать включенное в сообщение доказательство существования.

Структура сообщения channel_announcement

Сообщение `channel_announcement` в первую очередь сообщает о двух вещах:

1) доказательство того, что канал существует между узлом A и узлом B, причем оба узла контролируют мультиподписные ключи на выходе этого канала;

[87] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md.

2) набор возможностей канала (какие типы HTLC-контрактов он может маршрутизировать и т. д.).

При описании доказательства мы обычно будем ссылаться на узел 1 и узел 2. Из двух узлов, которые соединяет канал, «первый» узел – это узел, который имеет «более низкую» кодировку публичного ключа, если сравнивать публичный ключ двух узлов в шестнадцатеричном формате в сжатом формате, закодированном в лексикографическом порядке. Соответственно, в дополнение к публичному ключу узла в сети каждый узел также должен контролировать публичный ключ в блочной цепи Bitcoin.

Подобно сообщению `node_announcement`, все включенные подписи сообщения `channel_announcement` должны быть подписаны/верифицированы относительно сырой кодировки сообщения (за вычетом заголовка), которая следует после окончательной подписи (поскольку цифровая подпись не может подписывать сама себя).

С учетом сказанного сообщение `channel_announcement` содержит следующие поля:

`node_signature_1`

Подпись первого узла под дайджестом сообщения.

`node_signature_2`

Подпись второго узла под дайджестом сообщения.

`bitcoin_signature_1`

Подпись мультиподписного ключа (в финансовом выходе) первого узла под дайджестом сообщения.

`bitcoin_signature_2`

Подпись мультиподписного ключа (в финансовом выходе) второго узла под дайджестом сообщения.

`features`

Битовый вектор функциональностей, который описывает набор функциональностей уровня протокола, поддерживаемых этим каналом.

`chain_hash`

32-байтовый хеш, который обычно является хешем генезисного блока блочной цепи (например, главной mainnet-сети Bitcoin), в котором был открыт канал.

`short_channel_id`

`scid`, который однозначно определяет местоположение данного финансирующего канала в блочной цепи.

`node_id_1`

Публичный ключ первого узла в сети.

`node_id_2`

Публичый ключ второго узла в сети.

`bitcoin_key_1`

Сырой мультиподписной ключ для выхода финансирующего канала первого узла в сети.

`bitCoin_key_2`

Сырой мультиподписной ключ для выхода финансирующего канала второго узла в сети.

Валидация объявления канала

Теперь, когда мы знаем содержимое сообщения `channel_announcement`, можно посмотреть на то, как верифицировать существование канала в цепи.

Вооруженный информацией в channel_announcement, любой узел Lightning (даже без полной копии блочной цепи Bitcoin) может верифицировать существование и подлинность платежного канала.

Сначала верификатор будет использовать короткий ИД канала, чтобы определить Bitcoin-блок, который содержит выход финансирующего канала. Имея информацию о высоте блока, верификатор может запросить только этот конкретный блок из Bitcoin-узла. Затем блок может быть связан обратно с исходным блоком, следуя по цепочке заголовков блоков в обратном направлении (верифицируя доказательство работы), подтверждая, что этот блок на самом деле принадлежит блочной цепи Bitcoin.

Потом верификатор использует номер индекса транзакции для идентифицирования ИД транзакции, содержащей платежный канал. Большинство современных библиотек Bitcoin позволяют индексировать транзакцию блока, основываясь на индексе транзакции внутри большего блока.

Затем верификатор использует библиотеку Bitcoin (на языке верификатора) для извлечения соответствующей транзакции в соответствии с ее индексом внутри блока. Верификатор валидирует транзакцию (проверяя, что она правильно подписана и выдает тот же ИД транзакции при хешировании).

Далее верификатор извлекает выход адреса Pay-to-Witness-Script-Hash (P2WSH), на который ссылается номер выходного индекса короткого ИД канала. Это адрес выхода финансирующего канала. Кроме того, верификатор обеспечит, чтобы размер предполагаемого канала соответствовал значению выхода, произведенного в указанном индексе выхода.

Наконец, верификатор реконструирует мультиподписной скрипт из bitcoin_key_1 и bitcoin_key_2 и подтверждает, что он выдает тот же адрес, что и в выходе.

Теперь верификатор независимо подтвердил, что указанный в объявлении платежный канал финансируется и подтвержден в блочной цепи Bitcoin!

Сообщение channel_update

Третьим и последним сообщением, используемым в эпидемическом протоколе, является сообщение `channel_update`. Два из них генерируются для каждого платежного канала (по одному от каждого партнера канала) с указанием ожидаемых их комиссионных за маршрутизацию, привязки ко времени и возможностей.

Сообщение channel_update также содержит временную метку, позволяющую узлу обновлять свои комиссионные за маршрутизацию и другие ожидания и возможности, отправляя новое сообщение channel_update с более высокой

(более поздней) временной меткой, которая заменяет любые более старые обновления.

Сообщение channel_update содержит следующие поля:

`signature`

Цифровая подпись, соответствующая публичному ключу узла, для проверки подлинности источника и целостности обновления канала.

`chain_hash`

Хеш исходного (генезисного) блока цепи, содержащей канал.

`short_channel_id`

Короткий ИД канала для идентифицирования канала.

`timestamp`

Временная метка этого обновления, позволяющая получателям упорядочивать обновления и заменять более старые обновления.

`message_flags`

Битовое поле, указывающее на наличие дополнительных полей в сообщении `channel_update`.

`channel_flags`

Битовое поле, показывающее направление канала и другие опции канала.

`cltv_expiry_delta`

Ожидания в отношении дельты привязки ко времени этого узла для маршрутизации (см. главу 10).

`htlc_minimum_msat`

Минимальная сумма HTLC-контракта, которая будет маршрутизироваться.

`fee_base_msat`

Базовые комиссионные, которые будут взиматься за маршрутизацию.

`fee_proportional_millionths`

Пропорциональный комиссионный тариф, который будет взиматься за маршрутизацию.

`htlc_maximum_msat (option_channel_htlc_max)`

Максимальная сумма, которая будет маршрутизироваться.

Узел, получающий сообщение `channel_update`, может прикрепить эти метаданные к ребру канального графа, чтобы активировать отыскание пути, как мы увидим в главе 12.

Текущее сопровождение канального графа

Построение канального графа – это не разовое событие, а скорее непрерывная деятельность. Как только узел самозагрузится в сеть, он начнет получать «сплетни» в виде трех сообщений об обновлении. Он будет использовать эти

сообщения, чтобы немедленно начать построение валидированного канального графа.

Чем больше информации получает узел, тем лучше становится его «карта» сети Lightning и тем эффективнее он может быть при отыскании путей и доставке платежей.

Узел не только будет добавлять информацию в канальном графе. Он также будет отслеживать время последнего обновления канала и удалять «застоявшиеся» каналы, которые не обновлялись более двух недель. Наконец, если он увидит, что у какого-то узла больше нет каналов, он также удалит этот узел.

Информация, собранная из эпидемического протокола, – это не единственная информация, которая может быть сохранена в канальном графе. Разные имплементации узла Lightning могут присоединять другие метаданные к узлам и каналам. Например, некоторые имплементации узла вычисляют «балл», который оценивает «качество» узла как однорангового маршрутизационного узла. Этот балл используется как часть отыскания путей с целью приоретизации и деприоретизации путей.

Вывод

В этой главе мы узнали, как узлы Lightning обнаруживают друг друга, обнаруживают и обновляют статус своего узла, а также взаимодействуют друг с другом. Мы узнали, как создаются и поддерживаются канальные графы, и разведали несколько способов, с помощью которых сеть Lightning не позволяет злоумышленникам или нечестным узлам рассылать по сети спам.

Глава 12

Отыскание пути и доставка платежа

Доставка платежей в сети Lightning зависит от отыскания пути от отправителя к получателю, процесса, который так и называется – отысканием пути (*pathfinding*). Поскольку маршрутизация выполняется отправителем, отправитель должен найти подходящий путь, который позволяет достичь пункта назначения. Затем этот путь кодируется в луковице, как мы видели в главе 10.

В этой главе мы проинспектируем задачу отыскания пути, поймем, насколько неопределенность в отношении остатков средств в каналах усложняет эту задачу, и рассмотрим, как типичная имплементация отыскания пути пытается ее решить.

Отыскание пути в рамках комплекта протоколов Lightning

Отыскание пути, выбор пути, многокомпонентные платежи (MPP) и цикл попыток платежа методом проб и ошибок занимают бóльшую часть слоя платежей в верхней части комплекта протоколов.

Эти компоненты выделены контуром в наборе протоколов, показанном на рис. 12-1.

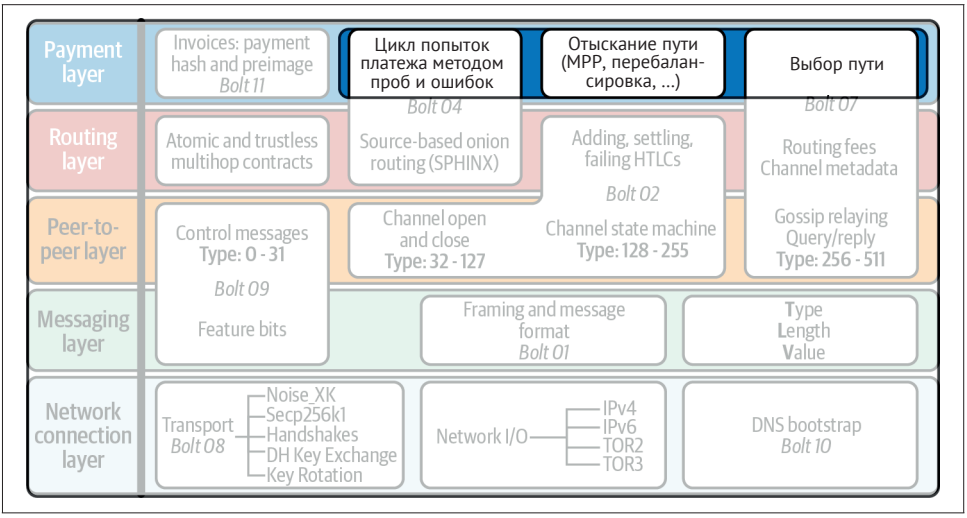


Рис. 12-1. Доставка платежей в комплекте протоколов Lightning

Где же BOLT?

На данный момент мы рассмотрели несколько технологий, которые являются частью сети Lightning, и увидели их точную спецификацию как часть стандарта BOLT. Вы, возможно, будете удивлены, обнаружив, что отыскание пути не является частью игры!

Это связано с тем, что отыскание пути не является деятельностью, требующей какой-либо формы координации или взаимодействия между разными имплементациями. Как мы уже увидели, путь выбирается отправителем. Несмотря на то что детали маршрута подробно указаны в документации, обнаружение и выбор пути полностью зависят от отправителя. Таким образом, каждая имплементация узла может выбирать для отыскания пути иную стратегию/алгоритм. Фактически разные имплемементации узла/клиента и кошелька могут даже конкурировать и использовать свой алгоритм отыскания пути в качестве точки дифференциации.

Отыскание пути: какую задачу мы решаем?

Термин «отыскание пути» может несколько дезориентировать, поскольку он подразумевает поиск единственного пути, соединяющего два узла. Вначале, когда сеть Lightning была небольшой и плохо взаимосвязанной, задача действительно заключалась в том, чтобы найти способ присоединиться к платежным каналам, дабы добраться до получателя.

Но по мере того, как сеть Lightning стремительно росла, характер задачи отыскания пути изменился. В середине 2021 года, когда мы заканчиваем эту книгу, сеть Lightning будет состоять из 20 000 узлов, соединенных по меньшей мере 55 000 публичных каналов с совокупной емкостью почти 2000 BTC. Узел имеет в среднем 8.8 канала, в то время как верхние 10 наиболее связ-

ных узлов имеют от 400 до 2000 каналов каждый. Визуализация лишь малого подмножества канального графа LN показана на рис. 12-2.

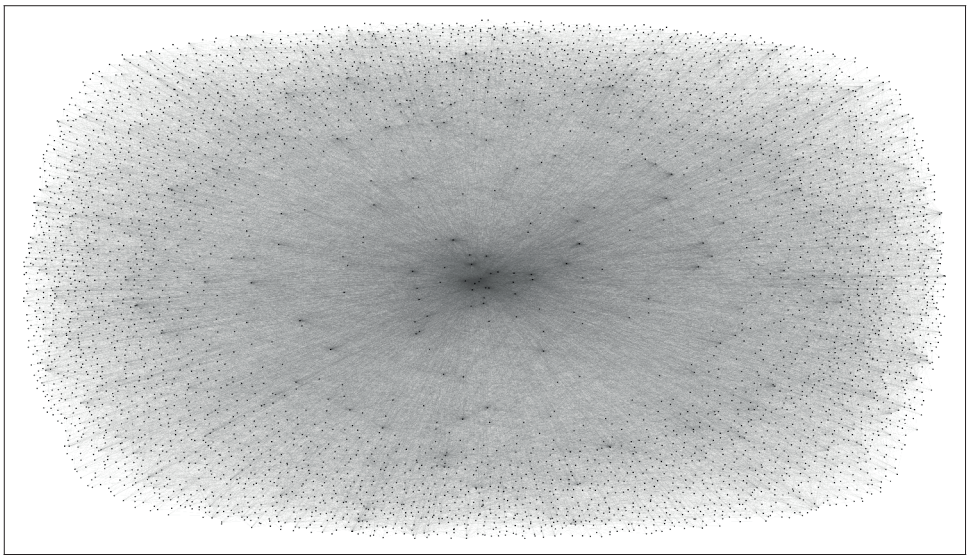


Рис. 12-2. Визуализация части сети Lightning по состоянию на июль 2021 года

Визуализация сети на рис. 12-2 была создана с помощью простого скрипта на Python, который можно найти в папке *code/lngraph* в репозитории книги.

Если отправитель и получатель подсоединены к другим высокосвязным узлам и имеют по меньшей мере один канал с достаточной емкостью, то будут тысячи путей. Задача заключается в выборе наилучшего пути, который приведет к успешной доставке платежа, из списка тысяч возможных путей.

Выбор наилучшего пути

Для того чтобы выбрать наилучший путь, мы должны сначала определить, что мы подразумеваем под «наилучшим». К этой категории может относиться целый ряд разных критериев, таких как:

- пути с достаточной ликвидностью. Очевидно, что если у пути недостаточно ликвидности для маршрутизирования нашего платежа, то этот путь не подходит;
- маршруты с низкими комиссионными. Если у нас есть несколько кандидатов, то мы можем выбрать тех, у кого более низкие комиссионные;
- пути с короткими привязками ко времени. Возможно, мы захотим избежать слишком длительной привязки наших средств и поэтому выберем пути с более короткими привязками ко времени.

Все эти критерии могут быть в какой-то степени желательными, и выбор путей, благоприятных во многих размерностях, является непростой задачей. Задачи оптимизации, подобные этой, бывают слишком сложными для отыскания «наилучшего» решения, но нередко решаются для некоторой аппроксимации оптимального, что является хорошей новостью, потому что в противном случае отыскание пути было бы непрослеживаемой задачей.

Отыскание путей в математике и информатике

Отыскание пути в сети Lightning подпадает под общую категорию теории графов в математике и более конкретную категорию обхода графов в информатике.

Сеть, такая как сеть Lightning, может быть представлена в виде математического конструкта, именуемого графом, где узлы соединены друг с другом ребрами (эквивалентными платежным каналам). Сеть Lightning образует ориентированный граф, поскольку узлы связаны асимметрично, так как остаток канала разделен между двумя партнерами по каналу, а ликвидность платежей различна в каждом направлении. Ориентированный граф с числовыми емкостными ограничениями на его ребрах называется транспортной сетью (поточной сетью), математическим конструктом, применяемым для оптимизации транспортировки и других подобных сетей. Транспортные сети могут использоваться в качестве каркаса, когда решения должны достигать определенного потока при минимизации стоимости. Такая задача называется задачей о потоке с минимальной стоимостью (minimum cost flow problem, аббр. MCFP).

Емкость, остаток, ликвидность

В целях более глубокого понимания задачи транспортировки сатоши из пункта A в пункт B нужно лучше определить три важных термина: емкость, остаток и ликвидность. Мы используем эти термины для описания способности платежного канала маршрутизировать платеж.

В платежном канале, соединяющем A←→B:

Емкость

Это агрегатная сумма сатоши, которые были профинансированы в мультиподпись «2 из 2» с помощью финансовой транзакции. Она представляет собой максимальное значение, хранящееся в канале. Емкость канала объявляется эпидемическим протоколом и известна узлам.

Остаток

Это сумма сатоши, хранящихся у каждого партнера канала, которая может быть отправлена другому партнеру канала. Подмножество остатка A может быть отправлено в направлении (A→B) узлу B. Подмножество остатка B может быть отправлено в противоположном направлении (A←B).

Ликвидность

Доступный остаток (его подмножество), который фактически может быть отправлен по каналу в одном направлении. Ликвидность A равна остатку A за вычетом резерва канала и любых ожидающих своей очереди HTLC-контрактов, совершенных A.

Единственное значение, известное сети (через эпидемические объявления), – это совокупная емкость канала. Некоторая неизвестная часть этой емкости распределяется как остаток каждого партнера. Некоторая подчасть этого остатка доступна для отправки по каналу в одном направлении:

```
емкость = остаток(A) + остаток(B)
ликвидность(A) = остаток(A) - резерв_канала(A) - ожидающие_HTLC_контракты(A)
```

Неопределенность остатков

Если бы мы знали точные остатки каждого канала, то могли бы вычислить один или несколько путей оплаты, используя любой из стандартных алгоритмов отыскания пути, которым обучают в хороших учебных программах по информатике. Но мы не знаем остатков каналов; мы знаем только агрегатную емкость канала, которая рекламируется узлами в объявлениях каналов. Для того чтобы платеж прошел успешно, на отправляющей стороне канала должен быть достаточный остаток. Если мы не знаем, как емкость распределяется между партнерами по каналу, то мы не знаем, достаточно ли остатка в том направлении, в котором мы пытаемся отправить платеж.

Остатки не объявляются в обновлениях канала по двум причинам: конфиденциальность и масштабируемость. Во-первых, объявление остатков снизило бы конфиденциальность сети Lightning, поскольку это позволило бы отслеживать платежи путем статистического анализа изменений в остатках. Во-вторых, если бы узлы объявляли остатки (глобально) при каждом платеже, то масштабирование сети Lightning было бы таким же плохим, как у внутрицепных Bitcoin-транзакций, которые широковещательно передаются всем участникам. Поэтому остатки не объявляются. В целях решения этой задачи отыскания пути в условиях неопределенности остатков нужны инновационные стратегии отыскания путей. Эти стратегии должны быть тесно связаны с используемым алгоритмом маршрутизации, то есть луковичной маршрутизации на основе источника, где ответственность за отыскание пути в сети несет отправитель.

Проблема неопределенности может быть описана математически как диапазон ликвидности, указывающий нижнюю и верхнюю границы ликвидности на основе известной информации. Поскольку мы знаем емкость канала и знаем резервный остаток канала (минимально допустимый остаток на каждом конце), ликвидность можно определить как

min(ликвидность) = резерв_канала,
max(ликвидность) = емкость – резерв_канала

или как диапазон:

резерв_канала <= ликвидность <= (емкость – резерв_канала).

Диапазон неопределенности ликвидности нашего канала – это диапазон между минимальной и максимально возможной ликвидностью. Он сети неизвестен, за исключением двух партнеров по каналу. Однако, как мы увидим, мы можем использовать сбойные HTLC-контракты, возвращенные после наших попыток оплаты, чтобы обновить нашу оценку ликвидности и уменьшить

неопределенность. Если, например, мы получаем код сбоя HTLC-контракта, который сообщает о том, что канал не может исполнить HTLC-контракт, который меньше нашей оценки максимальной ликвидности, то это означает, что максимальная ликвидность может быть обновлена до суммы сбойного HTLC-контракта. Проще говоря, если мы считаем, что ликвидность может справиться с HTLC-контрактом из *N* сатоши, и мы обнаруживаем, что она не может доставить M сатоши (где *M* меньше), тогда мы можем обновить нашу оценку до *M* – 1 в качестве верхней границы. Мы попытались найти потолок и на него наткнулись, так что он оказался ниже, чем мы думали!

Сложность отыскания пути

Отыскание пути в графе – это задача, которую современные компьютеры могут решать довольно эффективно. Если все ребра имеют одинаковый вес, то разработчики в основном выбирают поиск сперва в ширину. В тех случаях, когда ребра не имеют одинакового веса, используется алгоритм, основанный на алгоритме Дейкстры, например A* (произносится как «A-звездочка»)[88]. В нашем случае веса ребер могут быть представлены комиссионными за маршрутизацию. В поиск будут включены только те ребра, емкость которых превышает сумму, подлежащую отправке. В этой базовой форме процедура отыскания пути в сети Lightning очень проста и понятна.

Однако ликвидность канала отправителю неизвестна. Это превращает нашу простую теоретическую задачу в области информатики в довольно сложную задачу реального мира. Теперь мы должны решить задачу отыскания пути, обладая лишь частичными знаниями. Например, мы подозреваем, какие ребра могут переслать платеж, потому что их емкость кажется достаточно большой. Но мы не можем быть уверены, пока не попробуем это сделать либо не спросим напрямую владельцев каналов. Даже если бы мы могли спросить владельцев каналов напрямую, к тому времени, когда мы спросим других, вычислим путь, создадим луковицу и отправим ее дальше, их остаток может измениться. Мы не только располагаем ограниченной информацией, но и информация, которой мы располагаем, очень динамична и может измениться в любой момент времени без нашего ведома.

Без лишних сложностей

Механизм отыскания пути, имплементированный в узлах Lightning, заключается в том, чтобы сначала создать список возможных путей, отфильтрованных и отсортированных с помощью некоторой функции. Затем указанные пути будут прощупываться узлом или кошельком (пытаясь доставить платеж) в цикле проб и ошибок, пока не будет найден путь, который успешно доставит платеж.

Это прощупывание выполняется узлом Lightning или кошельком и не наблюдается непосредственно пользователем программного обеспечения. Однако если платеж не будет завершен мгновенно, то пользователь может заподозрить, что происходит прощупывание.

[88] См. https://en.wikipedia.org/wiki/A*_search_algorithm.

Хотя слепое прощупывание не является оптимальным и оставляет достаточно возможностей для совершенствования, следует отметить, что даже эта упрощенная стратегия работает на удивление хорошо для малых платежей и выскосвязных узлов.

Большинство имплементаций узлов Lightning и кошельков совершенствуют этот подход, упорядочивая/взвешивая список кандидатных путей. Некоторые имплементации упорядочивают кандидатные пути по стоимости (комиссионным) или некоторой комбинации стоимости и емкости.

Отыскание пути и процесс доставки платежа

Отыскание пути и доставка платежа предусматривают несколько шагов, которые мы перечислим ниже. В разных имплементациях могут использоваться разные алгоритмы и стратегии, но базовые шаги, скорее всего, будут очень похожими:

1. Создать канальный граф из объявлений и обновлений, содержащий емкость каждого канала, и отфильтровать граф, игнорируя любые каналы с недостаточной емкостью для суммы, которую мы хотим отправить.
2. Отыскать пути, соединяющие отправителя с получателем.
3. Упорядочить пути по некоторому весу (это может быть частью алгоритма предыдущего шага).
4. Попробовать каждый путь по порядку, пока платеж не завершится успешно (цикл проб и ошибок).
5. При необходимости использовать возвраты сбоев HTLC-контрактов, чтобы обновлять граф, уменьшая неопределенность.

Эти шаги можно сгруппировать в три первичных вида деятельности:

- построение канального графа;
- отыскание путей (отфильтрованных и упорядоченных с помощью некоторых эвристик);
- попытка(и) платежа.

Эти три действия могут быть повторены в платежном раунде, если мы используем возврат сбоев с целью обновления графа или если выполняем многокомпонентные платежи (см. раздел «Многокомпонентные платежи» на стр. 314).

В следующих далее разделах мы рассмотрим каждый из этих шагов подробнее, а также более продвинутые стратегии платежа.

Построение канального графа

В главе 11 мы рассмотрели три главных сообщения, которые узлы используют в своих сплетнях: `node_announcement`, `channel_announcement` и `channel_update`. Эти три способа позволяют любому узлу постепенно конструировать «карту» сети Lightning в виде канального графа. Каждое из этих сообщений предоставляет важную информацию для канального графа:

`node_announcement`

Содержит информацию об узле в сети Lightning, такую как его идентификатор (публичный ключ), сетевой адрес (например, IPv4/ 6 или Tor), возможности/признаки и т. д.

`channel_announcement`

Содержит емкость и идентификатор публичного (объявленного) канала между двумя узлами, а также доказательство существования канала и владение.

`channel_update`

Содержит ожидания в отношении комиссионных узла и привязки ко времени (CLTV) для маршрутизации исходящего (с точки зрения этого узла) платежа по указанному каналу.

В терминах математического графа сообщение node_announcement – это информация, необходимая для создания узлов или вершин графа. Сообщение channel_announcement позволяет нам создавать ребра графа, представляющие платежные каналы. Поскольку каждое направление платежного канала имеет свой собственный остаток, мы создаем ориентированный граф. Сообщение channel_update позволяет включать комиссионные и привязки ко времени для установки стоимости или веса ребер графа.

В зависимости от алгоритма, который мы будем использовать для отыскания пути, для ребер графа можно задать ряд разных стоимостных функций.

На данный момент давайте проигнорируем стоимостную функцию и просто создадим канальный граф, показывающий узлы и каналы, используя сообщения node_announcement и channel_announcement.

В этой главе мы увидим, как Селена пытается найти способ заплатить Рашиду миллион сатоши. Для начала Селена строит канальный граф, используя информацию из сплетен сети Lightning с целью обнаружения узлов и каналов. Затем Селена проведет разведывательный анализ своего канального графа, чтобы найти способ отправить платеж Рашиду.

Это канальный граф Селены. Такого понятия, как универсальный канальный граф, не существует, всегда существует только некий канальный граф, и он всегда строится с точки зрения узла, который его построил (см. вставку «Взаимосвязь между картой и территорией» ниже).

Селена не создает канальный граф только при отправке платежа. Скорее наоборот, узел Селены постоянно строит и обновляет канальный граф. С того момента, как узел Селены запустится и подсоединится к любому одноранговому узлу в сети, он будет участвовать в сплетнях и использовать каждое сообщение, чтобы узнать о сети как можно больше.

Взаимосвязь между картой и территорией

Со страницы Википедии, посвященной взаимосвязи между картой и территорией[89]: «Взаимосвязь между картой и территорией описывает взаимосвязь между объектом и представлением этого объекта, как во взаимосвязи между географической территорией и ее картой».

[89] См. https://en.wikipedia.org/wiki/Map–territory_relation.

> Взаимосвязь между картой и территорией лучше всего проиллюстрирована в коротком рассказе Льюиса Кэрролла «Сильвия и Бруно пришли к выводу», в котором описывается вымышленная карта территории в масштабе 1:1, поэтому обладающая идеальной точностью, но становящаяся совершенно бесполезной, поскольку она охватывала бы всю территорию, если ее развернуть.
>
> Что это означает для сети Lightning? Сеть Lightning – это территория, а канальный граф – это карта этой территории.
>
> В то время как мы могли бы представить себе теоретический (платонический идеальный) канальный граф, который представляет полную, актуальную карту сети Lightning, такая карта – это просто сама сеть Lightning. Каждый узел имеет свой собственный канальный граф, который строится на основе объявлений и с неизбежностью является неполным, неправильным и устаревшим!
>
> Карта никогда не сможет полностью и точно описать территорию.

Селена прослушивает сообщения `node_announcement` и обнаруживает четыре других узла (в дополнение к Рашиду, намеченному получателю). Результирующий граф представляет собой сеть из шести узлов: Селена и Рашид являются соответственно отправителем и получателем; Алиса, Боб, Ксавье и Ян являются промежуточными узлами. Первоначальный граф Селены – это просто список узлов, показанный на рис. 12-3.

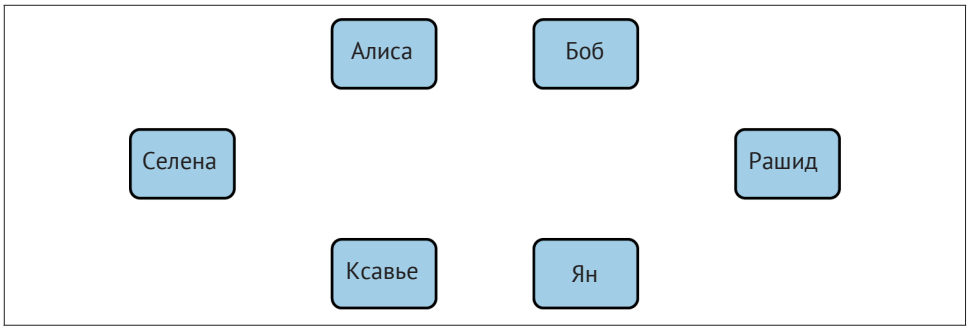


Рис. 12-3. Объявления узлов

Селена также получает семь сообщений `channel_announcement` с соответствующими емкостями каналов, что позволяет ей построить базовую «карту» сети, показанную на рис. 12-4. (Имена Алиса, Боб, Селена, Ксавье, Ян и Рашид были заменены их инициалами на английском, соответственно: A, B, S, X и R.)

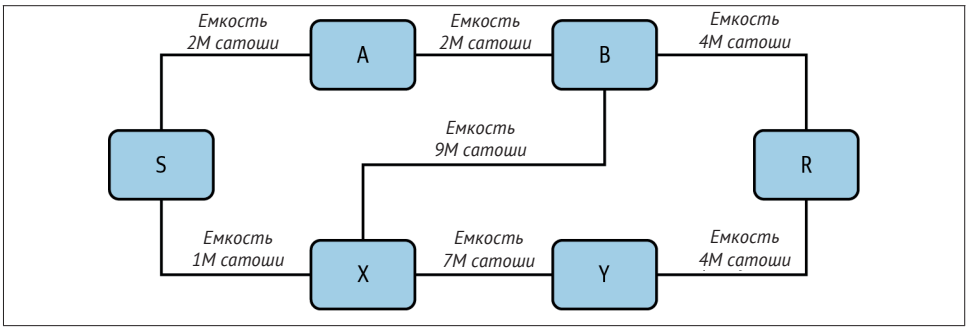


Рис. 12-4. Канальный граф

Неопределенность в канальном графе

Как видно по рис. 12-4, Селена не знает ни одного остатка каналов. Ее начальный канальный граф содержит самый высокий уровень неопределенности.

Но подождите: Селена действительно знает некоторые остатки каналов! Она знает остатки каналов, которые ее собственный узел соединил с другими узлами. Хотя это кажется не так уж и много, на самом деле это очень важная для построения пути информация – Селена знает фактическую ликвидность своих собственных каналов. Давайте обновим канальный граф, чтобы показать эту информацию. Для представления неизвестных остатков мы будем использовать символ «?», как показано на рис. 12-5.

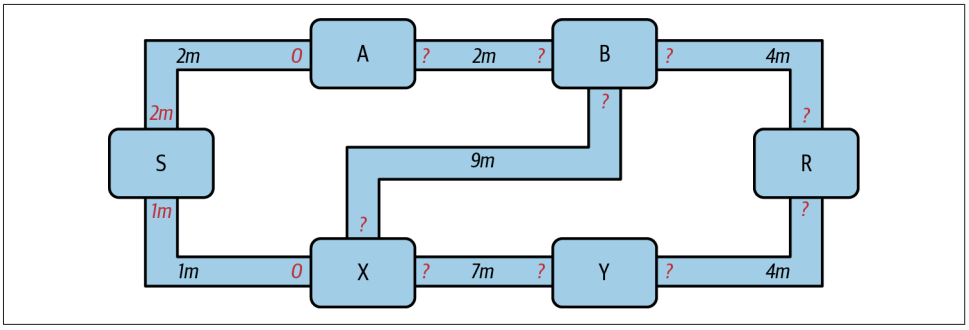


Рис. 12-5. Канальный граф с известными и неизвестными остатками

Хотя символ «?» выглядит зловещим, отсутствие уверенности – это не то же самое, что полное невежество. Неопределенность можно квантифицировать и ее сократить, обновив граф с помощью успешных/безуспешных HTLC-контрактов, которые мы пробуем.

Неопределенность может быть квантифицирована, поскольку мы знаем максимальную и минимальную возможную ликвидность и можем рассчитать вероятности для меньших (более точных) диапазонов.

Как только мы попытаемся отправить HTLC-контракт, мы сможем узнать больше об остатках каналов: если нам он будет успешным, то, значит, остаток был, по меньшей мере, достаточным для передачи определенной суммы.

Между тем если мы получаем ошибку «временный сбой канала», то наиболее вероятной причиной является нехватка ликвидности на определенную сумму.

Возможно, вы думаете, какой, дескать, смысл учиться на успешном HTLC-контракте. В конце концов, если он был успешен, то мы «закончили». Но учтите, что мы, возможно, отправляем одну часть многокомпонентного платежа. Мы также, возможно, отправляем другие платежи отдельными частями в течение короткого времени. Все, что мы узнаем о ликвидности, пригодится для следующей попытки!

Неопределенность ликвидности и вероятность

В целях квантификации неопределенности ликвидности канала можно применить теорию вероятностей. Базовая модель вероятности доставки платежа приведет к некоторым довольно очевидным, но важным выводам:

- меньшие платежи имеют больше шансов на успешную доставку по всему маршруту;
- каналы с большей емкостью дадут нам больше шансов на доставку платежей на определенную сумму;
- чем больше каналов (переходов), тем меньше шансов на успех.

Хотя они могут быть очевидными, они имеют важные последствия, в особенности для использования многокомпонентных платежей (см. раздел «Многокомпонентные платежи» на стр. 314). Понять математику будет нетрудно.

Давайте воспользуемся теорией вероятностей, чтобы понять, как мы пришли к этим выводам.

Сначала давайте предположим, что канал с емкостью c имеет ликвидность на одной стороне с неизвестным значением в диапазоне (0, c) или «диапазоне от 0 до c». Например, если емкость равна 5, то ликвидность будет находиться в диапазоне (0, 5). Теперь, исходя из этого, мы видим, что если мы хотим отправить 5 сатоши, то наши шансы на успех составляют всего 1 из 6 (16.66 %), потому что мы добьемся успеха только в том случае, если ликвидность будет ровна в точности 5.

Проще говоря, если возможные значения ликвидности равны 0, 1, 2, 3, 4, и 5, то только одного из этих шести возможных значений будет достаточно для отправки нашего платежа. Продолжая этот пример, если бы сумма нашего платежа составляла 3, то мы добились бы успеха, если бы ликвидность была 3, 4 или 5. Таким образом, наши шансы на успех составляют 3 из 6 (50 %). Выраженная в математике функция вероятности успеха для одного канала равна:

$$P_c(a) = (c + 1 - a) / (c + 1),$$

где a – это сумма, а c – емкость.

Из уравнения мы видим, что если сумма близка к 0, то вероятность близка к 1, тогда как если сумма превышает емкость, то вероятность равна нулю.

Другими словами: «Меньшие платежи имеют больше шансов на успешную доставку» или «Каналы с большей емкостью дают нам больше шансов на доставку на определенную сумму» и «Невозможно отправить платеж по каналу с недостаточной емкостью».

Теперь давайте подумаем о вероятности успеха на пути, состоящем из нескольких каналов. Допустим, первый канал имеет 50%-ную вероятность успеха ($P = 0.5$). Тогда если второй канал имеет 50%-ный шанс на успех ($P = 0.5$), то интуитивно понятно, что совокупный шанс составляет 25 % ($P = 0.25$).

Это можно выразить в виде уравнения, которое вычисляет вероятность успешного платежа как произведение вероятностей каждого канала в пути (путях):

$$P_{\text{платеж}} = \prod_i^n = 1P_i,$$

где P_i – это вероятность успеха по одному пути или каналу, а $P_{\text{платеж}}$ – совокупная вероятность успешного платежа по всем путям/каналам.

Из уравнения мы видим, что поскольку вероятность успеха по одному каналу всегда меньше или равна 1, то вероятность по многочисленным каналам будет падать экспоненциально.

Другими словами, «чем больше каналов (переходов) вы используете, тем меньше шансов на успех».

За неопределенностью ликвидности в каналах стоит много математической теории и моделирования. Фундаментальную работу по моделированию интервалов неопределенности ликвидности канала можно найти в статье «Безопасность и конфиденциальность платежей в сети Lightning с неопределенными остатками каналов» (соавтора этой книги) Пикхардта и соавт.[90]

Комиссионные и другие метрики канала

Далее наш отправитель добавит в граф информацию из сообщений `channel_update`, полученных от промежуточных узлов. Напомним, что сообщение `channel_update` содержит много информации о канале и ожиданиях одного из партнеров канала.

На рис. 12-6 мы видим, как Селена может обновлять канальный граф на основе сообщений об обновлении channel_update от A, B, X и Y. Обратите внимание, что ИД канала и направление канала (включенные в channel_flags) сообщают Селене, к какому каналу и в каком направлении относится это обновление. Каждый партнер по каналу передает одно или несколько сообщений channel_update, чтобы объявить о своих ожиданиях и другой информации о канале. Например, в левом верхнем углу мы видим channel_update, отправленный Алисой для канала A–B и направления «A в B». С помощью этого обновления Алиса сообщает сети, сколько она будет брать комиссионными за маршрутизацию HTLC-контракта Бобу по этому конкретному каналу. Боб может объявить об обновлении канала (не показано на этой диаграмме) для противоположного направления с совершенно другими ожиданиями комиссионных. Любой узел может отправить новый channel_update, изменяя ожидания в отношении комиссионных или привязки ко времени в любое время.

[90] Security and Privacy of Lightning Network Payments with Uncertain Channel Balances, Pickhardt et al. См. https://arxiv.org/abs/2103.08576.

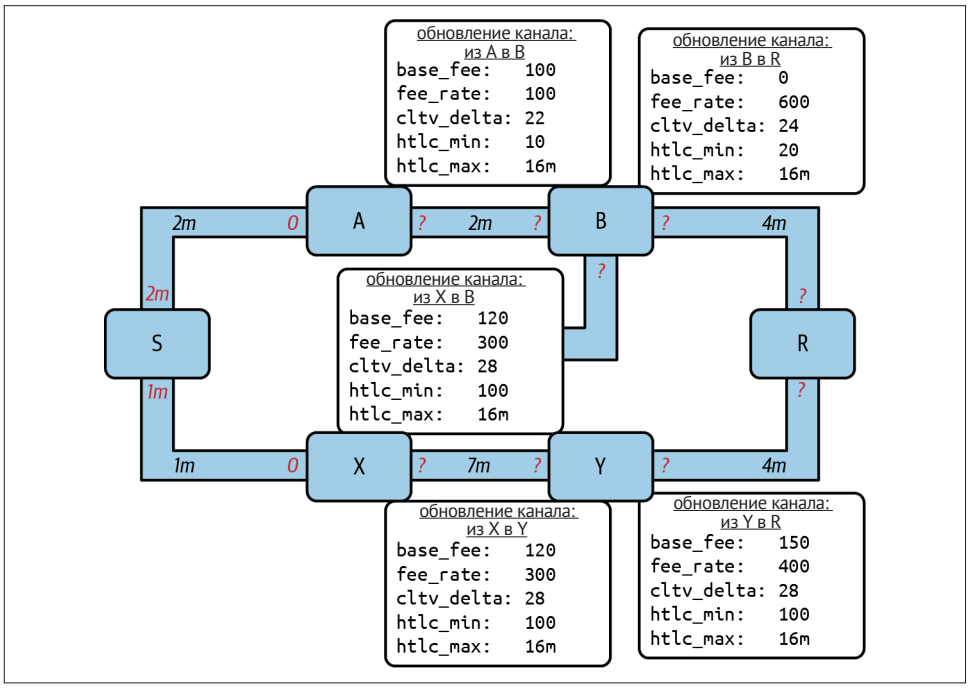


Рис. 12-6. Комиссионные в канальном графе и другие метрики канала

Информация о комиссионных и привязки ко времени очень важна, и не только как метрики выбора пути. Как мы видели в главе 10, отправителю необходимо суммировать комиссионные и привязки ко времени (`cltv_expiry_delta`) при каждом переходе, чтобы создать луковицу. Процесс расчета комиссионных происходит от получателя к отправителю в обратном направлении пути, потому что каждый промежуточный переход ожидает входящий HTLC-контракт с большей суммой и временем истечения срока, чем исходящий HTLC-контракт, который он отправит на следующий переход. Так, например, если Боб хочет получить 1000 сатоши в виде комиссионных и 30 блоков с дельтой истечения срока привязки ко времени, чтобы отправить платеж Рашиду, то эта сумма и дельта истечения срока должны быть добавлены в HTLC-контракт *от* Алисы.

Также важно отметить, что канал должен обладать ликвидностью, достаточной не только для суммы платежа, но и для кумулятивных комиссионных за все последующие переходы. Несмотря на то что канал Селены к Ксавье (S → X) имеет достаточную ликвидность для оплаты в размере 1M сатоши, у него недостаточно ликвидности, как только мы учтем комиссионные. Нам нужно знать комиссионные, потому что будут рассматриваться только те пути, которые обладают достаточной ликвидностью как для платежа, так и для всех комиссионных.

Отыскание кандидатных путей

Отыскание подходящего пути через подобного рода ориентированный граф является хорошо изученной задачей информатики (широко известной как задача о кратчайшем пути), которая может быть решена с помощью различных алгоритмов в зависимости от желаемой оптимизации и ресурсных ограничений.

Самый известный решающий эту задачу алгоритм был изобретен голландским математиком Э. В. Дейкстрой в 1956 году и известен просто как алгоритм Дейкстры[91]. В дополнение к изначальному алгоритму Дейкстры существует множество вариаций и оптимизаций, таких как A* («A-звездочка»)[92], который представляет собой алгоритм, основанный на эвристике.

Как упоминалось ранее, «поиск» должен применяться в обратном порядке для учета комиссионных, которые накапливаются от получателя к отправителю. Таким образом, Дейкстра, A* либо какой-нибудь другой алгоритм будут искать путь от получателя к отправителю, используя комиссионные, оценочную ликвидность и дельту привязки ко времени (или некоторую их комбинацию) в качестве стоимостной функции для каждого перехода.

Используя один из таких алгоритмов, Селена вычисляет несколько возможных путей к Рашиду, отсортированных по кратчайшему пути:

1. S→A→B→R.
2. S→X→Y→R.
3. S→X→B→R.
4. S→A→B→X→Y→R.

Но, как мы видели ранее, канал S → X не обладает достаточной ликвидностью для выплаты 1M сатоши после учета комиссионных. Таким образом, пути 2 и 3 нежизнеспособны. Это оставляет пути 1 и 4 в качестве возможных путей платежа.

Имея два возможных пути, Селена готова предпринять попытку доставки!

Доставка платежа (цикл проб и ошибок)

Узел Селены запускает цикл проб и ошибок, конструируя HTLC-контракты, собирая луковицу и пытаясь отправить платеж. Для каждой попытки есть три возможных исхода:

- успешный результат (`update_fulfill_htlc`);
- ошибка (`update_fail_htlc`);
- «застрявший» платеж без ответа (ни успеха, ни неудачи).

Если платеж завершится сбоем, то его можно повторить по другому пути, обновив граф (изменив метрики канала) и пересчитав альтернативный путь.

Мы рассмотрели, что произойдет, если платеж «застрял», в разделе «Застрявшие платежи» на стр. 278. Важная деталь заключается в том, что застрявший платеж является наихудшим результатом, потому что мы не можем повторить попытку с другим HTLC-контрактом, поскольку оба (застрявший и повторный) могут в конечном итоге пройти и вызвать двойной платеж.

[91] См. https://en.wikipedia.org/wiki/Dijkstra's_algorithm.

[92] См. https://en.wikipedia.org/wiki/A*_search_algorithm.

Первая попытка (путь №1)

Селена пытается пройти первый путь (S→A→B→R). Она конструирует луковицу и отправляет ее, но получает код сбоя от узла Боба. Боб сообщает о временном сбое канала с помощью `channel_update`, идентифицирующего канал B→R как тот, которого не может доставить. Эта попытка показана на рис. 12-7.

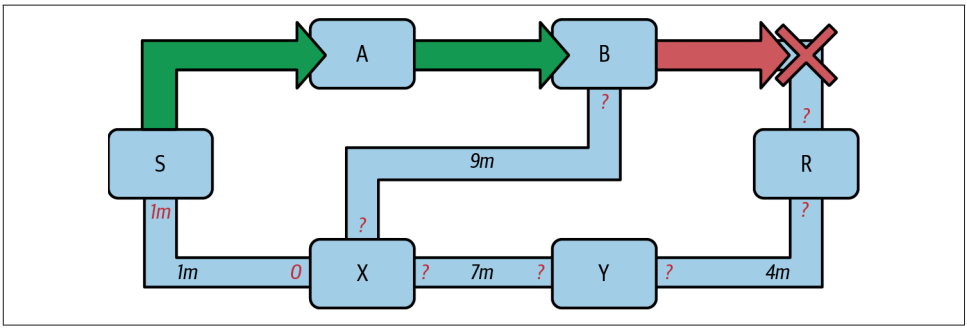


Рис. 12-7. Попытка пути № 1 завершается сбоем

Учеба на ошибках

Из этого кода сбоя Селена сделает вывод, что у Боба недостаточно ликвидности для доставки платежа Рашиду по этому каналу. Важно отметить, что этот сбой сужает неопределенность ликвидности данного канала! Ранее узел Селены исходил из того, что ликвидность канала на стороне Боба находилась где-то в диапазоне (0, 4M). Теперь она сможет допустить, что ликвидность находится в диапазоне (0, 999999). Аналогичным образом Селена теперь может допустить, что ликвидность этого канала на стороне Рашида находится в диапазоне (1M, 4M) вместо (0, 4M). Селена многому научилась из этого неуспешного платежа.

