The Open Network