Вторая попытка (путь № 4)

Теперь Селена пытается пройти четвертый кандидатный путь (S→A→B→X→Y→R). Это более длительный путь, он повлечет за собой дополнительные комиссионные, но сейчас это лучший вариант для доставки платежа.

К счастью, Селена получает сообщение `update_fulfill_htlc` от Алисы, указывающее на то, что платеж прошел успешно, как показано на рис. 12-8.

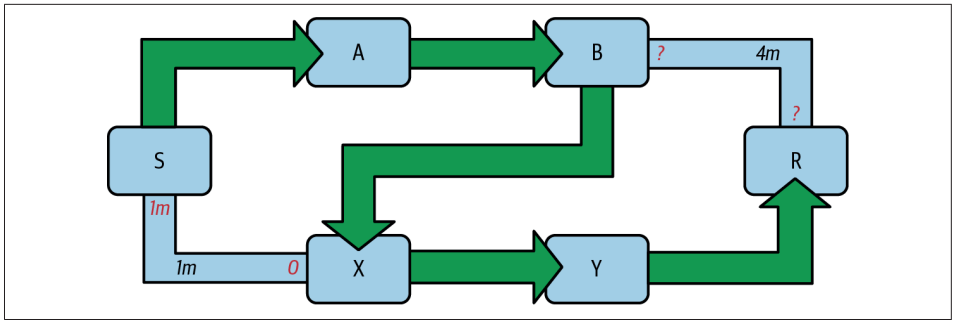


Рис. 12-8. Попытка пути № 4 завершилась успешно

Учеба на успехах

Селена также многому научилась из этого успешного платежа. Теперь она знает, что все каналы на пути S → A → B → X → Y → R имели ликвидность, достаточную для доставки платежа. Кроме того, она еще знает, что каждый из этих каналов переместил сумму HTLC-контракта (1M + комиссионные) на другой конец канала. Это позволяет Селене пересчитать диапазон ликвидности на принимающей стороне всех каналов на этом пути, заменяя минимальную ликвидность суммой 1M + комиссионные.

Застоявшиеся знания?

Теперь у Селены гораздо более точная «карта» сети Lightning (по меньшей мере в том, что касается этих семи каналов). Указанные знания будут полезны для любых последующих платежей, которые Селена попытается произвести.

Однако эти знания становятся несколько «застоявшимися», по мере того как другие узлы отправляют или маршрутизируют платежи. Селена никогда не увидит ни одного из этих платежей (если только она не является отправителем). Даже если она участвует в маршрутизации платежей, механизм луковичной маршрутизации означает, что она может видеть изменения лишь для одного перехода (ее собственные каналы).

Поэтому узел Селены должен учитывать продолжительность хранения этих знаний, прежде чем допускать, что они застоялись и больше не полезны.

Многокомпонентные платежи

Многокомпонентные платежи (Multipart payments, аббр. MPP) – это функциональность, которая была введена в сеть Lightning в 2020 году и уже очень широко доступна. Многокомпонентные платежи позволяют делить платеж на несколько частей, которые отправляются в виде HTLC-контрактов по нескольким разным путям намеченному получателю, сохраняя атомарность совокупного платежа. В этом контексте атомарность означает, что либо все HTLC-части платежа в конечном итоге будут исполнены, либо весь платеж завершается сбоем, и все HTLC-части завершаются сбоем. Нет никакой возможности для частично успешной оплаты.

Многокомпонентные платежи являются значительным усовершенствованием в сети Lightning, поскольку они позволяют отправлять суммы, которые не «поместятся» ни в одном отдельном канале, деля их на меньшие суммы, для которых имеется достаточная ликвидность. Кроме того, было показано, что многокомпонентные платежи повышают вероятность успешного платежа по сравнению с однопутным платежом.

Теперь, когда функциональность MPP доступна, лучше всего рассматривать однопутный платеж как подкатегорию MPP. По сути, один путь – это просто многокомпонентная часть размера 1. Все платежи могут рассматриваться как многокомпонентные платежи, если только размер платежа и доступная ликвидность не позволяют осуществлять доставку одной частью.

Использование MPP

MPP – это не то, что выберет пользователь, а скорее стратегия отыскания пути к узлу и доставки платежей. Имплементированы те же базовые шаги: создать граф, выбрать пути и пройти по циклу проб и ошибок. Разница в том, что при выборе пути мы также должны учитывать то, как делить платеж с целью оптимизации доставки.

В нашем примере можно увидеть несколько немедленных улучшений в задаче отыскания пути, которые становятся возможными благодаря MPP. Во-первых, мы можем использовать канал S → X, который, как известно, имеет недостаточную ликвидность для транспортировки 1M сатоши плюс комиссионные. Отправляя меньшую часть по этому каналу, мы можем использовать пути, которые ранее были недоступны. Во-вторых, у нас есть неизвестная ликвидность канала B → R, которой недостаточно для транспортировки суммы в 1M, но может быть достаточно для транспортировки меньшей суммы.

Разбивка платежей

Фундаментальный вопрос заключается в том, как разбить платежи, а конкретнее каково оптимальное число делений и оптимальные суммы для каждого деления.

Это область текущих исследований, в которой появляются новые стратегии. Многокомпонентные платежи приводят к другому алгоритмическому подходу, чем однопутные платежи, даже несмотря на то, что однопутные решения могут быть получены в результате многокомпонентной оптимизации (т. е. один путь может быть оптимальным решением, предложенным многокомпонентным алгоритмом отыскания пути).

Если вы помните, мы обнаружили, что неопределенность ликвидности/остатков приводит к некоторым (довольно очевидным) выводам, которые мы можем применить при отыскании пути методом MPP, а именно:

- меньшие платежи имеют более высокие шансы на успех;
- чем больше каналов вы используете, тем меньше шансов на успех (в геометрической прогрессии).

Исходя из первого из этих выводов, мы могли бы сделать вывод о том, что деление крупного платежа (например, 1M сатоши) на крошечные платежи увеличивает вероятность того, что каждый из этих малых платежей будет успешным. Число возможных путей с достаточной ликвидностью будет больше, если мы отправим меньшие суммы.

Доведя эту идею до крайности, почему бы не разделить платеж в 1M сатоши на миллион отдельных частей по одному сатоши? Что ж, ответ кроется в нашей второй идее: поскольку мы будем использовать больше каналов/путей для отправки наших миллионов HTLC-контрактов с одним сатоши, наши шансы на успех упадут в геометрической прогрессии.

Если это не очевидно, то две приведенные выше идеи создают «золотую средину», где мы можем максимизировать наши шансы на успех: деление на более мелкие платежи, но не слишком много делений!

Квантифицирование этого оптимального равновесия размера/числа делений для данного канального графа выходит за рамки данной книги, но это

активная область исследований. В некоторых текущих имплементациях используется очень простая стратегия деления платежа на две половины, четыре четверти и т. д.

Для того чтобы узнать больше о задаче оптимизации, именуемой потоками с минимальной стоимостью, связанными с делением платежей на разные размеры и распределением их по путям, см. статью «Оптимально надежные и дешевые платежные потоки в сети Lightning» (соавтора этой книги) Рене Пикхардт и Стефан Рихтер[93].

В нашем примере узел Селены попытается разделить платеж в 1M сатоши на две части соответственно по 600k и 400k сатоши и отправить их по двум разным путям. Это показано на рис. 12-9.

Поскольку канал S → X теперь можно использовать, и (к счастью для Селены) канал B → R обладает достаточной ликвидностью для 600k сатоши, две части успешно продвигаются по путям, которые ранее были невозможны.

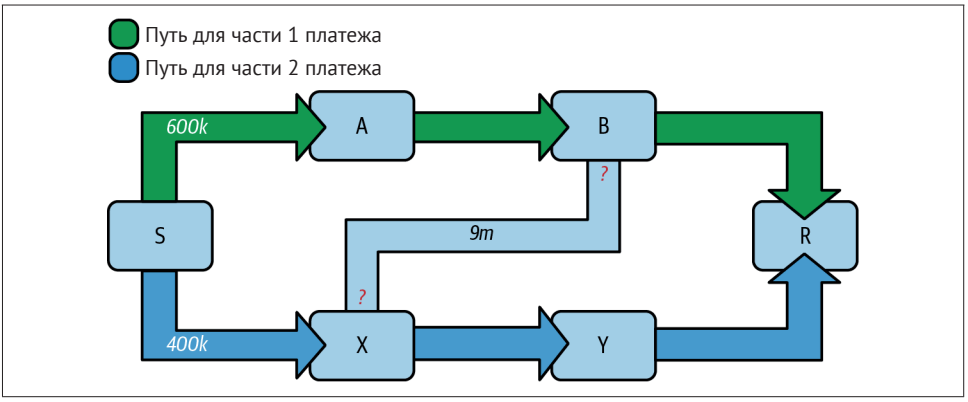


Рис. 12-9. Отправка двух частей многокомпонентного платежа

Метод проб и ошибок в течение нескольких «раундов»

Многокомпонентные платежи приводят к несколько измененному циклу проб и ошибок для доставки платежей. Поскольку мы пытаемся использовать несколько путей в каждой попытке, у нас есть четыре возможных исхода:

- все части выполнены успешно, оплата прошла успешно;
- некоторые части выполняются успешно, некоторые завершаются сбоем с возвращением ошибок;
- все части заканчиваются сбоем с возвращением ошибок;
- некоторые части «застряли», ошибки не возвращаются.

Во втором случае, когда некоторые части сбоят с возвращенными ошибками, а некоторые части завершаются успешно, мы можем повторить цикл проб и ошибок, но только для остаточной суммы.

[93] Optimally Reliable & Cheap Payment Flows on the Lightning Network (coauthor of this book) René Pickhardt, Stefan Richter. См. https://arxiv.org/abs/2107.05322.

Давайте допустим, например, что у Селены был гораздо более крупный канальный граф с сотнями возможных путей для достижения Рашида. Ее алгоритм отыскания пути мог бы найти оптимальное деление платежа, состоящее из 26 частей разного размера. После попытки отправить все 26 частей в первом раунде три из этих частей завершились с ошибкой.

Если бы эти три части состояли, скажем, из 155k сатоши, то Селена возобновила бы усилие по отысканию пути только для 155k сатоши. В следующем раунде можно было бы найти совершенно другие пути (оптимизированные для остаточной суммы в 155k) и разделить сумму в 155k на совершенно разные части!

Хотя кажется, что 26 разделенных частей – это много, тесты в сети Lightning успешно доставили платеж в размере 0.3679 BTC, разделив его на 345 частей.

Кроме того, узел Селены обновит канальный граф, используя информацию, полученную из успехов и ошибок первого раунда, чтобы найти наиболее оптимальные пути и деления для второго раунда.

Предположим, что узел Селены вычисляет, что самый лучший способ отправить остаток в 155k состоит в том, чтобы разделить на 6 частей по 80k, 42k, 15k, 11k, 6.5k и 500 сатоши. В следующем раунде Селена получает только одну ошибку, указывающую на то, что часть с 11k сатоши оказалась безуспешной. Опять же, Селена обновляет канальный граф на основе собранной информации и снова запускает отыскание пути, чтобы отправить остаток в 11k. На этот раз она преуспевает с двумя частями соответственно по 6k и 5k сатоши.

Этот многораундовый пример отправки платежа с использованием MPP показан на рис. 12-10.

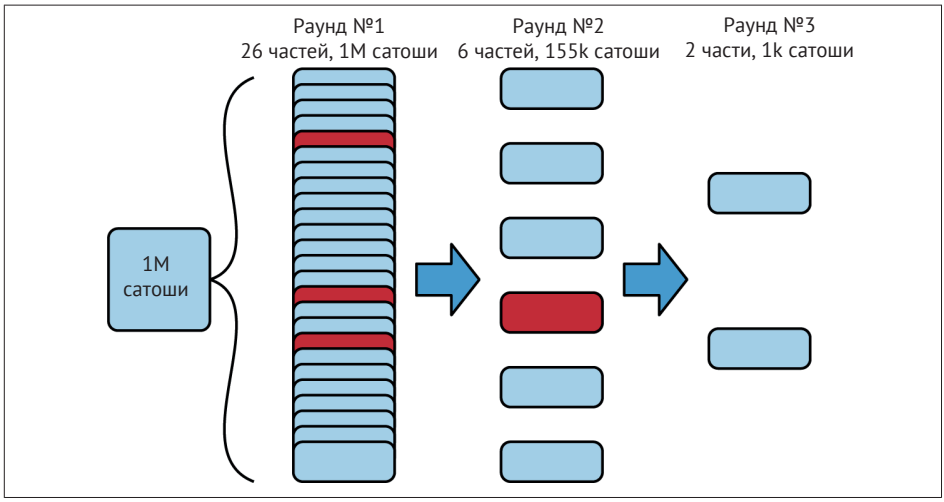


Рис. 12-10. Отправка платежа в несколько раундов с помощью MPP

В конце концов, узел Селены использовал три раунда отыскания пути, чтобы отправить 1M сатоши в 30 частях.

Вывод

В этой главе мы рассмотрели отыскание путей и доставку платежей. Увидели, как использовать канальный граф для отыскания путей от отправителя к получателю. Мы также увидели, как отправитель будет пытаться доставить платежи по выбранному пути и повторять их в цикле проб и ошибок.

Мы проинспектировали неопределенность ликвидности канала (с точки зрения отправителя) и последствия, которые это имеет для отыскания пути. Мы увидели, как можно квантифицировать неопределенность и использовать теорию вероятностей, чтобы сделать некоторые полезные выводы. Также мы увидели, как можно уменьшать неопределенность, извлекая уроки как из успешных, так и из безуспешных платежей.

Наконец, мы увидели, как недавно развернутая функциональность многокомпонентных платежей позволяет делить платежи на части, повышая вероятность успеха даже для более крупных платежей.

Глава 13

Проводной протокол: фреймирование и расширяемость

В этой главе мы углубимся в проводной протокол сети Lightning, а также рассмотрим все различные рычаги расширения, встроенные в протокол. К концу этой главы амбициозный читатель должен быть в состоянии написать свой собственный анализатор проводных протоколов для сети Lightning. В дополнение к возможности написать конкретно-прикладной анализатор проводного протокола читатель этой главы получит глубокое понимание различных механизмов модернизации, встроенных в протокол.

Слой обмена сообщениями в рамках комплекта протоколов Lightning

Слой обмена сообщениями, который подробно описан в этой главе, состоит из «фреймирования и формата сообщения», кодировки «тип–длина–значение» и «битов функциональностей». Эти компоненты выделены контуром в наборе протоколов, показанном на рис. 13-1.

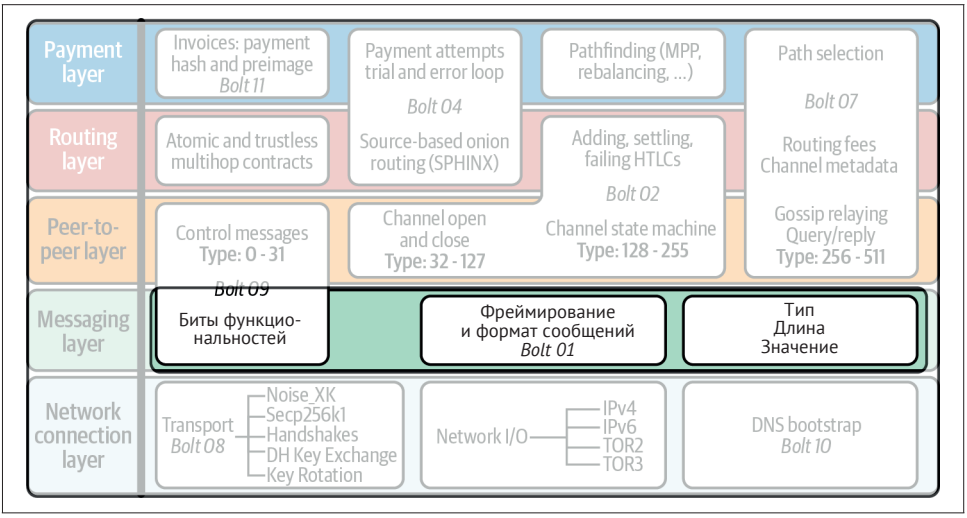


Рис. 13-1. Слой обмена сообщениями в рамках комплекта протоколов Lightning

Проводное фреймирование

Мы начнем с описания высокоуровневой структуры проводного фреймирования внутри протокола. Когда мы говорим «фреймирование» (обрамление, кадрирование – framing), то имеем в виду способ упаковки байтов по проводу (по линии передачи) для кодирования определенного протокольного сообщения. Без знания используемой в протоколе системы фреймирования цепочка байтов по проводу будет напоминать серию случайных байтов, поскольку структура не была предписана. Применяя правильное фреймирование для декодирования этих байтов по проводу, мы сможем извлекать структуру и, наконец, проводить лексический разбор этой структуры в протокольных сообщениях на нашем языке более высокого уровня.

Важно отметить, что сеть Lightning представляет собой сквозной шифрованный протокол, а фреймирование само по себе инкапсулировано внутрь слоя шифрованного транспорта сообщений. Как мы увидим в главе 14, для обработки транспортного шифрования в сети Lightning используется конкретно-прикладной вариант протокола Noise. В этой главе, всякий раз, когда мы приводим пример фреймирования, мы исходим из того, что слой шифрования уже удален (при декодировании) или что мы еще не зашифровали набор байтов перед отправкой их по проводу (кодирование).

Высокоуровневое фреймирование

С учетом сказанного мы готовы описать высокоуровневую схему, используемую для кодирования сообщений по проводу:

- сообщения по проводу начинаются с 2-байтового поля типа, за которым следует полезный груз сообщения;
- размер самого полезного груза сообщения может достигать 65 Кб;

- все целые числа кодируются в алфавитном порядке (сетевой порядок);
- любые байты, следующие за определенным сообщением, могут быть безопасно проигнорированы.

Да, вот и все. Поскольку протокол основан на инкапсулирующем слое протокола транспортного шифрования, нам не нужна явная длина каждого типа сообщения. Это связано с тем, что транспортное шифрование работает на уровне сообщения, поэтому к тому времени, когда мы будем готовы декодировать следующее сообщение, мы уже будем знать суммарное число байтов самого сообщения. Использование 2 байт для типа сообщения (закодированного в порядке от старшего в младшему) означает, что протокол может содержать до 2^16 – 1, или 65 535, разных сообщений. В продолжение темы, поскольку мы знаем, что все сообщения должны быть меньше 65 Кб, это упрощает лексический разбор сообщений, т. к. мы можем использовать буфер фиксированного размера и поддерживать строгие ограничения на суммарный объем памяти, необходимый для лексического разбора входящего проводного сообщения.

Последний пункт позволяет обеспечивать определенную степень обратной совместимости, поскольку новые узлы могут предоставлять информацию в проводных сообщениях, которую более старые узлы (могущие их не понимать) могут безопасно игнорировать. Как мы увидим впоследствии, эта функциональная возможность в сочетании с очень гибким форматом расширения проводных сообщений позволяет протоколу обеспечивать и прямую совместимость.

Кодировка типа

Получив эту высокоуровневую предпосылку, мы теперь начнем с самого примитивного слоя: лексического разбора примитивных типов. В дополнение к кодированию целых чисел протокол Lightning также позволяет кодировать широкий спектр типов, включая фрагменты байтов переменной длины, публичные ключи на основе эллиптической кривой, Bitcoin-адреса и подписи. Когда мы будем описывать структуру проводных сообщений позже в этой главе, то будем ссылаться на высокоуровневый тип (абстрактный тип), а не на представление указанного типа на более низком уровне. В данном разделе мы отслаиваем этот слой абстракции, чтобы убедиться, что наш будущий проводной лексический анализатор способен надлежаще кодировать/декодировать любой тип более высокого уровня.

В табл. 13-1 мы соотносим имя данного типа сообщения с высокоуровневой процедурой, используемой для кодирования/декодирования этого типа.

Таблица 13-1. Высокоуровневые типы сообщений

Высокоуровневый тип	Фреймирование	Комментарий
`node_alias`	32-байтовый срез фиксированной длины	При декодировании отклоняется, если содержимое не является валидным UTF-8
`channel_id`	32-байтовый срез фиксированной длины, который соотносит выходную точку с 32-байтовым значением	Имея выходную точку, ее можно конвертировать в `channel_id`, взяв TxID выходной точки и применив операцию XOR к ней и индексу (интерпретируемым как младшие 2 байта)

Окончание табл. 13-1

Высокоуровневый тип	Фреймирование	Комментарий
`short_chan_id`	Беззнаковое 64-битовое целое число (uint64)	Состоящий из высоты блока (24 бита), индекса транзакции (24 бита) и выходного индекса (16 бит), упакованный в 8 байт
`milli_satoshi`	Беззнаковое 64-битовое целое число (uint64)	Представляет 1000-ю часть сатоши
`satoshi`	Беззнаковое 64-битовое целое число (uint64)	Базовая единица биткойна
`pubkey`	Публичный ключ secp256k1, кодированный в сжатый формат, занимающий 33 байта	Занимает фиксированную длину 33 байта по проводу
`sig`	ECDSA-подпись эллиптической кривой secp256k1	Кодируется как фиксированный 64-байтовый срез, упакованный как R || S
`uint8`	8-битовое целое число	
`uint16`	16-битовое целое число	
`uint64`	64-битовое целое число	
`[]byte`	Байтовый срез переменной длины	Предваряется 16-битовым целым числом, обозначающим длину байтов
`color_rgb`	Цветовая кодировка RGB	Кодируется как серия 8-битовых целых чисел
`net_addr`	Кодировка сетевого адреса	Кодируется 1-байтовым префиксом, который обозначает тип адреса, за которым следует тело адреса

В следующем разделе мы опишем структуру каждого проводного сообщения, включая тип префикса сообщения вместе с содержимым его тела.

Расширения «Тип–длина–значение» для сообщений

Ранее в этой главе мы упоминали, что размер сообщений может достигать 65 Кб, и если при лексическом разборе сообщения остаются лишние байты, то эти байты следует игнорировать. На первый взгляд это требование может показаться несколько произвольным; однако оно допускает отстыкованную десинхронизированную эволюцию самого протокола Lightning. Мы обсудим это подробнее ближе к концу главы. Но сначала обратим наше внимание на то, для чего именно могут быть использованы эти «дополнительные байты» в конце сообщения.

Протокол буферизует формат сообщения

Формат сериализации сообщений Protobuf (Protocol Buffers, Протокольные буферы) начинался как внутренний формат, используемый в Google, и превратился в один из самых популярных форматов сериализации сообщений, ис-

пользуемых разработчиками по всему миру. Формат Protobuf описывает, как сообщение (обычно какая-то структура данных, связанная с API) кодируется по проводу и декодируется на другом конце. На десятках языков существует несколько «компиляторов Protobuf», которые действуют как мост, позволяющий любому языку кодировать Protobuf, который может декодироваться с помощью совместимого декодирования на другом языке. Такая межъязыковая совместимость структур данных обеспечивает широкий спектр инноваций, поскольку можно передавать структуру и даже типизированные структуры данных через границы языка и абстракции.

Протокольные буферы Protobuf также известны своей гибкостью в отношении того, как они обрабатывают изменения в базовой структуре сообщений. До тех пор, пока схема нумерации полей соблюдается, новое написание буферов Protobuf может включать информацию внутрь Protobuf, которая может быть неизвестна любым более старым читателям. Когда старый читатель сталкивается с новым сериализованным форматом, если есть типы/поля, которые он не понимает, то он их просто игнорирует. Это позволяет старым клиентам и новым клиентам сосуществовать, поскольку все клиенты могут проводить лексический разбор некоторой части нового формата сообщений.

Прямая и обратная совместимости

Протокольные буферы Protobuf чрезвычайно популярны среди разработчиков, потому что они имеют встроенную поддержку как прямой, так и обратной совместимостей. Большинство разработчиков, вероятно, знакомы с концепцией обратной совместимости. Проще говоря, принцип гласит, что любые изменения в формате сообщений или API должны делаться таким образом, чтобы не нарушать поддержку старых клиентов. В рамках наших предыдущих примеров расширяемости Protobuf обратная совместимость достигается за счет обеспечения того, чтобы новые дополнения к формату Protobuf не нарушали известные части старых программ чтения. С другой стороны, прямая совместимость также важна для десинхронизированных обновлений; однако она менее известна. Для того чтобы изменения были прямо совместимы, клиенты должны просто игнорировать любую информацию, которую они не понимают. Можно сказать, что механизм мягкого ответвления, используемый для модернизирования консенсусной системы Bitcoin, является прямо и обратно совместимым: любые клиенты, которые не обновляются, все равно могут использовать Bitcoin, и если они сталкиваются с какими-либо транзакциями, которых они не понимают, то просто их игнорируют, поскольку их денежные средства эти новые функциональности не используют.

Формат «Тип–длина–значение»

Для того чтобы иметь возможность модернизировать сообщения в прямо и обратно совместимой манере в дополнение к битам функциональностей (подробнее об этом позже), в сети Lightning используется конкретно-прикладной формат сериализации сообщений, который просто называется «Тип–длина–значение», или сокращенно TLV, от англ. Type-Length-Value. Указанный формат был навеян широко используемым форматом Protobuf и заимствует многие концепции, значительно упрощая имплементацию, а также программное

обеспечение, которое взаимодействует с лексическим разбором сообщений. Любопытный читатель может спросить: «Почему бы просто не использовать Protobuf?» В ответ разработчики Lightning ответили бы, что мы можем получить максимальную расширяемость Protobuf, а также воспользоваться преимуществами меньшей имплементации и, следовательно, меньшей атаки. Начиная с версии 3.15.6 вес компилятора Protobuf составляет более 656 671 строки кода. Для сравнения, имплементация TLV-формата сообщений в LND весит всего 2.3k строк кода (включая тесты).

Теперь, когда представлена необходимая справочная информация, мы готовы подробно описать TLV-формат. TLV-расширение сообщения является потоком отдельных TLV-записей. Одна TLV-запись состоит из трех компонентов: типа записи, длины записи и, наконец, непрозрачного значения записи:

`type`

Целое число, представляющее имя кодируемой записи.

`length`

Длина записи.

`value`

Непрозрачное значение записи.

Как type, так и length кодируются с использованием целого числа переменного размера, которое навеяно целым числом переменного размера (varint), используемым в P2P-протоколе Bitcoin, сокращенно именуемом BigSize.

Целочисленная кодировка BigSize

В своей полной форме целое число `BigSize` может представлять значение длиной до 64 бит. В отличие от применяемого в Bitcoin формата varint, формат `BigSize` вместо этого кодирует целые числа, используя упорядоченность байтов от старшего в младшему.

Тип varint BigSize состоит из двух компонентов: дискриминанта и тела. В контексте целого числа BigSize дискриминант сообщает декодировщику размер следующего целого числа переменного размера. Напомним, что уникальность целых чисел переменного размера заключается в том, что они предоставляют лексическому анализатору возможность использовать меньше байт для кодирования целых чисел меньшего размера, чем более крупных, экономя пространство. Кодирование целого числа BigSize следует одному из четырех следующих ниже вариантов:

1. Если значение меньше `0xfd` (`253`): тогда дискриминант на самом деле не используется, а кодировкой является просто само целое число. За счет этого можно кодировать очень малые целые числа без дополнительных накладных расходов.
2. Если значение меньше или равно `0xffff` (`65535`): дискриминант кодируется как `0xfd`, что указывает на то, что следующее значение будет больше `0xfd`, но меньше `0xffff`. Тогда число кодируется как 16-битовое целое число. Включая дискриминант, можно закодировать значение, которое больше 253, но меньше 65 535, используя 3 байта.

3. Если значение меньше `0xffffffff` (`4294967295`): дискриминант кодируется как `0xfe`. Тело кодируется с использованием 32-битового целого числа, включая дискриминант, и можно закодировать значение, которое меньше 4 294 967 295, используя 5 байт.
4. В противном случае мы просто кодируем значение как полноразмерное 64-битовое целое число.

Ограничения TLV-кодирования

В контексте TLV-сообщения типы записей ниже `2^16` считаются зарезервированными для использования в будущем. Типы, выходящие за пределы этого диапазона, должны использоваться для «конкретно-прикладных» расширений сообщений, используемых прикладными протоколами более высокого уровня.

Значение (`value`) записи зависит от ее типа (`type`). Другими словами, оно может принимать любую форму, потому что лексические анализаторы будут пытаться интерпретировать его в зависимости от контекста самого типа.

Каноническое TLV-кодирование

Одна из проблем с форматом Protobuf заключается в том, что при кодировании двумя разными версиями компилятора кодировки одного и того же сообщения могут выдавать совершенно разный набор байтов. Такие случаи неканонического кодирования неприемлемы в контексте Lightning, поскольку многие сообщения содержат подпись дайджеста сообщения. Если сообщение может быть закодировано двумя разными способами, то можно непреднамеренно нарушить аутентификацию подписи, перекодировав сообщение с использованием немного другого набора байтов на проводе.

В целях обеспечения того, чтобы все закодированные сообщения были каноническими, при кодировании определяются следующие ниже ограничения:

- все записи в потоке TLV должны быть закодированы в порядке строго возрастающего типа;
- все записи должны минимально кодировать поля `type` и `length`. Другими словами, всегда должно использоваться наименьшее представление `BigSize` целого числа;
- каждый `type` может появиться только один раз в пределах данного потока TLV.

В дополнение к этим ограничениям кодирования также определяется ряд требований к интерпретации более высокого уровня на основе арности данного целого числа `type`. Мы углубимся в эти детали ближе к концу главы, после того как опишем принцип модернизации протокола Lightning на практике и в теории.

Биты функциональностей и расширяемость протокола

Поскольку сеть Lightning является децентрализованной системой, ни одна сущность не может навязывать изменение или модификацию протокола для всех пользователей системы. Эта характеристика также наблюдается в других децентрализованных сетях, таких как Bitcoin. Однако, в отличие от Bitcoin, для изменения подмножества сети Lightning не требуется подавляющего консенсуса.

Lightning способна развиваться по своему желанию без строгих требований координации, потому что, в отличие от Bitcoin, в сети Lightning не требуется глобального консенсуса. Из-за этого факта и нескольких механизмов модернизации, встроенных в сеть Lightning, должны модернизироваться только те участники, которые хотят использовать эти новые функциональности сети Lightning, и тогда они смогут взаимодействовать друг с другом.

В этом разделе мы разведаем различные способы, с помощью которых разработчики и пользователи могут разрабатывать и развертывать новые функциональности в сети Lightning. Разработчики изначальной сети Lightning знали, что существует множество возможных направлений развития сети и лежащего в ее основе протокола. Как следствие они позаботились о том, чтобы имплементировать внутри системы несколько механизмов обеспечения расширяемости, которые можно использовать для ее частичной или полной модернизации несостыкованным, десинхронизированным и децентрализованным способом.

Биты функциональностей как механизм обеспечения обнаруживаемости модернизаций

Проницательный читатель, возможно, заметил различные места, где биты функциональностей включены в протокол Lightning. *Бит функциональности* (feature bit) – это битовое поле, которое может использоваться для рекламы понимания или соблюдения возможной модернизации сетевого протокола. Биты функциональностей обычно назначаются парами, то есть каждая потенциальная новая функциональность/модернизация всегда определяет два бита в битовом поле. Один бит сигнализирует о том, что рекламируемая функциональность является опциональной, означая, что узел знает об этой функциональности и может ее использовать, но не считает ее необходимой для нормальной работы. Другой бит сигнализирует о том, что вместо этого функциональность является обязательной, означая, что узел не продолжит работу, если потенциальный одноранговый узел не понимает эту функциональность.

Используя эти два бита (опциональный и обязательный), можно построить простую матрицу совместимости, с которой узлы/пользователи могут консультироваться, чтобы определять совместимость однорангового узла с желаемой функциональностью, как показано в табл. 13-2.

Таблица 13-2. Матрица совместимости битов функциональностей

Тип бита	Дистанционность опциональна	Дистанционность обязательна	Дистанционность неизвестна
Локальность опциональна	✓	✓	✓
Локальность обязательна	✓	✓	×
Локальность неизвестна	✓	×	×

Из этой упрощенной матрицы совместимости хорошо видно, что до тех пор, пока другая сторона знает о нашем бите функциональности, мы можем взаимодействовать с ней с помощью протокола. Если сторона даже не знает о том, на какой бит мы ссылаемся, и им требуется эта функциональность, то мы с ними несовместимы. В сети сигнализирование о дополнительных функ-

циональностях осуществляется с использованием *нечетного битового числа*, в то время как сигнализирование о требуемых функциональностях осуществляется с использованием *четного битового числа*. В качестве примера, если одноранговый узел сигнализирует о том, что он знает о функциональности, использующей бит 15, тогда мы знаем, что эта функциональность является опциональной, и мы можем взаимодействовать с ним или отвечать на его сообщения, даже если мы не знаем об этой функциональности. Если вместо этого он сигнализировал об этой функциональности, используя бит 16, то мы знаем, что эта функциональность является обязательной, и мы не можем взаимодействовать с ним, если наш узел также не понимает эту функциональность.

Разработчики Lightning придумали легкую для запоминания фразу, которая кодирует эту матрицу: «it's OK to be odd» (Быть нечетным – это нормально)[94]. Это простое правило обеспечивает богатый набор взаимодействий внутри протокола, так как простая операция с битовой маской между двумя битовыми векторами функциональностей позволяет одноранговым узлам определять, совместимы определенные взаимодействия друг с другом или нет. Другими словами, биты функциональностей используются в качестве механизма обеспечения обнаруживаемости модернизаций: они легко позволяют одноранговым узлам понимать, совместимы они или нет, основываясь на концепциях опциональных, обязательных и неизвестных битов функциональностей.

В рамках протокола биты функциональностей находятся в сообщениях node_announcement, channel_announcement и init. Как следствие эти три сообщения могут использоваться для сигнализации о знании и/или понимании модернизаций протокола на лету внутри сети. Находящиеся в сообщении node_announcement биты функциональностей могут позволять одноранговому узлу определять, совместимы их соединения или нет. Биты функциональностей в сообщениях channel_announcement позволяют одноранговому узлу определять, может данный тип платежа или HTLC-контракт проходить через данный одноранговый узел или нет. Биты функциональностей в сообщении init позволяют одноранговым узлам понимать, могут ли они поддерживать соединение, а также понимать, какие функциональности согласовываются на весь срок службы данного соединения.

TLV для прямой и обратной совместимостей

Как мы узнали ранее в этой главе, TLV-записи могут использоваться для расширения сообщений в прямо и обратно совместимой манерах. Со временем эти записи использовались для расширения существующих сообщений без нарушения протокола путем задействования «неопределенной» области внутри сообщения за пределами этого набора известных байтов.

В качестве примера, в изначальном протоколе Lightning не было концепции «наибольшей суммы HTLC-контракта», которая могла проходить по каналу в соответствии с политикой маршрутизации. Позже для пофазного внедрения этой концепции с течением времени в сообщение channel_update было добавлено поле max_htlc. Одноранговые узлы, которые получают сообщение channel_update, устанавливающее такое поле, но даже не знают о существовании модер-

[94] Это каламбур, в котором обыгрывается слово odd, имеющее второе значение – странный: быть странным – это нормально. – *Прим. перев.*

низации, не затронуты изменением, но их HTLC-контракты отклоняются, если они превышают лимит. С другой стороны, новые одноранговые узлы могут проводить лексический разбор, верифицировать и задействовать новое поле.

Те, кто знаком с концепцией мягких ответвлений (т. н. софт-форков) в Bitcoin, теперь могут увидеть некоторое сходство между этими двумя механизмами. В отличие от мягких ответвлений уровня консенсуса в Bitcoin, модернизация в сети Lightning не требует принятия подавляющего консенсуса. Вместо этого, как минимум, только два одноранговых узла в сети должны понимать новую модернизацию, чтобы начинать ее использовать. Обычно эти два одноранговых узла могут быть получателем и отправителем платежа или партнерами по новому платежному каналу.

Таксономия механизмов модернизации

Вместо того чтобы в сети существовал единый широко используемый механизм модернизации (например, мягкие ответвления для Bitcoin), в сети Lightning существует несколько возможных механизмов модернизации. В этом разделе мы перечислим эти механизмы модернизации и приведем реальный пример их использования в прошлом.

Модернизации внутренней сети

Мы начнем с типа модернизации, который требует наибольшей координации уровня протокола: модернизации внутренней сети. Модернизация внутренней сети характеризуется тем, что требует, чтобы каждый отдельный узел в пределах предполагаемого платежного пути понимал новую функциональность. Такая модернизация аналогична любой модернизации в интернете, которая требует модернизаций аппаратного уровня в рамках ключевой передаточной части модернизации. Однако в контексте сети Lightning мы имеем дело с чистым программным обеспечением, поэтому такие модернизации проще развертывать, но они по-прежнему требуют гораздо большей координации, чем любой другой механизм модернизации в сети.

Одним из примеров такой модернизации в сети было введение TLV-кодировки для информации о маршрутизации, закодированной в луковичных пакетах. Предыдущий формат использовал жестко закодированный формат сообщения фиксированной длины для передачи информации, такой как следующий переход. Поскольку этот формат был фиксированным, это означало, что новые модернизации уровня протокола были невозможны. Переход на более гибкий TLV-формат означал, что после этой модернизации любая функциональность, которая модифицировала тип информации, передаваемой на каждом переходе, могла быть развернута по желанию.

Стоит отметить, что луковичная TLV-модернизация была своего рода «мягкой» модернизацией внутренней сети, поскольку, если платеж не использовал какую-либо новую функциональность помимо этой новой кодировки информации о маршрутизации, тогда платеж мог передаваться с использованием смешанного набора узлов.

Сквозные модернизации

В отличие от модернизации внутренней сети, в этом разделе мы описываем сквозную модернизацию сети. Этот механизм модернизации отличается от

модернизации внутренней сети тем, что для модернизации требуются только «концы» платежа, отправитель и получатель.

Этот тип модернизации обеспечивает широкий спектр неограниченных инноваций в сети. Из-за луковично-шифрованного характера платежей в сети те, кто пересылает HTLC-контракты в центре сети, могут даже не знать, что используются новые функциональности.

Одним из примеров сквозной модернизации сети стало внедрение многокомпонентных платежей (MPP). MPP – это функциональность уровня протокола, которая позволяет делить один платеж на несколько частей или путей, которые будут собраны у получателя для расчета. Внедрение MPP было сопряжено с новым битом функциональности уровня node_announcement, который указывает на то, что получатель знает, как обрабатывать частичные платежи. Предполагая, что отправитель и получатель знают друг о друге (возможно, через счет согласно спецификации BOLT #11), тогда они смогут использовать новую функциональность без каких-либо дальнейших переговоров.

Другим примером сквозной модернизации являются различные типы спонтанных платежей, развернутые в сети. Один из ранних видов спонтанных платежей под названием keysend работал, просто помещая платежный прообраз в шифрованную луковицу. После получения получатель дешифрует прообраз, а затем использует его для улаживания платежа. Поскольку весь пакет зашифрован от конца до конца, этот тип платежа был безопасным, поскольку ни один из промежуточных узлов не может полностью развернуть луковицу, чтобы раскрыть платежный прообраз.

Модернизации уровня строительства канала

Последняя широкая категория модернизаций – это те, которые происходят на уровне строительства канала, но которые не изменяют структуру HTLC-контракта, широко используемого в сети. Когда мы говорим о строительстве канала, то имеем в виду, как канал финансируется или строится. В качестве примера, тип канала eltoo может быть развернут в сети с использованием нового бита функциональности уровня `node_announcement`, а также бита функциональности уровня `channel_announcement`. Только два партнера по обе стороны каналов должны понимать и рекламировать эти новые функциональности. Эта пара каналов затем может использоваться для пересылки любого типа платежа, если канал его поддерживает.

Еще одним является канальный формат *якорных выходов*, который позволяет увеличивать фиксационные комиссионные с помощью механизма управления комиссионными «Дитя платит за родителя» (Child-Pays-For-Parent, CPFP) в Bitcoin.

Вывод

Проводной протокол Lightning невероятно гибок и обеспечивает быстрые инновации и совместимость без строгого консенсуса. Это одна из причин того, что сеть Lightning развивается гораздо быстрее и привлекательна для многих разработчиков, которые в противном случае могли бы счесть стиль разработки Bitcoin слишком консервативным и медленным.

Глава 14

Шифрованный транспорт сообщений Lightning

В этой главе мы проведем обзор шифрованного транспорта сообщений сети Lightning, иногда именуемого бронтидным протоколом, который позволяет одноранговым узлам устанавливать сквозную шифрованную связь, аутентификацию и проверку целостности.

Часть этой главы включает в себя некоторые технические подробности о протоколе шифрования и алгоритмах шифрования, используемых в шифрованном транспорте Lightning. Если эти подробности вас не интересуют, то можете этот раздел пропустить.

Шифрованный транспорт в рамках комплекта протоколов Lightning

Транспортная составляющая сети Lightning и несколько ее компонентов показаны в самой левой части слоя сетевого соединения на рис. 14-1.

Введение

В отличие от ванильной P2P-сети Bitcoin, каждый узел в сети Lightning идентифицируется уникальным публичным ключом, который служит его идентификатором. По умолчанию этот публичный ключ используется для сквозного шифрования всех сообщений в сети. Шифрование на самом низком уровне протокола по умолчанию обеспечивает, чтобы все сообщения были аутентифицированы, защищены от атак «человек посередине» (man-in-the-middle, аббр. MITM)[95] и отслеживания третьими сторонами, а также обеспечивает конфиденциальность на базовом транспортном уровне. В этой главе мы подробно рассмотрим протокол шифрования, используемый в сети Lightning. По завершении данной главы читатель ознакомится с современным уровнем развития протоколов передачи шифрованных сообщений, а также с различными свой-

[95] Человек посередине – это атака безопасности, при которой злоумышленник перехватывает и, возможно, модифицирует данные, передаваемые между двумя пользователями. – *Прим. перев.*

ствами, которые такой протокол предоставляет сети. Стоит отметить, что ядро передачи шифрованных сообщений не зависит от его использования в контексте сети Lightning. Как следствие конкретно-прикладной шифрованный транспорт сообщений, который используется сетью Lightning, может быть перенесен в любой контекст, требующий шифрованной связи между двумя сторонами.

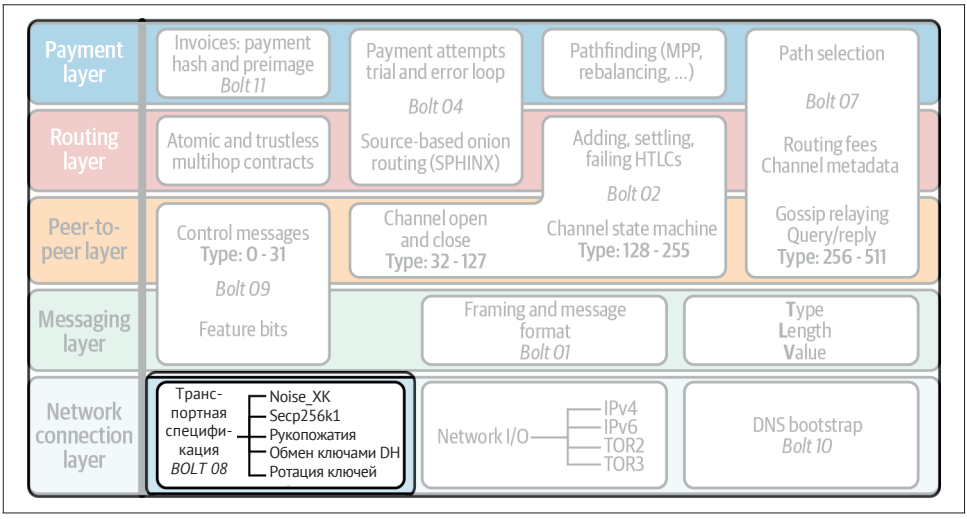


Рис. 14-1. Шифрованный транспорт сообщений в рамках комплекта протоколов Lightning

Канальный граф как децентрализованная инфраструктура публичных ключей

Как мы узнали в главе 8, каждый узел имеет долгосрочный идентификатор, который используется в качестве идентификатора вершины во время отыскания пути, а также в асимметричных криптографических операциях, связанных с созданием луковичных шифрованных пакетов маршрутизации. Этот публичный ключ, служащий долгосрочной идентификацией узла, включается в ответ при самозагрузке DNS, а также встроен в канальный граф. Как следствие, прежде чем узел попытается подсоединиться к другому узлу в P2P-сети, он уже знает публичный ключ узла, к которому хочет подсоединиться.

Вдобавок если узел, к которому происходит подсоединение, уже имеет ряд публичных каналов в графе, то подсоединяющийся узел может дополнительно вирифицировать идентичность узла. Поскольку весь канальный граф полностью аутентифицирован, его можно рассматривать как своего рода децентрализованную инфраструктуру публичных ключей (public key infrastructure, аббр. PKI): для регистрации ключа необходимо открыть публичный канал в блочной цепи Bitcoin, и после того как у узла больше не будет публичных каналов, они фактически удаляются из PKI.

Поскольку Lightning – это децентрализованная сеть, крайне важно, чтобы ни одна центральная сторона не была наделена полномочиями обеспечивать идентичность публичного ключа внутри сети. Вместо центральной стороны

в сети Lightning используется блочная цепь Bitcoin в качестве механизма смягчения Сивилловых атак[96], потому что получение идентичности в сети имеет ощутимую стоимость: комиссионные, необходимые для создания канала в блочной цепи, а также альтернативная стоимость капитала, выделяемого на их каналы. В процессе по существу развертывания PKI для конкретного домена сеть Lightning способна значительно упрощать свой протокол шифрованного транспорта, поскольку ей не нужно иметь дело со всеми сложностями, связанными с протоколом безопасности транспортного слоя (Transport Layer Security, аббр. TLS).

Почему не TLS?

Читатели, знакомые с TLS-системой, возможно, в этом месте зададутся вопросом: почему не использовался TLS, несмотря на недостатки существующей PKI-системы? Это действительно факт, что «самозаверяющие сертификаты» могут использоваться для эффективного обхода существующей глобальной PKI-системы путем простого подтверждения подлинности данного публичного ключа среди множества одноранговых узлов. Однако, даже если убрать в сторону существующую PKI-систему, TLS имеет несколько недостатков, которые побудили создателей сети Lightning вместо нее выбрать более компактный конкретно-прикладной протокол шифрования.

Начнем с того, что TLS – это протокол, который существует уже несколько десятилетий и в результате со временем эволюционировал по мере достижения новых продвижений в области транспортного шифрования. Однако со временем эта эволюция привела к увеличению размера и сложности протокола. За последние несколько десятилетий было обнаружено и исправлено несколько уязвимостей в TLS, причем каждая эволюция еще больше усложняла протокол. Вследствие возраста протокола существует несколько версий и итераций, а это означает, что клиенту необходимо понимать многие из предыдущих итераций протокола для связи с большой частью публичного интернета, что еще больше усложняет имплементацию.

В прошлом в широко используемых имплементациях SSL/TLS было обнаружено несколько уязвимостей безопасности памяти. Упаковка такого протокола в каждый узел Lightning послужила бы увеличению поверхности атаки узлов, выставленных в публичную одноранговую сеть. В целях повышения безопасности сети в целом и минимизирования эксплуатируемой поверхности атаки создатели сети Lightning вместо этого решили принять каркас криптосвязи на основе протокола Noise (Noise Protocol Framework). Noise как протокол усваивает некоторые уроки безопасности и конфиденциальности, извлеченные с течением времени благодаря постоянному тщательному изучению протокола TLS на протяжении десятилетий. В некотором смысле существование Noise позволяет сообществу эффективно «начать все сначала» с более компактного, упрощенного протокола, который сохраняет все добавленные выгоды TLS.

---

[96] Сивиллова атака (Sybil attack) определяется как небольшое число сущностей, подделывающих несколько элементов идентификационных данных одноранговых узлов, чтобы скомпрометировать непропорционально большую долю системы. – *Прим. перев.*

Каркас криптосвязи на основе протокола Noise

Каркас криптосвязи на основе протокола Noise (Noise Protocol Framework) – это современный, расширяемый и гибкий протокол шифрования сообщений, разработанный создателями протокола Signal. Протокол Signal является одним из наиболее широко используемых протоколов шифрования сообщений в мире. Он используется как в Signal, так и Whatsapp, которыми в совокупности пользуются более миллиарда человек по всему миру. Структура Noise является результатом десятилетий эволюции как в академических кругах, так и в индустрии протоколов шифрования сообщений. В Lightning каркас криптосвязи на основе протокола Noise используется для имплементации протокола шифрования, ориентированного на сообщения, используемого всеми узлами для связи друг с другом.

Сеанс связи с применением Noise состоит из двух отдельных фаз: фазы рукопожатия и фазы обмена сообщениями. Прежде чем две стороны смогут друг с другом общаться, им сначала необходимо прийти к совместному секрету, известному только им, который будет использоваться для шифрования и аутентифицирования сообщений, отправляемых друг другу. Разновидность соглашения об аутентифицированном ключе используется для прихода к окончательному совместному ключу между двумя сторонами. В контексте протокола Noise это соглашение об аутентифицированном ключе называется *рукопожатием*. Как только это рукопожатие будет завершено, оба узла теперь смогут отправлять друг другу шифрованные сообщения. Всякий раз, когда одноранговым узлам необходимо подсоединиться или переподсоединиться друг к другу, исполняется новая итерация протокола рукопожатия, обеспечивающая достижение прямой секретности (утечка ключа предыдущего транскрипта не ставит под угрозу какие-либо будущие транскрипты).

Поскольку протокол Noise позволяет разработчику протокола выбирать из нескольких криптографических примитивов, таких как симметричное шифрование и криптография с публичным ключом, обычно каждый вариант протокола Noise обозначается уникальным именем. В духе «Noise» для каждого варианта протокола подбирается имя, производное от какого-то «шума». В контексте сети Lightning разновидность используемого протокола Noise иногда называют бронтидой. Бронтида – это низкий вздымающийся шум, подобный тому, который можно услышать во время грозы, находясь очень далеко.

Шифрованный транспорт Lightning в деталях

В этом разделе мы разберем протокол шифрованного транспорта Lightning и углубимся в детали криптографических алгоритмов и протокола, используемых для установления шифрованной, аутентифицированной и гарантированной по целостности связи между одноранговыми узлами. Можно смело пропустить данный раздел, если вы находите такой уровень детализации пугающим.

Noise_XK: рукопожатие Noise в сети Lightning

Протокол Noise чрезвычайно гибок в том смысле, что он рекламирует несколько рукопожатий, каждое из которых обладает разными свойствами безопасности и конфиденциальности для потенциального разработчика протокола на выбор. Глубокий разведывательный анализ каждого из рукопожатий и их

различных компромиссов выходит за рамки этой главы. С учетом сказанного, сеть Lightning использует специфическое рукопожатие, именуемое `Noise_XK`. Уникальным свойством, предоставляемым этим рукопожатием, является сокрытие идентичности: для того чтобы узел инициировал соединение с другим узлом, он должен сначала знать свой публичный ключ. Механически это означает, что публичный ключ ответчика фактически никогда не передается в контексте рукопожатия. Вместо этого для аутентифицирования ответчика используется продуманная серия проверок Диффи–Хеллмана по эллиптической кривой (Elliptic Curve Diffie–Hellman, аббр. ECDH) и кода аутентификации сообщения (message authentication code, аббр. MAC).

Нотация рукопожатия и поток протокола

Каждое рукопожатие обычно состоит из нескольких шагов. На каждом шаге какой-либо (возможно) шифрованный материал отправляется противоположной стороне, выполняется ECDH (или несколько), в результате чего рукопожатие «смешивается» с транскриптом протокола. Этот транскрипт служит для аутентифицирования каждого шага протокола и помогает предотвращать ряд атак типа «человек посередине». В конце рукопожатия создаются два ключа, `ck` и `k`, которые используются для шифрования сообщений (`k`) и ротации ключей (`ck`) на протяжении всего срока сеанса.

В контексте рукопожатия `s` обычно является долгосрочным статическим публичным ключом. В нашем случае используемая криптосистема с публичным ключом представляет собой систему на основе эллиптической кривой, созданную с помощью кривой secp256k1, которая используется в других местах Bitcoin. Во время рукопожатия генерируется несколько эфемерных ключей. Мы используем `e` для обозначения нового эфемерного ключа. Операции ECDH между двумя ключами обозначаются как конкатенация двух ключей. В качестве примера `ee` представляет операцию ECDH между двумя эфемерными ключами.

Высокоуровневый обзор

Используя изложенную ранее нотацию, `Noise_XK` можно кратко описать следующим образом:

```
Noise_XK(s, rs):
  <- rs
  ...
  -> e, e(rs)
  <- e, ee
  -> s, se
```

Протокол начинается с «предварительной передачи» статического ключа (`rs`) ответчика инициатору. Перед исполнением рукопожатия инициатор должен сгенерировать свой собственный статический ключ (`s`). На каждом этапе рукопожатия все материалы, отправленные по проводу, и отправленные/использованные ключи постепенно хешируются в дайджест рукопожатия, `h`. Этот дайджест никогда не передается по проводу во время рукопожатия, а вместо этого используется в качестве «ассоциированных данных», когда AEAD (аутентифицированная шифровка с ассоциированными данными) отправляется по проводу. *Ассоциированные данные* (AD) позволяют протоколу шифрования

аутентифицировать дополнительную информацию вместе с шифрованным текстовым пакетом. В других доменах AD может быть доменно-именной, или просто текстовой, частью пакета.

Существование `h` обеспечивает, что если часть переданного сообщения о рукопожатии будет заменена, то другая сторона это заметит. На каждом шаге проверяется дайджест MAC. Если проверка MAC завершается успешно, то принимающая сторона знает, что до этого момента рукопожатие было успешным. В противном случае, если проверка MAC когда-либо завершится безуспешно, то процесс рукопожатия завершится сбоем, и соединение должно быть прервано.

Протокол также добавляет новую часть данных в каждое сообщение о рукопожатии: версию протокола. Начальная версия протокола равна 0. На момент написания этой главы никаких новых версий протокола создано не было. Как следствие, если одноранговый узел получает версию, отличную от 0, он должен отклонить попытку инициирования рукопожатия.

Что касается криптографических примитивов, то SHA-256 используется в качестве предпочтительной хеш-функции, `secp256k1` в качестве эллиптической кривой и `Chachapoly-130` в качестве конструкции AEAD (симметричного шифрования).

Каждый вариант протокола Noise имеет уникальный ASCII-литерал, используемый для ссылки на него. В целях обеспечения применения двумя сторонами одного и того же варианта протокола ASCII-литерал хешируется в дайджест, который используется для инициализации начального состояния рукопожатия. В контексте сети Lightning описывающий протокол ASCII-литерал имеет вид `Noise_XK_secp256k1_ChaChaPoly_SHA256`.

Рукопожатие в трех действиях

Часть рукопожатия может быть разделена на три отдельных «действия». Все рукопожатие занимает 1.5 пробега в оба конца между инициатором и ответчиком. При каждом действии между обеими сторонами отправляется одно и то же сообщение. Сообщение о рукопожатии представляет собой полезный груз фиксированного размера с префиксом версии протокола.

В протоколе Noise используется объектно-ориентированная нотация для описания протокола на каждом шаге. Во время настройки состояния рукопожатия каждая сторона инициализирует следующие переменные:

`ck`

: *Цепной ключ*. Это значение представляет собой накопленный хеш всех предыдущих выходов ECDH. В конце рукопожатия `ck` используется для вывода шифроключей для сообщений Lightning.

`h`

: *Хеш рукопожатия*. Это значение представляет собой накопленный хеш всех данных рукопожатия, которые были отправлены и получены до сих пор в процессе рукопожатия.

`temp_k1`, `temp_k2`, `temp_k3`

: *Промежуточные ключи*. Они используются для шифрования и дешифрования полезного груза AEAD нулевой длины в конце каждого сообщения о рукопожатии.

`e`

Пара эфемерных ключей участника. Для каждого сеанса узел должен генерировать новый эфемерный ключ с сильной криптографической случайностью/рандомностью.

`s`

Пара статических ключей участника (`ls` для локальной, `rs` для дистационной).

Учитывая это состояние сеанса рукопожатия плюс обмена сообщениями, мы затем определим ряд функций, которые будут оперировать на состоянии рукопожатия и обмена сообщениями. При описании протокола рукопожатия мы будем использовать эти переменные в режиме псевдокода, чтобы уменьшить детализацию объяснения каждого шага в протоколе. Мы определим функциональные примитивы рукопожатия следующим образом:

`ECDH(k, rk)`

Выполняет операцию Диффи–Хеллмана на эллиптической кривой, используя `k`, валидный приватный ключ `secp256k1`, и `rk`, валидный публичный ключ.

Возвращаемым значением является SHA-256 сжатого формата сгенерированной точки.

`HKDF(salt,ikm)`

Функция, определенная в RFC 5869, вычисляемая с помощью поля `info` нулевой длины.

Все вызовы HKDF неявно возвращают 64 байта криптографической случайности с использованием компонента «извлечь и расширить» функции `HKDF`.

`encryptWithAD(k, n, ad, простой_текст)`

Выводит `encrypt(k, n, ad, простой_текст)`,

где `encrypt` – это вычисление `ChaCha20-Poly1305` (вариант Инженерной рабочей группы интернета – Internet Engineering Task Force, аббр. IETF) с переданными аргументами, с одноразовым произвольным `n`, закодированным как 32 нулевых бита, за которыми следует 64-битовое значение с упорядочением от меньшего к большему. Примечание: оно соответствует соглашению в протоколе Noise, а не нашему обычному упорядочению.

`decryptWithAD(k, n, ad, шифротекст)`

Выводит `decrypt(k, n, ad, шифротекст)`,

где `decrypt` – это вычисление `ChaCha20-Poly1305` (вариант IETF) с переданными аргументами, с одноразовым произвольным `n`, закодированным как 32 нулевых бита, за которыми следует 64-битовое значение с упорядочением от меньшего к большему.

`generateKey()`

Генерирует и возвращает новую пару ключей `secp256k1`,

где объект, возвращаемый `generateKey`, имеет два атрибута: `.pub`, который возвращает абстрактный объект, представляющий публичный ключ;

и `.priv`, который представляет приватный ключ, используемый для генерирования публичного ключа,

где объект также имеет один метод: `.serializeCompressed()`.

`a || b`

Это обозначает конкатенацию двух байтовых строк `a` и `b`.

Инициализация состояния сеанса рукопожатия

Перед началом процесса рукопожатия обе стороны должны инициализировать начальное состояние, которое они будут использовать для продвижения процесса рукопожатия. Прежде всего обеим сторонам необходимо создать начальный дайджест рукопожатия `h`.

1. `h = SHA-256(имя_протокола)`
   где `имя_протокола = "Noise_XK_secp256k1_ChaChaPoly_SHA256"`, закодированное в виде ASCII-литерала.
2. `ck = h`
3. `h = SHA-256(h || пролог)`

   где `пролог` – это ASCII-литерал: `lightning`.

В дополнение к названию протокола мы также вносим добавочный «пролог», который используется для дальнейшей привязки контекста протокола к сети Lightning.

В завершение шага инициализации обе стороны смешивают публичный ключ ответчика с дайджестом рукопожатия. Поскольку этот дайджест используется при отправке ассоциированных данных с шифротекстом нулевой длины (только MAC), это обеспечивает, чтобы инициатор действительно знал публичный ключ ответчика.

- Инициирующий узел смешивает статический публичный ключ отвечающего узла, сериализованный в сжатый формат Bitcoin: `h = SHA-256(h || rs.pub.serializeCompressed())`.
- Отвечающий узел смешивает свой локальный статический публичный ключ, сериализованный в сжатый формат Bitcoin: `h = SHA-256(h || ls.pub.serializeCompressed())`.

Акты рукопожатия

После первоначальной инициализации рукопожатия можно приступить к фактическому исполнению процесса рукопожатия. Рукопожатие состоит из серии из трех сообщений, отправляемых между инициатором и ответчиком, отныне именуемых «актами». Поскольку каждый акт представляет собой одно сообщение, отправляемое между сторонами, рукопожатие совершается в общей сложности 1.5 раза в оба конца (0.5 для каждого акта).

Первый акт совершает начальную часть инкрементного тройного обмена ключами Диффи–Хеллмана (DH) (с использованием нового эфемерного ключа, сгенерированного инициатором), а также обеспечивает, чтобы инициатор действительно знал долгосрочный публичный ключ ответчика. Во время второго акта ответчик передает эфемерный ключ, который он хочет использовать для сеанса, инициатору и еще раз инкрементно смешивает этот новый ключ с тройным рукопожатием DH. Во время третьего (и заключительного) акта

инициатор передает свой долгосрочный статический публичный ключ ответчику и исполняет заключительную операцию DH, чтобы смешать его с окончательным результирующим совместным секретом.

Акт первый

```
-> e, es
```

Первый акт отправляется от инициатора ответчику. Во время первого акта инициатор пытается удовлетворить скрытый вызов со стороны ответчика. В целях выполнения этой задачи инициатор должен знать статический публичный ключ ответчика.

Сообщение о рукопожатии составляет ровно 50 байт: 1 байт для версии рукопожатия, 33 байта для сжатого эфемерного публичного ключа инициатора и 16 байт для тега `poly1305`.

Действия отправителя:

1. `e = GenerateKey()`
2. `h = SHA-256(h || e.pub.serializeCompressed())`
   Вновь сгенерированный эфемерный ключ накапливается в текущем дайджесте рукопожатия.
3. `es = ECDH(e.priv, rs)`
   Инициатор выполняет ECDH между своим недавно сгенерированным эфемерным ключом и статическим публичным ключом дистанционного узла.
4. `ck, temp_k1 = HKDF(ck, es)`
   Генерируется новый временный ключ шифрования, который используется для генерирования аутентифицирующего MAC.
5. `c = encryptWithAD(temp_k1, 0, h, zero)`
   где `zero` – это обычный текст нулевой длины.
6. `h = SHA-256(h || c)`
   Наконец, сгенерированный зашифрованный текст накапливается в дайджесте рукопожатия для проверки подлинности.
7. Отправить `m = 0 || e.pub.serializeCompressed() || c` ответчику через сетевой буфер.

Действия получателя:

1. Прочитать ровно 50 байт из сетевого буфера.
2. Провести лексический разбор прочитанного сообщения (`m`) в `v`, `re` и `c`:
   - где `v` – это первый байт сообщения `m`, `re` – следующие 33 байта `m`, а `c` – последние 16 байт `m`;
   - сырые байты эфемерного публичного ключа (`re`) дистанционной стороны должны быть десериализованы в точку на кривой с использованием аффинных координат, закодированных в сериализованном составном формате ключа.
3. Если `v` является нераспознанной версией рукопожатия, то ответчик должен прервать попытку подсоединения.
4. `h = SHA-256(h || re.serializeCompressed())`

Ответчик накапливает эфемерный ключ инициатора в дайджесте аутентификации рукопожатия.

5. `es = ECDH(s.priv, re)`
   Ответчик выполняет ECDH между своим статическим приватным ключом и эфемерным публичным ключом инициатора.
6. `ck, temp_k1 = HKDF(ck, es)`
   Генерируется новый временный ключ шифрования, который вскоре будет использован для проверки подлинности MAC.
7. `p = decryptWithAD(temp_k1, 0, h, c)`
   Если проверка MAC в этой операции завершается безуспешно, то инициатор не знает статический публичный ключ ответчика. Если это так, то ответчик должен прервать соединение без каких-либо дальнейших сообщений.
8. `h = SHA-256(h || c)`
   Полученный шифротекст смешивается с дайджестом рукопожатия. Этот шаг служит для обеспечения того, чтобы полезный груз не был модифицирован атакой «человек посередине» (MITM).

Акт второй

```
<- e, ee
```

Второй акт отправляется от ответчика инициатору. Второй акт состоится только в том случае, если первый акт прошел успешно. Первый акт был успешным, если ответчик смог правильно дешифровать и проверить MAC тега, отправленного в конце первого акта.

Рукопожатие составляет ровно 50 байт: 1 байт для версии рукопожатия, 33 байта для сжатого эфемерного публичного ключа ответчика и 16 байт для тега `poly1305`.

Действия отправителя:

1. `e = GenerateKey()`
2. `h = SHA-256(h || e.pub.serializeCompressed())`
   Вновь сгенерированный эфемерный ключ накапливается в текущем дайджесте рукопожатия.
3. `ee = ECDH(e.priv, re)`
   где `re` – это эфемерный ключ инициатора, который был получен во время первого акта.
4. `ck, temp_k2 = HKDF(ck, ee)`
   Генерируется новый временный ключ шифрования, который используется для генерирования аутентифицирующего MAC.
5. `c = encryptWithAD(temp_k2, 0, h, zero)`
   где `zero` – это обычный текст нулевой длины.
6. `h = SHA-256(h || c)`
   Наконец, сгенерированный зашифрованный текст накапливается в дайджесте рукопожатия для проверки подлинности.
7. Отправить `m = 0 || e.pub.serializeCompressed() || c` инициатору через сетевой буфер.

Действия получателя:

1. Прочитать ровно 50 байт из сетевого буфера.
2. Выполнить лексический разбор прочитанного сообщения (`m`) в `v`, `re` и `c`: где `v` – это первый байт сообщения `m`, `re` – следующие 33 байта `m`, а `c` – последние 16 байт `m`.
3. Если `v` является нераспознанной версией рукопожатия, то ответчик должен прервать попытку подсоединения.
4. `h = SHA-256(h || re.serializeCompressed())`
5. `ee = ECDH(e.priv, re)`
   где `re` – это эфемерный публичный ключ ответчика.
   Сырые байты эфемерного публичного ключа (`re`) дистанционной стороны должны быть десериализованы в точку на кривой с использованием аффинных координат, закодированных в сериализованном составном формате ключа.
6. `ck, temp_k2 = HKDF(ck, ee)`
   Генерируется новый временный ключ шифрования, который используется для создания аутентифицирующего MAC.
7. `p = decryptWithAD(temp_k2, 0, h, c)`
   Если проверка MAC в этой операции завершается безуспешно, то инициатор должен прервать соединение без каких-либо дальнейших сообщений.
8. `h = SHA-256(h || c)`
   Полученный шифротекст смешивается с дайджестом рукопожатия. Этот шаг служит для обеспечения того, чтобы полезный груз не был модифицирован атакой «человек посередине» (MITM).

Акт третий

```
-> s, se
```

Третий акт – это заключительная фаза соглашения об аутентифицированном ключе, описанного в этом разделе. Этот акт отправляется от инициатора ответчику в качестве заключительного шага. Третий акт выполняется тогда и только тогда, когда второй акт прошел успешно. Во время третьего акта инициатор передает свой статический публичный ключ ответчику, зашифрованный с высокой степенью секретности, используя накопленный секретный ключ, полученный из HKDF, в этой точке рукопожатия.

Рукопожатие составляет ровно 66 байт: 1 байт для версии рукопожатия, 33 байта для статического публичного ключа, зашифрованного потоковым шифром `ChaCha20`, 16 байт для тега зашифрованного публичного ключа, сгенерированного с помощью конструкции AEAD, и 16 байт для окончательного тега аутентификации.

Действия отправителя:

1. `c = encryptwithad(temp_k2, 1, h, s.pub.serializeCompressed())`
   где `s` – это статический публичный ключ инициатора.
2. `h = SHA-256(h || c)`
3. `se = ECDH(s.priv, re)`
   где `re` – это эфемерный публичный ключ ответчика.

4. `ck, temp_k3 = HKDF(ck, se)`
   Окончательный промежуточный совместный секрет смешивается с текущим цепным ключом.
5. `t = encryptWithAD(temp_k3, 0, h, zero)`
   где `zero` – это обычный текст нулевой длины.
6. `sk, rk = HKDF(ck, zero)`
   где `zero` – это обычный текст нулевой длины, `sk` – ключ, который будет использоваться инициатором для шифровки сообщений ответчику, а `rk` – ключ, который будет использоваться инициатором для дешифровки сообщений, отправленных ответчиком.
   Генерируются окончательные ключи шифрования, которые будут использоваться для отправки и получения сообщений в течение всего сеанса.
7. `rn = 0, sn = 0`
   Отправляющие и принимающие одноразовые произвольные числа (nonce) инициализируются значением 0.
8. Отправить `m = 0 || c || t` через сетевой буфер.

Действия получателя:

1. Прочитать ровно 66 байт из сетевого буфера.
2. Выполнить лексический разбор прочитанного сообщения (`m`) в `v`, `c` и `t`:
   где `v` – первый байт сообщения `m`, `c` – следующие 49 байт `m`, а `t` – последние 16 байт `m`.
3. Если `v` является нераспознанной версией рукопожатия, то ответчик должен прервать попытку подсоединения.
4. `rs = decryptWithAD(temp_k2, 1, h, c)`
   В этой точке ответчик восстановил статический публичный ключ инициатора.
5. `h = SHA-256(h || c)`
6. `se = ECDH(e.priv, rs)`
   где `e` – исходный эфемерный ключ ответчика.
7. `ck, temp_k3 = HKDF(ck, se)`
8. `p = decryptWithAD(temp_k3, 0, h, t)`
   Если проверка MAC в этой операции завершается безуспешно, то ответчик должен прервать соединение без каких-либо дальнейших сообщений.
9. `rk, sk = HKDF(ck, zero)`
   где `zero` – это обычный текст нулевой длины, `rk` – ключ, который будет использоваться ответчиком для дешифровки сообщений, отправленных инициатором, а `sk` – ключ, который будет использоваться ответчиком для шифровки сообщений инициатору.
   Генерируются окончательные ключи шифрования, которые будут использоваться для отправки и получения сообщений в течение всего сеанса.
10. `rn = 0, sn = 0`
    Отправляющие и принимающие одноразовые произвольные числа инициализируются значением 0.

Шифрование транспортных сообщений

По завершении третьего акта обе стороны получили ключи шифрования, которые будут использоваться для шифровки и дешифровки сообщений в течение оставшейся части сеанса.

Фактические сообщения протокола Lightning заключены в шифротексты AEAD. Каждому сообщению предшествует другой шифротекст AEAD, который кодирует суммарную длину следующего сообщения Lightning (не включая его MAC).

Максимальный размер любого сообщения Lightning не должен превышать 65 535 байт. Максимальный размер, равный 65 535, упрощает тестирование, делает проще управление памятью и помогает смягчать атаки на истощение памяти.

В целях усложнения анализа трафика префикс длины для всех шифрованных сообщений Lightning также зашифрован. Дополнительно к зашифрованному префиксу длины добавляется 16-байтовый тег Poly-1305 в целях обеспечения того, чтобы длина пакета не была модифицирована на лету, а также чтобы избежать создания дешифровочного оракула.

Структура пакетов на проводе напоминает схему на рис. 14-2.

Рис. 14-2. Структура шифрованного пакета

Длина сообщения с префиксом кодируется как 2-байтовое целое число с упорядоченностью от большего к меньшему, при этом суммарная максимальная длина пакета составляет 2 + 16 + 65,535 + 16 = 65 569 байт.

Шифрование и отправка сообщений. В целях шифровки и отправки сообщения Lightning (`m`) в сетевой поток при наличии отправляющего ключа (`sk`) и одноразового произвольного числа (`sn`) выполняются следующие действия:

1. Пусть `l = len(m)`,
   где `len` получает длину сообщения Lightning в байтах.
2. Сериализовать `l` в 2 байта, закодированных как целое число с упорядочением от большего к меньшему.
3. Зашифровать `l` (используя `ChaChaPoly-1305`, `sn` и `sk`), чтобы получить `lc` (18 байт).
   - Одноразовое произвольное число `sn` кодируется как 96-битовое число с упорядочением от меньшего к большему. Поскольку декодированное произвольное число равно 64 битам, 96-битовое одноразовое произвольное число кодируется как 32 бита начальных нулей, за которыми следует 64-битовое значение.

- Одноразовое произвольное число `sn` после этого шага должно быть увеличено на единицу.
- Байтовый срез нулевой длины должен быть передан в качестве AD (ассоциированных данных).

4. Наконец, зашифровать само сообщение (`m`), используя ту же процедуру, которая использовалась для шифрования префикса длины. Обозначим этот шифротекст через `c`.
   Одноразовое произвольное число `sn` после этого шага должно быть увеличено на единицу.
5. Отправить `lc || c` через сетевой буфер.

Получение и дешифровка сообщений. В целях дешифровки следующего сообщения в сетевом потоке выполнить такие действия:

1. Прочитать ровно 18 байт из сетевого буфера.
2. Обозначить зашифрованный префикс длины через `lc`.
3. Дешифровать `lc` (используя `ChaCha20-Poly1305`, `rn` и `rk`), чтобы получить размер шифрованного пакета `l`.
   - Байтовый срез нулевой длины должен быть передан в качестве AD (ассоциированных данных).
   - Одноразовое произвольное число `rn` после этого шага должно быть увеличено на единицу.
4. Прочитать ровно `l` + 16 байт из сетевого буфера и обозначить эти байты через `c`.
5. Дешифровать `c` (используя `ChaCha20-Poly1305`, `rn` и `rk`), чтобы получить дешифрованный пакет `p` с обычным текстом.

Одноразовое произвольное число `rn` после этого шага должно быть увеличено на единицу.

Ротация ключей сообщений Lightning

Регулярная смена ключей и забывание предыдущих ключей полезны для предотвращения дешифровки старых сообщений в случае последующей утечки ключей (т. е. обратной секретности).

Ротация ключа выполняется для каждого ключа (sk и rk) индивидуально. Ключ должен быть ротирован после того, как участник зашифрует или дешифрует его 1000 раз (т. е. каждые 500 сообщений). Это можно правильно учитывать, ротируя ключ, как только выделенное ей произвольное число превысит 1000.

Ротация ключа k выполняется в соответствии со следующими ниже шагами:

1. Пусть `ck` будет цепным ключом, полученным в конце третьего акта.
2. `ck', k' = HKDF(ck, k)`
3. Обнулить одноразовое произвольное число ключа, `n = 0`.
4. `k = k'`
5. `ck = ck'`

Вывод

Базовое транспортное шифрование Lightning основано на протоколе Noise и обеспечивает надежные гарантии конфиденциальности, подлинности и целостности для всех коммуникаций между одноранговыми узлами Lightning.

В отличие от системы Bitcoin, где одноранговые узлы часто общаются «в открытую» (без шифрования), все коммуникации Lightning шифруются от одноранговых узла к одноранговому узлу. В дополнение к транспортному шифрованию (между одноранговыми узлами) в сети Lightning платежи также шифруются в луковичные пакеты (между переходами), а платежные реквизиты отправляются вне полосы между отправителем и получателем (в сквозном порядке). Комбинация всех этих механизмов безопасности является кумулятивной и обеспечивает многослойную защиту от деанонимизации, атак «человек посередине» и сетевого наблюдения.

Разумеется, никакая безопасность не является совершенной, и в главе 16 мы увидим, что эти свойства могут быть ухудшены и атакованы. Тем не менее сеть Lightning значительно совершенствует конфиденциальность Bitcoin.

Глава 15

Платежные запросы Lightning

В этой главе мы рассмотрим платежные запросы Lightning, или, как их чаще называют, счета Lightning.

Счета в комплекте протоколов Lightning

Платежные запросы, т. н. счета, являются частью платежного слоя и показаны в левом верхнем углу рис. 15-1.

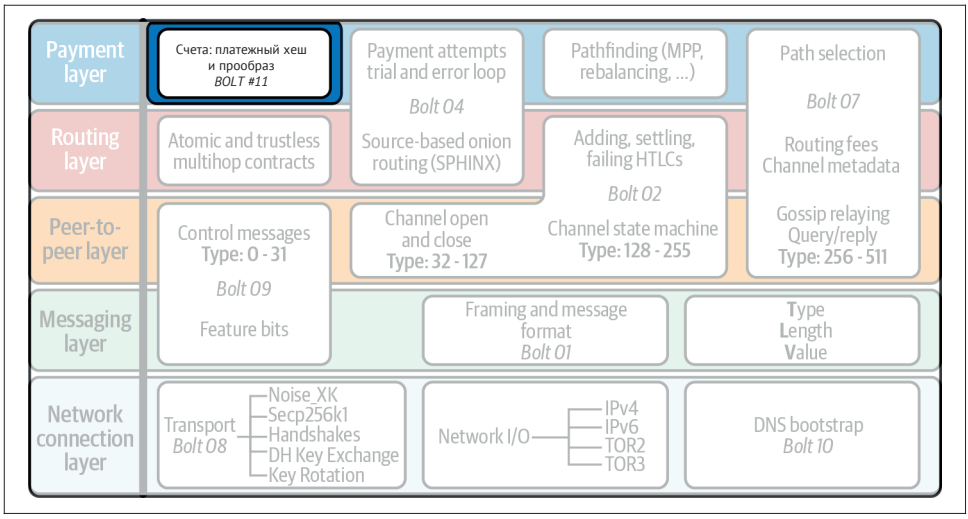


Рис. 15-1. Платежные запросы в комплекте протоколов Lightning

Введение

Как мы узнали из книги, для совершения платежа Lightning требуются минимум две части данных: платежный хеш и пункт назначения. Поскольку SHA-256 используется в сети Lightning для имплементации HTLC-контрактов, для передачи этой информации требуется 32 байта. Пункты назначения, с другой стороны, – это просто публичный ключ `secp256k1` узла, который желает получить платеж. Цель платежного запроса в контексте сети Lightning состоит

в том, чтобы передать эти две части информации от отправителя получателю. Удобный формат QR-кода для передачи информации, необходимой для совершения платежа от получателя отправителю, описан в спецификации «BOLT #11: протокол выставления счетов для платежей Lightning»[97]. На практике в платежном запросе сообщается нечто большее, чем просто платежный хеш и пункт назначения, чтобы сделать кодировку более полнофункциональной.

Платежные запросы Lightning против Bitcoin-адресов

Часто задаваемый вопрос, когда люди впервые сталкиваются с платежным запросом Lightning, звучит так: почему вместо этого нельзя использовать обычный формат статического адреса?

Для ответа на этот вопрос вы должны сначала понять, чем Lightning отличается от базового слоя Bitcoin в качестве метода платежа. По сравнению с Bitcoin-адресом, который может использоваться для совершения потенциально неограниченного числа платежей (хотя реиспользование Bitcoin-адреса может нарушить конфиденциальность пользователя), платежный запрос Lightning следует использовать только один раз. Это связано с тем фактом, что при отправке платежа на Bitcoin-адрес, по сути, используется криптосистема с публичным ключом, «кодирующая» платеж таким образом, чтобы только истинный «владелец» этого Bitcoin-адреса мог его погасить.

В отличие от этого, в целях совершения платежа Lightning получатель должен раскрыть «секрет» для всего маршрута платежа, включая отправителя. Это может быть истолковано как использование своего рода специфичной для домена симметричной криптографии, поскольку платежный прообраз для практических целей является одноразовым произвольным числом (число используется только один раз). Если отправитель попытается произвести еще один платеж, используя этот идентичный платежный хеш, то он рискует потерять средства, поскольку платеж может фактически не быть доставлен в пункт назначения. Можно с уверенностью предположить, что после обнаружения прообраза все узлы в пути сохранят его навсегда, а затем, вместо того чтобы переслать HTLC-контракт для взимания комиссионных за маршрутизацию, если платеж будет совершен, они могут просто произвести платеж в этом экземпляре и заполучить всю сумму платежа взамен. В результате использовать платежный запрос когда-либо более одного раза небезопасно.

Существуют новые варианты изначального платежного запроса Lightning, которые позволяют отправителю реиспользовать их столько раз, сколько он захочет. Эти варианты переворачивают обычный платежный поток, поскольку отправитель передает прообраз внутри зашифрованного луковичного полезного груза получателю, который является единственным, кто способен его дешифровать и произвести платеж. В качестве альтернативы, если допустить механизм, который позволяет отправителю обычным образом запрашивать новый платежный запрос от получателя, можно использовать интерактивный протокол, позволяющий в определенной степени реиспользовать платежный запрос.

[97] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md.

BOLT #11: сериализация и интерпретация платежных запросов Lightning

В этом разделе мы опишем механизм, используемый для кодирования набора информации, необходимой для совершения платежа в сети Lightning. Как упоминалось ранее, платежный хеш и пункт назначения – это минимальный объем информации, необходимый для совершения платежа. Однако на практике также передается дополнительная информация, такая как информация о привязке ко времени, срок действия платежного запроса и, возможно, резервный адрес в сети. Все это в полной мере задокументировано в спецификации «BOLT #11: протокол выставления счетов для платежей Lightning»[98].

Кодирование платежного запроса на практике

Сперва давайте посмотрим на то, как выглядит реальный платежный запрос на практике. Ниже приведен действительный платежный запрос, который мог быть использован для совершения платежа в mainnet-сети Lightning в момент его создания:

```
lnbc2500u1pvjluezpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypqdq
5xysxxatsyp3k7enxv4jsxqzpuaztrnwngzn3kdzw5hydlzf03qdgm2hdq27cqv3agm2awhz5
se903vruatfhq77w3ls4evs3ch9zw97j25emudupq63nyw24cg27h2rspfj9srp
```

Человекочитаемый префикс

Глядя на эту цепочку символов, можно выделить часть, которую можно разобрать глазами, в то время как остальная часть выглядит просто как случайный набор символов. Часть, которая в некоторой степени поддается анализу человеком, называется *человекочитаемым префиксом*. Он позволяет человеку быстро извлекать некоторую релевантную информацию из платежного запроса с первого взгляда. В данном случае мы видим, что этот платеж предназначен для экземпляра mainnet-сети Lightning (`lnbc`) и запрашивает 2500 uBTC (микробиткойн), или 25 000 000 сатоши. Последняя часть называется порцией данных и использует расширяемый формат для кодирования информации, необходимой для совершения платежа.

Каждая версия экземпляра сети Lightning (mainnet, testnet и т. д.) имеет свой собственный человекочитаемый префикс (см. табл. 15-1). В силу этого клиентское программное обеспечение, а также люди имеют возможность быстро определять, может ли платежный запрос быть удовлетворен их узлом или нет.

Таблица 15-1. BOLT №11. Сетевые префиксы

Сеть	Префикс по BOLT №11
mainnet	lnbc
testnet	lntb
simnet/regtest	lnbcrt

[98] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md.

Первая часть человекочитаемого префикса представляет собой компактное выражение суммы платежного запроса. Компактная сумма закодирована в двух частях. Сперва в качестве *базовой* суммы используется целое число. Затем следует множитель, который позволяет указывать несовпадающие увеличения на порядок величины, уравновешиваемые базовой суммой. Если вернуться к нашему первоначальному примеру, то можно взять порцию `2500u` и уменьшить ее в 1000 раз, чтобы вместо этого использовать `2500m` или (2500 mBTC). В качестве общего правила, чтобы с первого взгляда определить сумму счета, надо взять базовый коэффициент и умножить его на множитель.

Полный список множителей, определенных в настоящее время, приведен в табл. 15-2.

Таблица 15-2. Множители суммы по BOLT №11

Множитель	Биткойн-единица	Коэффициент умножения
m	милли	0.001
u	микро	0.000001
n	нано	0.000000001
p	пико	0.000000000001

bech32 и сегмент данных

Если «нечитаемая» порция выглядит знакомой, то это потому, что в ней используется та же схема кодирования, что и в современных Bitcoin-адресах, совместимых с SegWit, а именно bech32. Описание схемы кодирования bech32 выходит за рамки данной главы. Если вкратце, то это изощренный способ кодирования коротких строк, который обладает очень хорошими свойствами исправления ошибок, а также обнаружения.

Порция данных может быть разделена на три раздела:

- временная метка;
- ноль или более тегированных пар ключ–значение;
- подпись под всем счетом.

Временная метка выражается в секундах с 1970 года, или эпохе Unix. Указанная временная метка позволяет отправителю определять возраст счета и, как мы увидим позже, дает возможность получателю принудительно делать счет валидным только в течение определенного периода времени, если он того пожелает.

Подобно формату TLV, о котором мы узнали в разделе «Формат “Тип–длина–значение”» на стр. 323, в формате счета по спецификации BOLT # 11 используется ряд расширяемых пар ключ–значение, кодирующих информацию, необходимую для удовлетворения платежа. Поскольку используются пары ключ–значение, в будущем легко добавлять новые значения, если будет введен новый тип платежа или дополнительные требования/функциональность.

Наконец, прилагается подпись, которая охватывает весь счет, подписанный получателем платежа. Эта подпись позволяет отправителю верифицировать, что платежный запрос действительно был создан получателем платежа. В отличие от платежных запросов Bitcoin, которые не подписаны, это позволяет нам обеспечивать, чтобы платежный запрос был подписан конкретным субъ-

ектом. Сама подпись кодируется с использованием ИД восстановления, что позволяет использовать более компактную подпись, позволяющую извлекать публичный ключ. При проверке подписи ИД восстановления извлекает публичный ключ, а затем сверяет его с публичным ключом, включенным в счет.

Тегированные поля счета

Тегированные поля счета кодируются в главном теле счета. Указанные поля представляют разные пары ключ–значение, которые выражают либо дополнительную информацию, которая способна помочь совершить платеж, либо информацию, *требуемую* для совершения платежа. Поскольку используется легкий вариант bech32, каждое из этих полей фактически находится в домене «основание 5».

Данное поле тега состоит из трех компонентов:

- тип поля (5 бит);
- длина данных поля (10 бит);
- сами данные, которые имеют длину * 5 байт в размере.

Полный список всех определенных в настоящее время тегированных полей приведен в табл. 15-3.

Таблица 15-3. Тегированные поля счета по BOLT №11

Тег поля	Длина данных	Применение
p	52	Платежный хеш SHA-256
s	52	256-битовый секрет, который повышает сквозную конфиденциальность платежа за счет смягчения прощупывания со стороны промежуточных узлов
d	Переменная	Описание, короткий литерал в формате UTF-8, указывающий назначение платежа
n	53	Публичный ключ узла назначения
h	52	Хеш, представляющий описание самого платежа. Он может использоваться для фиксации на описании длиной более 639 байт
x	Переменная	Время истечения срока платежа в секундах. Дефолтное значение равно 1 часу (3600), если не указано
c	Переменная	min_cltv_expiry, используемая для последнего перехода по маршруту. Дефолтное значение равно 9, если не указано
f	Переменная	Резервный сетевой адрес, который будет использоваться для завершения платежа, если платеж не может быть совершен через сеть Lightning
r	Переменная	Одна или несколько записей, которые позволяют получателю предоставлять отправителю дополнительные эфемерные ребра с целью завершения платежа
9	Переменная	Набор 5-битовых значений, содержащих биты функциональностей, требуемые для завершения платежа

Элементы, содержащиеся в поле r, обычно называются *маршрутизационными подсказками*. Они позволяют получателю передавать дополнительный набор данных, которые способны помочь отправителю совершить платеж. Под-

сказки обычно используются, когда у получателя есть некоторые / все приватные каналы и он хочет направить отправителя в эту «некартированную» часть канального графа. Маршрутизационная подсказка эффективно кодирует ту же информацию, что и обычное сообщение `channel_update`. Само обновление упаковывается в одно значение со следующими ниже полями:

- публичный ключ исходящего узла в ребре (264 бита);
- идентификатор `short_channel_id` «виртуального» ребра (64 бита);
- базовые комиссионные (`fee_base_msat`) ребра (32 бита);
- пропорциональные комиссионные (`fee_proportional_millionths`) (32 бита);
- дельта истечения срока CLTV (`cltv_expiry_delta`) (16 бит).

Заключительной порцией сегмента данных является набор битов функциональностей, которые сообщают отправителю функциональность, необходимую для завершения платежа. В качестве примера, если в будущем будет добавлен новый тип платежа, который не совместим с изначальным типом платежа, получатель сможет установить требуемый бит функциональности, чтобы сообщить, что плательщику для совершения платежа необходимо понимать эту функциональность.

Вывод

Как мы уже увидели, счета – это гораздо больше, чем просто запрос на определенную сумму. Они содержат важную информацию о том, как произвести платеж, такую как маршрутизационные подсказки, публичный ключ узла назначения, эфемерные ключи для повышения безопасности и многое другое.

Глава 16

Безопасность и конфиденциальность сети Lightning

В этой главе мы обратимся к нескольким наиболее важным вопросам, связанным с безопасностью и конфиденциальностью сети Lightning. Сначала мы рассмотрим конфиденциальность, что это значит, как ее оценивать, и некоторые вещи, которые можно сделать, чтобы защитить свою личную жизнь при использовании сети Lightning. Затем мы разведаем несколько распространенных атак и методов борьбы с ними.

Почему важна конфиденциальность?

Ключевое ценностное предложение криптовалюты – это деньги, устойчивые к цензуре. Система Bitcoin предлагает участникам возможность хранить и передавать свои богатства без вмешательства правительств, банков или корпораций. Сеть Lightning продолжает эту миссию.

В отличие от тривиальных решений по масштабированию, таких как депозитарные Bitcoin-банки, сеть Lightning нацелена на масштабирование системы Bitcoin без ущерба для самоопеки, что должно приводить к большей устойчивости к цензуре в экосистеме Bitcoin. Однако сеть Lightning работает в рамках другой модели безопасности, которая бросает новые вызовы в отношении безопасности и конфиденциальности.

Определения конфиденциальности

Вопрос «Является ли Lightning конфиденциальной?» не имеет прямого ответа. Конфиденциальность (приватность) – сложная тема; нередко бывает трудно точно определить, что мы подразумеваем под конфиденциальностью, в особенности если вы не являетесь исследователем конфиденциальности. К счастью, исследователи конфиденциальности используют процессы для анализа и оценивания конфиденциальностных характеристик систем, и мы тоже можем их использовать! Давайте посмотрим на то, как исследователь безопас-

ности может попытаться ответить на вопрос «Является ли Lightning конфиденциальной?» в два общих шага.

Прежде всего исследователь конфиденциальности определил бы *модель безопасности*, которая детализирует то, на что способен злоумышленник и чего он стремится достичь. Затем он опишет релевантные свойства системы и проверит, согласуется ли она с требованиями.

Процесс оценивания конфиденциальности

Модель безопасности основана на наборе опорных допущений о безопасности. В криптографических системах эти допущения часто сосредоточены вокруг математических свойств криптографических примитивов, таких как шифры, подписи и хеш-функции. Допущения о безопасности сети Lightning заключаются в том, что используемые в протоколе ECDSA-подписи, хеш-функция SHA-256 и другие криптографические функции работают в соответствии с их определениями безопасности. Например, мы исходим из допущения, что практически невозможно найти прообраз (и второй прообраз) хеш-функции. Благодаря этому сеть Lightning может опираться на механизм HTLC-контракта (который использует прообраз хеш-функции) с целью обеспечения атомарности многопереходных платежей: никто, кроме конечного получателя, не может раскрыть платежный секрет и урегулировать HTLC-контракт. Мы также допускаем определенную степень связности в сети, а именно что каналы Lightning образуют связанный граф. И поэтому можно отыскать путь от любого отправителя к любому получателю. Наконец, мы исходим из допущения о том, что сетевые сообщения распространяются в течение определенных тайм-аутов.

Теперь, когда мы выявили несколько опорных допущений, давайте рассмотрим некоторых возможных противников.

Вот некоторые возможные модели противников в сети Lightning. Пересылающий узел «честный, но любопытный» может отслеживать суммы платежей, непосредственно предшествующие и последующие узлы, а также граф объявленных каналов с их емкостью. Очень высокосвязный узел может делать то же самое, но в большей степени. Например, возьмем разработчиков популярного кошелька, поддерживающих узел, к которому их пользователи подсоединяются по умолчанию. Этот узел будет отвечать за маршрутизацию значительной доли платежей пользователям этого кошелька и от них. Что делать, если несколько узлов находятся под контролем противника? Если два вступающих в сговор узла находятся на одном и том же платежном пути, то они поймут, что пересылают HTLC-контракты, принадлежащие одному и тому же платежу, потому что HTLC-контракты имеют одинаковый платежный хеш.

Многокомпонентные платежи (см. раздел «Многокомпонентные платежи» на стр. 314) позволяют пользователям запутывать суммы своих платежей с учетом их неравномерных размеров деления.

Каковы могут быть цели злоумышленника Lightning? Информационная безопасность часто описывается в терминах трех основных свойств: конфиденциальности, целостности и доступности.

Конфиденциальность

Информация попадает только к намеченным получателям.

Целостность

Информация не изменяется при передаче.

Доступность

Система функционирует бóльшую часть времени.

Важные свойства сети Lightning в основном сосредоточены на конфиденциальности и доступности. Некоторые наиболее важные свойства, которые подлежат защите, таковы:

- только отправитель и получатель знают сумму платежа;
- никто не может связать отправителей и получателей;
- честный пользователь не может быть заблокирован от отправки и получения платежей.

Для каждой цели конфиденциальности и модели безопасности существует определенная вероятность того, что злоумышленник добьется успеха. Эта вероятность зависит от различных факторов, таких как размер и структура сети. При прочих равных условиях, как правило, легче успешно атаковать малую сеть, чем крупную. Точно так же чем более централизована сеть, тем более способным может быть злоумышленник, если «центральные» узлы находятся под его контролем. Разумеется, термин «централизация» должен быть точно определен, чтобы вокруг него строить модели безопасности, и существует немало возможных определений того, насколько централизованной сеть является. Наконец, как платежная сеть, сеть Lightning зависит от экономических стимулов. Размер и структура комиссионных влияют на алгоритм маршрутизации и, следовательно, могут либо помочь злоумышленнику, направляя большинство платежей через свои узлы, либо предотвратить это.

Анонимностное множество

Что значит деанонимизировать кого-то? Проще говоря, деанонимизация подразумевает привязку какого-либо действия к реальной идентичности человека, такой как его имя или физический адрес. В исследованиях конфиденциальности понятие деанонимизации имеет более тонкие нюансы. Во-первых, мы не обязательно говорим об именах и адресах. Обнаружение чьего-либо IP-адреса или номера телефона тоже может рассматриваться как деанонимизация. Часть информации, которая позволяет связать действие пользователя с его предыдущими действиями, называется *идентичностью*. Во-вторых, деанонимизация не является двоичной; пользователь не является ни полностью анонимным, ни полностью деанонимизированным. Вместо этого исследование конфиденциальности рассматривает анонимность в сравнении с анонимностным множеством.

Анонимностное множество является центральным понятием в исследованиях конфиденциальности. Оно относится к множеству идентичностей таким образом, что, с точки зрения злоумышленника, данное действие может соответствовать любому из этого множества. Рассмотрим пример из реальной жизни. Представьте, что вы встречаете человека на городской улице. Каков, с вашей

точки зрения, уровень его анонимности? Если вы не знаете его лично и не имеете какой-либо дополнительной информации, то его анонимностное множество примерно равно населению города, включая путешественников. Если вы дополнительно рассмотрите его внешность, то сможете примерно оценить его возраст и исключить из анонимностного множества жителей города, которые явно старше или моложе рассматриваемого человека. Кроме того, если вы заметите, что человек входит в офис компании X, используя электронный бейдж, то анонимностное множество сжимается до числа сотрудников и посетителей компании X. Наконец, вы можете заметить номер машины, на которой он приехал на место. Если вы – случайный наблюдатель, то это мало что вам даст. Однако если вы являетесь городским чиновником и имеете доступ к базе данных, которая соотносит номерные знаки с именами, то сможете ограничить анонимностное множество всего несколькими людьми: владельцем автомобиля и любыми близкими друзьями и родственниками, которые могли одолжить автомобиль.

Этот пример иллюстрирует несколько важных моментов. Во-первых, каждый бит информации может приближать противника к его цели. Возможно, нет необходимости сжимать множество параметров до размера единицы. Например, если злоумышленник планирует целенаправленную атаку типа «отказ в обслуживании» (DoS) и может вывести из строя 100 серверов, то достаточно анонимностного множества из 100 элементов. Во-вторых, злоумышленник может соотнести информацию из разных источников. Даже если утечка конфиденциальной информации выглядит относительно безобидной, мы никогда не знаем, чего она может достичь в сочетании с другими источниками данных. Наконец, в особенности в криптографических обстановках, у злоумышленника всегда есть «последнее средство» – поиск методом грубой силы. Криптографические примитивы сконструированы таким образом, что практически невозможно угадывать секрет, такой как приватный ключ. Тем не менее каждый бит информации приближает противника к этой цели, и в какой-то момент она становится достижимой.

С точки зрения Lightning, деанонимизация обычно означает получение соответствия между платежами и пользователями, идентифицируемыми по идентификаторам узлов. Каждому платежу может быть назначено анонимностное множество отправителя и анонимностное множество получателя. В идеале анонимностное множество состоит из всех пользователей сети. За счет этого гарантируется, что у злоумышленника нет никакой информации вообще. Однако в реальной сети происходит утечка информации, которая позволяет злоумышленнику сужать круг поиска. Чем меньше анонимностное множество, тем выше вероятность успешной деанонимизации.

Различия между сетями Lightning и Bitcoin с точки зрения конфиденциальности

Хотя верно, что транзакции в сети Bitcoin не ассоциируют реально существующие идентификаторы с Bitcoin-адресами, все транзакции передаются открытым текстом и могут быть проанализированы. Для деанонимизации пользователей Bitcoin и других криптовалют были созданы многочисленные компании.

На первый взгляд, Lightning обеспечивает более высокую конфиденциальность, чем Bitcoin, потому что платежи Lightning не передаются по всей сети

широковещательно. В то время как это улучшает базовый уровень конфиденциальности, другие свойства протокола Lightning могут усложнять анонимные платежи. Например, у более крупных платежей может быть меньше вариантов маршрутизации. Вследствие этого злоумышленник, который контролирует узлы с высокой капитализацией, получает потенциальную возможность маршрутизировать большинство крупных платежей и обнаруживать суммы платежей и, возможно, другие детали. Со временем, по мере роста сети Lightning, это может стать меньшей проблемой.

Еще одно важное различие между Lightning и Bitcoin заключается в том, что узлы Lightning поддерживают постоянную идентичность, в то время как узлы Bitcoin этого не делают. Опытный пользователь Bitcoin может легко переключать узлы, используемые для получения данных блочной цепи и широковещательных транзакций. Пользователь Lightning, напротив, отправляет и получает платежи через узлы, которые он использовал для открытия своих платежных каналов. Более того, протокол Lightning исходит из допущения, что маршрутизационные узлы объявляют свой IP-адрес в дополнение к их ИД узла. Это создает постоянную связь между идентификаторами узлов и IP-адресами, что бывает опасно, учитывая, что IP-адрес нередко является промежуточным шагом в атаках на анонимность, связанных с физическим местоположением пользователя и, в большинстве случаев, реальной идентичностью. Есть возможность использовать Lightning поверх Tor, но многие узлы не используют эту функциональность, как видно из статистики, собранной из объявлений узлов.

У пользователя Lightning при отправке платежа в его анонимностном множестве находятся его соседи. В частности, маршрутизационный узел знает только непосредственно предшествующий и следующий узлы. Маршрутизационный узел не знает, являются ли его непосредственные соседи по маршруту платежа конечным отправителем или получателем. Следовательно, анонимностное множество узла в Lightning примерно равно его соседям (см. рис. 16-1).

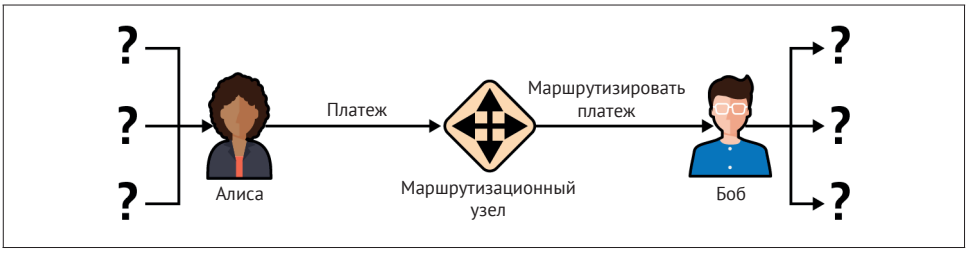


Рис. 16-1. Анонимностное множество Алисы и Боба составлено из их соседей

Аналогичная логика применима и к получателям платежей. Многие пользователи открывают лишь несколько платежных каналов, тем самым ограничивая свои возможности анонимности. Более того, в Lightning анонимностное множество статично или, по меньшей мере, меняется медленно. Напротив, во внутрицепных CoinJoin-транзакциях можно добиваться значительно большего уровня анонимности. CoinJoin-транзакции с анонимностными множествами, превышающими 50, довольно часты. Как правило, анонимностные мно-

жества в CoinJoin-транзакции соответствуют динамически изменяющемуся множеству пользователей.

Наконец, пользователям Lightning также может быть отказано в обслуживании, поскольку их каналы заблокированы или истощены злоумышленником. Для пересылки платежей требуется капитал (дефицитный ресурс!), который должен быть временно заперт в HTLC-контрактах вдоль маршрута. Злоумышленник может отправлять много платежей, но не завершать их, занимая капитал честных пользователей в течение длительного времени. Этот вектор атаки отсутствует (или, по меньшей мере, не так очевиден) в Bitcoin.

Подводя итог, в то время как некоторые аспекты архитектуры сети Lightning предполагают, что она является шагом вперед с точки зрения конфиденциальности по сравнению с Bitcoin, другие свойства протокола могут облегчать атаки на конфиденциальность. Необходимы тщательные исследования, чтобы оценить, какие гарантии конфиденциальности предоставляет сеть Lightning, и улучшить положение дел.

Вопросы, обсуждаемые в этой части главы, обобщают исследования, имевшиеся на середину 2021 года. Однако указанная область исследований и разработок быстро развивается. Мы рады сообщить, что авторам известно о нескольких исследовательских группах, которые в настоящее время работают над конфиденциальностью сети Lightning.

Теперь давайте рассмотрим несколько атак на конфиденциальность LN, которые были описаны в научной литературе.

Атаки на Lightning

Недавние исследования описывают различные способы, с помощью которых безопасность и конфиденциальность сети Lightning могут быть скомпрометированы.

Наблюдение за суммами платежей

Одной из целей платежной системы, сберегающей конфиденциальность, является сокрытие суммы платежа от посторонних лиц. Сеть Lightning в этом отношении является усовершенствованием по сравнению со слоем 1. В то время как Bitcoin-транзакции транслируются широковещательно открытым текстом и могут наблюдаться кем угодно, платежи Lightning проходят только через несколько узлов по платежному пути. Однако промежуточные узлы действительно видят сумму платежа, хотя эта сумма платежа может не соответствовать фактической общей сумме платежа (см. раздел «Многокомпонентные платежи» на стр. 314). Это необходимо для создания нового HTLC-контракта на каждом переходе. Доступность сумм платежей для промежуточных узлов не представляет непосредственной угрозы. Однако честный, но любопытный промежуточный узел может его использовать как часть более масштабной атаки.

Связывание отправителей и получателей

Злоумышленник может быть заинтересован в том, чтобы узнать отправителя и/или получателя платежа, дабы выявить определенные экономические отношения. Это нарушение конфиденциальности может нанести ущерб сопро-

тивлению цензуре, поскольку промежуточный узел может подвергать цензуре платежи определенным получателям или отправителям либо от них. В идеале связывание отправителей с получателями не должно быть возможным ни для кого, кроме отправителя и получателя.

В следующих разделах мы рассмотрим два типа противников: противника, находящегося вне пути, и противника, находящегося на пути. Злоумышленник вне пути пытается оценить отправителя и получателя платежа, не участвуя в процессе маршрутизирования платежа. Злоумышленник на пути может использовать любую информацию, которую он может получить, маршрутизируя выплату процентов.

Во-первых, рассмотрим противника вне пути. На первом шаге этого сценария атаки сильный злоумышленник вне пути логически выводит индивидуальные остатки в каждом платежном канале посредством прощупывания (описано в следующем разделе) и формирует снимок сети во время t_1. Для простоты давайте сделаем t_1 равным 12:05. Затем он снова прощупывает сеть некоторое время спустя во время t_2, которое мы сделаем равным 12:10. Потом злоумышленник сравнивает снимки, взятые в 12:10 и 12:05, и использует различия между двумя снимками, чтобы получить информацию о совершенных платежах, просмотрев пути, которые изменились. В простейшем случае, если бы между 12:10 и 12:05 произошел только один платеж, злоумышленник наблюдал бы единственный путь, по которому остатки изменились на те же суммы. Следовательно, злоумышленник узнает об этом платеже почти все: отправителя, получателя и сумму. Если несколько платежных путей друг на друга накладываются, то противнику необходимо применить эвристику, чтобы идентифицировать такое наложение и отделить платежи.

Теперь мы обратим наше внимание на *противника*, находящегося *на пути*. Такой противник может показаться заумным. Однако в июне 2020 года исследователи отметили, что на одном самом центральном узле наблюдалось около 50 % всех платежей LN[99], в то время как на четырех самых центральных узлах наблюдалось в среднем 72 % платежей[100]. Эти результаты подчеркивают актуальность модели злоумышленника на пути. Несмотря на то что посредники в платежном пути узна́ют только о своем преемнике и предшественнике, существует несколько утечек, которые злонамеренный или честный, но любопытный посредник может использовать для логического выведения отправителя и получателя.

Злоумышленник на пути может отслеживать сумму любого маршрутизируемого платежа, а также дельты привязки ко времени (см. главу 10). Следовательно, злоумышленник может исключать любые узлы из анонимностного множества отправителя или получателя с емкостью ниже, чем маршрутизируемая сумма. Поэтому мы наблюдаем компромисс между конфиденциальностью и суммами платежей. В типичной ситуации чем больше сумма платежа, тем меньше анонимностное множество. Мы отмечаем, что эта утечка может быть минимизирована с помощью многокомпонентных платежей либо с помощью платежных каналов большой емкости. Аналогичным образом платеж-

[99] См. https://arxiv.org/pdf/2006.12143.pdf.

[100] См. https://arxiv.org/pdf/1909.06890.pdf.

ные каналы с малыми дельтами привязки ко времени могут быть исключены из платежного пути. Точнее, платежный канал нельзя относить к платежу, если оставшееся время, на которое платеж может быть привязан, больше, чем то, что узел пересылки был бы готов принять. Эту утечку можно было бы устранить, придерживаясь так называемых теневых маршрутов.

Одной из самых тонких и в то же время мощных утечек, которые может спровоцировать злоумышленник на пути, является анализ времени. Злоумышленник на пути может вести журнал для каждого маршрутизируемого платежа, а также количество времени, которое требуется узлу для ответа на запрос HTLC-контракта. Перед началом атаки злоумышленник изучает характеристики задержки каждого узла в сети Lightning, отправляя им запросы. Естественно, это может помочь в установлении точного положения противника на платежном пути. Более того, как недавно было показано, злоумышленник может успешно определять отправителя и получателя платежа из множества возможных отправителей и получателей, используя оценщиков, основанных на времени.

Наконец, важно признать, что, вероятно, существуют неизвестные или неизученные утечки, которые способны помочь попыткам деанонимизации. Например, поскольку разные кошельки Lightning применяют разные алгоритмы маршрутизации, даже знание применяемого алгоритма маршрутизации может помочь исключать определенные узлы из числа отправителей и/или получателей платежа.

Раскрытие остатков каналов (прощупывание)

Остатки каналов Lightning должны быть скрыты по соображениям конфиденциальности и эффективности. Узел Lightning знает остатки только своих соседних каналов. Протокол не предоставляет стандартного способа запроса остатка дистанционного канала.

Однако злоумышленник может раскрыть остаток дистанционного канала в ходе *атаки прощупыванием*. В информационной безопасности прощупывание относится к технике отправки запросов в целевую систему и принятия решений о ее приватном состоянии на основе полученных ответов.

Каналы Lightning подвержены прощупыванию. Напомним, что стандартный платеж Lightning начинается с того, что получатель создает случайный платежный секрет и отправляет его хеш отправителю. Обратите внимание, что для промежуточных узлов все хеши выглядят случайными. Нет никакого способа определить, соответствует ли хеш реальному секрету или был сгенерирован случайным образом.

Атака прощупыванием протекает следующим образом. Допустим, злоумышленник Мэллори хочет раскрыть остаток Алисы в публичном канале между Алисой и Бобом. Предположим, что суммарная емкость этого канала составляет 1 млн сатоши. Остаток Алисы может составлять от нуля до 1 млн сатоши (если быть точным, то оценка немного жестче из-за резерва канала, но ради простоты здесь мы его не учитываем). Мэллори открывает канал с Алисой с 1 млн сатоши и отправляет 500 000 сатоши Бобу через Алису, используя случайное число в качестве платежного хеша. Конечно же, это число не соответствует какому-либо известному платежному секрету. Следовательно, платеж не будет произведен. Вопрос в том, как именно он даст сбой?

Есть два сценария. Если у Алисы есть более 500 000 сатоши на ее стороне канала для Боба, то она пересылает платеж. Боб дешифрует платежный код и понимает, что платеж предназначен для него. Он просматривает свое локальное хранилище платежных секретов и ищет прообраз, соответствующий платежному хешу, но не находит его. Следуя протоколу, Боб возвращает ошибку «неизвестный платежный хеш» Алисе, которая передает ее обратно Мэллори. В результате Мэллори знает, что платеж мог бы быть успешным, если бы платежный хеш был реальным. Таким образом, Мэллори может обновить свою оценку остатка Алисы с «от нуля до 1 млн» до «от 500 000 до 1 млн». Еще один сценарий происходит, если остаток Алисы ниже 500 000 сатоши. В этом случае Алиса не может переслать платеж и возвращает Мэллори сообщение об ошибке «недостаточный остаток». Мэллори обновляет свою оценку с «от нуля до 1 млн» до «от нуля до 500 000».

Обратите внимание, что в любом случае оценка Мэллори становится в два раза точнее всего после одного прощупывания! Она может продолжить прощупывание, выбрав следующую величину прощупывания таким образом, чтобы она делила текущий интервал оценки пополам. Этот хорошо известный метод поиска называется *двоичным поиском*. При двоичном поиске число щупов является логарифмическим с требуемой точностью. Например, чтобы получить остаток Алисы в канале от 1 млн сатоши до одного сатоши, Мэллори нужно было бы всего лишь выполнить $\log_2(1\,000\,000) \approx 20$ прощупываний. Если одно прощупывание занимает 3 секунды, то один канал может быть точно прощупан всего за минуту!

Прощупывание канала можно сделать еще более эффективным. В простейшем варианте Мэллори напрямую подсоединяется к каналу, который она хочет прощупать. Можно ли прощупать канал, не открывая канал для одной из его конечных точек? Представьте, что Мэллори теперь хочет прощупать канал между Бобом и Чарли, но не хочет открывать другой канал, который требует оплаты внутрицепных комиссионных и ожидания подтверждений финансовых транзакций. Вместо этого Мэллори реиспользует свой существующий канал с Алисой и отправляет щуп по маршруту Мэллори → Алиса → Боб → Чарли. Мэллори может интерпретировать ошибку «неизвестный платежный хеш» так же, как и раньше: щуп достиг пункта назначения; следовательно, все каналы вдоль маршрута имеют достаточные остатки для его пересылки. Но что, если Мэллори получит сообщение об ошибке «недостаточный остаток»? Означает ли это, что остаток недостаточен между Алисой и Бобом или же между Бобом и Чарли?

В текущем протоколе Lightning сообщения об ошибках говорят не только о том, какая ошибка произошла, но и о том, где она произошла. Итак, при более тщательной обработке ошибок Мэллори теперь знает, какой канал отказал. Если это целевой канал, то она обновляет свои оценки; если нет, то она выбирает другой маршрут к целевому каналу. Она даже получает дополнительную информацию об остатках промежуточных каналов в дополнение к остатку целевого канала.

Атака прощупыванием может быть дополнительно использована для связывания отправителей и получателей, как описано в предыдущем разделе.

В этом месте вы, возможно, спросите: почему сеть Lightning так плохо справляется с защитой конфиденциальных данных своих пользователей? Не лучше ли было бы не сообщать отправителю, почему и где произошел сбой платежа?

И действительно, это могло бы стать потенциальной контрмерой, но у нее есть существенные недостатки. Сеть Lightning должна соблюдать тщательное равновесие между безопасностью и эффективностью. Помните, что обычные узлы не знают распределения остатка в дистанционных каналах. Таким образом, платежи могут завершаться и (часто действительно) завершаются безуспешно из-за недостаточного остатка на промежуточном переходе. Сообщения об ошибках позволяют отправителю исключать сбойный канал из рассмотрения при построении другого маршрута. Один популярный кошелек Lightning даже выполняет внутреннее прощупывание, чтобы проверять, действительно ли построенный маршрут способен справиться с платежом.

Существуют и другие потенциальные контрмеры против прощупывания канала. Во-первых, злоумышленнику трудно нацеливаться на необъявленные каналы. Во-вторых, узлы, которые имплементируют маршрутизацию точно в срок (JIT), могут быть менее подвержены атаке. Наконец, поскольку многокомпонентные платежи уменьшают проблему недостаточной емкости, разработчики протокола могут рассмотреть возможность сокрытия некоторых деталей ошибок без ущерба для эффективности.

Отказ в обслуживании

Когда ресурсы становятся публично доступными, существует риск того, что злоумышленники могут попытаться сделать этот ресурс недоступным, выполнив атаку типа «отказ в обслуживании» (denial-of-service, аббр. DoS). Как правило, это достигается за счет того, что злоумышленник бомбардирует ресурс запросами, которые неотличимы от законных запросов. Атаки редко приводят к тому, что цель терпит финансовые убытки, помимо альтернативных издержек, связанных с отключением их службы, и просто предназначены для того, чтобы нанести ущерб цели.

Типичные меры по смягчению последствий DoS-атак требуют аутентификации запросов, чтобы отделять законных пользователей от вредоносных. Эти меры по смягчению последствий влекут за собой незначительные затраты для обычных пользователей, но будут служить достаточным сдерживающим фактором для злоумышленника, запускающего масштабные запросы. Меры по борьбе с атаками «отказ в обслуживании» можно увидеть повсюду в интернете – веб-сайты устанавливают ограничения скорости в целях обеспечения того, чтобы ни один пользователь не смог потреблять все внимание своего сервера, веб-сайты для просмотра фильмов требуют аутентификации на входе, чтобы держать сердитых участников r/prequelmemes (группы Reddit) в загоне, а службы передачи данных продают API-ключи с целью ограничения числа запросов.

DoS в Bitcoin

В Bitcoin емкость, которую узлы используют для ретрансляции транзакций, и пространство, которое они предоставляют сети в виде своей очереди mempool, являются публично доступными ресурсами. Любой узел в сети может потреблять емкость и пространство mempool, отправив валидную транзакцию. Если эта транзакция добывается в валидном блоке, то они будут платить комиссионные за транзакцию, что увеличивает стоимость использования этих совместных сетевых ресурсов.

В прошлом сеть Bitcoin столкнулась с попыткой DoS-атаки, когда злоумышленники рассылали спам по сети с помощью транзакций с низкими комиссионными. Многие из этих транзакций не были выбраны майнерами из-за их низких комиссионных за транзакции, поэтому злоумышленники могли потреблять сетевые ресурсы, не платя комиссионных. В целях решения этой проблемы были установлены минимальные комиссионные за ретрансляцию транзакций, которые устанавливают пороговую комиссию, требуемую узлами для распространения транзакций. Эта мера в значительной степени обеспечила, чтобы потребляющие сетевые ресурсы транзакции в конечном итоге платили свои цепные комиссионные. Минимальные комиссионные за ретрансляцию приемлемы для обычных пользователей, но наносят финансовый ущерб злоумышленникам, если они попытаются рассылать спам по сети. Хотя некоторые транзакции могут не попадать в валидные блоки в условиях высоких комиссионных, эти меры в значительной степени эффективны для предотвращения такого рода спама.

DoS в Lightning

Аналогично Bitcoin, сеть Lightning взимает комиссионные за использование своих публично доступных ресурсов, но в данном случае ресурсы являются публично доступными каналами, а комиссионные поступают в виде комиссионных за маршрутизацию. Возможность маршрутизировать платежи через узлы в обмен на комиссионные обеспечивает сети большое преимущество в масштабируемости – узлы, которые не подсоединены напрямую, все равно могут совершать транзакции, но это происходит за счет раскрытия публично доступного ресурса, который должен быть защищен от DoS-атак.

Когда узел Lightning пересылает платеж от вашего имени, он использует данные и емкость платежа, чтобы обновлять свою фиксационную транзакцию, и сумма платежа резервируется на остатке его канала до тех пор, пока она не будет погашена или не завершится безуспешно. При успешных платежах это приемлемо, потому что узлу в конечном итоге выплачиваются его комиссионные. Безуспешные платежи не облагаются комиссионными в соответствии с текущим протоколом. Благодаря этому узлы могут беззатратно маршрутизировать безуспешные платежи по любым каналам. Это отлично подходит для законных пользователей, которые не хотели бы платить за безуспешные попытки, но также позволяет злоумышленникам беззатратно потреблять ресурсы узлов – так же, как низкокомиссионные транзакции в Bitcoin, которые никогда не заканчиваются оплатой комиссионных майнеру.

На момент написания этой главы в списке рассылки lightning-dev продолжается обсуждение вопроса о том, как наилучшим образом решить эту проблему.

Известные DoS-атаки

Известны две DoS-атаки на публично доступные каналы LN, которые делают целевой канал или набор целевых каналов непригодными для использования. Обе атаки включают маршрутизацию платежей по публично доступному каналу, а затем удержание их до истечения тайм-аута, что максимизирует продолжительность атаки. Требование отказывать в платежах, чтобы не платить комиссионные, довольно просто выполняется, потому что вредоносные узлы могут просто ремаршрутизировать платежи на себя. В отсутствие комиссион-

ных за безуспешные платежи единственной стоимостью для злоумышленника являются затраты на открытие канала с целью отправки этих платежей, что делается тривиально в условиях низких комиссионных.

Заклинивание фиксаций

Узлы Lightning обновляют свое совместное состояние с помощью асимметричных фиксационных транзакций, при которых HTLC-контракты добавляются и удаляются с целью облегчения платежей. Каждая сторона ограничена в общей сложности 483[101] HTLC-контрактами в фиксационной транзакции за раз. Атака с заклиниванием канала (channel jamming) позволяет злоумышленнику делать канал непригодным для использования, направляя 483 платежа по целевому каналу и удерживая их до истечения времени ожидания.

Следует отметить, что это ограничение было выбрано в спецификации в обеспечение того, чтобы все HTLC-контракты могли обрабатываться в рамках одной законной транзакции[102]. Хотя этот лимит может быть увеличен, транзакции все еще ограничены размером блока, поэтому число доступных слотов, скорее всего, останется ограниченным.

Запирание ликвидности канала

Атака с запиранием ликвидности канала (channel liquidity lockup) сравнима с атакой с заклиниванием каналов в том смысле, что она маршрутизирует платежи по каналу и удерживает их так, что канал становится непригодным для использования. Вместо того чтобы запирать слоты на канальной фиксации, эта атака маршрутизирует крупные HTLC-контракты через целевой канал, потребляя всю доступную полосу пропускания канала. Фиксации капитала со стороны атаки выше, чем при атаке с заклиниванием фиксаций, поскольку атакующему узлу требуется больше средств для маршрутизирования безуспешных платежей через цель.

Межслоевая деанонимизация

Компьютерные сети нередко бывают многослойными. Многослойность позволяет подразделять задачи и делает всю систему управляемой. Никто не смог бы создать веб-сайт, если бы для этого требовалось понимание всего стека TCP/IP вплоть до физического кодирования битов в оптическом кабеле. Каждый слой призван обеспечивать функциональность вышестоящего слоя чистым способом. В идеале верхний слой должен воспринимать нижний слой как черный ящик. На самом деле, однако, имплементации не идеальны, и детали просачиваются в верхний слой. Это проблема негерметичных абстракций.

В контексте Lightning протокол LN опирается на протокол Bitcoin и P2P-сеть LN. До этого момента мы рассматривали гарантии конфиденциальности, предлагаемые сетью Lightning, только изолированно. Однако создание и закрытие платежных каналов по своей сути выполняются в блочной цепи

[101] См. https://github.com/lightningnetwork/lightning-rfc/blob/c053ce7afb4cbf88615877a0d5fc7b8dbe2b9ba0/02-peer-protocol.md#the-open_channel-message.

[102] См. https://github.com/lightningnetwork/lightning-rfc/blob/master/05-onchain.md#penalty-transaction-weight-calculation.

Bitcoin. Следовательно, для полного анализа обеспечения конфиденциальности в сети Lightning необходимо рассмотреть каждый слой технологического стека, с которым пользователи могут взаимодействовать. В частности, деанонимизирующий злоумышленник может и будет использовать внутрицепные и внецепные данные для кластеризации или связывания узлов LN с соответствующими Bitcoin-адресами.

Злоумышленники, пытающиеся деанонимизировать пользователей LN, могут иметь различные цели в межслоевом контексте:

- кластеризировать Bitcoin-адреса, принадлежащие одному и тому же пользователю (слой 1). Мы называем их Bitcoin-сущностями;
- кластеризировать узлы LN, принадлежащие одному и тому же пользователю (слой 2);
- однозначно связывать наборы узлов LN с наборами Bitcoin-сущностей, которые их контролируют.

Несколько эвристик и шаблонов использования позволяют злоумышленнику кластеризовывать Bitcoin-адреса и узлы LN, принадлежащие одним и тем же пользователям LN. Более того, эти кластеры могут быть связаны между слоями с использованием других мощных эвристик межслоевого связывания. Последний тип эвристики, методы межслоевого связывания, подчеркивает необходимость целостного представления о конфиденциальности. В частности, мы должны рассматривать конфиденциальность в контексте обоих слоев вместе взятых.

Внутрицепная кластеризация Bitcoin-сущностей

Взаимодействия сети Lightning с блочной цепью постоянно отражаются в графе Bitcoin-сущностей. Даже если канал закрыт, злоумышленник может наблюдать, с какого адреса канал финансировался и куда монеты были потрачены после его закрытия. В рамках указанного анализа давайте рассмотрим четыре отдельные сущности. Открытие канала вызывает денежный поток от источниковой сущности к финансирующей сущности; закрытие канала вызывает поток от расчетной сущности к целевой сущности.

В начале 2021 года Ромити (Romiti) и соавт.[103] выявили четыре эвристики, которые позволяют кластеризовывать эти сущности. Две из них улавливают определенное поведение с негерметичным финансированием, а две другие описывают поведение с негерметичным улаживанием расчетов.

Звездная эвристика (финансирование)

> Если компонент содержит одну источниковую сущность, которая направляет средства одной или нескольким финансовым сущностям, то эти финансовые сущности, скорее всего, контролируются одним и тем же пользователем.

Змеиная эвристика (финансирование)

> Если компонент содержит одну источниковую сущность, которая направляет средства одному или нескольким сущностям, которые сами используются в качестве источниковых и финансовых сущностей, то все эти сущности, скорее всего, контролируются одним и тем же пользователем.

[103] См. https://arxiv.org/pdf/2007.00764.pdf.

Коллекторская эвристика (улаживание платежа)

Если компонент содержит одну целевую сущность, которая получает средства от одной или нескольких улаживающих сущностей, то эти улаживающие сущности, скорее всего, контролируются одним и тем же пользователем.

Прокси-эвристика (улаживание платежа)

Если компонент содержит одну целевую сущность, которая получает средства от одной или нескольких сущностей, которые сами используются в качестве улаживающих и целевых сущностей, то эти сущности, скорее всего, контролируются одним и тем же пользователем.

Стоит отметить, что эти эвристические методы могут давать ложноположительные результаты. Например, если транзакции нескольких разрозненных пользователей объединяются в CoinJoin-транзакцию, то звездная или прокси-эвристика может производить ложноположительные результаты. Это может происходить, если пользователи финансируют платежный канал с помощью CoinJoin-транзакции. Еще одним потенциальным источником ложноположительных результатов может быть то, что сущность может представлять нескольких пользователей, если кластеризованные адреса контролируются службой (например, биржей) или от имени своих пользователей (опекаемый кошелек). Однако эти ложноположительные результаты могут быть эффективно отфильтрованы.

Контрмеры

Если выходы финансовых транзакций не реиспользуются для открытия других каналов, то змеиная эвристика не работает. Если пользователи воздерживаются от каналов финансирования из одного внешнего источника и избегают взимания средств в одной внешней целевой сущности, то другие эвристические методы не дадут каких-либо существенных результатов.

Внецепная кластеризация узлов Lightning

Узлы LN рекламируют псевдонимы, например *LNBig.com*. Псевдонимы могут улучшать удобство использования системы. Однако пользователи, как правило, применяют для своих собственных разных узлов похожие псевдонимы. Например, *LNBig.com Выставление счетов*, скорее всего, принадлежит тому же пользователю, что и узел с псевдонимом *LNBig.com*. Учитывая это наблюдение, можно кластеризировать узлы LN, применяя их узловые псевдонимы. В частности, можно кластеризовать узлы LN в единый адрес, если их псевдонимы схожи по некоторой метрике сходства строковых литералов.

Еще одним методом кластеризации узлов LN является применение их IP-адресов или Tor-адресов. Если одни и те же IP-адреса или Tor-адреса соответствуют разным узлам LN, то эти узлы, скорее всего, контролируются одним и тем же пользователем.

Контрмеры

Для большей конфиденциальности псевдонимы должны достаточно отличаться друг от друга. Хотя публичное объявление IP-адресов может быть неизбежным для тех узлов, которые хотят иметь входящие каналы в сети Lightning, воз-

можность подсоединения между узлами одного и того же пользователя может быть снижена, если клиенты каждого узла размещаются у разных поставщиков служб и, следовательно, IP-адресов.

Межслоевое связывание: узлы Lightning и Bitcoin-сущности

Ассоциирование узлов LN с Bitcoin-сущностями является серьезным нарушением конфиденциальности, которое усугубляется тем фактом, что большинство узлов LN раскрывают свои IP-адреса публично. Как правило, IP-адрес можно рассматривать как уникальный идентификатор пользователя. Две широко наблюдаемые модели поведения выявляют связи между узлами LN и Bitcoin-сущностями:

Реиспользование монет

Всякий раз, когда пользователи закрывают платежные каналы, они получают свои соответствующие монеты обратно. Однако многие пользователи реиспользуют эти монеты при открытии нового канала. Эти монеты могут быть эффективно увязаны с общим узлом LN.

Реиспользование сущностей

Как правило, пользователи пополняют свои платежные каналы с Bitcoin-адресов, соответствующих одной и той же Bitcoin-сущности.

Эти алгоритмы межслоевого связывания могут быть сорваны, если пользователи владеют несколькими некластеризованными адресами или используют несколько кошельков для взаимодействия с сетью Lightning.

Возможная деанонимизация Bitcoin-сущностей иллюстрирует, насколько важно учитывать конфиденциальность обоих слоев одновременно, а не по одному за раз.

Граф Lightning

Сеть Lightning, как следует из названия, представляет собой одноранговую сеть платежных каналов. Следовательно, многие из ее свойств (конфиденциальность, устойчивость, связность, эффективность маршрутизации) зависят и характеризуются ее сетевой природой.

В этом разделе мы обсуждаем и анализируем сеть Lightning с точки зрения науки о сетях. Мы особенно заинтересованы в понимании канального графа LN, его устойчивости, связности и других важных характеристик.

Как выглядит граф Lightning в реальности?

Можно было бы ожидать, что сеть Lightning представляет собой случайный граф, где ребра формируются между узлами случайным образом. Если бы это было так, то степенное распределение сети Lightning соответствовало бы нормальному распределению Гаусса. В частности, большинство узлов имели бы примерно одинаковую степень, и мы не ожидали бы узлов с чрезвычайно большими степенями. Это происходит потому, что нормальное распределение экспоненциально уменьшается для значений за пределами интервала вокруг среднего значения распределения. Изображение случайного графа (как мы видели на рис. 12-2) выглядит как топология ячеистой сети. Она выглядит де-

централизованной и неиерархичной: кажется, что каждый узел имеет равное значение. Кроме того, случайные графы имеют большой диаметр. В частности, маршрутизация в таких графах является сложной задачей, поскольку кратчайший путь между любыми двумя узлами имеет умеренно большую длину.

Однако, в отличие от этого, граф LN совершенно другой.

Граф Lightning сегодня

Lightning – это финансовая сеть. Отсюда на рост и формирование сети также влияют экономические стимулы. Всякий раз, когда узел присоединяется к сети Lightning, он, возможно, захочет максимально увеличивать свою связность с другими узлами, чтобы повышать эффективность маршрутизации. Это явление называется преференциальным прикреплением. Означенные экономические стимулы приводят к созданию принципиально иной сети, чем случайный граф.

Основываясь на моментальных снимках публично объявленных каналов, степенное распределение сети Lightning подчиняется функции возведения в степень. В таком графе подавляющее большинство узлов имеют очень мало соединений с другими узлами, в то время как только несколько узлов имеют многочисленные соединения. На высоком уровне эта графовая топология напоминает звезду: сеть имеет хорошо связанное ядро и слабо связанную периферию. Сети со степенным (degree) распределением согласно функции возведения в степень (power function) также называются безмасштабными сетями. Эта топология выгодна для эффективной маршрутизации платежей, но подвержена определенным атакам на основе топологии.

Атаки на основе топологии

Злоумышленник, возможно, захочет нарушить работу сети Lightning и может решить, что его цель состоит в том, чтобы разбить всю сеть на множество более мелких компонентов, что сделает маршрутизацию платежей практически невозможной во всей сети. Менее амбициозной, но все же опасной и серьезной целью может быть выведение из строя только определенных сетевых узлов. Такое нарушение может произойти на уровне узла или на уровне ребра.

Давайте предположим, что злоумышленник может вывести из строя любой узел в сети Lightning. Например, он может атаковать их с помощью распределенной атаки типа «отказ в обслуживании» (DDoS) или вывести их из строя любыми способами. Оказывается, что если злоумышленник выбирает узлы случайным образом, то безмасштабные сети, такие как сеть Lightning, устойчивы к атакам на удаление узлов. Это происходит потому, что случайный узел находится на периферии с малым числом соединений, поэтому он играет незначительную роль в связности сети. Однако если злоумышленник более осмотрителен, то он может нацелиться на наиболее хорошо связанные узлы. Неудивительно, что сеть Lightning и другие безмасштабные сети не защищены от целенаправленных атак на удаление узлов.

С другой стороны, противник мог бы действовать более скрытно. Несколько атак на основе топологии нацеливаются на один узел или один платежный канал. Например, злоумышленник может быть заинтересован в том, чтобы намеренно исчерпать емкость определенного платежного канала. В более общем случае злоумышленник может истощить всю исходящую емкость узла, чтобы вывести его с рынка маршрутизации. Это можно было бы легко получить,

маршрутизируя платежи через узел жертвы с суммами, равными емкости каждого платежного канала. После завершения этой так называемой атаки с изоляцией узла жертва больше не сможет отправлять или маршрутизировать платежи, если она не получит платеж или не восстановит остаток своих каналов.

В заключение темы: удаление ребер и узлов из маршрутизируемой сети Lightning предусмотрено даже по конструкции. Однако в зависимости от используемого вектора атаки противнику может потребоваться предоставить больше или меньше ресурсов для проведения атаки.

Темпоральность сети Lightning

Сеть Lightning – это динамично меняющаяся безразрешительная сеть. Узлы могут свободно присоединяться к сети или покидать ее, они могут открывать и создавать платежные каналы в любое удобное для них время. Следовательно, один статический снимок графа LN дезориентирует. Нам необходимо учитывать темпоральность и постоянно меняющуюся природу сети. На данный момент граф LN растет с точки зрения числа узлов и платежных каналов. Его эффективный диаметр также уменьшается; то есть узлы становятся ближе друг к другу, как хорошо видно по рис. 16-2.

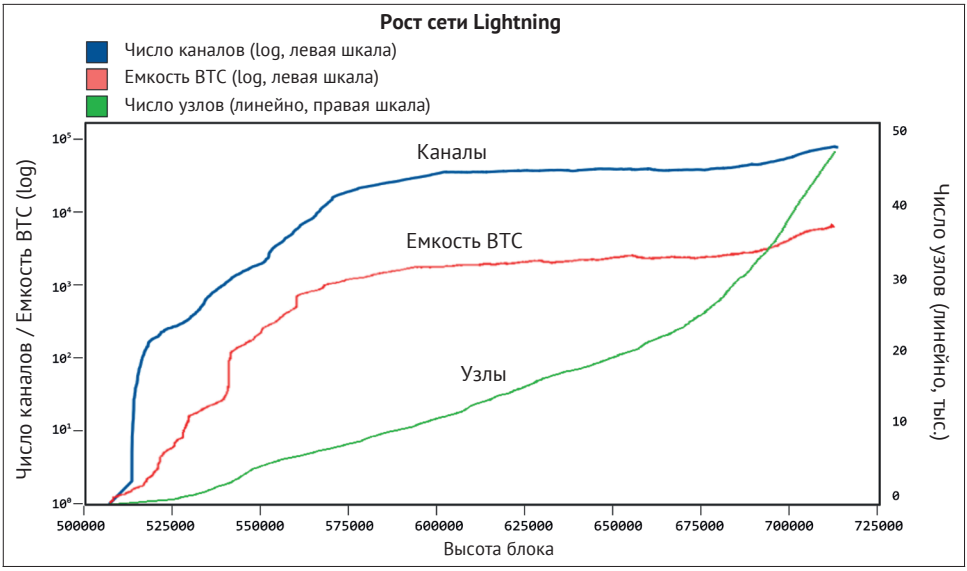


Рис. 16-2. Устойчивый рост сети Lightning в узлах, каналах и привязанной (запертой) емкости (по состоянию на сентябрь 2021 года)

В социальных сетях поведение замыкания треугольника (триадическое замыкание) является обычным явлением. В частности, в графе, где узлы представляют людей, а дружеские отношения представлены в виде ребер, ожидается, что на графе появятся треугольники. Треугольник в данном случае представляет собой попарную дружбу между тремя людьми. Например, если Алиса знает Боба, а Боб знает Чарли, то вполне вероятно, что в какой-то момент Боб познакомит Алису с Чарли. Однако такое поведение было бы странным в сети Lightning. Узлы прос-

то не заинтересованы в замыкании треугольников, потому что они могли бы просто маршрутизировать платежи вместо открытия нового платежного канала. Удивительно, но замыкание треугольника является обычной практикой в сети Lightning. Число треугольников неуклонно росло до имлементации многокомпонентных платежей. Это противоречит здравому смыслу и удивительно, учитывая, что узлы могли бы просто маршрутизировать платежи через две стороны треугольника вместо открытия третьего канала. Это может означать, что неэффективности маршрутизации побуждали пользователей замыкать треугольники и не возвращаться к маршрутизации. Надо надеяться, что многокомпонентные платежи помогут повысить эффективность маршрутизации платежей.

Централизация в сети Lightning

Распространенной метрикой оценивания центральности узла в графе является его *центральность на основе промежуточности*. Доминантность центральной точки – это метрика, выводимая из центральности на основе промежуточности, используемой для оценивания центральности сети. Точное определение понятия доминантности центральной точки читатель может найти в работе Фримена (Freeman)[104].

Чем больше доминантность центральной точки в сети, тем более централизованной является сеть. Можно заметить, что сеть Lightning имеет более высокую доминантность центральной точки (т. е. она более централизована), чем случайный граф (граф Эрдеша–Рени) или безмасштабный граф (граф Барабаши–Альберта) равного размера.

В целом наше понимание динамической природы канального графа LN довольно ограничено. Полезно проанализировать, как изменения протокола, такие как многокомпонентные платежи, могут повлиять на динамику сети Lightning. Было бы полезно подробнее разведать темпоральную природу графа LN.

Экономические стимулы и графовая структура

Граф LN формируется спонтанно, и узлы соединяются друг с другом на основе взаимного интереса. Как следствие стимулы двигают развитие графа. Давайте взглянем на несколько релевантных стимулов.

- Рациональные стимулы:
  - Узлы устанавливают каналы для отправки, получения и маршрутизирования платежей (зарабатывают комиссионные).
  - Что повышает вероятность установления канала между двумя узлами, которые действуют рационально?
- Альтруистические стимулы:
  - Узлы устанавливают каналы «во благо сети».
  - Хотя мы не должны основывать наши допущения о безопасности на альтруизме, в определенной степени альтруистическое поведение движет Bitcoin (прием входящих соединений, раздача блоков).
  - Какую роль он играет в Lightning?

[104] См. https://doi.org/10.2307/3033543.

На ранних стадиях сети Lightning многие операторы узлов утверждали, что комиссионные, заработанные за маршрутизацию, не компенсируют альтернативные издержки, связанные с запиранием ликвидности. Это указывало на то, что оперирование узлом может быть обусловлено главным образом альтруистическими стимулами «во благо сети». Данная ситуация может измениться в будущем, если сеть Lightning будет иметь значительно больший трафик или если появится рынок комиссионных за маршрутизацию. С другой стороны, если узел желает оптимизировать свои комиссионные за маршрутизацию, то он должен минимизировать среднюю длину кратчайшего пути до всех последующих узлов. Иными словами, узел, ищущий прибыль, попытается расположиться в центре канального графа или близко к нему.

Практические советы пользователям по защите их конфиденциальности

Мы все еще находимся на ранних стадиях развития сети Lightning. Многие из проблем, перечисленных в этой главе, вероятно, будут решены по мере ее взросления и роста. В то же время ниже приведено несколько мер, которые вы можете предпринять для защиты вашего узла от злоумышленников; что-то столь простое, как обновление дефолтных параметров, с которыми работает ваш узел, может значительно повысить прочность вашего узла.

Необъявленные каналы

Если вы намерены использовать сеть Lightning для отправки и получения средств между узлами и кошельками, которые вы контролируете, и не заинтересованы в маршрутизации платежей других пользователей, то нет особой необходимости сообщать о своих каналах остальной части сети. Вы могли бы открыть канал, скажем, между вашим настольным компьютером, на котором работает полноценный узел, и вашим мобильным телефоном, на котором работает Lightning-кошелек, и просто отказаться от объявления канала, обсуждаемого в главе 3. Иногда такие каналы называют «приватными» каналами; однако правильнее называть их «необъявленными» каналами, поскольку они не являются строго приватными (конфиденциальными).

Необъявленные каналы не будут известны остальной части сети и обычно не будут использоваться для маршрутизирования платежей других пользователей. Они все еще могут использоваться для маршрутизировани платежей, если о них известно другим узлам; например, счет может содержать маршрутизационные подсказки, которые указывают путь с необъявленным каналом. Однако если допустить, что вы открыли необъявленный канал только с самим собой, то вы получаете некоторую степень конфиденциальности. Поскольку вы не выставляете свой канал в сеть, то снижаете риск атаки типа «отказ в обслуживании» на ваш узел. Вы также можете легче управлять емкостью этого канала, поскольку он будет использоваться только для приема или отправки непосредственно на ваш узел.

Есть еще преимущества в открытии необъявленного канала с известной стороной, с которой вы часто совершаете сделки. Например, если Алиса и Боб часто играют в покер на биткойны, то они могли бы открыть канал для отправки своих

выигрышей туда и обратно. В нормальных условиях этот канал не будет использоваться для маршрутизирования платежей от других пользователей или взимания комиссионных. И поскольку канал не будет известен остальной части сети, любые платежи между Алисой и Бобом невозможно будет логически вывести путем отслеживания изменений емкости канала. За счет этого обеспечивается некоторая конфиденциальность Алисе и Бобу; однако если один из них решит сообщить о канале другим пользователям, например включив его в маршрутизационные подсказки счета, тогда эта конфиденциальность будет потеряна.

Следует также отметить, что для открытия необъявленного канала необходимо совершить публичную транзакцию в блочной цепи Bitcoin. Следовательно, можно сделать вывод о существовании и размере канала, если злоумышленник отслеживает блочную цепь на предмет транзакций, открывающих канал, и пытается соотнести их с каналами в сети. Кроме того, когда канал будет закрыт, окончательный остаток канала будет обнародован, как только он будет привязан к блочной цепи Bitcoin. Однако поскольку открывающая и фиксационная транзакции являются псевдонимными, будет непросто соединить их обратно с Алисой или Бобом. Кроме того, обновление Taproot (Стержневой корень) от 2021 года затрудняет различие между транзакциями открытия и закрытия канала и другими конкретными видами Bitcoin-транзакций. Следовательно, хотя необъявленные каналы не являются полностью приватными, они обеспечивают некоторые преимущества конфиденциальности при их осторожном использовании.

Соображения по маршрутизации

Как описано в разделе «Отказ в обслуживании» на стр. 360, узлы, которые открывают публичные каналы, подвергаются риску серии атак на свои каналы. В то время как на уровне протокола разрабатываются соглашения, узел может предпринять целый ряд шагов для защиты от атак типа «отказ в обслуживании» на свои публичные каналы:

Минимальный размер HTLC-контракта

При открытии канала ваш узел может установить минимальный размер HTLC-контракта, который он будет принимать. Установка более высокого значения обеспечивает, чтобы каждый имеющийся у вас слот канала не мог быть занят очень малым платежом.

Ограничение частоты

Многие имплементации узлов позволяют узлам динамически принимать или отклонять HTLC-контракты, которые пересылаются через ваш узел. Ниже приведены некоторые полезные рекомендации для конкретно-прикладного ограничителя скорости:

- ограничивать число фиксационных слотов, которые может использовать один одноранговый узел;
- отслеживать частоту отказов одного однорангового узла и ограничивать частоту, если их сбои внезапно резко возрастают.

Теневые каналы

Узлы, которые хотят открыть крупные каналы для одной цели, могут вместо этого открыть один публичный канал для этой цели и поддерживать его

дополнительными приватными каналами, именуемыми теневыми каналами. Эти каналы по-прежнему могут использоваться для маршрутизации, но не объявляются потенциальным злоумышленникам.

Принятие каналов

В настоящее время узлы Lightning с трудом справляются с самозагрузкой входящей ликвидности. Хотя существуют некоторые платные решения по получению входящей ликвидности, такие как своповые службы, рынки каналов и платные услуги по открытию каналов от известных узлов, многие узлы с радостью примут любой законный запрос на открытие канала, чтобы увеличить свою входящую ликвидность.

Возвращаясь к контексту Bitcoin, это можно сравнить с тем, как ядро Bitcoin по-разному обрабатывает свои входящие и исходящие соединения из опасения, что узел может быть заслонен. Если узел открывает входящее соединение с вашим Bitcoin-узлом, то у вас нет возможности узнать, выбрал ли вас инициатор случайно или же он специально нацелен на ваш узел со злым умыслом. К вашим исходящим соединениям не нужно относиться с таким подозрением, потому что либо узел был выбран случайным образом из пула многих потенциальных одноранговых узлов, либо вы намеренно подсоединились к одноранговому узлу вручную.

То же самое можно сказать и о Lightning. Когда вы открываете канал, это делается с намерением, но когда дистанционная сторона открывает канал для вашего узла, у вас нет возможности узнать, будет этот канал использоваться для атаки на ваш узел или нет. Как отмечается в нескольких статьях, относительно низкая стоимость развертывания узла и открытия каналов для целей является одним из важных факторов, облегчающих атаки. Если вы принимаете входящие каналы, то разумно установить некоторые ограничения на узлы, с которых вы принимаете входящие каналы. Многие имплементации предоставляют перехватчики (зацепки, hook) приемки каналов, которые позволяют адаптировать политику приема каналов к вашим предпочтениям.

Вопрос о принятии и отклонении каналов является философским. Что, если в итоге мы получим сеть Lightning, в которой новые узлы не смогут участвовать, потому что они не смогут открывать какие-либо каналы? Наше предложение состоит не в том, чтобы устанавливать эксклюзивный список «мегахабов», из которых вы будете принимать каналы, а в том, чтобы принимать каналы таким образом, который соответствует вашим предпочтениям в отношении риска.

Некоторые потенциальные стратегии заключаются в следующем:

Никакого риска

Не принимать никаких входящих каналов.

Низкий риск

Принимать каналы из известного набора узлов, с которыми у вас ранее были успешно открыты каналы.

Средний риск

Принимать каналы только от тех узлов, которые присутствуют в графе в течение более длительного периода и имеют некоторые долговечные каналы.

Более высокий риск

Принимать любые входящие каналы и применять меры по смягчению последствий, описанные в разделе «Соображения по маршрутизации» на стр. 370.

Вывод

Подводя итог: конфиденциальность и безопасность – это нюансированные, сложные темы, и хотя многие исследователи и разработчики ищут усовершенствования в масштабах всей сети, важно, чтобы все участники сети понимали, что конкретно они могут сделать для защиты собственной конфиденциальности и повышения безопасности на уровне отдельного узла.

Справочные материалы и дальнейшее чтение

В этой главе мы использовали целый ряд справочных материалов из текущих исследований по безопасности сети Lightning. Указанные полезные работы и статьи, перечисленные по темам, можно найти в следующих ниже списках.

Конфиденциальность и атаки прощупыванием

- *Хорди Эррера-Джоанкомарти и соавт.* О сложности сокрытия остатка каналов сети Lightning (Jordi Herrera-Joancomartí et al., On the Difficulty of Hiding the Balance of Lightning Network Channels. Asia CCS '19: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (July 2019): 602–612.), https://eprint.iacr.org/2019/328.
- *Утц Нисслмюллер и соавт.* Об активных и пассивных атаках на конфиденциальность в автономных сетях криптовалют (Utz Nisslmueller et al., Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-Chain Networks), arXiv preprint (2020), https://arxiv.org/abs/2003.00003.
- *Сергей Тихомиров и соавт.* Прощупывание остатка каналов в сети Lightning (Sergei Tikhomirov et al., Probing Channel Balances in the Lightning Network), arXiv preprint (2020), https://arxiv.org/abs/2004.00333.
- *Джордж Каппос и соавт.* Эмпирический анализ конфиденциальности в сети Lightning (George Kappos et al. An Empirical Analysis of Privacy in the Lightning Network), arXiv preprint (2021), https://arxiv.org/abs/2003.12470.
- Исходный код Zap с функцией прощупывания, https://github.com/LN-Zap/zap-desktop/blob/v0.7.2-beta/services/grpc/router.methods.js.

Атаки переполнением

- *Айелет Мизрахи и Авив Зоар.* Атаки переполнением в сетях платежных каналов (Ayelet Mizrahi and Aviv Zohar. Congestion Attacks in Payment Channel Networks), arXiv preprint (2020), https://arxiv.org/abs/2002.06564.

Соображения по маршрутизации

- *Марти Бент*: интервью с Йостом Ягером, рассказы из крипты, аудиоподкаста (Marty Bent, interview with Joost Jager, Tales from the Crypt, podcast audio, October 2, 2020), https://anchor.fm/tales-from-the-crypt/episodes/197-Joost-Jager-ekghn6.

Глава 17

Заключение

Всего за несколько лет сеть Lightning превратилась из официального документа в быстро растущую глобальную сеть. В качестве второго слоя Bitcoin она оправдала надежды в отношении быстрых, недорогих и приватных платежей. Вдобавок она вызвала цунами инноваций, поскольку освобождает разработчиков от ограничений жестко регламентированного консенсуса, которые существуют при разработке Bitcoin.

Инновации в сети Lightning происходят на нескольких разных уровнях:

- в стержневом протоколе Bitcoin, обеспечивающем использование и спрос на новые операционные коды Bitcoin Script, алгоритмы подписания и оптимизации;
- на уровне протокола Lightning с быстрым развертыванием новых функциональностей по всей сети;
- на уровне платежного канала с новыми конструктами и улучшениями каналов;
- как отдельные функциональности в порядке выбора, развернутые сквозными независимыми имплементациями, которые отправители и получатели могут использовать, если захотят;
- с новыми и захватывающими Lightning-приложениями (LApp), построенными поверх клиентов и протоколов.

Давайте посмотрим, как эти инновации меняют Lightning сейчас и будут менять в ближайшем будущем.

Децентрализованные и асинхронные инновации

Lightning не связана жестко регламентированным консенсусом, как в случае с Bitcoin. Это означает, что разные клиенты Lightning могут имплементировать разные функциональности и согласовывать их взаимодействие (см. раздел «Биты функциональностей и расширяемость протокола» на стр. 325). Как следствие инновации в сети Lightning происходят гораздо быстрее, чем в Bitcoin.

Lightning не только быстро движется вперед, но и создает спрос на новые функциональности в системе Bitcoin. Многие недавние и планируемые инновации в Bitcoin были мотивированы и оправданы их использованием в сети Lightning. Фактически сеть Lightning часто упоминается в качестве примера использования многих новых функциональностей.

Инновации в Bitcoin-протоколе и в Bitcoin Script

Система Bitcoin по необходимости является консервативной системой, которая должна сохранять совместимость с консенсусными правилами, чтобы избегать незапланированных ответвлений блочной цепи или разделов P2P-сети. Как следствие новые функциональности требуют большой координации и тестирования, прежде чем они будут имплементированы в mainnet-сети, реальной производственной системе.

Вот несколько текущих или предлагаемых инноваций в Bitcoin, которые мотивированы вариантами использования в сети Lightning:

Neutrino

Облегченный клиентский протокол с улучшенными функциональными возможностями обеспечения конфиденциальности по сравнению с унаследованным SPV-протоколом. Neutrino в основном используется клиентами Lightning для доступа к блочной цепи Bitcoin.

Подписи Шнорра

Представленные в рамках мягкого ответвления (софт-форка) Taproot, подписи Шнорра позволят создавать гибкие контракты с точечной привязкой ко времени (PTLCS) для строительства каналов в Lightning. Это может, в частности, привести к использованию отзывных подписей вместо отзывных транзакций.

Taproot

Также является частью мягкого ответвления ноября 2021 года, который вводит подписи Шнорра и Taproot (Стержневой корень), позволяет сложным скриптам появляться как платежи с одним плательщиком, с одним получателем и неотличим от наиболее распространенного типа платежей в Bitcoin. Это позволит транзакциям с кооперативным (взаимным) закрытием каналов Lightning выглядеть неотличимыми от простых платежей и повышать конфиденциальность для пользователей LN.

Перепривязка/Rebinding

Также известная под названиями `SIGHAS_NOINPUT` или `SIGHASH_ANYPREVOUT`, эта запланированная модернизация языка Bitcoin Script в первую очередь обусловлена передовыми умными контрактами, такими как канальный протокол eltoo.

Ковенанты

В настоящее время находящиеся на ранних стадиях исследований, ковенанты (особые соглашения) позволяют транзакциям создавать выходы, которые ограничивают будущие расходующие их транзакции. Этот механизм мог бы повысить безопасность каналов Lightning, сделав возможным принудительное внесение адресов в белый список в фиксационных транзакциях.

Инновация в протоколе Lightning

P2P-протокол Lightning обладает высокой степенью расширяемости и претерпел целый ряд изменений с момента своего создания. Правило «Быть нечетным – нормально», используемое в битах функциональностей (см. раздел

«Биты функциональностей и расширяемость протокола» на стр. 325), обеспечивает узлам возможность согласовывать поддерживаемые ими функциональности, позволяя выполнять несколько независимых модернизаций протокола.

Расширяемость TLV

Механизм «Тип–длина–значение» (см. раздел «Формат Тип–длина–значение» на стр. 323), служащий для расширения протокола обмена сообщениями, является чрезвычайно мощным и уже позволил внедрить несколько новых возможностей в Lightning, сохраняя при этом как прямую, так и обратную совместимость. Ярким примером, который в настоящее время разрабатывается и его использует, является ослепление пути и батутные платежи. Он позволяет получателю скрываться от отправителя, но также дает возможность мобильным клиентам отправлять платежи без необходимости сохранения полного канального графа на своих устройствах, используя третью сторону, которой им не нужно раскрывать конечного получателя.

Строительство платежного канала

Платежные каналы – это абстракция, которой оперируют два партнера по каналу. До тех пор, пока эти двое готовы выполнять новый код, они могут одновременно имплементировать разнообразные канальные механизмы. Фактически недавние исследования показывают, что каналы можно даже динамически модернизировать до нового механизма, не закрывая старый канал и не открывая новый тип канала.

eltoo

Предлагаемый канальный механизм, который использует перепривязку входа, значительно упрощает работу платежных каналов и устраняет необходимость в механизме штрафных санкций. Прежде чем его можно будет имплементировать, ему нужен новый тип Bitcoin-подписи.

Сквозные функциональности в порядке выбора

Контракты с точечной привязкой ко времени

Другой подход к HTLC-контрактам, PTLC-контракты могут повышать конфиденциальность, уменьшать утечку информации на промежуточные узлы и работать эффективнее, чем каналы на основе HTLC-контракта.

Крупные каналы

Крупные, или Wumbo-, каналы были динамично введены в сеть, не требуя координации. Каналы, поддерживающие крупные платежи, рекламируются как часть сообщений об объявлении канала и могут использоваться в порядке выбора.

Многокомпонентные платежи (MPP)

MPP-платеж также был введен в порядке выбора, но что еще лучше, требуется, чтобы отправитель и получатель платежа могли выполнять MPP-платеж. Остальная часть сети просто направляет HTLC-контракты, как если бы они были однокомпонентными платежами.

JIT-маршрутизация

Опциональный метод, который может использоваться маршрутизационными узлами для повышения их надежности и защиты от рассылки спама.

Keysend

Модернизация, независимо введенная клиентскими имплементациями Lightning, позволяет отправителю отправлять деньги «незапрошенным» и асинхронным способом, не требуя сначала выставлять счет.

Счета HODL[105]

Платежи, при которых окончательный HTLC-контракт не взимается, привязывая отправителя к платежу, но позволяя получателю отложить взимание до тех пор, пока не будет соблюдено какое-либо другое условие, или отменить счет без взимания. Он также был имплементирован независимо разными клиентами Lightning и может использоваться в порядке выбора.

Службы луковичных маршрутизируемых сообщений

Механизм луковичной маршрутизации и опорная база данных публичных ключей узлов могут использоваться для отправки данных, не связанных с платежами, таких как текстовые сообщения или сообщения на форуме. Использование Lightning для задействования платных сообщений в качестве решения для спамных постов и Сивилловых атак (спама) – еще одно новшество, которое было имплементировано независимо от стержневого протокола.

Offers/Предложения

В настоящее время предлагаемый как BOLT #12, но уже имплементированный некоторыми узлами, это протокол связи для запроса (повторяющихся) счетов с дистанционных узлов через луковичные сообщения.

Lightning-приложения (LApps)

Хотя они все еще находятся в зачаточном состоянии, мы уже наблюдаем появление интересных приложений, которые можно выделить. В широком смысле определяемые как приложение, использующее протокол Lightning или клиент Lightning в качестве компонента, LApp являются прикладным слоем Lightning. Ярким примером является приложение LNURL, которое предоставляет функциональность, аналогичную той, которая предлагается в спецификации BOLT #12, но только по адресам HTTP и Lightning. Оно работает поверх предложений, предоставляя пользователям адрес электронной почты, на который другие могут отправлять средства, пока программное обеспечение в фоновом режиме запрашивает счет у конечной точки узла LNURL. Создаются дополнительные LApp для простых игр, приложений для обмена сообщениями, микрослужб, платных API, платных диспенсеров (например, топливных насосов), систем торговли деривативами и многого другого.

[105] Слово HODL происходит от взволнованного неправильного произнесения слова «HOLD», выкрикнутого на форуме, чтобы побудить людей не продавать биткойн в панике.

На старт, внимание, марш!

Будущее выглядит светлым. Сеть Lightning выводит Bitcoin на новые неизведанные рынки и приложения. Вооружившись знаниями, изложенными в этой книге, вы сможете разведать этот новый рубеж или, возможно, даже присоединиться к нему в качестве первопроходца и проложить новый путь.

Приложение А

Обзор основных принципов системы Bitcoin

Сеть Lightning способна работать поверх нескольких блочных цепей, но в первую очередь заякорена на систему Bitcoin. В целях понимания сети Lightning вам необходимо понимание основных принципов системы Bitcoin и ее строительных блоков.

Вы можете использовать целый ряд неплохих ресурсов, чтобы узнать о Bitcoin больше, включая сопутствующую книгу Андреаса М. Антонопулоса «Освоение системы Bitcoin», 2-е изд., которую можно найти на GitHub под лицензией с открытым исходным кодом[106]. При этом вам не придется читать еще одну, чтобы быть готовым к этой!

В данной главе мы собрали наиболее важные понятия, которые вам нужно знать о системе Bitcoin, и объяснили их в контексте сети Lightning. Благодаря этому вы сможете узнать именно то, что вам нужно, чтобы разобрастся в сети Lightning, не отвлекаясь ни на что другое.

В этом приложении рассматривается несколько важных понятий системы Bitcoin, в том числе:

- ключи и цифровые подписи;
- хеш-функции;
- транзакции Bitcoin и их структура;
- выстраивание транзакций Bitcoin в цепь;
- выходные точки транзакций;
- Bitcoin Script: привязывающие и отвязывающие скрипты;
- базовые привязывающие скрипты;
- сложные и условные привязывающие скрипты;
- привязки ко времени.

Ключи и цифровые подписи

Возможно, вы слышали, что биткойн основан на *криптографии*, разделе математики, широко используемом в компьютерной безопасности. Криптография также может использоваться для доказательства знания секрета без раскрытия этого секрета (цифровая подпись) или подтверждения подлинности данных (цифровой отпечаток). Эти типы криптографических доказательств являются

[106] См. https://github.com/bitcoinbook/bitcoinbook.

математическими инструментами, критически важными для системы Bitcoin, и широко используются в Bitcoin-приложениях.

Владение биткойном устанавливается посредством *цифровых ключей*, *биткойновых адресов* и *цифровых подписей*. Цифровые ключи на самом деле не хранятся в сети, а вместо этого создаются и хранятся пользователями в файле или простой базе данных, именуемой *кошельком*. Цифровые ключи в кошельке пользователя полностью независимы от протокола Bitcoin и могут генерироваться и управляться программным обеспечением кошелька пользователя, не ссылаясь на блочную цепь и не обращаясь к интернету.

Большинство транзакций Bitcoin требуют включения в блочную цепь валидной цифровой подписи, которая может быть сгенерирована только с помощью секретного ключа; поэтому любой, у кого есть копия этого ключа, имеет контроль над биткойном. Цифровая подпись, используемая для расходования средств, также называется *свидетелем* – данный термин широко используется в криптографии. Свидетельские данные в биткойновой транзакции говорят об истинном владении расходуемыми средствами. Ключи существуют парами, состоящими из приватного (секретного, закрытого) ключа и публичного (открытого) ключа. Думайте о публичном ключе как о номере банковского счета, а о приватном ключе – как о секретном PIN-коде.

Приватные и публичные ключи

Приватный ключ – это просто число, взятое случайным образом. На практике, чтобы упростить управление многими ключами, большинство кошельков генерируют последовательность приватных ключей из одного случайного начального числа (seed) с использованием детерминированного алгоритма выведения чисел. Проще говоря, одно случайное число используется для продуцирования повторяемой последовательности чисел, кажущихся случайными, которые используются в качестве приватных ключей. Это позволяет пользователям создавать резервные копии только начального значения и иметь возможность извлекать все необходимые ключи из этого начального значения.

Bitcoin, как и многие другие криптовалюты и блочные цепи, для обеспечения безопасности использует *эллиптические кривые*. В системе Bitcoin умножение эллиптической кривой на эллиптическую кривую *secp256k1* используется как *однопутная функция*. Проще говоря, природа математики эллиптических кривых делает тривиальным вычисление скалярного умножения точки, но невозможным вычисление обратного (деление или дискретный логарифм).

Каждый приватный ключ имеет соответствующий *публичный ключ*, который вычисляется из приватного ключа с использованием скалярного умножения на эллиптической кривой. Проще говоря, имея приватный ключ k, можно умножить его на константу G, чтобы получить публичный ключ K:

$$K = k*G.$$

Выполнить этот расчет в противоположную сторону невозможно. Имея публичный ключ K, нельзя вычислить приватный ключ k. Поделить на G в математике эллиптических кривых невозможно. Вместо этого нужно было бы попробовать все вероятные значения k в исчерпывающем процессе, именуемом *атакой грубой силой*. Поскольку k – это 256-битовое число, исчерпание всех

возможных значений с помощью любого классического компьютера потребовало бы больше времени и энергии, чем доступно в нашей Вселенной.

Хеши

Еще одним важным инструментом, широко используемым в системе Bitcoin и в сети Lightning, являются *криптографические хеш-функции*, в частности хеш-функция SHA-256.

Хеш-функция, т. н. дайджестная функция, – это функция, которая принимает данные произвольной длины и преобразовывает их в результат фиксированной длины, именуемый *хешем*, *дайджестом* или *отпечатком* (см. рис. A-1). Важно отметить, что хеш-функции являются *однопутными* функциями, а это означает, что невозможно вычислить их в обратную сторону и просчитать входные данные по отпечатку.

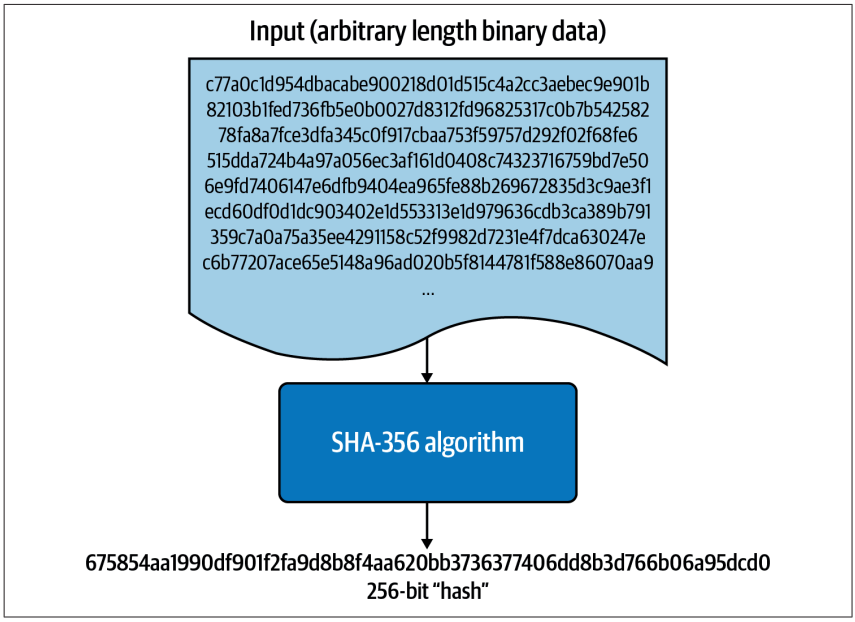


Рис. A-1. Криптографический алгоритм хеширования SHA-256

Например, если использовать терминал командной строки для подачи текста «Mastering the Lightning Network» (Освоение сети Lightning) в функцию SHA-256, то она произведет следующий ниже отпечаток:

```
$ echo -n "Mastering the Lightning Network" | shasum -a 256

ce86e4cd423d80d054b387aca23c02f5fc53b14be4f8d3ef14c089422b2235de -
```

Данные на входе в функцию, используемые для вычисления хеша, также называются прообразом (preimage).

Конечно же, длина данных на входе может быть намного больше. Давайте попробуем то же самое с PDF-файлом технического документа системы Bitcoin от Сатоши Накамото:

```
$ wget http://bitcoin.org/bitcoin.pdf
$ cat bitcoin.pdf | shasum -a 256

b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553 -
```

Хотя это и занимает больше времени, чем для одного предложения, функция SHA-256 обрабатывает 9-страничный PDF-файл, «переваривая» его в 256-битовый отпечаток.

Теперь, на этом этапе, вам, возможно, будет интересно знать, как функция, которая обрабатывает данные неограниченного размера, может создавать уникальный отпечаток, представляющий собой число фиксированного размера.

Теоретически, поскольку существует бесконечное число возможных прообразов (входов) и только конечное число отпечатков, должно быть много прообразов, которые создают один и тот же 256-битовый отпечаток. Когда два прообраза создают один и тот же хеш, это называется *коллизией*.

На практике 256-битовое число настолько велико, что вы никогда не обнаружите столкновение преднамеренно. Криптографические хеш-функции работают, исходя из того, что поиск коллизии – это попытка на основе грубой силы, которая отнимает столько много энергии и времени, что практически невозможна.

Криптографические хеш-функции широко используются в различных приложениях, поскольку они обладают некоторыми полезными функциональными свойствами. Они:

детерминированы.

Один и тот же вход всегда выдает один и тот же хеш;

необратимы.

Невозможно вычислить прообраз хеша;

защищены от коллизий.

Вычислительно невозможно найти два сообщения с одинаковым хешем;

некоррелированы.

Небольшое изменение входа производит такое значительное изменение выхода, что выход кажется некоррелированным со входом;

равномерны/случайны.

Криптографическая хеш-функция создает хеши, которые равномерно распределены по всему 256-битовому пространству возможных выходов. Выходы хеш-функции кажутся случайными, хотя на самом деле они таковыми не являются.

Используя эти свойства криптографических хешей, можно построить несколько интересных приложений:

отпечатки.

Хеш можно использовать для взятия отпечатка файла или сообщения, чтобы иметь возможность однозначно их идентифицировать. Хеши могут использоваться в качестве универсальных идентификаторов любого набора данных;

доказательство целостности.

Отпечаток файла или сообщения демонстрирует его целостность, поскольку файл или сообщение нельзя подделать или изменить каким-либо образом без изменения отпечатка. Это часто используется для обеспечения неподдельности программного обеспечения перед его инсталляцией на ваш компьютер;

обязательство/отсутствие возможности отказа.

Можно зафиксировать конкретный прообраз (например, число или сообщение), не раскрывая его, опубликовав его хеш. Позже можно раскрыть секрет, и каждый сможет убедиться, что это то же самое, что было зафиксировано ранее, потому что оно производит опубликованный хеш;

доказательство работы / измельчение хеша.

Хеш можно использовать, чтобы доказать, что вычислительная работа была выполнена, показав неслучайную закономерность в хеше, которую можно произвести только путем повторных догадок на прообразе. Например, хеш заголовка блока Bitcoin начинается с большого числа нулевых битов. Единственный способ его произвести – это изменить часть заголовка и хешировать его триллионы раз до тех пор, пока он случайно не произведет эту закономерность;

атомарность.

Секретный прообраз можно сделать обязательным условием для расходования средств в нескольких связанных транзакциях. Если какая-либо из сторон раскрывает прообраз, чтобы провести одну из транзакций, то все остальные стороны теперь тоже могут удерживать свои транзакции. Расходуемым становится все или ничего, достигая атомарности в нескольких транзакциях.

Цифровые подписи

Приватный ключ используется для создания подписей, которые необходимы, чтобы расходовать биткойн, подтвердив владение используемыми в транзакции средствами.

Цифровая подпись – это число, которое вычисляется на основе применения приватного ключа к конкретному сообщению.

Имея сообщение m и приватный ключ k, функция подписи F_{sign} может произвести подпись S:

$$S = F_{sign}(m, k).$$

Полученная подпись S может быть независимо верифицирована любым, у кого есть публичный ключ K (соответствующий приватному ключу k), и сообщение:

$$F_{verify}(m, K, S).$$

Если F_{verify} возвращает истинный результат, то верификатор может подтвердить, что сообщение m было подписано кем-то, кто имел доступ к приватному ключу k. Важно отметить, что цифровая подпись подтверждает владение приватным ключом k во время подписания, не раскрывая k.

Цифровые подписи используют криптографический алгоритм хеширования. Подпись применяется к хешу сообщения, в результате чего сообщение m «резюмируется» в хеш фиксированной длины H(m), который служит отпечатком.

Применяя цифровую подпись к хешу транзакции, подпись не только подтверждает авторизацию, но и «привязывает» данные транзакции, обеспечивая их целостность. Подписанная транзакция не может быть модифицирована, поскольку любое изменение приведет к другому хешу и сделает подпись невалидной.

Типы подписей

Подписи не всегда применяются ко всей транзакции. В целях обеспечения подписанию гибкости цифровая подпись Bitcoin содержит префикс, именуемый типом подписного хеша, который указывает, какая часть данных транзакции включена в хеш. Это позволяет подписи фиксировать или «привязывать» все или только некоторые данные в транзакции. Наиболее распространенным типом подписного хеша является `SIGHASH_ALL`, который привязывает в транзакции все, включая все транзакционные данные в подписанном хеше. Для сравнения, `SIGHASH_SINGLE` привязывает все входы транзакции, но только один выход (подробнее о входах и выходах – в следующем разделе). Разные типы подписного хеша можно совмещать для продуцирования шести разных «шаблонов» транзакционных данных, которые привязываются подписью.

Более подробную информацию о типах подписного хеша можно найти в разделе «Типы подписного хеша» в главе 6 книги «Освоение системы Bitcoin», 2-е изд.[107]

Транзакции Bitcoin

Транзакции – это структуры данных, которые кодируют передачу стоимости между участниками системы Bitcoin.

Входы и выходы

Основополагающим строительным блоком биткойновой транзакции является ее выход. *Выходы транзакций* представляют собой неделимые фрагменты биткойновой валюты, зарегистрированные в блочной цепи и признанные валидными всей сетью. Транзакция расходует входы и создает выходы. Входы транзакции – это просто ссылки на выходы ранее зарегистрированных транзакций. Благодаря этому каждая транзакция расходует выходы предыдущих транзакций и создает новые выходы (см. рис. А-2).

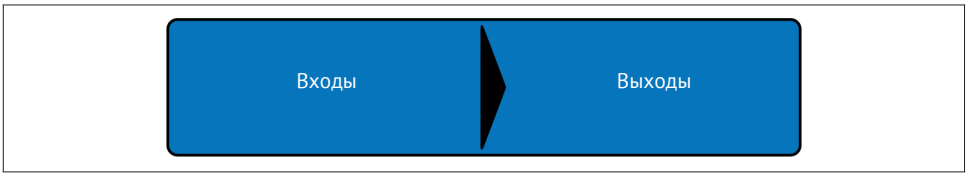


Рис. А-2. Транзакция переводит стоимость из входа в выход

[107] См. https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch06.asciidoc#sighash_types.

Полные узлы Bitcoin отслеживают все доступные и расходуемые выходы, именуемые неизрасходованными транзакционными выходами (unspent transaction outputs, аббр. UTXO). Коллекция всех UTXO называется множеством UTXO, которое в настоящее время насчитывает миллионы UTXO. Множество UTXO увеличивается по мере создания новых UTXO и сжимается при потреблении UTXO. Каждая транзакция представляет собой изменение (переход состояния) в множестве UTXO путем потребления одного или нескольких UTXO в качестве входов транзакций и создания одного или нескольких UTXO в качестве выходов транзакций.

Например, давайте допустим, что у пользователя Алиса есть 100 000 сатоши UTXO, которые она может потратить. Алиса может заплатить Бобу 100 000 сатоши, создав транзакцию с одним входом (потребив свой существующий вход в размере 100 000 сатоши) и одним выходом, который «оплатит» Бобу 100 000 сатоши. Теперь у Боба есть 100 000 сатоши UTXO, и он их может потратить, создав новую транзакцию, которая потребляет этот новый UTXO и тратит его на другой UTXO в качестве платежа другому пользователю и т. д. (см. рис. A-3).

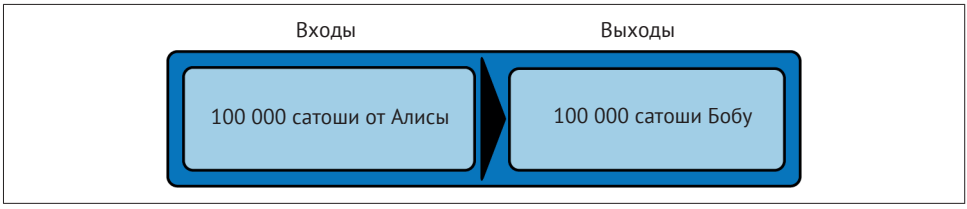


Рис. A-3. Алиса платит Бобу 100 000 сатоши

Выходы транзакции могут иметь произвольное (целочисленное) значение, выраженное в сатоши. Точно так же, как доллары, которые можно разделить с точностью до двух знаков после запятой в виде центов, биткойн можно разделить с точностью до восьми знаков после запятой в виде сатоши. Хотя выходы могут иметь любое произвольное значение, после создания они становятся *неделимыми*. Это важная характеристика выходов, которую необходимо подчеркнуть: выходы представляют собой дискретные и неделимые единицы стоимости, выраженные в целочисленных сатоши. Неизрасходованный выход может быть потреблен транзакцией только полностью.

Что делать, если Алиса захочет заплатить Бобу 50 000 сатоши, но у нее есть только неделимые 100 000 сатоши UTXO? Алисе нужно будет создать транзакцию, которая потребляет (в качестве входа) 100 000 сатоши UTXO и имеет два выхода: один выплачивает 50 000 сатоши Бобу, а другой выплачивает 50 000 сатоши обратно Алисе в качестве «сдачи» (см. рис. A-4).

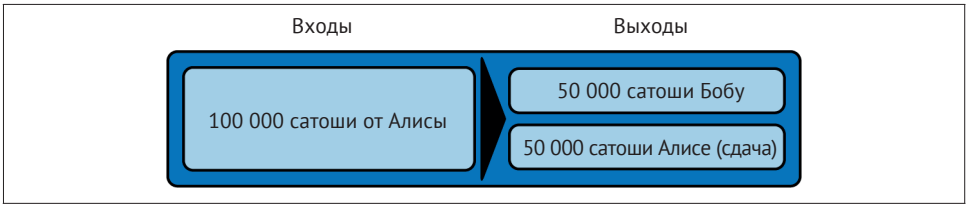


Рис. A-4. Алиса платит 50k сатоши Бобу и 50k сатоши себе в качестве сдачи

В сдаче на выходе нет ничего особенного или какого-либо способа отличить ее от любого другого выхода. Этот выход не обязательно должен быть последним. Может быть несколько выходов со сдачей или никаких выходов со сдачей. Только создатель транзакции знает, какие выходы предназначены для других, а какие – для адресов, которыми он владеет, и, следовательно, куда «выдаются» остатки.

Точно так же, если Алиса захочет заплатить Бобу 85 000 сатоши, но у нее есть два доступных UTXO по 50 000 сатоши, то она должна создать транзакцию с двумя входами (потребляя оба своих 50 000 сатоши UTXO), и два выхода, заплатив Бобу 85 000 и отправив 15 000 сатоши обратно себе в качестве сдачи (см. рис. A-5).

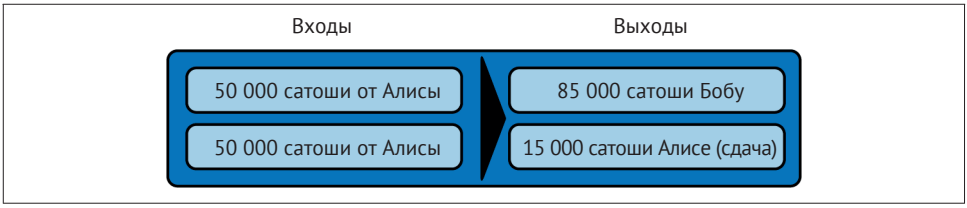


Рис. A-5. Алиса использует два входа по 50k сатоши, чтобы заплатить 85k сатоши Бобу и 15k сатоши себе в качестве сдачи

Приведенные выше иллюстрации и примеры показывают, как транзакция Bitcoin совмещает (расходует) один или несколько входов и создает один или несколько выходов. Транзакция может иметь сотни или даже тысячи входов и выходов.

Хотя создаваемые сетью Lightning транзакции имеют многочисленные выходы, они не имеют «сдач» как таковых, поскольку весь имеющийся остаток канала делится между двумя партнерами по каналу.

Транзакционные цепочки

Каждый выход может быть использован в качестве входа в последующую транзакцию. Так, например, если Боб решил потратить 10 000 сатоши на транзакцию, заплатив Чану, а Чан потратил 4 000 сатоши, чтобы заплатить Дине, то все будет выглядеть так, как показано на рис. A-6.

Выход считается потраченным, если на него ссылаются как на вход в другую транзакцию, которая зарегистрирована в блочной цепи. Выход считается неизрасходованным (и доступным для расходования), если на него не ссылается ни одна зарегистрированная транзакция.

Единственный тип транзакции, который не имеет входа, – это специальная транзакция, создаваемая майнерами в системе Bitcoin, именуемая *базово-монетарной транзакцией*, или coinbase-транзакцией. Базово-монетарная транзакция имеет только выходы и никаких входов, потому что она создает новый биткойн в результате майнинга. Людая другая транзакция использует

один или несколько ранее зарегистрированных выходов в качестве своих входов.

Поскольку транзакции выстраиваются в цепь, если выбрать транзакцию случайным образом, то можно проследить любой ее вход назад к предыдущей транзакции, которая ее создала. Если вы будете продолжать в том же духе, то в конечном итоге достигнете базово-монетарной транзакции, где биткойн был добыт впервые.

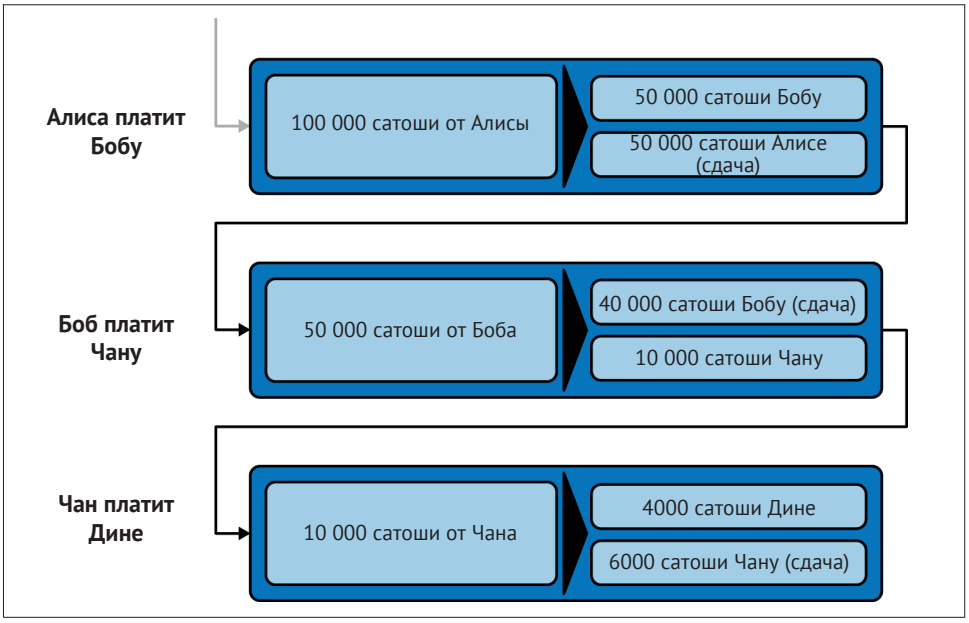


Рис. А-6. Алиса платит Бобу, который платит Чану, который платит Дине

TxID: идентификаторы транзакций

Каждая транзакция в системе Bitcoin идентифицируется уникальным идентификатором (при условии существования BIP-0030), именуемым *ИД транзакции*, или сокращенно *TxID*. В целях генерирования уникального идентификатора используется криптографическая функция хеширования SHA-256 для создания хеша транзакционных данных. Их «отпечаток» служит универсальным идентификатором. На транзакцию можно ссылаться по ее ИД, и как только транзакция регистрируется в блочной цепи Bitcoin, каждый узел в сети Bitcoin знает, что эта транзакция валидна.

Например, ИД транзакции может выглядеть следующим образом:

e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c

Это реально существующая транзакция (созданная в качестве примера для книги «Освоение системы Bitcoin»), которую можно найти в блочной цепи Bitcoin. Попробуйте ее найти, введя приведенный ниже TxID в проводник блоков:

https://blockstream.info/tx/e31e4e214c3f436937c74b8663b3ca58f7ad5b3fce7783eb84fd9a5ee5b9a54c

либо используйте короткую ссылку (с учетом регистра):

http://bit.ly/AliceTx

Выходные точки: выходные идентификаторы

Поскольку каждая транзакция имеет уникальный ИД, мы также можем однозначно идентифицировать выход транзакции в рамках этой транзакции по ссылке на TxID и номер выходного индекса. Первый выход в транзакции – это выходной индекс 0, второй выход – выходной индекс 1 и т. д. Выходной идентификатор общепринято называть *выходной точкой* (outpoint).

По традиции мы записываем выходную точку в виде TxID, двоеточия и номера выходного индекса:

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18:0
```

Выходные идентификаторы (выходные точки) – это механизмы, которые выстраивают транзакции в цепь. Каждый вход в транзакцию является ссылкой на определенный выход из предыдущей транзакции. Эта ссылка является выходной точкой: TxID и номером выходного индекса. Таким образом, транзакция «расходует» конкретный выход (по номеру индекса) из конкретной транзакции (по TxID), чтобы создать новые выходы, которые сами по себе могут быть потрачены по ссылке на выходную точку.

На рис. А-7 представлена цепь транзакций от Алисы к Бобу, от Чана к Дине, на этот раз с выходными точками в каждом входе.

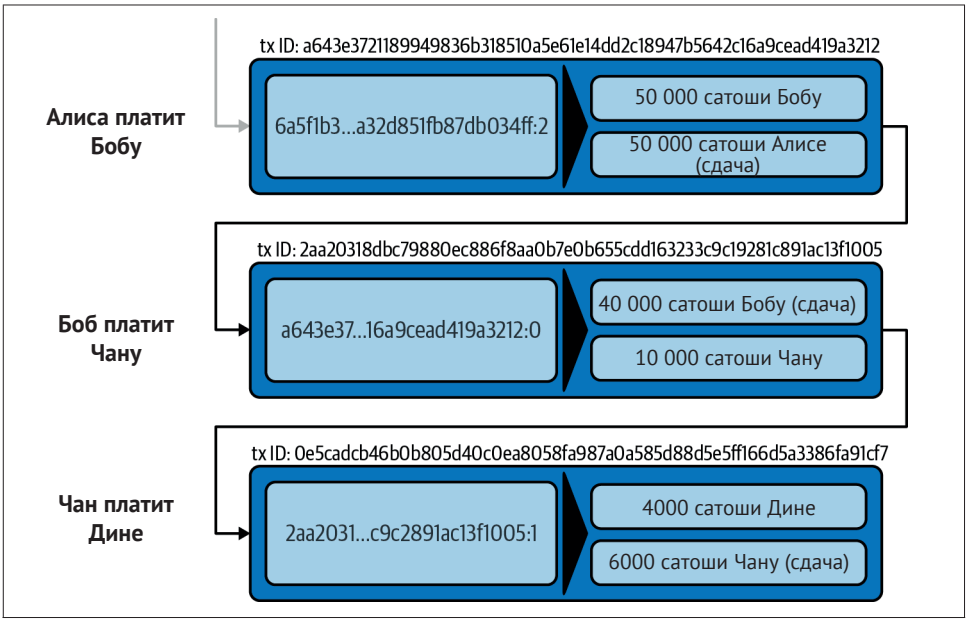


Рис. А-7. Входы транзакции ссылаются на конечные точки, образуя цепь

Вход в транзакции Боба ссылается на транзакцию Алисы (по TxID) и выход с индексом 0.

Вход в транзакции Чана ссылается на TxID транзакции Боба и первый индексированный выход, потому что платеж Чана является выходом под #1. В платеже Боба Чану сдача Боба является выходом под #0[108].

Теперь, если посмотреть на платеж Алисы Бобу, то можно увидеть, что Алиса расходует выходную точку, которая была третьей (выходной индекс # 2) в транзакции, идентификатор которой равен 6a5f1b3[...]. Мы не видим эту транзакцию, на которую ссылается диаграмма, но можем вывести эти детали из выходной точки.

Bitcoin Script

Последним элементом системы Bitcoin, необходимым для нашего полного понимания, является скриптовый язык, который управляет доступом к выходным точкам. До сих пор мы упрощали описание, говоря: «Алиса подписывает транзакцию, чтобы заплатить Бобу». Однако за кулисами существует некоторая скрытая сложность, которая позволяет имплементировать более сложные условия расходования. Самое простое и наиболее распространенное условие расходования – «предоставить подпись, совпадающую со следующим публичным ключом». Подобного рода условие расходования регистрируется в каждом выходе в виде *привязывающего скрипта*, написанного на скриптовом языке, именуемом Bitcoin Script.

Bitcoin Script – это чрезвычайно простой скриптовый язык на основе стека. Он не содержит ни циклов, ни рекурсии и, следовательно, является неполным по Тьюрингу (то есть он не может выражать произвольную сложность и имеет предсказуемое исполнение). Те, кто знаком с (ныне древним) языком программирования FORTH, узнают синтаксис и стиль.

Работа языка Bitcoin Script

Проще говоря, система Bitcoin вычисляет исходный код Bitcoin Script, исполняя скрипт в стеке; если конечный результат является истинным, `TRUE`, то она считает условие расходования удовлетворенным, а транзакцию валидной.

Давайте рассмотрим очень простой пример исходного кода Bitcoin Script, который складывает числа 2 и 3, а затем сравнивает результат с числом 5:

```
2 3 ADD 5 EQUAL
```

На рис. А-8 мы видим, как этот скрипт исполняется (слева направо).

[108] Напомним, что сдача не обязательно должна быть последним выходом транзакции и фактически неотличима от других выходов.

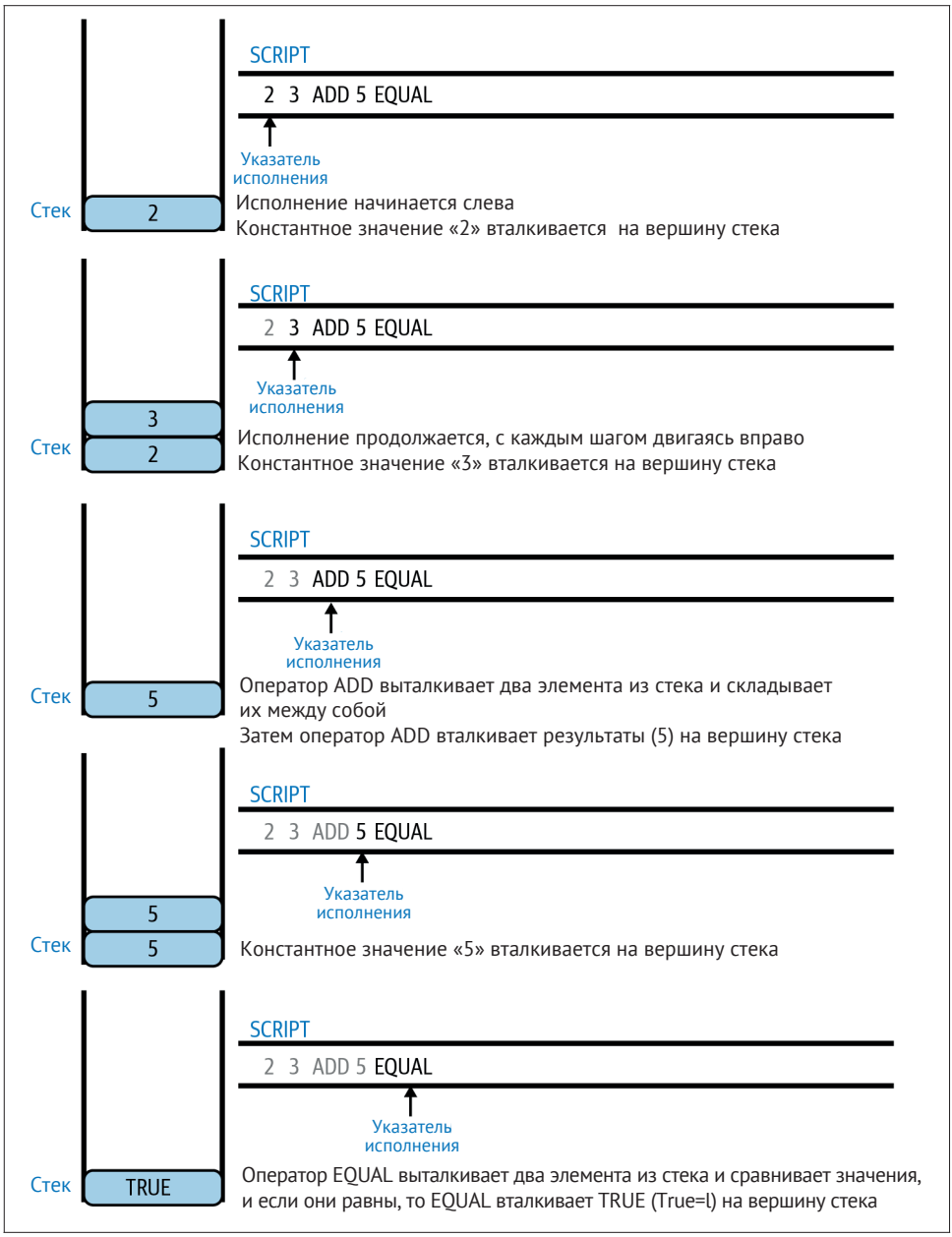


Рис. А-8. Пример исполнения исходного кода Bitcoin Script

Привязывающие и отвязывающие скрипты

Исходный код Bitcoin Script состоит из двух частей:

привязывающие скрипты.

Они встроены в выходы транзакции, устанавливая условия, которые должны быть удовлетворены, чтобы израсходовать эти выходы. Например, кошелек Алисы добавляет привязывающий скрипт в выход оплаты Бобу, который устанавливает условие, что для расходования платежа требуется подпись Боба;

отвязывающие скрипты.

Они встроены во входы транзакции, выполняя условия, установленные привязывающим скриптом выхода, на которые есть ссылка. Например, Боб может отпереть упомянутый выше выход, предоставив отвязывающий скрипт, содержащий цифровую подпись.

Используя упрощенную модель, при валидации отвязывающий скрипт и привязывающий скрипт конкатенируются и исполняются (исключениями являются P2SH и SegWit). Например, если кто-то привязал выход транзакции привязывающим скриптом "3 ADD 5 EQUAL", то мы могли бы его израсходовать с помощью отвязывающего скрипта "2" во входе в транзакцию. Любой, кто будет валидировать эту транзакцию, сделает конкатенацию нашего отвязывающего (2) и привязывающего скриптов (3 ADD 5 EQUAL) и пропустит результат через механизм исполнения исходного кода Bitcoin Script. Будет получено TRUE, и мы сможем израсходовать выход.

Очевидно, что этот упрощенный пример был бы очень плохим вариантом выбора для привязывания фактического результата в системе Bitcoin, потому что в нем нет секрета, только базовая арифметика. Любой желающий сможет израсходовать выход, предоставив ответ "2". Поэтому большинство привязывающих скриптов требуют демонстрации знания секрета.

Привязывание к публичному ключу (подписи)

Самая простая форма привязывающего скрипта требует подпись. Давайте рассмотрим транзакцию Алисы, которая платит Бобу 50 000 сатоши. Выход, который Алиса создает для оплаты Бобу, будет содержать привязывающий скрипт, требующий подписи Боба, и будет выглядеть следующим образом:

```
<Bob Public Key> CHECKSIG
```

Оператор `CHECKSIG` берет два элемента из стека: подпись и публичный ключ. Как вы видите, публичный ключ Боба находится в привязывающем скрипте, поэтому не хватает только подписи, соответствующей этому публичному ключу. Этот привязывающий скрипт может быть использован только Бобом, потому что лишь у Боба есть соответствующий приватный ключ, необходимый для генерирования цифровой подписи, соответствующей публичному ключу.

Для отвязывания этого привязывающего скрипта Боб предоставит отвязывающий скрипт, содержащий только его цифровую подпись:

```
<Bob Signature>
```

На рис. A-9 вы видите привязывающий скрипт в транзакции Алисы (в выходе, который выполняет оплату Бобу) и отвязывающий скрипт (во входе, который расходует этот выход) в транзакции Боба.

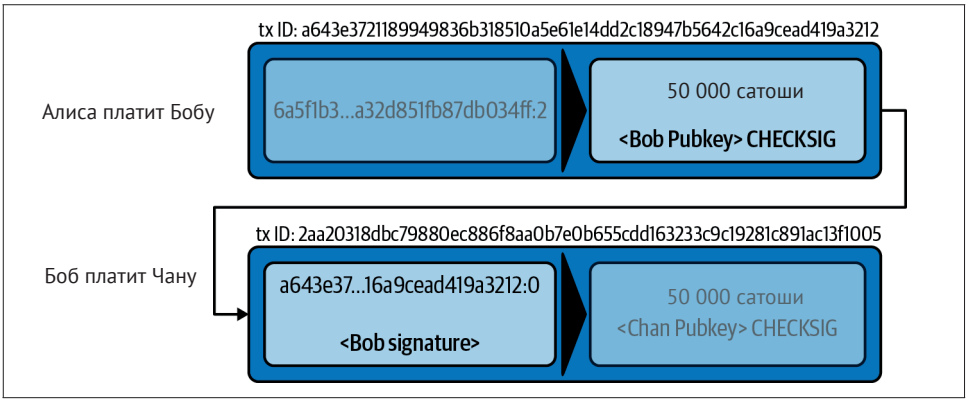


Рис. A-9. Цепь транзакций, показывающая привязывающий скрипт (выход) и отвязывающий скрипт (вход)

Для валидирования транзакции Боба узел Bitcoin должен выполнить следующее.

1. Извлечь отвязывающий скрипт из входа (`<Bob Signature>`).
2. Найти выходную точку, которую он пытается израсходовать (`a643e37...3213:0`). Это транзакция Алисы, и она будет найдена в блочной цепи.
3. Извлечь из этой выходной точки привязывающий скрипт (`<Bob pubkey> CHECKSIG`).
4. Конкатенировать в единый скрипт, поместив отвязывающий скрипт перед привязывающим скриптом (`<Bob Signature> <Bob PubKey> CHECKSIG`).
5. Исполнить этот скрипт в механизме исполнения исходного кода Bitcoin Script, чтобы увидеть, какой выход будет получен.
6. Если выходом будет `TRUE`, то сделать вывод о том, что транзакция Боба валидна, поскольку она смогла выполнить условие расходования, чтобы израсходовать эту выходную точку.

Привязывание к хешу (секрету)

Еще одним типом привязывающего скрипта, который используется в сети Lightning, является *привязка к хешу*. В целях его отвязывания необходимо знать секретный *прообраз* хеша.

Для того чтобы это продемонстрировать, давайте попросим Боба сгенерировать случайное число R и сохранить его в секрете:

```
R = 1833462189
```

Теперь Боб вычисляет хеш SHA-256 этого числа:

```
H = SHA256(R) =>
H = SHA256(1833462189) =>
```

```
H = 0ffd8bea4abdb0deafd6f2a8ad7941c13256a19248a7b0612407379e1460036a
```

Боб передает Алисе хеш H, который мы рассчитали ранее, но сохраняет число R в секрете. Напомним, что из-за свойств криптографических хешей Алиса не может «обратить назад» вычисление хеша и угадать число R.

Алиса создает выход, выплачивая 50 000 сатоши с помощью привязывающего скрипта:

```
HASH256 H EQUAL
```

где H – это фактическое значение хеша (`0ffd8...036a`), которое Боб передал Алисе.

Давайте объясним этот скрипт.

Оператор HASH256 извлекает значение из стека и вычисляет хеш SHA-256 этого значения. Затем он вталкивает результат в стек.

Значение H вталкивается в стек, а затем оператор EQUAL выполняет проверку этих двух значений на совпадение и вталкивает в стек соответственно TRUE либо FALSE.

Следовательно, этот привязывающий скрипт будет работать только в том случае, если он совмещен с отвязывающим скриптом, который содержит R, чтобы при их совмещении мы имели:

```
R HASH256 H EQUAL
```

Только Боб знает R, поэтому лишь Боб может произвести транзакцию с привязывающим скриптом, раскрывающим секретное значение R.

Интересно, что Боб может отдать значение R любому другому, кто затем сможет израсходовать этот биткойн. Это делает секретное значение R почти похожим на биткойновый «ваучер», поскольку любой, у кого он есть, может потратить созданный Алисой выход. Мы увидим всю полезность этого свойства для сети Lightning!

Мультиподписные скрипты

Язык Bitcoin Script предоставляет мультиподписной строительный блок (примитив), который можно использовать для служб условного депонирования и сложных конфигураций владения между несколькими участвующими сторонами. Схема, требующая для расходования биткойнов нескольких подписей, называется *мультиподписной схемой*, далее обозначаемой как схема *K из N*, где:

- N – это суммарное число подписантов, идентифицированных в мультиподписной схеме, и
- K – это *кворум*, или *порог*: минимальное число подписей для авторизации расходовования.

Скрипт для мультиподписи K из N таков:

```
K <PubKey1> <PubKey2> ... <PubKeyN> N CHECKMULTISIG
```

где N – это суммарное число перечисленных в списке публичных ключей (от публичного ключа 1 до публичного ключа N), а K – порог необходимых подписей для расходования выхода.

Для строительства платежного канала сеть Lightning использует мультиподписную схему «2 из 2». Например, платежный канал между Алисой и Бобом был бы собран на основе следущей ниже мультиподписи «2 из 2»:

```
2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

Приведенный выше привязывающий скрипт может быть удовлетворен отвязывающим скриптом, содержащим пару подписей[109]:

```
0 <Sig Alice> <Sig Bob>
```

Два сценария вместе сформируют совмещенный валидационный скрипт:

```
0 <Sig Alice> <Sig Bob> 2 <PubKey Alice> <PubKey Bob> 2 CHECKMULTISIG
```

Мультиподписной привязывающий скрипт может быть представлен адресом Bitcoin, кодирующим хеш привязывающего скрипта. Например, первоначальная финансовая транзакция платежного канала Lightning – это транзакция, которая выполняет оплату по адресу, кодирующему мультиподписной привязывающий скрипт «2 из 2» двух канальных партнеров.

Скрипты привязки ко времени

Еще одним важным строительным блоком, который существует в системе Bitcoin и широко используется в сети Lightning, является привязка ко времени. Привязка ко времени – это ограничение на расходование, которое требует, чтобы прошло определенное время либо была достигнута определенная высота блока, прежде чем расходование будет разрешено. Это немного похоже на чек с отсрочкой платежа, выписанный с банковского счета, который нельзя обналичить до даты, указанной в чеке.

Биткойн имеет два уровня привязок ко времени: привязки ко времени уровня транзакции и привязки ко времени уровня выхода.

Привязка ко времени уровня транзакции регистрируется в поле транзакции nLockTime и предотвращает принятие всей транзакции до истечения времени, указанного в привязке ко времени. Привязки ко времени уровня транзакций сегодня являются наиболее часто используемым механизмом наложения привязок ко времени в системе Bitcoin.

Привязка ко времени уровня выхода создается скриптовым оператором. Существует два типа выходных привязок ко времени: привязки к абсолютному времени и привязки к относительному времени.

Привязки к абсолютному времени уровня выхода имплементируются оператором CHECKLOCKTIMEVERIFY, который в разговоре нередко сокращается до CLTV. Привязки к абсолютному времени имплементируют ограничение по времени с меткой абсолютного времени или абсолютной высотой блока, выражая эквивалент «не может быть израсходовано до блока 800 000».

Привязки к относительному времени уровня выхода имплементируются оператором CHECKSEQUENCEVERIFY, который в разговоре нередко сокраща-

[109] Первый аргумент (0) не имеет никакого смысла, но требуется из-за дефекта в имплементации мультиподписи в системе Bitcoin. Эта проблема описана в главе 7 книги «Освоение системы Bitcoin».

ется до CSV. Привязки к относительному времени имплементируют ограничение расходования, которое относится к подтверждению транзакции, выражающее эквивалент «не может быть израсходовано до наступления 1024 блоков после подтверждения».

Скрипты с несколькими условиями

Одной из наиболее мощных функциональностей языка Bitcoin Script является управление потоком исполнения скриптов, т. н. условные конструкции. Вы, вероятно, знакомы с управлением потоком исполнения в различных языках программирования, в которых используется конструкция `IF...THEN...ELSE`. Условные инструкции языка Bitcoin Script выглядят немного по-другому, но по сути являются той же самой конструкцией.

На базовом уровне условные операционные коды Bitcoin Script позволяют строить привязывающий скрипт, который имеет два способа отвязывания, в зависимости от результата TRUE/FALSE оценивания логического условия. Например, если x равно TRUE, то привязывающим скриптом будет A, иначе (ELSE) привязывающим скриптом будет B.

В дополнение к этому условные выражения Bitcoin Script могут вкладываться на бесконечную глубину, и, стало быть, условная инструкция может содержать внутри себя другое, которое содержит другое, и т. д. Управление потоком Bitcoin Script может использоваться для создания очень сложных скриптов с сотнями или даже тысячами возможных путей исполнения. Вложенность не имеет лимита, но консенсусные правила налагают ограничение на максимальный размер скрипта в байтах.

Bitcoin Script имплементирует управление потоком с использованием операционных кодов IF, ELSE, ENDIF и NOTIF. Кроме того, условные выражения могут содержать логические операторы, такие как BOOLAND, BOOLOR и NOT.

На первый взгляд, скрипты управления потоками Bitcoin Script могут показаться запутанными. Это потому, что Bitcoin Script – это стековый язык. Если арифметическую операцию 1 + 1 выразить в Bitcoin Script, то она будет выглядеть «задом наперед» (1 1 ADD). С конструкцией управления потоком исполнения в Bitcoin Script точно та же история – она также выглядит «задом наперед».

В большинстве традиционных (процедурных) языков программирования управление потоком выглядит следующим образом:

```
if (условие):
  исходный код, выполняемый, когда условие является истинным
else:
  исходный код, выполняемый, когда условие является истинным
исходный код, выполняемый в любом случае
```

В стековом языке, таком как Bitcoin Script, логическое условие предшествует `IF`, заставляя его выглядеть «задом наперед», вот так:

```
условие
IF
  исходный код, выполняемый, когда условие является истинным
ELSE
  исходный код, выполняемый, когда условие является истинным
ENDIF
исходный код, выполняемый в любом случае
```

При чтении исходного кода Bitcoin Script помните, что вычисляемое условие предшествует операционному коду IF.

Использование управления потоком в скриптах

Очень распространенным применением управления потоком в Bitcoin Script является построение привязывающего скрипта, который предлагает несколько путей исполнения, каждый из которых имеет свой способ погашения UTXO.

Давайте рассмотрим простой пример, где у нас есть два подписанта, Алиса и Боб, и любой из них имеет возможность выполнить погашение. При использовании мультиподписи это было бы выражено в виде мультиподписного скрипта 1 из 2. Для демонстрации мы сделаем то же самое с инструкцией IF:

```
IF
  <Pubkey Алисы> CHECKSIG
ELSE
  <Pubkey Боба> CHECKSIG
ENDIF
```

Глядя на этот привязывающий скрипт, вы, возможно, зададитесь вопросом: а где, собственно, условие? Ведь нет ничего, что предшествовало бы инструкции IF!

Условие не является частью привязывающего скрипта. Вместо этого условие будет предложено в отвязывающем скрипте, что позволит Алисе и Бобу «выбирать» путь исполнения, который они хотят.

Алиса погашает это с помощью отвязывающего скрипта:

```
<Sig алисы> 1
```

1 в конце служит условием (TRUE), которое заставит инструкцию IF исполнить первый путь погашения, для которого у Алисы есть подпись.

Для того чтобы Боб смог этим воспользоваться, ему пришлось бы выбрать второй путь исполнения, передав инструкции IF значение FALSE:

```
<Sig Боба> 0
```

Отвязывающий скрипт Боба помещает 0 в стек, заставляя инструкцию IF исполнить второй (ELSE) скрипт, для которого требуется подпись Боба.

Поскольку каждое из двух условий также требует подписи, Алиса не может использовать вторую инструкцию, а Боб не может использовать первую инструкцию; у них для этого нет необходимых подписей!

Поскольку условные потоки могут быть вложенными, то же самое можно сделать со значениями TRUE/FALSE в отвязывающем скрипте для навигации по сложному пути условий.

В примере А-1 вы видите пример сложного скрипта, который используется в сети Lightning, с несколькими условиями[110]. Скрипты, используемые в сети Lightning, высокооптимизированы и компактны, чтобы минимизировать нагрузку на сеть, поэтому их нелегко читать и понимать. Тем не менее попробуйте, сможете ли вы определить некоторые концепции языка Bitcoin Script, о которых мы узнали в этой главе.

[110] Взято из спецификации BOLT #3 (https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md).

Пример А-1. Сложный скрипт, используемый в сети Lightning

```
  # На дистанционный узел с ключом отзыва
  DUP HASH160 <RIPEMD160(SHA256(revocationpubkey))> EQUAL
  IF
      CHECKSIG
  ELSE
      <remote_htlcpubkey> SWAP SIZE 32 EQUAL
      NOTIF
          # На локальный узел посредством транзакции с HTLC-тайм-аутом (привязан-
ной ко времени).
          DROP 2 SWAP <local_htlcpubkey> 2 CHECKMULTISIG
      ELSE
          # На дистанционный узел с прообразом.
          HASH160 <RIPEMD160(payment_hash)> EQUALVERIFY
          CHECKSIG
      ENDIF
  ENDIF
```

Приложение B

Базовая инсталляция и использование Docker

Эта книга содержит ряд примеров, которые выполняются внутри Docker-контейнеров с целью стандартизации в различных операционных системах.

Данное приложение поможет вам проинсталлировать Docker и ознакомиться с несколькими наиболее часто используемыми командами Docker, чтобы вы могли запускать примеры контейнеров из книги.

Инсталляция Docker

Прежде чем мы начнем, вы должны установить контейнерную систему Docker на свой компьютер. Docker – это открытая система, которая распространяется бесплатно как редакция для сообщества (Community Edition), предназначенная для работы во многих разных операционных системах, включая Windows, macOS и Linux. Версии для Windows и Macintosh называются Docker Desktop и состоят из настольного приложения с графическим интерфейсом и инструментом командной строки. Версия Linux называется Docker Engine и состоит из демона сервера и инструментов командной строки. Мы будем использовать инструменты командной строки, которые идентичны на всех платформах.

Продолжайте и проинсталлируйте Docker в свою операционную систему, следуя инструкциям «Get Docker» (Получить Docker) с веб-сайта Docker[111].

Выберите свою операционную систему из списка и следуйте инструкциям по инсталляции.

Если вы инсталлируете в Linux, то следуйте инструкциям после инсталляции, чтобы убедиться, что вы можете запускать Docker как обычный пользователь, а не как корневой пользователь (`root`). В противном случае вам нужно будет предварять все команды docker командой `sudo`, запуская их от имени `root`, например `sudo docker`.

[111] См. https://docs.docker.com/get-docker.

После инсталляции Docker можно протестировать свою инсталляцию, запустив демонстрационный контейнер `hello-world` следующим образом:

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

Базовые команды Docker

В этом приложении мы довольно широко используем Docker. Мы будем применять следующие ниже команды и аргументы Docker.

Сборка контейнера

```
docker build [-t тег] [каталог]
```

тег – это то, как мы идентифицируем собранный нами контейнер, а *каталог* – это место, где находится контекст контейнера (папки и файлы) и файл определения (Dockerfile).

Оперирование контейнером

```
docker run -it [--network имя_сети] [--name имя] тег
```

имя_сети – это имя сети Docker, *имя* – это каноническое имя, которое мы выбираем для этого экземпляра контейнера, а *тег* – это тег имени, который мы дали контейнеру при его сборке.

Исполнение команды в контейнере

```
docker exec имя команда
```

имя_спате – это имя, которое мы дали контейнеру в команде run, а *команда* – это исполняемый файл или скрипт, который мы хотим выполнить внутри контейнера.

Остановка и запуск контейнера

В большинстве случаев, если мы запускаем контейнер как в интерактивном, так и в терминальном режиме, т. е. с установленными флагами `i` и `t` (комбинированно как `it`), контейнер можно остановить простым нажатием **Ctrl-C** или выходом из оболочки с помощью `exit` либо **Ctrl-D**. Если контейнер не завершается, то вы можете его остановить из другого терминала следующим образом:

```
docker stop имя
```

В целях возобновления работы уже существующего контейнера используйте команду start следующим образом:

```
docker start имя
```

Удаление контейнера по имени

Если вы даете контейнеру имя, вместо того чтобы позволить Docker назвать его случайным образом, вы не сможете повторно использовать это имя до тех пор, пока контейнер не будет удален. Docker вернет ошибку, подобную этой:

```
docker: Error response from daemon: Conflict. The container name "/bitcoind" is
already in use...
```

Для того чтобы ее исправить, удалите существующий экземпляр контейнера:

```
docker rm имя
```

имя_cname – это имя, заданное контейнеру (bitcoind в примере сообщения об ошибке).

Выведение списка оперируемых контейнеров

```
docker ps
```

Эта команда показывает текущие запущенные контейнеры и их имена.

Выведение списка Docker-образов

```
docker образ ls
```

Эта команда показывает Docker-образы, которые были собраны или скачаны на ваш компьютер.

Вывод

Приведенных выше базовых команд Docker будет достаточно, чтобы начать работу, и они позволят выполнить все примеры в этой книге.

Приложение C

Сообщения проводного протокола

В этом приложении перечислены все определенные в настоящее время типы сообщений, используемые в P2P-протоколе Lightning. Кроме того, мы показываем структуру каждого сообщения, собирая сообщения в логические группировки на основе потоков протокола.

Сообщения протокола Lightning расширяемы, и их структура может изменяться во время всесетевых модернизаций. Для получения достоверной информации обратитесь к последней версии спецификации BOLT, находящейся в репозитории Lightning-RFC на GitHub[112].

Типы сообщений

Определенные в настоящее время типы сообщений перечислены в табл. C-1.

Таблица C-1. Типы сообщений

Целое число типа	Имя сообщения	Категория
16	`init`	Установление соединения
17	`error`	Сообщение об ошибке
18	`ping`	Оживленность соединения
19	`pong`	Оживленность соединения
32	`open_channel`	Финансирование канала
33	`accept_channel`	Финансирование канала
34	`funding_created`	Финансирование канала
35	`funding_signed`	Финансирование канала
36	`funding_locked`	Финансирование канала + Операция канала
38	`shutdown`	Закрытие канала
39	`closing_signed`	Закрытие канала

[112] См. https://github.com/lightningnetwork/lightning-rfc.

Окончание табл. C-1

Целое число типа	Имя сообщения	Категория
128	update_add_htlc	Операция канала
130	update_fulfill_hltc	Операция канала
131	update_fail_htlc	Операция канала
132	commit_sig	Операция канала
133	revoke_and_ack	Операция канала
134	update_fee	Операция канала
135	update_fail_malformed_htlc	Операция канала
136	channel_reestablish	Операция канала
256	channel_announcement	Операция канала
257	node_announcement	Операция канала
258	channel_update	Анонс канала
259	announce_signatures	Операция канала
261	query_short_chan_ids	Синхронизация канального графа
262	reply_short_chan_ids_end	Синхронизация канального графа
263	query_channel_range	Синхронизация канального графа
264	reply_channel_range	Синхронизация канального графа
265	gossip_timestamp_range	Синхронизация канального графа

В табл. 13-1 поле Категории позволяет быстро классифицировать сообщение на основе его функциональности в рамках самого протокола. На высоком уровне мы помещаем сообщение в один из восьми (неисчерпаемых) сегментов, включая:

установление соединения.

Отправляется при первом установлении однорангового соединения. Также используется для согласования набора функциональностей, поддерживаемых новым соединением;

сообщение об ошибке.

Используется одноранговыми узлами для передачи сообщений друг другу о возникновении ошибок уровня протокола;

оживленность соединения.

Используется одноранговыми узлами для проверки того, что данное транспортное соединение все еще работает;

финансирование канала.

Используется одноранговыми узлами для создания нового платежного канала. Этот процесс также называется процессом финансирования канала;

операция канала.

Акт обновления данного канала вне цепи. Сюда входят отправка и получение платежей, а также пересылка платежей внутри сети;

объявление канала.

Процесс объявления нового публичного канала для более широкой сети, чтобы его можно было использовать для целей маршрутизации;

синхронизация канального графа.

Процесс скачивания и верифицирования канального графа.

Обратите внимание, что сообщения, принадлежащие к одной и той же категории, обычно также коллективно используют смежный тип сообщений. Это сделано специально для того, чтобы сгруппировать семантически похожие сообщения вместе в самой спецификации.

Структура сообщения

Теперь мы подробно рассмотрим каждую категорию сообщений, чтобы определить точную структуру и семантику всех сообщений, определенных в рамках протокола LN.

Сообщения об установлении соединения

Сообщения в этой категории являются самым первым сообщением, отправляемым между одноранговыми узлами после установления транспортного соединения. На момент написания этой главы в данной категории существовало только одно сообщение – сообщение `init`. Сообщение `init` отправляется обеими сторонами соединения после того, как оно только что было установлено. Никакие другие сообщения не должны отправляться до того, как сообщение `init` будет отправлено обеими сторонами.

Сообщение init

Структура сообщения `init` определяется следующим образом:

- Тип: 16
- Поля:
  - `uint16: global_features_len`
  - `global_features_len*byte: global_features`
  - `uint16: features_len`
  - `features_len*byte: features`
  - `tlv_stream_tlvs`

Структурно сообщение `init` состоит из двух байтовых срезов переменного размера, в каждом из которых хранится набор битов функциональностей. Как мы видим в разделе «Биты функциональностей и расширяемость протокола» на стр. 325, биты функциональностей – это примитив, используемый в протоколе для объявления набора функциональностей протокола, которые узел либо понимает (опциональные функциональности), либо требует (требуемые функциональности).

Обратите внимание, что современные имплементации узлов будут использовать только поле `features`, а элементы будут находиться в векторе `global_features` в основном для *исторических* целей (обратной совместимости).

То, что следует за стержневым сообщением, представляет собой серию записей «Тип–длина–значение» (TLV), которые могут использоваться для расширения сообщения в прямо и обратно совместимой манере в будущем. Мы дадим описание того, что такое TLV-записи и как они используются, позже в этом приложении.

Затем сообщение `init` инспектируется одноранговым узлом, чтобы определить, правильно ли соединение определено на основе набора опциональных и требуемых битов функциональностей, объявленных обеими сторонами.

Опциональная функциональность означает, что одноранговый узел знает о какой-либо функциональности, но не считает ее критически важной для операций нового соединения. Примером одной из них может быть что-то вроде способности понимать семантику поля, недавно добавленного в существующее сообщение.

С другой стороны, требуемые функциональности указывают на то, что если другой одноранговый узел не знает об этой функциональности, то соединение недостаточно четко определено. Примером такой функциональности может быть теоретический новый тип канала в рамках протокола: если ваш одноранговый узел не знает об этой функциональности, то вы не хотите поддерживать соединение, потому что он не сможет открыть ваш новый предпочтительный тип канала.

Сообщения об ошибке

Сообщения этой категории используются для отправки ошибок уровня соединения между двумя одноранговыми узлами. В протоколе существует еще один тип ошибки: ошибка уровня пересылки HTLC-контракта. Ошибки уровня соединения могут сигнализировать о таких вещах, как несовместимость битов функциональностей или намерение принудительно закрыть (в одностороннем порядке выполнить широковещательную передачу последней подписанной фиксации).

Сообщение error

Единственным сообщением в этой категории является сообщение `error`.

- Тип: 17
- Поля:
  - `channel_id : chan_id`
  - `uint16 : data_len`
  - `data_len*byte : data`

Сообщение `error` может быть отправлено в пределах определенного канала, установив поле `channel_id` равным `channel_id` канала, находящегося в этом новом состоянии ошибки. В качестве альтернативы, если ошибка относится к соединению в целом, то поле `channel_id` должно быть установлено равным нулям. Этот полностью нулевой `channel_id` также называется идентификатором уровня соединения для ошибки.

В зависимости от природы ошибки отправка сообщения `error` одноранговому узлу, с которым у вас есть канал, может указывать на то, что канал не может быть продолжен без ручного вмешательства, поэтому единственным вариантом на этом этапе является принудительное закрытие канала путем широковещательной передачи последнего состояния фиксации канала.

Оживленность соединения

Сообщения в этом разделе используются для проверки того, работает соединение по-прежнему или нет. Поскольку протокол LN несколько абстрагируется от базового транспорта, используемого для передачи сообщений, определяется набор сообщений `ping` и `pong` уровня протокола.

Сообщение ping

Сообщение `ping` используется для проверки того, является ли другая сторона в соединении «живой». Оно содержит следующие ниже поля:

- Тип: 18
- Поля:
  - `uint16 : num_pong_bytes`
  - `uint16 : ping_body_len`
  - `ping_body_len*bytes : ping_body`

Следующий его спутник – сообщение `pong`.

Сообщение pong

Сообщение `pong` отправляется в ответ на сообщение `ping` и содержит следующие ниже поля:

- Тип: 19
- Поля:
  - `uint16 : pong_body_len`
  - `ping_body_len*bytes : pong_body`

Сообщение `ping` может быть отправлено любой стороной в любое время.

Сообщение `ping` содержит поле `num_pong_bytes`, которое используется для указания принимающему узлу величины полезного груза, который он отправляет в своем сообщении `pong`. Сообщение `ping` также включает в себя непрозрачный набор байтов `ping_body`, который может быть безопасно проигнорирован. Он служит только для того, чтобы позволить отправителю заполнять отправляемые им сообщения `ping`, что бывает полезно при попытке помешать определенным методам деанонимизации, основанным на размерах пакетов в проводе.

Сообщение `pong` должно быть отправлено в ответ на полученное сообщение `ping`. Получатель должен прочитать набор случайных байтов `num_pong_bytes`, чтобы отправить обратно в качестве поля `pong_body`. Разумное использование этих полей/сообщений может позволить заботящемуся о конфиденциальности маршрутизационному узлу попытаться помешать определенным классам попыток деанонимизации сети, поскольку они могут создавать «поддельную» дешифровку, которая напоминает другие сообщения, основанные на размерах передаваемых пакетов. Напомним, что по умолчанию сеть Lightning использует шифрованный транспорт, поэтому пассивный сетевой монитор не может читать байты обычного текста и, следовательно, может отключать только хронометраж и размеры пакетов.

Финансирование канала

По мере продвижения мы вступаем на территорию стержневых сообщений, которые определяют функциональность и семантику протокола Lightning. В этом разделе мы разведаем сообщения, отправляемые в процессе создания нового канала. Мы опишем только используемые поля, поскольку оставляем углубленный анализ процесса финансирования для главы 7.

Сообщения, отправляемые во время потока финансирования канала, относятся к следующему набору из пяти сообщений: open_channel, accept_channel, funding_created, funding_signed и funding_locked.

Подробный протокол, использующий эти сообщения, описан в главе 7.

Сообщение open_channel

Сообщение `open_channel` запускает процесс финансирования канала и содержит следующие ниже поля:

- Тип: 32
- Поля:
  - `chain_hash : chain_hash`
  - `32*byte : temp_chan_id`
  - `uint64 : funding_satoshis`
  - `uint64 : push_msat`
  - `uint64 : dust_limit_satoshis`
  - `uint64 : max_htlc_value_in_flight_msat`
  - `uint64 : channel_reserve_satoshis`
  - `uint64 : htlc_minimum_msat`
  - `uint32 : feerate_per_kw`
  - `uint16 : to_self_delay`
  - `uint16 : max_accepted_htlcs`
  - `pubkey : funding_pubkey`
  - `pubkey : revocation_basepoint`
  - `pubkey : payment_basepoint`
  - `pubkey : delayed_payment_basepoint`
  - `pubkey : htlc_basepoint`
  - `pubkey : first_per_commitment_point`
  - `byte : channel_flags`
  - `tlv_stream : tlvs`

Это первое сообщение, отправляемое, когда узел желает исполнить новый поток финансирования с другим узлом. Это сообщение содержит всю требующуюся информацию, необходимую обоим одноранговым узлам для строительства как финансовой транзакции, так и фиксационной транзакции.

На момент написания данной главы в наборе опциональных TLV-записей, которые могут быть добавлены в конец определенного сообщения, определена единственная TLV-запись:

- Тип: 0
- Данные: `upfront_shutdown_script`

`upfront_shutdown_script` – это байтовый срез переменного размера, который должен быть валидным скриптом публичного ключа, принятым консенсусным алгоритмом сети Bitcoin. Предоставляя такой адрес, отправляющая сторона может эффективно создать «замкнутый цикл» для своего канала, поскольку ни одна из сторон не подпишет транзакцию кооперативного закрытия, которая оплачивается по любому другому адресу. На практике этот адрес обычно выводится из кошелька холодного хранения.

Поле `channel_flags` – это битовое поле, из которого на момент написания только первый бит имеет какое-либо значение. Если этот бит установлен, то этот канал должен быть объявлен в публичной сети как маршрутизируемый канал. В противном случае канал считается неразрекламированным, также обычно именуемым приватным каналом.

Сообщение accept_channel

Сообщение `accept_channel` является ответом на сообщение `open_channel`.

- Тип: 33
- Поля:
  - `32*byte : temp_chan_id`
  - `uint64 : dust_limit_satoshis`
  - `uint64 : max_htlc_value_in_flight_msat`
  - `uint64 : channel_reserve_satoshis`
  - `uint64 : htlc_minimum_msat`
  - `uint32 : minimum_depth`
  - `uint16 : to_self_delay`
  - `uint16 : max_accepted_htlcs`
  - `pubkey : funding_pubkey`
  - `pubkey : revocation_basepoint`
  - `pubkey : payment_basepoint`
  - `pubkey : delayed_payment_basepoint`
  - `pubkey : htlc_basepoint`
  - `pubkey : first_per_commitment_point`
  - `tlv_stream : tlvs`

Сообщение `accept_channel` – это второе сообщение, отправляемое в процессе потока финансирования. Оно служит для подтверждения намерения открыть канал с новым дистанционным одноранговым узлом. Сообщение в основном повторяет набор параметров, которые ответчик желает применить к своей версии фиксационной транзакции. В главе 7, когда мы подробно рассмотрим процесс финансирования, мы проводим разведку последствий различных параметров, которые могут быть установлены при открытии нового канала.

Сообщение funding_created

В ответ инициатор отправит сообщение `funding_created`.

- Тип: 34
- Поля:
  - `32*byte : temp_chan_id`
  - `32*byte : funding_txid`

- uint16 : funding_output_index
- sig : commit_sig

После того как инициатор канала получает сообщение accept_channel от получателя, у него есть все материалы, необходимые для сборки фиксационной транзакции, а также финансовой транзакции. Поскольку каналы по умолчанию являются единственным источником финансирования (только одна сторона выделяет средства), только инициатору необходимо построить финансовую транзакцию. Как следствие, для того чтобы дать ответчику подписать версию фиксационной транзакции для инициатора, инициатору необходимо отправить только финансовую выходную точку канала.

Сообщение funding_signed

В заключение ответчик отправляет сообщение funding_signed.

- Тип: 34
- Поля:
  - channel_id : channel_id
  - sig : signature

В заключение, после того как ответчик получит сообщение funding_created, теперь у него есть валидная подпись фиксационной транзакции инициатора. С помощью этой подписи он может выйти из канала в любое время, подписав свою половину мультиподписного выхода финансирования и выполнив широковещательную передачу транзакции. Это называется принудительным закрытием. И наоборот, чтобы предоставить инициатору возможность закрыть канал, ответчик также подписывает фиксационную транзакцию инициатора.

После того как это сообщение будет получено инициатором, для него будет безопасно выполнить широковещательную передачу финансовой транзакции, потому что теперь он может выйти из канального соглашения в одностороннем порядке.

Сообщение funding_locked

После того как финансовая транзакция получит достаточное число подтверждений, отправляется сообщение funding_locked.

- Тип: 36
- Поля:
  - channel_id : channel_id
  - pubkey : next_per_commitment_point

После того как финансовая транзакция получит минимальное число подтверждений, обе стороны должны отправить сообщение funding_locked. Канал можно будет начать использовать только после того, как это сообщение будет получено и отправлено.

Закрытие канала

Закрытие канала – это многошаговый процесс. Один узел инициируется отправкой сообщения shutdown. Затем два партнера по каналу обмениваются се-

рией сообщений `closing_signed` для согласования взаимоприемлемых комиссионных за закрывающую транзакцию. Финансист канала отправляет первое сообщение `closing_signed`, а другая сторона может его принять, отправив сообщение `closing_signed` с теми же значениями комиссионных.

Сообщение shutdown

Сообщение `shutdown` инициирует процесс закрытия канала и содержит следующие ниже поля:

- Тип: 38
- Поля:
  - `channel_id : channel_id`
  - `u16 : len`
  - `len*byte : scriptpubkey`

Сообщение closing_signed

Сообщение `closing_signed` отправляется каждым партнером канала до тех пор, пока они не договорятся о комиссионных.

Оно содержит следующие ниже поля:

- Тип: 39
- Поля:
  - `channel_id : channel_id`
  - `u64 : fee_satoshis`
  - `signature : signature`

Операция канала

В этом разделе мы кратко опишем набор сообщений, используемых для того, чтобы узлы могли оперировать каналом. Под операцией мы подразумеваем возможность отправлять, получать и пересылать платежи по данному каналу.

Для того чтобы отправить, получить или переслать платеж по каналу, сначала необходимо добавить HTLC-контракт в обе фиксационные транзакции, которые содержат ссылку на канал.

Сообщение update_add_htlc

Сообщение `update_add_htlc` позволяет любой стороне добавить новый HTLC-контракт в противоположную фиксационную транзакцию.

- Тип: 128
- Поля:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `uint64 : amount_msat`
  - `sha256 : payment_hash`
  - `uint32 : cltv_expiry`
  - `1366*byte : onion_routing_packet`

Отправка этого сообщения позволяет одной стороне инициировать либо отправку нового платежа, либо пересылку существующего платежа, поступивше-

го по входящему каналу. В сообщении указывается сумма (amount_msat) вместе с платежным хешем, который отвязывает сам платеж. Набор инструкций по пересылке следующего перехода луковично зашифрован в поле onion_routing_packet. В главе 10, посвященной многошаговой переадресации HTLC-контракта, мы подробно рассмотрим протокол луковичной маршрутизации, используемый в сети Lightning.

Обратите внимание, что каждый отправленный HTLC-контракт использует автоматически наращиваемый ИД, применяемый любым сообщением, которое модифицирует HTLC-контракт (улаживает или отменяет) для ссылки на HTLC-контракт уникальным способом, ограниченным диапазоном канала.

Сообщение update_fulfill_hltc

Сообщение update_fulfill_hltc разрешает погашение (получение) активного HTLC-контракта.

- Тип: 130
- Поля:
  - channel_id : channel_id
  - uint64 : id
  - 32*byte : payment_preimage

Это сообщение отправляется получателем HTLC-контракта участнику, предлагающему активировать активный HTLC-контракт. Указанное сообщение ссылается на id рассматриваемого HTLC-контракта, а также предоставляет прообраз (который отвязывает HLTC-контракт).

Сообщение update_fail_htlc

Сообщение update_fail_htlc отправляется для удаления HTLC-контракта из фиксационной транзакции.

- Тип: 131
- Поля:
  - channel_id : channel_id
  - uint64 : id
  - uint16 : len
  - len*byte : reason

Сообщение update_fail_htlc противоположно сообщению update_fulfill_hltc в том смысле, что оно позволяет получателю HTLC-контракта удалить тот же самый HTLC-контракт. Это сообщение обычно отправляется, когда HTLC-контракт не может быть правильно перенаправлен вверх по потоку и его необходимо отправить обратно отправителю, чтобы распутать HTLC-цепочку. Как мы рассмотрели в разделе «Сообщения о сбоях» на стр. 276, указанное сообщение содержит шифрованную причину сбоя (reason), которая может позволить отправителю либо скорректировать свой платежный маршрут, либо его завершить, если сам сбой является терминальным.

Сообщение committement_signed

Сообщение committement_signed используется для проставления штампа о создании новой фиксационной транзакции.

- Тип: 132
- Поля:
  - `channel_id : channel_id`
  - `sig : signature`
  - `uint16 : num_htlcs`
  - `num_htlcs*sig : htlc_signature`

В дополнение к отправке подписи для следующей фиксационной транзакции отправителю этого сообщения также необходимо отправить подпись для каждого HTLC-контракта, присутствующего в фиксационной транзакции.

Сообщение revoke_and_ack

Сообщение `revoke_and_ack` отправляется для отзыва датированной фиксации.

- Тип: 133
- Поля:
  - `channel_id : channel_id`
  - `32*byte : per_commitment_secret`
  - `pubkey : next_per_commitment_point`

Поскольку сеть Lightning использует фиксационную транзакцию с заменой при отзыве (replace by revoke), после получения новой фиксационной транзакции посредством сообщения `commit_sig` сторона должна отозвать свою предыдущую фиксацию, прежде чем она сможет получить другую. Отзывая фиксационную транзакцию, отзывающая сторона также предоставляет следующую точку фиксации, которая требуется для того, чтобы другая сторона могла отправить ей новое фиксационное состояние.

Сообщение update_fee

Сообщение `update_fee` отправляется для обновления комиссионных по текущим фиксационным транзакциям.

- Тип: 134
- Поля:
  - `channel_id : channel_id`
  - `uint32 : feerate_per_kw`

Это сообщение может быть отправлено только инициатором канала; именно он будет оплачивать фиксационные комиссионные канала, как только тот будет открыт.

Сообщение update_fail_malformed_htlc

Сообщение `update_fail_malformed_htlc` отправляется для удаления поврежденного HTLC-контракта.

- Тип: 135
- Поля:
  - `channel_id : channel_id`
  - `uint64 : id`
  - `sha256 : sha256_of_onion`
  - `uint16 : failure_code`

Это сообщение похоже на сообщение `update_fail_htlc`, но оно редко используется на практике. Как упоминалось ранее, каждый HTLC-контракт несет луковично-шифрованный маршрутизационный пакет, который также обеспечивает целостность частей самого HTLC-контракта. Если сторона получает луковичный пакет, который каким-то образом был поврежден по пути, то она не сможет пакет дешифровать. Как следствие она также не сможет правильно переслать HTLC-контракт; поэтому она отправит это сообщение, чтобы указать на то, что HTLC-контракт был поврежден где-то по пути обратно к отправителю.

Объявление канала

Сообщения в этой категории используются для объявления компонентов структуры данных, прошедших проверку подлинности в канальном графе, в более широкой сети. Канальный граф обладает рядом уникальных свойств вследствие условия, что все данные, добавляемые в канальный граф, также должны быть заякорены в базовой блочной цепи Bitcoin. Как следствие, для того чтобы добавить новую запись в канальный граф, агент должен быть внутрицепными комиссионными за транзакцию. Это служит естественным сдерживающим фактором для спама в сети Lightning.

Сообщение channel_announcement

Сообщение `channel_announcement` используется для объявления нового канала в более широкой сети.

- Тип: 256
- Поля:
  - `sig : node_signature_1`
  - `sig : node_signature_2`
  - `sig : bitcoin_signature_1`
  - `sig : bitcoin_signature_2`
  - `uint16 : len`
  - `len*byte : features`
  - `chain_hash : chain_hash`
  - `short_channel_id : short_channel_id`
  - `pubkey : node_id_1`
  - `pubkey : node_id_2`
  - `pubkey : bitcoin_key_1`
  - `pubkey : bitcoin_key_2`

Серия подписей и публичных ключей в сообщении служит доказательством того, что канал действительно существует в базовой блочной цепи Bitcoin. Как мы подробно рассмотрели в разделе «Короткий ИД канала» на стр. 294, каждый канал уникально идентифицируется с помощью локатора, который кодирует его местоположение в блочной цепи. Этот локатор называется `short_channel_id` и может помещаться в 64-битовое целое число.

Сообщение node_announcement

Сообщение `node_announcement` позволяет узлу объявлять/обновлять свою вершину внутри графа большего канала.

- Тип: 257
- Поля:
  - `sig : signature`
  - `uint64 : flen`
  - `flen*byte : features`
  - `uint32 : timestamp`
  - `pubkey : node_id`
  - `3*byte : rgb_color`
  - `32*byte : alias`
  - `uint16 : addrlen`
  - `addrlen*byte : addresses`

Обратите внимание, что если у узла нет рекламируемого канала в канальном графе, то это сообщение игнорируется, обеспечивая, чтобы добавление элемента в канальный граф было сопряжено с внутрицепной стоимостью. В этом случае внутрицепная стоимость будет равна стоимости создания канала, к которому этот узел подсоединен.

В дополнение к рекламированию своего набора функциональностей это сообщение также позволяет узлу объявлять/обновлять набор сетевых адресов, по которым до него можно добраться.

Сообщение channel_update

Сообщение `channel_update` отправляется для обновления свойств и политик активного канального ребра в канальном графе.

- Тип: 258
- Поля:
  - `signature : signature`
  - `chain_hash : chain_hash`
  - `short_channel_id : short_channel_id`
  - `uint32 : timestamp`
  - `byte : message_flags`
  - `byte : channel_flags`
  - `uint16 : cltv_expiry_delta`
  - `uint64 : htlc_minimum_msat`
  - `uint32 : fee_base_msat`
  - `uint32 : fee_proportional_millionths`
  - `uint16 : htlc_maximum_msat`

В дополнение к возможности активировать/деактивировать канал это сообщение позволяет узлу обновлять свои комиссионные за маршрутизацию, а также другие поля, которые определяют тип платежа, разрешенный для прохождения по этому каналу.

Сообщение announce_signatures

Сообщением `announce_signatures` обмениваются одноранговые узлы канала для сбора набора подписей, необходимых для создания сообщения `channel_announcement`.

- Тип: 259
- Поля:
  - `channel_id : channel_id`
  - `short_channel_id : short_channel_id`
  - `sig : node_signature`
  - `sig : bitcoin_signature`

После отправки сообщения `funding_locked`, если обе стороны желают прорекламировать свой канал в сети, то каждая из них отправит сообщение `announce_signatures`, которое позволяет обеим сторонам разместить четыре подписи, необходимые для создания сообщения `announce_signatures`.

Синхронизация канального графа

Узлы создают локальную перспективу графа каналов, используя пять сообщений: `query_short_chan_ids`, `reply_short_chan_ids_end`, `query_channel_range`, `reply_channel_range` и `gossip_timestamp_range`.

Сообщение query_short_chan_ids

Сообщение `query_short_chan_ids` позволяет одноранговому узлу получать информацию о канале, относящуюся к серии коротких идентификаторов канала.

- Тип: 261
- Поля:
  - `chain_hash : chain_hash`
  - `u16 : len`
  - `len*byte : encoded_short_ids`
  - `query_short_channel_ids_tlvs : tlvs`

Как мы узнали в главе 11, указанные канальные идентификаторы могут быть серией каналов, которые были новыми для отправителя или устаревшими, что позволяет отправителю получать последний набор информации для набора каналов.

Сообщение reply_short_chan_ids_end

Сообщение `reply_short_chan_ids_end` отправляется после того, как одноранговый узел завершит отвечать на предыдущее сообщение `query_short_chan_ids`.

- Тип: 262
- Поля:
  - `chain_hash : chain_hash`
  - `byte : full_information`

Это сообщение сигнализирует принимающей стороне о том, что если она хочет отправить еще одно сообщение с запросом, то теперь она может это сделать.

Сообщение query_channel_range

Сообщение `query_channel_range` позволяет узлу запрашивать набор каналов, открытых в пределах диапазона блоков.

- Тип: 263
- Поля:
  - `chain_hash : chain_hash`
  - `u32 : first_blocknum`
  - `u32 : number_of_blocks`
  - `query_channel_range_tlvs : tlvs`

Поскольку каналы представлены с использованием короткого ИД канала, который кодирует местоположение канала в цепи, узел в сети может использовать высоту блока в качестве своего рода курсора для поиска в цепи, чтобы обнаруживать набор недавно открытых каналов.

Сообщение reply_channel_range

Сообщение `reply_channel_range` является ответом на сообщение `query_channel_range` и включает набор коротких идентификаторов каналов для известных каналов в этом диапазоне.

- Тип: 264
- Поля:
  - `chain_hash : chain_hash`
  - `u32 : first_blocknum`
  - `u32 : number_of_blocks`
  - `byte : sync_complete`
  - `u16 : len`
  - `len*byte : encoded_short_ids`
  - `reply_channel_range_tlvs : tlvs`

В качестве ответа на `query_channel_range` это сообщение отправляет обратно набор каналов, которые были открыты в этом диапазоне. Указанный процесс может быть повторен запрашивающей стороной, перемещая свой курсор дальше по цепи, чтобы продолжить синхронизацию канального графа.

Сообщение gossip_timestamp_range

Сообщение `gossip_timestamp_range` позволяет одноранговому узлу начать получать новые входящие эпидемические сообщения в сети.

- Тип: 265
- Поля:
  - `chain_hash : chain_hash`
  - `u32 : first_timestamp`
  - `u32 : timestamp_range`

После того как одноранговый узел синхронизировал канальный граф, он может отправить это сообщение, если хочет получать обновления об изменениях в канальном графе в реальном времени. Он также может задать поля `first_timestamp` и `timestamp_range`, если хочет получать отложенные обновления, которые он, возможно, пропустил во время отключения.

Приложение D

Источники и уведомления о лицензиях

Это приложение содержит уведомления об атрибуции и лицензии на материалы, включенные с разрешения, предоставленные через открытые лицензии.

Источники

Материалы были получены из различных публичных источников и источников с открытой лицензией:

- вики-страница сети ION Lightning[113];
- «Lightning 101: что такое счет Lightning?» от Suredbits[114];
- разрабатываемые спецификации сети Lightning на GitHub; лицензия Creative Commons (CC-BY 4.0)[115];
- страница в Википедии «Эллиптическая кривая Диффи–Хеллмана»[116];
- страница в Википедии «Цифровая подпись»[117];
- страница в Википедии «Криптографическая хеш-функция»[118];
- страница в Википедии «Луковичная маршрутизация»[119];
- вики-склад «Комплект протоколов сети Lightning»[120];
- вики-склад «Введение в протокол сети Lightning и основы технологии Lightning»[121].

[113] См. https://wiki.ion.radar.tech/.

[114] См. https://medium.com/suredbits/lightning-101-what-is-a-lightning-invoice-d527db1a77e6.

[115] См. https://github.com/lightningnetwork/lightning-rfc.

[116] См. https://w.wiki/4QCL.

[117] См. https://w.wiki/4QCX.

[118] См. https://w.wiki/4QCb.

[119] См. https://w.wiki/4QCc.

[120] См. https://w.wiki/4QCd.

[121] См. https://w.wiki/4QCf.

Сервер BTCPay Server

Логотип, снимки экрана и другие изображения[122] сервера BTCPay Server, используемые в рамках лицензии MIT[123]:

Лицензия MIT

Авторское право (c) 2018 BTCPay Server

Настоящим бесплатно предоставляется разрешение любому лицу, получающему копию этого программного обеспечения и связанных с ним файлов документации («Программное обеспечение»), на использование Программного обеспечения без ограничений, включая, помимо прочего, права на использование, копирование, изменение, объединение, публикацию, распространение, сублицензирование и/или продажу копии Программного обеспечения и разрешение лицам, которым Программное обеспечение предоставляется, делать это при соблюдении следующих ниже условий:

Вышеупомянутое уведомление об авторских правах и это уведомление о разрешении должны быть включены во все копии или существенные части Программного обеспечения.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРЕДОСТАВЛЯЕТСЯ «КАК ЕСТЬ», БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ, ПРИГОДНОСТИ ДЛЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ И ОТСУТСТВИЯ НАРУШЕНИЙ. НИ В КОЕМ СЛУЧАЕ АВТОРЫ ИЛИ ПРАВООБЛАДАТЕЛИ НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ЗА КАКИЕ-ЛИБО ПРЕТЕНЗИИ, УБЫТКИ ИЛИ ИНУЮ ОТВЕТСТВЕННОСТЬ, БУДЬ ТО В РЕЗУЛЬТАТЕ ДЕЙСТВИЯ ДОГОВОРА, ДЕЛИКТА ИЛИ ИНЫМ ОБРАЗОМ, ВЫТЕКАЮЩИЕ ИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЛИ В СВЯЗИ С НИМ ИЛИ ИСПОЛЬЗОВАНИЕМ ИЛИ ДРУГИМИ СДЕЛКАМИ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ.

Lamassu Industries AG

Изображения биткойн-банкомата Gaia[124], показанные на рис. 2-3, используются с разрешения Lamassu Industries AG. Использование этих изображений не является одобрением продукта или компании, но предоставляется в качестве наглядного примера биткойн-банкомата.

[122] См. https://github.com/btcpayserver/btcpayserver-media.

[123] См. https://github.com/btcpayserver/btcpayserver-media/blob/master/LICENSE.

[124] См. https://lamassu.is/product/gaia.

Глоссарий

В представленном ниже кратком глоссарии содержатся многие термины, применяемые по отношению к системе Bitcoin и сети Lightning. Эти термины используются по всей книге, поэтому добавьте его в закладки для быстрого доступа.

Адрес (address)

В Bitcoin-адрес компактно кодируется информация, необходимая для оплаты получателю. Современный адрес состоит из цепочки букв и цифр, которая начинается с bc1 и выглядит как bc1qw508d6qejxtdg4y5r3zarvary0c5xw 7kv8f3t4. Адрес является кратким обозначением привязывающего скрипта получателя, который может использоваться отправителем для подписания поручения о перечислении средств получателю. Большинство адресов представляют публичный ключ получателя или какую-либо форму скрипта, который определяет более сложные условия расходования средств. Приведенный выше пример представляет собой адрес bech32, кодирующий свидетельскую программу, привязывающую средства в хеше публичного ключа (см. Оплата по хешу свидетельского публичного ключа). Существуют также более старые форматы адресов, начинающиеся с 1 или 3, в которых представление хешей публичных ключей или хешей скриптов обеспечивается за счет кодировки адресов в формате Base58Check.

Асимметричная криптографическая система (asymmetric cryptographic system)

Асимметричная криптография, или криптография с публичным ключом, – это криптографическая система, в которой используются пары ключей: публичные ключи, которые могут разноситься широко, и приватные ключи, которые известны только владельцу. Генерация таких ключей зависит от криптографических алгоритмов, основанных на математических задачах по генерированию функций, которые легко решаются в одну сторону, но очень трудно решаются в обратную сторону. Эффективная безопасность требует хранения приватным только приватного ключа; публичный ключ может распространяться открыто без ущерба для безопасности.

Автопилот (autopilot)

Автопилот – это рекомендательный механизм для узлов Lightning, в котором используется статистика топологии сети Lightning, чтобы подсказывать, с какими узлами им следует открывать каналы. В зависимости от имплементации автопилота также может быть рекомендована емкость канала. Автопилот не является частью протокола LN.

Остаток средств, баланс (balance)

Остаток платежного канала – это сумма биткойна, принадлежащая каждому партнеру канала. Например, Алиса могла бы открыть канал с Бобом на сумму 1 BTC. Тогда остаток канала составляет 1 BTC для Алисы и 0 BTC для

Боба. Остаток канала обновляется по мере того, как пользователи совершают транзакции. Например, если Алиса отправляет Бобу 0.2 BTC, то остаток теперь составляет 0.8 BTC для Алисы и 0.2 для Боба. При закрытии канала биткойн в канале делится между двумя партнерами по каналу в соответствии с последним остатком средств, закодированным в фиксационной транзакции. В сети Lightning возможность отправлять и получать платежи ограничена остатками каналов. См. Емкость.

bech32

bech32 относится к генерическому формату в кодировке base32 с контрольным суммированием, который обеспечивает надежные гарантии обнаружения ошибок. Хотя bech32 изначально был разработан для использования в качестве формата адресов для нативных выходных данных SegWit (BIP-173), он также используется для кодирования счетов Lightning (спецификация BOLT #11). В то время как в нативных выходных данных SegWit версии 0 (P2WPKH и P2WSH) используется bech32, в выходах более высоких версий SegWit (например, оплата по стержневому корню, или P2TR) используется улучшенный вариант bech32m (BIP-350). Адреса bech32m иногда называются адресами «bc1», т. к. это значение является префиксом таких адресов. Нативные выходы SegWit более эффективны в блочном пространстве, чем более старые адреса, и, следовательно, могут снижать плату за транзакции для владельца такого адреса.

Предложение по усовершенствованию системы Bitcoin (BIP)

Предложение по усовершенствованию системы Bitcoin (Bitcoin Improvement Proposal, аббр. BIP) – это предложение, представленное членами Bitcoin-сообщества, по усовершенствованию Bitcoin. Например, BIP-21 – это предложение по улучшению схемы единого идентификатора Bitcoin-ресурса (URI). Предложения BIP находятся на GitHub[125].

Биткойн, Bitcoin

В зависимости от контекста выступает в качестве названия денежной единицы (монеты), сети или опорного активирующего протокола. При написании «биткойн» с буквой «б» в нижнем регистре обычно относится к денежной единице, тогда как «Bitcoin» с заглавной буквой «Б» обычно относится к протоколу или системе.

Майнинг в системе Bitcoin (Bitcoin mining)

Майнинг в системе Bitcoin – это процесс сборки блока из недавних Bitcoin-транзакций, а затем решения вычислительной задачи, необходимой в качестве доказательства работы. Это процесс, посредством которого обновляется совместный реестр биткойнов (т. е. блочная цепь Bitcoin) и с помощью которого в реестр включаются новые транзакции. Он также является процессом, посредством которого выпускается новый биткойн. Всякий раз, когда создается новый блок, майнинговый узел будет получать новый биткойн, созданный в рамках базово-монетарной транзакции (coinbase-транзакции) этого блока.

[125] См. https://github.com/bitcoin/bips.

Блок (block)

Блок – это структура данных в блочной цепи Bitcoin, которая состоит из заголовка и тела транзакций Bitcoin. Блок помечается меткой времени и фиксируется на определенном предшествующем (родительском) блоке. При хешировании заголовок блока обеспечивает доказательство работы, которое делает блочную цепь вероятностно немутируемой. Блоки должны соответствовать правилам, соблюдение которых обеспечивается сетевым консенсусом в целях расширения блочной сети. Когда блок добавляется в блочную цепь, считается, что включенные транзакции получили свое первое подтверждение.

Блочная цепь, блокчейн (blockchain)

Блочная цепь (блокчейн)[126] – это распределенный реестр, или база данных всех транзакций Bitcoin. Транзакции группируются в дискретные обновления, именуемые блоками, лимитированные количеством вплоть до 4 млн весовых единиц. Блоки производятся примерно каждые 10 минут посредством стохастического процесса, именуемого майнингом. Каждый блок включает в себя вычислительно-емкое «доказательство работы». Требование доказательства работы используется для регулирования блочных интервалов и защиты блочной цепи от атак с целью переписывания истории: злоумышленнику нужно будет преодолеть существующее доказательство работы, чтобы заменить уже опубликованные блоки, что делает каждый блок вероятностно немутируемым, поскольку он погребен под последующими блоками.

Базис технологии Lightning (BOLT)

Базис технологии Lightning (Basis of Lightning Technology, аббр. BOLT) является формальной спецификацией сети Lightning. В отличие от протокола системы Bitcoin, который имеет референтную имплементацию и также служит спецификацией протокола, различные имплементации LN подчиняются спецификации BOLT, поэтому они могут работать друг с другом, образуя одну и ту же сеть. Данная спецификация имеется на GitHub[127].

Емкость (capacity)

Емкость платежного канала эквивалентна сумме биткойна, обеспечиваемой финансовой транзакцией. Поскольку финансовая транзакция в блочной цепи видна публично, а канал объявляется через эпидемический протокол, емкость является публичной информацией. Она не раскрывает никакой информации о сумме биткойна, принадлежащей каждому канальному партнеру в канале, т. е. об остатке. Высокая емкость не гарантирует, что канал может использоваться для маршрутизации в обоих направлениях.

[126] Для справки: изначально блочная цепь – это тип велосипедной цепи, т. е. гибкое изделие, состоящее из многочисленных однотипных, последовательно шарнирно соединенных жестких звеньев, которые могут быть различной конструкции (круглые или фасоннозвенные, пластинчатые, шарнирные и др.), в основном применяющейся в трековых гонках. – *Прим. перев.*

[127] См. https://github.com/lightningnetwork/lightning-rfc.

c-lightning

Имплеменация протокола LN компанией Blockstream[128], базирующейся в Виктории, Сейшельские острова. Она написана на C. Ее исходный код находится на GitHub[129].

Закрывающая транзакция (closing transaction)

Если оба партнера по каналу соглашаются закрыть канал, они создадут расчетную транзакцию, которая отражает самую последнюю фиксационную транзакцию. После обмена подписями под закрывающей транзакцией никаких дальнейших обновлений канала делаться не должно. Взаимное закрытие канала с помощью закрывающей транзакции имеет то преимущество, что для получения всех средств требуется меньше транзакций с блочной цепью, по сравнению с односторонним принудительным закрытием канала путем публикации фиксационной транзакции. Кроме того, средства обеих сторон из закрывающей транзакции могут расходоваться немедленно.

CLTV

CLTV – это аббревиатура для оператора Bitcoin Script `OP_CHECKLOCKTIMEVERIFY`. Она определяет абсолютную высоту блока до того, как выходные данные могут быть израсходованы. Атомарность процесса маршрутизации в значительной степени зависит от CLTV в HTLC-контрактах. Узлы маршрутизации объявляют через эпидемический протокол о своих ожидаемых дельтах истечения срока действия CLTV, которые они желают иметь для любых входящих и исходящих HTLC-контрактов.

Монетарная база (coinbase)

Монетарная база – это специальное поле, используемое только при единоличном вводе базово-монетарных транзакций. Монетарная база допускает до 100 байт произвольных данных, но начиная с BIP-34 она должна сначала указывать текущую высоту блока, чтобы гарантировать уникальность базово-монетарных транзакций. Монетарную базу не следует путать с базово-монетарной транзакцией.

Базово-монетарная транзакция (coinbase transaction)

Первая транзакция в блоке, который всегда создается майнером и который включается в единую монетарную базу. Базово-монетарная транзакция может требовать блочного вознаграждения и назначать его одному или нескольким выходам. Блочное вознаграждение состоит из блочной субсидии (только что созданного биткойна) и суммы всех транзакционных комиссий из транзакций, включенных в блок. Результат монетарной базы можно тратить только после погашения 100 блоков. Если блок включает в себя какие-либо транзакции SegWit, то базово-монетарная транзакция должна включать фиксацию на идентификаторах свидетельских транзакций в дополнительном выходе.

[128] См. https://blockstream.com/.

[129] См. https://github.com/ElementsProject/lightning.

Холодное хранение (cold storage)

Относится к хранению определенной суммы биткойна в офлайновом режиме. Холодное хранение достигается, когда приватные ключи Bitcoin создаются и хранятся в безопасной офлайновой среде. Холодное хранение имеет важность для защиты биткойновых активов. Онлайновые компьютеры уязвимы перед хакерами и не должны использоваться для хранения значительного объема биткойна.

Фиксационная транзакция (commitment transaction)

Фиксационная транзакция – это транзакция Bitcoin, подписанная обоими партнерами по каналу, которая кодирует самый последний остаток канала. Всякий раз, когда новый платеж производится или пересылается по каналу, остаток канала обновляется, и обе стороны подписывают новую фиксационную транзакцию. Важно отметить, что в канале между Алисой и Бобом и Алиса, и Боб хранят свою собственную версию фиксационной транзакции, которая также подписывается другой стороной. В любой момент канал может быть закрыт Алисой либо Бобом, если они отправят свою фиксационную транзакцию в блочную цепь Bitcoin. Отправка более старой (устаревшей) фиксационной транзакции считается обманом (т. е. нарушением протокола) в сети Lightning и может быть оштрафована другой стороной путем истребования всех средств в канале для себя посредством штрафной транзакции.

Подтверждения (confirmations)

После включения транзакции в блок она получает одно подтверждение. Как только в блочной цепи добывается еще один блок, транзакция получает два подтверждения и т. д. Шесть или более подтверждений считаются достаточным доказательством того, что транзакция не может быть отменена.

Контракт (contract)

Контракт – это набор транзакций Bitcoin, которые вместе приводят к определенному желаемому поведению. Примерами могут служить RSMC для создания бездоверительного двунаправленного платежного канала или HTLC-контракта с целью создания механизма, который допускает бездоверительную пересылку платежей через третьих лиц.

Обмен ключами Диффи–Хеллмана (Diffie–Hellman Key Exchange, аббр. DHKE)

В сети Lightning используется метод Диффи–Хеллмана эллиптической кривой (ECDH). Это протокол соглашения об анонимном ключе, который позволяет двум сторонам, каждая из которых имеет пару публичный–приватный ключ на основе эллиптической кривой, устанавливать совместный секрет по небезопасному каналу связи. Этот совместный секрет может использоваться непосредственно в качестве ключа или для производного выведения еще одного ключа. Ключ или производный ключ затем может использоваться для шифрования последующих обменов сообщений с использованием шифра с симметричным ключом. Примером производного ключа может быть совместный секрет между эфемерным сеансовым ключом отправителя луковицы и публичным ключом перехода (переходного узла, чешуйки – hop) луковицы, как описано и используется в формате SPHINX Mix.

Цифровая подпись (digital signature)

Цифровая подпись – это математическая схема для верифицирования подлинности и целостности цифровых сообщений или документов. Ее можно рассматривать как криптографическое обязательство, в котором сообщение не скрыто.

Двойное расходование (double-spending)

Двойное расходование – это результат успешного расходования денежных средств более одного раза. Bitcoin защищает от двойного расходования, верифицируя, что каждая добавленная в блочную цепь транзакция соответствует правилам консенсуса; это означает проверку того, что входы транзакции ранее не были потрачены.

Алгоритм цифровой подписи на основе эллиптической кривой (ECDSA)

Алгоритм цифровой подписи на основе эллиптической кривой (Elliptic Curve Digital Signature Algorithm, аббр. ECDSA) – это криптографический алгоритм, используемый системой Bitcoin для обеспечения того, чтобы средства могли расходоваться только владельцем правильного приватного ключа.

Eclair

Имплементация протокола LN компанией ACINQ[130], базирующейся в Париже. Написан на языке Scala. Исходный код находится на GitHub[131].

Кодирование (encoding)

Кодирование – это процесс конвертирования сообщения в другую форму. Например, конвертирование числа из десятичного в шестнадцатеричное.

Сервер Electrum (Electrum server)

Сервер Electrum – это узел Bitcoin с дополнительным интерфейсом (API). Он часто требуется биткойновыми кошельками, которые не оперируют полноценным узлом. Например, эти кошельки проверяют статус конкретных транзакций или широковещательно транслируют транзакции в очередь mempool с помощью API-интерфейсов сервера Electrum. Некоторые кошельки Lightning тоже используют серверы Electrum.

Эфемерный ключ (ephemeral key)

Эфемерные ключи – это ключи, которые используются только в течение короткого времени и не сохраняются после использования. Они часто выводятся производно из другого ключа, который хранится в течение длительного времени, для использования в одном сеансе. Эфемерные ключи в основном применяются в формате SPHINX Mix и луковичной маршрутизации в сети Lightning. Это повышает безопасность транспортируемых сообщений или платежей. Даже если происходит утечка эфемерного ключа, публичной становится информация только об одном единственном сеансе.

[130] См. https://acinq.co/.

[131] См. https://github.com/ACINQ/eclair.

Биты функциональностей (feature bits)

Двоичная строка, которую узлы Lightning используют для обмена сообщениями друг с другом о том, какие функциональности они поддерживают. Биты функциональностей включены во многие сообщения Lightning, а также в спецификацию BOLT #11. Они декодируются с помощью спецификации BOLT #9 и будут сообщать узлам, какие функциональности активированы узлом и являются ли они обратно совместимыми. Также называются флагами функциональностей.

Комиссионные (fees)

В контексте сети Lightning узлы будут взимать маршрутизационные комиссионные за пересылку платежей других пользователей. Отдельные узлы могут устанавливать свои собственные политики взимания комиссионных, которые будут рассчитываться как сумма фиксированной базовой комиссии (base_fee) и комиссионного тарифа (fee_rate), который зависит от суммы платежа. В контексте системы Bitcoin отправитель транзакции платит майнерам транзакционную комиссию за включение транзакции в блок. Комиссия за транзакцию Bitcoin не включает базовую комиссию и линейно зависит от веса транзакции, но не от суммы.

Финансовая транзакция (funding transaction)

Финансовая транзакция[132] используется для открытия платежного канала. Сумма (в биткойнах) финансовой транзакции в точности соответствует емкости платежного канала. Выходом финансовой транзакции является мультиподписной (multisig) скрипт «2 из 2», в котором каждый партнер канала управляет одним ключом. Из-за его мультиподписной природы он может быть потрачен только по взаимному согласию между партнерами канала. В конечном итоге он будет потрачен одной из фиксационных транзакций или закрывающей транзакцией.

Глобальные признаки (поле `globalfeatures`)

Глобальные признаки узла Lightning – это признаки, представляющие интерес для всех остальных узлов. Чаще всего связаны с поддерживаемыми форматами маршрутизации. Они объявляются в сообщении инициализации `init` одноранговогого протокола, а также в сообщениях `channel_announcement` и `node_announcement` эпидемического протокола обмена сообщениями.

Эпидемический протокол (gossip protocol)

Узлы LN отправляют и получают информацию о топологии сети Lightning посредством эпидемических сообщений, которыми обмениваются со своими одноранговыми узлами. Эмидемический протокол[133] в основном определен

[132] Финансовая транзакция – это начальная транзакция, которая создается, когда одна или обе стороны финансируют канал. – *Прим. перев.*

[133] Эпидемический протокол обмена сообщениями, или протокол по принципу сплетен, – это процедура или процесс компьютерной одноранговой (равноправной) связи, основанный на принципе распространения эпидемий. В некоторых распределенных системах одноранговая эмидемическая передача сообщений используется, чтобы обеспечивать распространение данных среди всех членов группы. – *Прим. перев.*

в спецификации BOLT #7 и определяет формат сообщений `node_announcement`, `channel_announcement` и `channel_update`. В целях предотвращения спама сообщения об объявлениях узла будут перенаправляться только в том случае, если у узла уже есть канал, а сообщения об объявлениях канала будут пересылаться только в том случае, если финансовая транзакция канала была подтверждена системой Bitcoin. Обычно узлы Lightning соединяются со своими партнерами по каналу, но для обработки эмидемических сообщений можно подсоединяться к любому другому узлу Lightning.

Аппаратный кошелек (hardware wallet)

Аппаратный кошелек – это особый тип кошелька Bitcoin, который хранит приватные ключи пользователя на защищенном аппаратном устройстве. На момент написания этой книги аппаратные кошельки для узлов LN недоступны, поскольку, для того чтобы участвовать в протоколе, используемые сетью Lightning ключи должны быть подключены к сети.

Хеш (hash)

Цифровой отпечаток фиксированного размера некоторого двоичного входа произвольной длины. Также именуется дайджестом.

Программный код аутентификации сообщений на основе хеша (HMAC)

Программный код аутентификации сообщений на основе хеша (hash-based message authentication code, аббр. HMAC) – это алгоритм верификации целостности и подлинности сообщения, основанный на хеш-функции и криптографическом ключе. Он используется в луковичной маршрутизации для обеспечения целостности пакета в каждом переходе (переходном узле – hop), а также в рамках варианта протокола Noise, используемого для шифрования сообщений.

Хеш-функция (hash function)

Криптографическая хеш-функция – это математический алгоритм, который соотносит данные произвольного размера с битовой строкой фиксированного размера (хеш) и предназначен для однопутной функции, то есть функции, которую невозможно инвертировать. Единственный способ воссоздать входы из выходов идеальной криптографической хеш-функции – это попытаться выполнить исчерпывающий перебор (атаку грубой силой) возможных входных данных, чтобы увидеть, не дают ли они совпадение.

Привязка к хешу (hashlock)

Привязка к хешу – это заложенное в Bitcoin Script условие расходования, которое ограничивает расходование выхода до тех пор, пока не будет раскрыт указанный фрагмент данных. Привязки к хешу обладают полезным свойством, заключающимся в том, что как только любая привязка к хешу раскрывается посредством расходования, любые другие привязки к хешу, защищенные с помощью того же ключа, также могут быть потрачены. Это позволяет создавать многочисленные выходы, которые обременены одной и той же хеш-привязкой и которые могут использоваться одновременно.

Контракт с привязкой к хешу и времени (HTLC)

Контракт с привязкой к хешу и времени (hash time-locked contract, аббр. HTLC) – это исходный код Bitcoin Script, состоящий из привязок к хешу и привязок ко времени, требующий, чтобы получатель платежа израсходовал платеж до истечения крайнего срока, представив хеш-прообраз, либо отправитель может потребовать возврата средств после истечения срока привязки ко времени. В сети Lightning HTLC-контракты являются выходами в фиксационной транзакции платежного канала и используются для обеспечения бездоверительной маршрутизации платежей.

Счет, счет-фактура (invoice)

Процесс оплаты в сети Lightning инициируется получателем (получателем платежа), который выставляет счет, также именуемый платежным запросом. Счета включают хеш платежа, сумму, описание и время истечения срока действия. Счета Lightning определены в спецификации BOLT #11. Счета также могут содержать запасной Bitcoin-адрес, на который можно произвести платеж в случае, если маршрут не будет найден, а еще подсказки для маршрутизации платежа по приватному каналу.

Маршрутизация точно в срок (just-in-time routing)

Маршрутизация точно в срок (just-in-time, аббр. JIT) – это альтернатива маршрутизации на основе источника, которая была впервые предложена соавтором Рене Пикхардтом. С помощью JIT-маршрутизации промежуточные узлы вдоль маршрута могут приостанавливать происходящую оплату, чтобы выполнять балансировку своих каналов, прежде чем продолжать платеж. За счет этого они могут обеспечивать успешную пересылку платежей, которые в противном случае могли бы завершиться безуспешно из-за нехватки исходящей емкости.

Сообщение Lightning (Lightning message)

Сообщение Lightning – это зашифрованная строка данных, которая может быть отправлена между двумя одноранговыми узлами в сети Lightning. Подобно другим протоколам связи, сообщения Lightning состоят из заголовка и тела. Заголовок и тело имеют свой собственный HMAC-код. Сообщения Lightning являются основным строительным блоком слоя обмена сообщениями.

Сеть Lightning, сетевой протокол Lightning, протокол Lightning (Lightning Network, Lightning Network Protocol, Lightning Protocol)

Сеть Lightning – это протокол поверх системы Bitcoin (или других криптовалют). Он создает сеть платежных каналов, которая обеспечивает надежную пересылку платежей по сети с помощью HTLC-контрактов и луковичной маршрутизации. Другими компонентами сети Lightning являются эпидемический протокол обмена сообщениями, транспортный слой и платежные запросы.

Комплект протоколов сети Lightning (Lightning Network protocol suite)

Комплект протоколов сети Lightning состоит из пяти слоев, которые отвечают за различные части протокола. Снизу (первый слой) доверху (пятый слой) эти слои называются слоем сетевой связи, слоем обмена сообщениями, одноранговым слоем, маршрутизационным слоем и платежным слоем. Различные спецификации BOLT определяют части одного или нескольких слоев.

Узел сети Lightning, узел Lightning (Lightning Network node, Lightning node)

Компьютер, участвующий в сети Lightning по одноранговому протоколу Lightning. Узлы сети Lightning имеют возможность открывать каналы с другими узлами, отправлять и получать платежи, а также маршрутизировать платежи от других пользователей. Как правило, пользователь узла сети Lightning также выполняет узел сети Bitcoin.

lnd

Имплементация протокола LN компанией Lightning Labs[134] из Сан-Франциско. Она написана на языке Go. Ее исходный код находится на GitHub[135].

Локальные признаки (поле: `localfeatures`)

Локальные признаки узла LN – это конфигурируемые признаки, представляющие непосредственный интерес для его одноранговых участников. Они объявляются в инициализационном сообщении `init` протокола одноранговой сетевого взаимодействия, а также в сообщениях `channel_announcement` и `node_announcement` эпидемического протокола.

Время привязки (locktime)

Время привязки, или, более технически, nLockTime[136], – это часть транзакции Bitcoin, которая указывает самое раннее время или самый ранний блок, когда эта транзакция может быть добавлена в блочную цепь.

Слой обмена сообщениями (messaging layer)

Слой обмена сообщениями строится поверх слоя сетевого соединения в рамках комплекта протоколов сети Lightning. Он отвечает за обеспечение зашифрованной и безопасной связи и обмена информацией по выбранному протоколу слоя сетевого соединения. Слой обмена сообщениями определяет структуру и формат сообщений Lightning, как определено в спецификации BOLT #1. Биты функциональностей, определенные в спецификации BOLT #9, также являются частью этого слоя.

Миллисатоши (millisatoshi)

Самая малая учетная единица в сети Lightning. Миллисатоши – это одна стомиллиардная часть одного биткойна, представляющая одну тысячную одного сатоши. Миллисатоши не существуют в системе Bitcoin, и ими нельзя расплачиваться в указанной сети.

Многокомпонентные платежи (multipart payments)

Многокомпонентные платежи (multipart payments, аббр. MPP), часто также именуемые многопутевыми платежами, представляют собой метод разбивки суммы платежа на несколько меньших частей и доставки их по одному или нескольким путям. Поскольку MPP может отправлять многие или все части по

[134] См. https://lightning.engineering/.

[135] См. https://github.com/lightningnetwork/lnd.

[136] Время привязки используется в обеспечение того, чтобы транзакция была привязана к определенной высоте блока или определенному моменту времени. – *Прим. перев.*

одному пути, термин «многокомпонентный платеж» более точен, чем многопутевой платеж. В информатике многокомпонентные платежи моделируются как сетевые потоки.

Мультиподпись (multisignature)

Мультиподпись (multisig) относится к скрипту, который требует более одной подписи для авторизации расходования. Платежные каналы всегда кодируются как мультиподписные адреса, требующие одной подписи от каждого партнера платежного канала. В стандартном случае двустороннего платежного канала используется мультиподписной адрес «2 из 2».

Узел (node)

См. раздел «Узел сети Lightning».

Емкость сети (network capacity)

Емкость LN – это общая сумма биткойна, привязанная и циркулирующая внутри сети Lightning. Это сумма емкостей каждого публичного канала. В какой-то степени она отражает используемость сети Lightning, потому что мы ожидаем, что люди будут вкладывать биткойн в каналы Lightning, чтобы их тратить или пересылать платежи других пользователей. Следовательно, чем больше сумма биткойна в каналах Lightning, тем выше ожидаемая используемость сети Lightning. Обратите внимание, что, поскольку можно наблюдать емкость только публичного канала, истинная емкость сети неизвестна. Кроме того, поскольку емкость канала может обеспечивать неограниченное число платежей туда и обратно, из емкости сети не следует лимит стоимости, передаваемой по сети Lightning.

Слой сетевого соединения (network connection layer)

Самый низкий слой комплекта протоколов сети Lightning. В его обязанность входит поддержание интернет-протоколов, таких как IPv4, IPv6, TOR2 и TOR3, и их использование для создания защищенного криптографического канала связи, как определено в спецификации BOLT #8, или для использования DNS для самозагрузки сети, как определено в спецификации BOLT #10.

Noise_XK

Шаблон каркаса протокола Noise для установления аутентифицированного и зашифрованного канала связи между двумя одноранговыми узлами сети Lightning. X означает, что инициатору соединения не требуется знать публичный ключ. K означает, что публичный ключ получателя должен быть известен.

Луковичная маршрутизация (onion routing)

Луковичная маршрутизация – это метод анонимной связи по компьютерной сети. В луковичной сети сообщения инкапсулируются в слои шифрования, аналогичные чешуйкам луковицы. Зашифрованные данные передаются через ряд сетевых узлов, именуемых луковичными маршрутизаторами, каждый из которых снимает один слой, открывая следующее предназначение данных. Когда последний слой дешифрован, сообщение прибывает по своему назначению. Отправитель остается анонимным, поскольку каждый посредник знает только местоположение непосредственно предшествующего и следующего узлов.

Выход, результат (output)

Выход Bitcoin-транзакции, также именуемый неизрасходованным транзакционным выходом (unspent transaction output, аббр. UTXO). Выход представляет собой неделимую сумму биткойна, которую можно потратить, а также скрипт, который определяет, какие условия необходимо выполнить, чтобы этот биткойн был потрачен. Каждая биткойновая транзакция потребляет некоторые выходы ранее зарегистрированных транзакций и создает новые выходы, которые могут быть потрачены позже последующими транзакциями. Для расходования типичного биткойнового выхода требуется подпись, при этом для выходов может потребоваться выполнение более сложных скриптов. Например, для мультиподписного скрипта требуются подписи двух или более держателей ключа, перед тем как можно будет потратить выход, что является фундаментальным строительным блоком сети Lightning.

Оплата по хешу публичного ключа (Pay-to-Public-Key-Hash)

Оплата по хешу публичного ключа (Pay-to-Public-Key-Hash, аббр. P2PKH) – это тип выхода, который привязывает биткойн к хешу публичного ключа. Выход, привязанный скриптом P2PKH, можно отвязать (израсходовать) путем представления публичного ключа, совпадающего с хешем, и цифровой подписи, созданной соответствующим приватным ключом[137].

Оплата по хешу скрипта (P2SH)

Оплата по хешу скрипта (Pay-to-Script-Hash, аббр. P2SH) – это универсальный тип выхода, который позволяет использовать сложные скрипты Bitcoin Script. В случае P2SH сложный сценарий, который детализирует условия расходования выхода (скрипт погашения), не представлен в привязывающем скрипте. Вместо этого значение привязывается к хешу скрипта, который должен быть представлен и выполнен, чтобы получить возможность расходовать выход.

P2SH-адрес (P2SH address)

P2SH-адреса являются кодировками в формате Base58Check 20-байтового хеша скрипта. P2SH-адреса начинаются с «3». P2SH-адреса скрывают всю сложность, вследствие чего отправитель платежа не видит скрипт.

Оплата по хешу публичного ключа свидетеля (Pay-to-Witness-Public-Key-Hash)

Оплата по хешу публичного ключа свидетеля (Pay-to-Witness-Public-Key-Hash, аббр. P2WPKH) – это SegWit-эквивалент оплаты P2PKH, использующий сегрегированного свидетеля. Подпись для расходования выхода оплаты P2WPKH помещается в свидетельское дерево вместо поля ScriptSig. См. раздел SegWit.

P2WPKH-адрес (P2WPKH address)

Формат адреса в версии «нативного SegWit v0». P2WPKH-адреса имеют кодировку bech32 и начинаются с «bc1q».

[137] В системе Bitcoin хеш публичного ключа также называется адресом, а P2PKH является наиболее распространенной транзакцией. См. https://river.com/learn/terms/p/p2wpkh/. – *Прим. перев.*

Оплата по хешу скрипта свидетеля (Pay-to-Witness-Script-Hash)

Оплата по хешу скрипта свидетеля (Pay-to-Witness-Script-Hash, аббр. P2WSH) – это SegWit-эквивалент оплаты P2SH с использованием сегрегированного свидетеля. Подпись и скрипт расходования выхода оплаты P2WSH помещаются в свидетельское дерево вместо поля ScriptSig. См. раздел SegWit.

P2WSH-адрес (P2WSH address)

Формат адреса скрипта в версии «нативного Segwi v0». P2WSH-адреса имеют кодировку bech32 и начинаются с «bc1q».

Оплата по стержневому корню (Pay-to-Taproot, аббр. P2TR)

Модернизация Taproot (Стержневой корень), которая была введена в ноябре 2021 года, представляет собой новый тип выхода, который привязывает биткойн к дереву условий расходования либо к единственному корневому условию.

P2TR-адрес (P2TR address)

Формат адреса Taproot, представляющий версию SegWit v1, являет собой адрес в кодировке bech32m и начинается с «bc1p».

Платеж (payment)

Платеж в сети Lightning происходит, когда биткойн переводится внутри сети Lightning. Платежи, как правило, не видны в блочной цепи Bitcoin.

Платежный канал (payment channel)

Платежный канал – это финансовая взаимосвязь между двумя узлами сети Lightning, созданная с использованием биткойновой транзакции, оплачивающей мультиподписной адрес. Партнеры по каналу могут использовать канал для отправки биткойнов туда и обратно друг другу, не фиксируя все транзакции в блочной цепи Bitcoin. В типичном платежном канале в блочную цепь добавляются только две транзакции: финансовая транзакция и фиксационная транзакция. Платежи, отправляемые по каналу, не записываются в блочную цепь; при этом принято говорить, что они происходят «вне цепи».

Платежный слой (payment layer)

Верхний и пятый слой комплекта протоколов сети Lightning работает поверх маршрутизационного слоя. В его обязанности входит активирование процесса оплаты посредством счетов в спецификации BOLT #11. Несмотря на то что он в значительной степени использует канальный граф из эмидемического протокола, как определено в спецификации BOLT #7, фактические стратегии доставки платежа не являются частью спецификации протокола и оставлены на усмотрение имплементаций. Поскольку эта тема очень важна для обеспечения надежности процесса доставки платежей, мы включили его в эту книгу.

Равноправный участник (peer)

Участники одноранговой сети. В сети Lightning одноранговые узлы соединяются друг с другом посредством зашифрованной, аутентифицированной связи через сокет TCP, по IP или Tor.

Одноранговый слой (peer-to-peer layer)

Одноранговый слой является третьим слоем комплекта протоколов сети Lightning и работает поверх слоя обмена сообщениями. Он отвечает за определение синтаксиса и семантики информации, которой обмениваются одноранговые узлы с помощью сообщений Lightning. Она состоит из управляющих сообщений, как определено в спецификации BOLT #9; сообщений об установлении канала, его операции и закрытии, как определено в спецификации BOLT #2; а также эпидемических и маршрутизационных сообщений, как определено в спецификации BOLT #7.

Приватный канал (private channel)

Канал, не объявленный для остальной части сети. Технически термин «приватный» является неправильным, поскольку эти каналы все-таки можно идентифицировать с помощью подсказок о маршрутизации и фиксационных транзакций. Их лучше описывать как «необъявленные» каналы. С помощью необъявленного канала партнеры канала могут отправлять и получать платежи друг от друга в обычном режиме. Однако остальная часть сети не будет знать об этом канале и поэтому обычно не сможет использовать его для маршрутизации платежей. Поскольку количество и емкость необъявленных каналов неизвестны, суммарное количество публичных каналов и емкость составляют лишь часть суммарной сети Lightning.

Прообраз (preimage)

В контексте криптографии и, в частности, в сети Lightning прообраз относится ко входу в хеш-функцию, которая производит конкретный хеш. Вычислить прообраз из хеша невозможно (хеш-функции работают только в одну сторону). Отобрав секретное случайное значение в качестве прообраза и вычислив его хеш, можно зафиксировать этот прообраз и позже его раскрыть. Любой желающий может подтвердить, что раскрытый прообраз правильно производит хеш.

Подтверждение работы (Proof of Work, аббр. PoW)

Данные, для отыскания которых требуются значительные вычисления и которые могут быть легко проверены любым человеком, чтобы доказать объем работы, который потребовался для их получения. В системе Bitcoin майнеры должны найти числовое решение алгоритма SHA-256, которое соответствует общесетевой цели, именуемой целевым уровнем сложности. Дополнительную информацию см. в разделе «Майнинг в системе Bitcoin».

Контракт с точечной привязкой ко времени (PTLC)

Контракт с точечной привязкой ко времени (Point Time-Locked Contract, аббр. PTLC) – это Bitcoin Script, который позволяет условно расходовать по предоставлению секрета либо после прохождения определенной высоты блока, аналогично HTLC-контракту. В отличие от HTLC-контрактов, PTLC-контракты зависят не от прообраза хеш-функции, а от приватного ключа из точки на эллиптической кривой. Следовательно, допущение о безопасности основывается на дискретном логарифме. PTLC-контракты в сети Lightning пока что не имплементированы.

Привязка к относительному времени (relative timelock)

Привязка к относительному времени – это тип привязки ко времени, который позволяет входу указывать самое раннее время, когда вход может быть добавлен в блок. Время является относительным и основано на том, когда выход, на который ссылается этот вход, был записан в блоке. Привязки к относительному времени задаются в транзакционном поле `nSequence` и операционным кодом языка Bitcoin Script `CHECKSEQUENCEVERIFY` (CSV), который был введен предложением BIP-68/112/113.

Отзываемый контракт со сроком погашения последовательности (RSMC)

Отзываемый контракт со сроком погашения последовательности (Revocable Sequence Maturity Contract, аббр. RSMC) используется для выстраивания платежного канала между двумя пользователями систем Bitcoin или LN, которым не нужно доверять друг другу. Название контракта происходит от последовательности состояний, которые кодируются как фиксационные транзакции и могут быть отозваны, если они неправильно опубликованы и добыты системой Bitcoin.

Отзывной ключ (revocation key)

Каждый RSMC-контракт содержит два отзывных ключа. Каждый партнер по каналу знает один отзывной ключ. Зная оба отзывных ключа, выход RSMC-контракта может быть израсходован в пределах заданной привязки ко времени. При согласовании нового состояния канала старые отзывные ключи используются совместно, тем самым «отзывая» старое состояние. Отзывные ключи используются для того, чтобы запретить партнерам канала выполнять широковещательную передачу старого состояния канала.

RIPEMD-160

RIPEMD-160 – это криптографическая хеш-функция, которая создает 160-битный (20-байтовый) хеш.

Маршрутизационный слой (routing layer)

Четвертый слой комплекта протоколов сети Lightning работает поверх одноранговой слоя. В его обязанности входит определение криптографических примитивов и необходимого протокола связи, обеспечивающего безопасную и атомарную передачу биткойна с отправляющего узла на узел получателя. В то время как спецификация BOLT #4 определяет луковичный формат, который используется для передачи транспортной информации дистанционным одноранговым узлам, с которыми не существует прямых соединений, фактическая луковичная транспортировка и криптографические примитивы определены в спецификации BOLT # 2.

Топология (topology)

Топология сети Lightning описывает форму сети Lightning в виде математического графа. Узлами графа являются узлы Lightning (участники сети / одноранговые узлы). Ребрами графа являются платежные каналы. Топология сети Lightning публично транслируется с помощью эпидемического протокола, за исключением необъявленных каналов. Это означает, что сеть Lightning может

быть значительно больше, чем заявленное число каналов и узлов. Знание топологии представляет особый интерес для процесса маршрутизации платежей на основе источника, в котором отправитель определяет маршрут.

Сатоши (satoshi)

Сатоши – это наименьшая единица (номинал) биткойна, которая может быть записана в блочной цепи. Один сатоши составляет 1/100 миллиона (0.00000001) биткойна и назван в честь создателя сети Bitcoin Сатоши Накамото.

Сатоши Накамото (Satoshi Nakamoto)

Сатоши Накамото – это имя, используемое человеком или группой людей, которые разработали систему Bitcoin и создали ее оригинальную референтную имплементацию. В рамках своей имплементации они также разработали первую базу данных блочной цепи. В ходе работы они были первыми, кто решил проблему двойного расходования цифровой валюты. Их настоящая личность остается неизвестной.

Подпись Шнорра (Schnorr signature)

Новая схема цифровой подписи, которая планировалась к активации в Bitcoin в ноябре 2021 года. Она позволяет имплементировать инновации в сети Lightning, такие как эффективные PTLC-контракты (усовершенствование HTLC-контрактов).

Скрипт, Bitcoin Script

В Bitcoin используется скриптовая подсистема для транзакций, именуемая Bitcoin Script. Похожий на язык программирования Forth, он прост, основан на стеке и обрабатывается слева направо. Он намеренно неполон по Тьюрингу и не имеет ни циклов, ни рекурсии.

ScriptPubKey (он же скрипт pubkey)

ScriptPubKey, или скрипт pubkey, – это скрипт, включенный в выходы, устанавливающий условия, которые должны быть соблюдены для того, чтобы эти выходы были израсходованы. Данные для выполнения условий могут быть предоставлены в подписном скрипте. См. также ScriptSig.

ScriptSig (он же скрипт подписи)

ScriptPubKey, или скрипт подписи, – это данные, генерируемые получателем, которые почти всегда используются в качестве переменных для выполнения скрипта pubkey.

Секретный ключ (он же приватный ключ)

Секретный номер, который отвязывает биткойн, отправленный на соответствующий адрес. Секретный ключ выглядит так: 5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibvpxh.

Сегрегированный свидетель (SegWit)

Сегрегированный свидетель (SegWit) – это обновление протокола Bitcoin, введенное в 2017 году, которое добавляет новую функциональность для подписей и других доказательств авторизации транзакций. Указанное новое свидетельское поле изъято из вычисления ИД транзакции, что решает большинство

классов проблем с деформируемостью транзакций третьих сторон. Сегрегированный свидетель был развернут как мягкое ответвление и вносит изменение, которое технически делает правила протокола Bitcoin строже.

Безопасный алгоритм хеширования (SHA)

Безопасный алгоритм хеширования (Secure Hash Algorithm, аббр. SHA) – это семейство криптографических хеш-функций, опубликованных Национальным институтом стандартов и технологий (NIST). В протоколе Bitcoin в настоящее время используется SHA-256, который создает 256-битовый хеш.

Короткий идентификатор канала (scid)

После установления канала индекс финансовой транзакции в блочной цепи используется в качестве короткого ИД канала для уникальной идентификации канала. Короткий ИД канала (short channel ID, аббр. scid) состоит из 8 байт, относящихся к трем числам. В своей сериализованной форме он изображает эти три числа в виде десятичных значений, разделенных буквой «x» (например, 600123x01x00). Первое число (4 байта) – это высота блока. Второе число (2 байта) – это индекс финансовой транзакции с блоками. Последнее число (2 байта) – это выход транзакции.

Упрощенная верификация платежа (SPV)

Упрощенная верификация платежа (simplified payment verification, аббр. SPV) – это метод верифицирования того, что те или иные транзакции были включены в блок, без скачивания всего блока. Этот метод используется некоторыми облегченными кошельками систем Bitcoin и Lightning.

Маршрутизация на основе источника (source-based routing)

В сети Lightning отправитель платежа определяет маршрут платежа. Хотя этот факт снижает вероятность успеха процесса маршрутизации, он увеличивает вероятность платежей. Благодаря формату SPHINX Mix, используемому луковичной маршрутизацией, все узлы маршрутизации не знают ни отправителя платежа, ни конечного получателя. Маршрутизация на основе источника принципиально отличается от того, как маршрутизация работает в интернет-протоколе.

Мягкое ответвление (soft fork)

Мягкое ответвление, или изменение на основе мягкого ответвления, – это прямо и обратно совместимое обновление протокола, поэтому оно позволяет как старым, так и новым узлам продолжать использовать одну и ту же цепь.

Формат SPHINX Mix (SPHINX Mix Format)

Особый метод луковичной маршрутизации, используемый в сети Lightning и изобретенный Джорджем Данезисом (George Danezis) и Яном Голдбергом (Ian Goldberg) в 2009 году[138]. В формате SPHINX Mix каждое сообщение луковичного пакета дополняется некоторыми случайными данными, в результате чего ни один отдельный переход (переходный узел – hop) не может оценить, как далеко оно прошло по маршруту. Хотя конфиденциальность отправителя

[138] См. https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf.

и получателя платежа защищена, каждый узел по-прежнему может возвращать сообщение об ошибке вдоль пути к отправителю сообщения.

Подводный своп, субмаринный своп (submarine swap)

Подводный своп – это бездоверительный атомарный обмен между внутрицепными Bitcoin-адресами и внецепными платежами в сети Lightning. Подобно тому, как платежи LN используют HTLC-контракты, которые делают окончательное требование о средствах, при условии что получатель раскрывает секрет (хеш-прообраз), подводные свопы используют тот же механизм для перевода средств через барьер «внутри цепи / вне цепи» с минимальным доверием. Обратные подводные свопы позволяют совершать свопы в противоположном направлении, от внецепного платежа LN к внутрицепному Bitcoin-адресу.

Привязка ко времени (timelock)

Привязка ко времени – это тип обременения, который ограничивает расходование биткойна до определенного времени в будущем или высоты блока. Привязки ко времени занимают видное место во многих Bitcoin-контрактах, включая платежные каналы и HTLC-контракты.

Транзакция (transaction)

Транзакции – это структуры данных, используемые сетью Bitcoin для передачи биткойна с одного адреса на другой. Несколько тысяч транзакций агрегируются в блок, который затем регистрируется (добывается) в блочной цепи. Первая транзакция в каждом блоке, именуемая базово-монетарной транзакцией, генерирует новый биткойн.

Деформируемость транзакции (transaction malleability)

Деформируемость транзакции – это свойство, которое хеш транзакции может изменять без изменения семантики транзакции. Например, изменение подписи может изменить хеш транзакции. Для фиксационной транзакции требуется хеш финансовой транзакции, и если хеш финансовой транзакции изменится, то зависящие от него транзакции станут невалидными. Это лишит пользователей возможности истребовать возврата средств, если таковые имеются. Мягкое ответвление в виде сегрегированного свидетеля (Segregated Witness) решает эту проблему и, следовательно, является важной модернизацией в поддержку сети Lightning.

Транспортный слой (transport layer)

В компьютерных сетях транспортный слой – это концептуальное разделение методов, используемых компьютерами (и в конечном счете приложениями) для взаимодействия друг с другом. Транспортный уровень предоставляет службы связи между компьютерами, такие как управление потоками, верификация и мультиплексирование (чтобы позволить нескольким приложениям работать на компьютере одновременно).

Неизрасходованный выход транзакции (UTXO)

Неизрасходованный транзакционный выход (unspent transaction output, аббр. UTXO). См. Выход.

Кошелек (wallet)

Кошелек – это часть программного обеспечения, которая содержит приватные ключи системы Bitcoin. Используется для создания и подписания транзакций Bitcoin. В контексте сети Lightning также хранит отзывные секреты старого состояния канала и последних предварительно подписанных фиксационных транзакций.

Сторожевая вышка (watchtower)

Сторожевые вышки – это служба безопасности в сети Lightning, которая отслеживает платежные каналы на предмет потенциальных нарушений протокола. Если один из партнеров канала переходит в офлайновый режим или теряет свою резервную копию, то сторожевая вышка сохраняет резервные копии и может восстановить информацию о его канале.
Сторожевые вышки также отслеживают блочную цепь Bitcoin и могут отправлять штрафную транзакцию, если один из партнеров попытается «обмануть», передав устаревшее состояние. Сторожевые вышки могут управляться самими партнерами канала или в качестве платной услуги, предлагаемой третьей стороной. Сторожевые вышки не имеют никакого контроля над средствами в самих каналах.

Некоторые представленные в глоссарии определения были процитированы в рамках лицензии CC-BY из вики-учебника по системе Bitcoin, Википедии, репозитория книги «Освоение системы Bitcoin»[139] и других публикаций с открытым исходным кодом.

[139] См. https://en.bitcoin.it/wiki/Main_Page, https://en.wikipedia.org/ и https://github.com/bitcoinbook/bitcoinbook.

Предметный указатель

А

Автоматизация узла Lightning 154
Автопилот 157
Автопилот lnd 157
Адгоритм Дейкстры 304
Адрес мультиподписной 69, 181
Архитектура сети Lightning 168
  комплект протоколов 168
  набросок деталей 169
  слои 169
Атака на основе топологии 366
Атака прошупыванием 358

Б

Безопасность и конфиденциальность
  риск со стороны горячего кошелька 150
Безопасность и конфиденциальность
  безопасность операционной системы 146
  узел Lightning 146
Безопасность и конфиденциальность 351
  анонимностное множество 353
  атаки на Lightning 356. См. Нарушение конфиденциальности
  важность конфиденциальности 351
  граф Lightning 365
  межслоевая деанонимизация 362
  необъявленные каналы 369
  определения конфиденциальности 351
  практический совет пользователям в отношении защиты конфиденциальности 369
  процесс оценивания конфиденциальности 352
  разница между сетью Lightning и Bitcoin в смысле конфиденциальности 354
  соображения по маршрутизированию 370
  централизация в сети Lightning 368
  экономические стимулы и графовая структура 368
Биткойн testnet (tBTC) 46
Биткойн (валюта)
  владение и контроль в платежных каналах 173
  загрузка в кошелек Lightning 51
  получение для кошелька Lightning 52
  предотращение привязанности и нерасходуемости биткойна 174
  разнообразие форм независимого владения и мультиподпись 174
  совместное владение без независимого контроля 174
Бит функциональности 325, 402
  механизм обеспечения обнаруживаемости модернизации 326
  протокол расширяемости 325

В

Верификация платежей упрощенная (SPV) 42
Верификация хеша 218
Вес блока 91
Вес транзакции 92
Возврат ошибки
  луковичная маршрутизация 275
  сообщение о сбое 276
Вход транзакции 383
Выход транзакции 383
Выходы транзакции неизрасходованные (UTXO) 90, 384
Вышка сторожевая 155

Г

Генерирование корневого приватного ключа 175
Граф Lightning 365
  атаки на основе топологии 366

реальность против теоретического внешнего вида 365
темпоральность сети Lightning 367
экономические стимулы и графовая структура 368
Граф канальный 281
взаимосвязь между картой и территорией 306
децентрализованная инфраструктура публичных ключей 331
комиссионные и другие метрики канала 310
конструкция 305
неопределенность 308
неопределенность ликвидности и вероятность 309
ориентированный граф 289
сообщения эпидемического протокола 290
структура 289
текущее сопровождение 297
Граф ориентированный 289, 302
Груз переходный полезный 253

Д

Данные ассоциированные (AD) 265
Данные перехода 253
Деанонимизация 353
Деанонимизация межслоевая 362
внецепных кластеризация узлов Lightning 364
внутрицепная кластеризация Bitcoin-сущностей 363
межслоевое связывание с узлами Lightning и Bitcoin-сущностями 365
Демон узла сети Lightning (LND)
Docker-контейнер LND 111
инсталлирование LND из исходного кода 114
компилирование исходного кода LND 115
копирование исходного кода LND 115
оперирование контейнерами bitcoind и LND 112
проект 111
Деформируемость транзакций 184
Диапазон ликвидности 302
Доверие
сеть Lightning 89
Доказательство работы (PoW), алгоритм 35
Документация по стандартам BOLT (Базис технологии Lightning) 97
Доминантность центральной точки 368
Допущение о безопасности 352
Доставка платежа 299. См. Отыскание пути
многокомпонентные платежи 314
одноранговый эпидемический протокол 82
отыскание пути и маршрутизация 83
отыскание пути и процесс доставки 305
отыскание пути на основе источника 84
пересылка платежей, алгоритм 87
протокол луковичной маршрутизации 85
цикл проб и ошибок 312, 316

Е

Емкость
платежный канал 302
Емкость канала 69

З

Задача о коммивояжере 84
задержка привязки ко времени 73
Заклинивание фиксационное 362
Закрытие канала 75
взаимное 76
нарушение протокола 78
принудительное закрытие 77
сообщение closing_signed 202
сообщение shutdown 201
транзакция кооперативного закрытия 202
Закрытие принудительное 77, 161
Закрытие транзакций 77, 200
Запирание канальной ликвидности 362
Зачистка средств
внецепная зачистка 151
внутрицепная зачистка 151
горячие кошельки 150
зачистка на основе подводного свопа 151

подводные свопы с помощью петли 152
Зашифровать и затем аутентифицировать 265
Злоумышленник вне пути 356
Злоумышленник на пути 356

И

Идентификатор узла 176
Инновации в Lightning
Lightning-приложения 376
P2P-протокол Lightning 374
варианты использования сети 373
децентрализованная/асинхронная природа 373
инновации в Bitcoin, мотивированные вариантами использования сети Lightning 374
расширяемость TLV 375
сквозные функциональности в порядке выбора 375
строительство платежного канала 375
Инфраструктура публичных ключей (PKI) 331

К

Канал необъявленный 75, 293, 369
Канал платежный 29, 66, 171. См. Данные канала
асимметричная фиксационная транзакция 190
владение биткойном и контроль над ним 173
возврат до финансирования 182
выведение отзывного и фиксационного секретов 194
выстраивание транзакций в цепь без широковещательной передачи 183
задержанное расходование выхода to_self 191
закрытие канала 75, 200
идентификаторы узлов 176
инновации в строительстве 375
ключи публичный/приватный узла 175
конкурирующие фиксации 188
кооперативное закрытие 200
локальный канал против многочисленных маршрутизируемых каналов 229
маршрутизирование платежей по каналам 67
мультиподписные адреса 69, 181
необъявленные каналы 369
обман со старыми фиксационными транзакциями 189
обман с предыдущим состоянием 72
обновление фиксаций с помощью HTLC-контрактов 229
объявление канала 75
ограничения 68
основы 66
отзывные ключи 192
отзыв старых фиксационных транзакций 189
открытие в сети Lightning 122
отправка платежей 187
полезные свойства 68
поток сообщений об установлении канала 177
пример плохой процедуры открытия канала 70
продвижение состояния канала вперед 195
работа 228
разделение платежных остатков 187
сборка заранее подписанных возвратных транзакций 182
сборка финансовой транзакции 181
с двойным финансированием 182
сетевой адрес узла 175
сеть Lightning как другой способ использования системы Bitcoin 172
соединение узлов как одноранговых участников 176
строительство 177
транзакционная деформируемость и сегрегированный свидетель 184
удержание подписанных транзакций без широковещательной передачи 182
фиксационная транзакция 70, 193
финансовая транзакция 69, 181
широковещательная передача финансовой транзакции 186
элементы 175
Канал платежный с двойным финансированием 182

Канал публичный
объявление 75
Канал Пуна–Дриджа 68
Канал сети Lightning
открытие канала 56, 58
резервные копии 148
Каркас криптосвязи на основе протокола
Noise_XK 333
шифрованный транспорт сообщений 333
Каркас криптосвязи на основе протокола Noise
сообщения Lightning 88
Кластеризация Bitcoin-сущностей внутрицепная 363
Кластеризация узлов Lightning 365
Кластеризация узлов Lightning внецепная 364
Ключ 379
генерирование для луковичной маршрутизации 256
кошелек Lightning 48
луковичная маршрутизация 256
Ключ отзывной 192
Ключ приватный 379
владение биткойном 173
генерирование 175
Ключ публичный узла 175
Ключ сеансовый 257
Коллизия 381
Комиссионные
канальный граф 310
сравнение системы Bitcoin и сети Lightning 94
Комиссионные базовые 91
Компонент условный в скрипте 394
Контракт с привязкой к хешу и времени (HTL)
исполнение 240
Контракт с привязкой к хешу и времени (HTLC)
Bitcoin Script 217
внутрицепное и внецепное улаживание платежей 216
декрементирование привязок ко времени 226
добавление HTLC-контракта 231
кооперативный отказ / отказ тайм-аута 225
локальный платеж 245
механизм работы 216
многочисленные контракты 239
новая фиксация с выходом HTLC-контракта 233
обновление фиксаций 229
обратное распространение секрета 220
оптимизация хеша 223
основы луковичной маршрутизации 251
пересылка платежей 230
платежный прообраз и верификация хеша 218
поток фиксационных сообщений 230
привязка подписи для предотвращения кражи 222
признание новой фиксации / отзыв старой фиксации 235
протокол справедливости 214
распространение 240
распространение по сети 219
сообщение update_add_HTLC 231
удаление из-за ошибки / истечения срока 244
фиксационная транзакция 232
Контракт с точечной привязкой ко времени (PTLC) 215, 279
Конфигурирование портов 142
Конфигурирование сети
Tor для входящих соединений 144
автоматическая переадресация портов с помощью UPnP 143
ручное конфигурирование узла Bitcoin или узла Lightning 145
узел Lightning 141
Копирование каналов статическое резервное (SCB) 148
Копия резервная 148
Кошелек Eclair
скачивание/инсталлирование 48
Кошелек Lightning
скачивание/инсталлирование 48
Кошелек Lightning
биткойн testnet 46
загрузка биткойна 51
каналы LN 56
мнемоническая фраза 49
основы 43
получение биткойна 52

приобретение биткойна 51
Биткойн (валюта)
приобретение для кошелька Lightning 51
соединение системы Bitcoin и сети Lightnings мостом 56
создание нового кошелька 49
счета 62
Счет Lightning 62
уравновешивание сложности и контроля 47
хранение мнемонической фразы 50
Кошелек горячий
выпросы безопасности 150
зачистка средств 150
Кошелек неопекаемый (некустодиальный) 44
Кошелек опекаемый (кустодиальный) 43
Кошелек самоопекаемый 44
Кривая эллиптическаая Диффи–Хеллмана (ECDH) 258
Кривая эллиптическая 379

Л

Ликвидность
неопределенность и вероятность 309
платежный канал 302
Ликвидность канала 92
Логит-регрессия. См. Регрессия логистическая

М

Маршрутизатор луковичный (The Onion Router, Tor) 144
Маршрутизация 205
внутрицепное и внецепное улаживание HTLC-контрактов 216
доставка платежа 83
имплементирование атомарных бездоверительных многопереходных платежей 214
комиссионные 162
контракт с привязкой к хешу и времени
механизм работы 216
маршрутизированный платеж 205
по сравнению с отправкой 30
по сравнению с отысканием пути 207
пример сети Lightning 122
принятие каналов 371
протокол справедливости 214
реальный физический пример 208
создание сети платежных каналов 207
соображения по безопасности/конфиденциальности 370
Маршрутизация луковичная 246
HTLC-контракты 251
keysend и конкретно-прикладные записи в приложениях Lightning 280
возвращение ошибок 275
выбор пути 247, 251
генерирование ключей 256
заключительный луковичный пакет 268
застрявшие платежи 278
защита/обнаружение повторного воспроизведения 265
конкретно-прикладные луковичные TLV-записи 279
конструирование полезного груза 253
луковицы фиксированной длины 260
обертывание луковичных слоев 260
обертывание переходного полезного груза 262
общее описание процесса обертывания 261
отправка луковицы 269
отправка/получение платежей keysend 280
отслаивание слоев 250
сборка слоев 248
сообщение update_add_htlc 269
спонтанные платежи keysend 279
физический пример 247
Маршрутизация на основе источника 248
Метаданные счета Lightning 81
Механизм штрафных санкций 189
Милисатоши 95
Множество анонимностное 353
Модернизация
внутренняя сеть 328
сквозная 328
Мониторинг
доступность узла 154
сторожевые вышки 155

Н

Надежность узла Lightning 126
Накопитель твердотельный (SSD) 129
Нарушение конфиденциальности
  атака отказом в обслуживании 360
  запирание канальной ликвидности 362
  межслоевая деанонимизация 362
  межслоевое связывание с узлами Lightning и Bitcoin-сущностями 365
  наблюдение за суммами платежей 356
  раскрытие остатков каналов 358
  связывание отправителей и получателей 356
  фиксационное заклинивание 362
Нарушение протокола 78
  сторожевая вышка 155
Настройка универсальная автоматическая сетевых устройств (UPnP) 142

О

Обеспечение программно-информационное узла Lightning 97
  Bitcoin Core и regtest 102
  Docker-контейнеры 100
  командная строка 98
  проект демона узла сети Lightning 111
  проект узла сети Lightning c-lightning 105
  проект узла сети Lightning Eclair 116
  сборка полной сети из различных узлов Lightning 120
  среда разработки 98
Облако
  оперирование узлом Lightning 127
Обман
  мониторинг обмана 197
  со старыми транзакциями 189
  с предыдущим состоянием 72
Обмен ключей Диффи–Хеллмана (DHKE) 258
Обнаружение однорангового узла 283
  самозагрузка P2P-узлов 283
  самозагрузка адресов DNS-серверов 284
Обнаружение однорангового узла первоначальное 283
Оборудование узла Lightning 129
Общая основа. См. Перекрытие
Операция канала
  сообщение commitment_signed 409
  сообщение revoke_and_ack 410
  сообщение update_add_htlc 408
  сообщение update_fail_htlc 409
  сообщение update_fail_malformed_htlc 410
  сообщение update_fee 410
  сообщение update_fulfill_hltc 409
Оперирование узлом Lightning 125
  lndmon 164
  Ride The Lightning (RTL) 164
  ThunderHub 165
  безопасность 146
  вопросы надежности 126
  время безотказной работы и доступность узла 153
  выбор имплементации 136
  выбор операционной системы 136
  выбор платформы 127
  доступ к узлу 147
  запуск узла 139
  изоляция процесса 138
  инсталлирование узла Bitcoin или узла Lightning 137
  использование инсталлятора и помощника 131
  комиссионные за маршрутизацию 162
  конфигурирование сети 140
  конфигурирование узла 140
  межслоевое связывание с Bitcoin-сущностями 365
  мониторинг доступности 154
  оперирование в облаке 127
  оперирование узлом дома 128
  отказоустойчивость и автоматизация 154
  переключение серверной конфигурации в облаке 130
  постоянное хранилище данных 131
  резервное копирование узла и каналов 148
  риск со стороны горячего кошелька 150
  сторожевые вышки 155
  типы оборудования для узлов Lightning 127
  требования к оборудованию 129

управление каналами 156
управление узлом 164
фоновые службы 138
Остаток
платежный канал 302
Остаток канала
раскрытие 358
Отказ в обслуживании (DoS), атака 360
DoS в сети Lightning 361
DoS в системе Bitcoin 361
защита от атак 370
известные DoS-атаки 361
Отказ кооперативный 225
Отказ оборудования 127
Отказоустойчивость узла Lightning 154
Отказ тайм-аута 225
Отправка
по сравнению с маршрутизацией 30
Отыскание пути 83, 299
выбор наилучшего пути 301
емкость, остаток и ликвидность 302
комплект протоколов Lightning 299
математика и информатика 302
на основе источника 84
неопределенность остатков 303
отыскание кандидатных путей 312
по сравнению
с маршрутизацией 207
природа задачи и решения 300
простота 304
протокол луковичной
маршрутизации 85
процесс доставки платежа 305
сложность 304
стандарт BOLT 300
строительство канального графа 305
Отыскание пути на основе источника 84

П

Перебалансировка каналов 161
Перебалансировка на основе кругового
маршрута 161
Пересылка платежа
исполнение HTLC-контракта 240
локальные платежи 245
распространение HTLC-
контракта 240
Пересылка платежей
с помощью HTLC-контрактов 230
Переход 251
Петля с помощью подводных
свопов 152
Плагин автопилотный c-lightning 159
Платеж 29
деление 314
доставка 82
Платеж без застревания 278
Платеж внецепной 29
Платеж внутрицепной
внецепное улаживание платежей 216
Платеж застрявший 278
Платеж локальный 245
Платеж многокомпонентный
(MPP) 93, 314
деление платежей 314
метод проб и ошибок в течение
нескольких раундов 316
Платежный секрет (прообраз) 80
Платеж спонтанный keysend 279
Повторная выборка данных. См.
Бутстрап
Подпись
привязка к публичному ключу 390
Подпись цифровая 29, 382
Подсказка маршрутизационная 81
Подсказка о состоянии 197
Подсказка о состоянии запутанная 197
Поиск двоичный 359
Помощник (инсталляция/
конфигурация программного
обеспечения) 131
BTCPay Server 134
myNode 133
RaspiBlitz 131
Umbrel 133
узел Bitcoin против облегченного
узла Lightning 135
Порог. См. Точка разделения
Поток сообщений об установлении
канала 177
сообщение accept_channel 180
сообщение open_channel 179
Префикс человекочитаемый 347
Привязка к абсолютному времени 393
Привязка ко времени уровня
выхода 393
Привязка ко времени уровня
транзакции 393
Привязка к относительному
времени 194, 393

Привязка подписи 222
Привязывание к хешу (секрету) 391
Проводник по сети Lightning 42
Проект узла сети Lightning c-lightning
  инсталлирование c-lightning из исходного кода 108
  инсталлирование необходимых библиотек и пакетов 108
  компилирование исходного кода c-lightning 109
  копирование последней версии исходного кода c-lightning 109
  настройка сети Docker 106
  оперирование контейнерами bitcoind и c-lightning 107
  сборка c-lightning в качестве Docker-контейнера 105
Прообраз (платежный секрет) 80, 210, 217
Протокол безопасности транспортного слоя (TLS) 332
Протокол бронтидный 330, 333
Протокол криптографический 66
Протокол луковичной маршрутизации 85
Протокол одноранговый Lightning для управления каналами 177
Протокол проводной 319
  модернизации уровня строительства канала 329
  проводное фреймирование 320
  слой передачи сообщений в рамках комплекта протоколов Lightning 319
Протокол сегрегированного свидетеля (SegWit) 184
  сообщение funding_created 185
  сообщение funding_signed 186
Протокол сети Lightning
  инновации 374
  отыскание пути 299
  слой передачи сообщений 319
Протокол сети Lightning Protocol счета Lightning 345
Протокол справедливости 33
  доверительные протоколы без посредников 32
  имплементирование атомарных бездоверительных многопереходных платежей 214
  маршрутизация 214
  пример доказательства работы 35
  примитивы безопасности как строительные блоки 34
  реальный пример 33
  свойства 214
Протокол транспортный шифрованный Lightning 330
  Noise_XK 333
  канальный граф как децентрализованная инфраструктура публичных ключей 331
  каркас криптосвязи на основе протокола 333
  комплект протоколов Lightning 330
  нотация рукопожатия и поток протокола 334
  ротация ключей сообщений Lightning 343
  уязвимости/ограничения TLS 332
  элементы 333
Протокол шифрованного транспорта Lightning
  акты рукопожатия 337
  высокоуровневый обзор 334
  инициализация состояния сеанса рукопожатия 337
  рукопожатие в трех актах 335
  шифрование транспортного сообщения 342
Протокол эпидемический 281
  обнаружение одноранговой узла 283
  одноранговый 82
  сообщение channel_announcement 292
  сообщение channel_update 296
  сообщение node_announcement 291
  сообщения 290
Прощупывание канала 358

Р

Работа узла Lightning 42
Расходование двойное 188
Расширение сообщений «Тип–длина–значение» (TLV)
  прямая/обратная совместимость 323
  формат сообщений Protobuf 322
  формат сообщений Protobuf

Расширение сообщений «Тип–длина–значение» (TLV)
расширения сообщений в проводном протоколе 322
Протокол проводной TLV-расширения сообщений 322
Формат сериализации сообщений Protobuf (Протокольные буферы) 322
Резерв канала 200
Рукопожатие 333
акты 337
инициализация состояния сеанса 337
нотация и поток протокола 334

C

Самозагрузка
DNS 284
P2P-узлы 284
Самозагрузка адресов DNS-серверов 284
Самозагрузка одноранговой узла первоначальная 284
Сатоши 95
Своп подводный 151
Связность канала 92
Секрет отзывной 73
Секрет платежный (прообраз) 210, 217
Секрет совместный (ss) 256
Сервер виртуальный частный (VPS) 127
Сеть Lightning (в целом) 28. См. Инновации в сети Lightning
базовые понятия 28
варианты использования и пользователи 39
доверие 89
доверие в децентрализированных сетях 30
доставка платежа 82
кошелек Lightning 41
маршрутизирование платежей по каналам 67
механизм работы 65
мотивация 36
Сеть Lightning (в целом)
масштабирование блочной цепи (блокчейна) 36
определяющие признаки 38
основы платежного канала 66
платежный канал 66
пример 41
покупка чашки кофе 61
Кошелек Lightning
пример 61
справедливость без центральной власти 31
таксономия механизмов модернизации 328
шифрование однорангового обмена информацией 88
Сеть Lightning (пример)
запуск сети 121
конфигурация docker-compose 121
открытие каналов и маршрутизирование платежа 122
применение docker-compose для оркестровки Docker-контейнеров 120
сборка полной сети из различных узлов Lightning 120
Сеть транспортная 302
Система бездоверительная
доверие в децентрализированных сетях 30
Система операционная
безопасность 146
для узла Lightning 136
Скрипт запуска 139
Скрипт мультиподписной 392
Скрипт отвязывающий 390
Скрипт привязки ко времени 393
Скрипт привязывающий 390
привязка к публичному ключу (подпись) 390
Служба фоновая 138
Сокрытие идентичности 334
Сообщение о сбое
луковичная маршрутизация 276
Сообщение о синхронизации канального графа 413
сообщение gossip_timestamp_range 414
сообщение query_channel_range 413
сообщение query_short_chan_ids 413
сообщение reply_channel_range 414
сообщение reply_short_chan_ids_end 413
Сообщение проводного протокола 400
закрытие канала 407
объявление канала 411

операция канала 408
синхронизация канального графа 413
сообщение accept_channel 406
сообщение closing_signed 408
сообщение error 403
сообщение funding_created 406
сообщение funding_locked 407
сообщение funding_signed 407
сообщение init 402
сообщение open_channel 405
сообщение ping 404
сообщение pong 404
сообщение shutdown 408
сообщение update_add_htlc 408
сообщения об оживленности соединения 404
сообщения об ошибках 403
сообщения об установлении соединения 402
структура сообщения 402
типы сообщений 400
финансирование канала 405
Состояние канала
обман и штраф на практике 197
отзыв и рефиксация 196
продвижение вперед 195
резерв канала 200
сообщение commitment_signed 196
сообщение revoke_and_ack 196
Справедливость
обеспечение 31
Сравнение системы Bitcoin и сети Lightning 89
адреса против счетов 79, 89
безразрешительная работа 96
блочная цепь (блокчейн) реестр против судебной системы 94
варьирующиеся комиссионные против объявленных комиссионных 91
выбор выходов против отыскания пути 90
выходы со сдачей 91
доверие и риск контрагента 96
майнинговые комиссионные против маршрутизационных комиссионных 91
минимальный размер платежа сатоши против миллисатоши 95
необратимость и окончательность платежей 96
общая денежная единица 95
общие черты 95
ожидание подтверждений против денежного расчета Lightning 93
открытый источник / открытая система 96
отправка произвольных сумм против ограничений по емкости 93
платежная деятельность асинхронность против синхронности 94
публичные Bitcoin-транзакции против приватных платежей Lightning 92
структуры комиссионных 94
транзакции против платежей 89
Среда разработки
командная строка 98
программно-информационное обеспечение узла Lightning 98
скачивание репозитория книги 99
Стек протоколов 168
Строка командная 98
Схема мультиподписная 392
Схема ослепления 259
Счет Lightning 79, 345
bech32 и сегмент данных 348
дополнительные метаданные 81
кодирование платежного запроса на практике 347
комплект протоколов Lightning 345
платежный хеш/прообраз 80
против Bitcoin-адреса 346
сериализация/интерпретация платежного запроса 347
тегированные поля счета 349
человекочитаемый префикс 347

Т

Тариф комиссионный 91
Темпоральность сети Lightning 367
Теория игр 32
«Тип–длина–значение» (TLV), формат 323
инновации 375
конкретно-прикладные луковичные TLV-записи 279
прямая/обратная совместимость 327

Типическая группа. См. Страта
Тип подписного хеша 383
Точка выходная (выходной идентификатор) 387
Транзакция 30
Транзакция возвратная 182
Транзакция кооперативного закрытия 202
Транзакция фиксационная 70, 193
  HTLC-контракт 232
  асимметричная 190
  во время принудительного закрытия 77
  возвратные транзакции 182
  конкурирующие фиксации 188
  новая фиксация с выходом HTLC-контракта 233
  обман со старыми транзакциями 189
  обновление фиксаций с помощью HTLC-контрактов 229
  отзыв старых транзакций 189
  разделение остатков 187
Транзакция финансовая 58, 69, 181
Транзакция штрафная 78

У

Удаление HTLC-контракта из-за ошибки / истечения срока 244
Узел 29
Узел Bitcoin 135
  инсталляция/конфигурация 137
Узел Bitcoin или узел Lightning
  автоматическая переадресация портов 143
  ручное конфигурирование сети 145
Узел Lightning облегченный 135
Узел демона сети Lightning (LND)
  SCB 150
Узел заключительный 251, 253
Узел изначальный 251
Узел канала одноранговый 56
Узел маршрутизационный 205
Улаживание внецепной против внутрицепного платежа 216
Управление каналами 156
  автопилот 157
  закрытие каналов 161
  открытие исходящих каналов 157
  перебалансировка каналов 161
  получение входящей ликвидности 160
Управление потоком 395
Управление узлом 164
  lndmon 164
  Ride The Lightning (RTL) 164
  ThunderHub 165

Ф

Фраза мнемоническая 49
Фраза первоначальная мнемоническая 49
Фреймирование проводное 320
  высокоуровневая схема 320
  кодировка типа 321
Функция дайджестная 380
Функция однопутная 379

Х

Хранилище данных 129
Хеш 380
Хеш платежный 80, 210, 217
Хеш-функция 29, 380
Хеш-функция криптографическая 380

Ц

Центральность
  сеть Lightning 368
Центральность на основе промежуточности 368
Цепочка транзакций 385
Цепь блочная (блокчейн) 29
  короткий ИД канала 294
  локализация канала в блочной цепи Bitcoin) 293
  масштабирование 36
Цикл проб и ошибок 312, 316

Ш

Шифрование однорангового обмена информацией 88

A

accept_channel, сообщение 180
AD (ассоциированные данные) 265
announce_signatures, сообщение 412
API вызова дистанционных процедур (RPC) 147

B

bech32
  счета Lightning 348

BigSize, целочисленная кодировка 324
«Тип–длина–значение» (TLV), формат ограничения кодировки 324
Bitcoin Core 102
bitcoind, контейнер 102
контейнеры c-lightning 107
контейнеры Eclair 117
контейнеры LND 111
Bitcoin Script 388
HTLC-контракты 217
выполнение 388
инновации, мотивированные вариантами использования сети Lightning 373
использование управления потоком 395
мультиподписной скрипт 392
привязывание к публичному ключу (подпись) 390
привязывающий/отвязывающий скрипт 390
скрипт привязки ко времени 393
скрипты с несколькими условиями 394
Bitcoin-адрес
против счета Lightning 346
Bitcoin-банкомат 52
Bitcoin (система)
DoS-атака 360
инновации, мотивированные вариантами использования сети Lightning 373
ключи и цифровая подпись 378
основы 378
приватные ключи 379
скрипт 388
транзакции 383
хеши 380
цифровая подпись 382
Bitcoin-сущность
кластеризация сущностей 363
межслоевое связывание с узлами Lightning 365
Bitcoin-транзакции 383
Bitcoin-транзакция
выходные точки (выходные идентификаторы) 387
идентификаторы транзакций 386
цепочки транзакций 385
входы и выходы 383
BOLT (Базис технологии Lightning)
документы по стандартам
отыскание пути 300
сериализация/интерпретация платежного запроса Lightning 347
BTCPay Server 134

C

channel_announcement, сообщение 292, 411
валидация 296
канальный граф 305
локализация канала в блочной цепи Bitcoin 293
необъявленные (приватные) каналы 293
одноранговый эпидемический протокол 82
сообщение announce_signatures 412
сообщение channel_update 411
сообщение node_announcement 411
структура сообщения 294
channel_update, сообщение 82, 296, 305, 412
c-lightning, проект узла сети Lightning 105
closing_signed, сообщение 202, 408
commitment_signed, сообщение 196, 409

D

Docker
базовая инсталляция и использование 397
базовые команды 398
инсталлирование 397
сборка контейнера 398
docker-compose
конфигурация 121
оркестровки Docker-контейнеров 120
Docker-контейнер
программно-информационное обеспечение узла Lightning 100
сборка контейнера c-lightning 105
Docker-контейнеры
вывод списка Docker-образов 399
вывод списка оперируемых контейнеров 399
исполнение команды в контейнере 398

контейнер Bitcoin Core 102
оперирование контейнером 398
остановка/запуск контейнера 398
сборка контейнера 398
удаление контейнера по имени 399

E

ECDH (Эллиптическая кривая Диффи–Хеллмана) 258
Eclair, проект узла сети Lightning 116
Docker-контейнер 116
инсталлирование Eclair из исходного кода 118
компилирование исходного кода Eclair 119
копирование исходного кода Eclair 119
оперирование контейнерами bitcoind и Eclair 117

F

funding_created, сообщение 185, 406
funding_locked, сообщение 187, 407
funding_signed, сообщение 186, 407

K

K-из-N, схема 392

L

Lightning-приложение (LApp) 376
Linux
инсталлирование узла Bitcoin или узла Lightning 137
lndmon 164

M

myNode 133

N

node_announcement, сообщение 291, 305, 411
валидирование 292
одноранговый эпидемический протокол 82
структура 291
Noise_XK 333
акты рукопожатия 337
высокоуровневый обзор 334
инициализация состояния сеанса рукопожатия 337
нотация рукопожатия и поток протокола 333
ротация ключей сообщений Lightning 343
рукопожатие в трех актах 335
шифрование транспортного сообщения 342
NVMe (Non-Volatile Memory Express) 131

O

open_channel, сообщение 179

P

P2P-узел
самозагрузка 284
P2WSH-адрес (Pay-to-Witness-Script-Hash) 282
Pay-to-Witness-Script-Hash (P2WSH) 282
PKI (инфраструктура публичных ключей) 331
PoW (Доказательство работы), алгоритм 35
PTLC (Контракт с точечной привязкой ко времени) 215, 279

Q

query_channel_range, сообщение 413
query_short_chan_ids, сообщение 413

R

RaspiBlitz 131
regtest, режим 102
reply_channel_range, сообщение 414
reply_short_chan_ids_end, сообщение 413
revoke_and_ack, сообщение 196, 235, 410
Ride The Lightning (RTL) 164
RTL (Ride The Lightning) 164

S

SCB (статическое резервное копирование каналов) 148
shutdown, сообщение 201
SPHINX Mix, формат. См. Луковичная маршрутизация
SPV (упрощенная верификация платежей) 42
SSD (твердотельный накопитель) 129

T

ThunderHub 165
TLS (протокол безопасности транспортного слоя) 332

Tor (Маршрутизатор луковичный) 144
TxID (идентификаторы транзакций) 386

U

Umbrel 133
update_add_htlc, сообщение 87, 269
update_add_HTLC, сообщение 231
update_fail_htlc, сообщение 409
update_fail_malformed_htlc, сообщение 410
update_fee, сообщение 410
update_fulfill_hltc, сообщение 409
UTXO (неизрасходованные выходы транзакции) 90, 384

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «КТК Галактика» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, пр. Андропова д. 38 оф. 10.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.galaktika-dmk.com**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Андреас Н. Антонопулос,
Олаулува Осунтокун
и Рене Пикхард

Освоение Lightning Network

Протокол второслойной блочной цепи
для мгновенных Bitcoin-платежей

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Логунов А. В.*
Корректор *Синяева Г. И.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 36,56. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com

O'REILLY®

Сеть Lightning (Lightning Network, LN) – это быстро развивающийся второслойный платежный протокол, который работает поверх системы Bitcoin, обеспечивая почти мгновенные транзакции между двумя сторонами. Авторы объясняют, как это усовершенствование позволит перейти на следующий уровень масштабируемости системы Bitcoin, повысив скорость и конфиденциальность при одновременном снижении комиссии. Вы узнаете, как LN может поддерживать гораздо больше транзакций, чем современные финансовые сети.

Издание адресовано разработчикам, системным архитекторам, инвесторам и предпринимателям, желающим лучше понять LN.

- Как сеть Lightning решает проблему масштабирования блочной цепи
- Основы LN, включая кошельки, узлы и способы оперирования одним из них
- Пять слоев протокола сети Lightning
- Платежные каналы Lightning, луковичная маршрутизация и эпидемический протокол обмена сообщениями
- Поиск путей по платежным каналам для транспортировки биткойнов вне цепи от отправителя к получателю
- Документы по стандарту Базиса технологии Lightning (BOLT)

«Этой книги вам будет достаточно для того, чтобы постичь волшебство атомарных и бездоверительных платежей, проходящих через несколько переходных узлов, и разобраться в том, как сеть Lightning воплощает это поверх сети блочных цепей Bitcoin».

Хорхе Лесмес, директор и глобальный руководитель блочной цепи для банковской деятельности, NTT Data

«Это бесценный ресурс, учитывая невероятный опыт авторов, а также тот факт, что они являются архитекторами этой технологии».

Доктор Джефф Флауэрс, профессор Колледжа Сан-Матео

Андреас М. Антонопулос – автор, преподаватель и один из ведущих мировых экспертов по системе Bitcoin и открытой блочной цепи. **Олаолува Осунтокун** – разработчик системы Bitcoin, исследователь и технический директор Lightning Labs. **Рене Пикхардт** – консультант и исследователь в области науки о данных, чья работа над надежной маршрутизацией платежей привела к открытию оптимально дешевых и надежных платежных потоков (так называемых «платежей Пикхардта»).

ISBN 978-5-93700-144-3

Интернет-магазин: www.dmkpress.com

Оптовая продажа: КТК «Галактика»
books@alians-kniga.ru

ДМК ПРЕСС ИЗДАТЕЛЬСТВО
www.дмк.рф

9 785937 001443 >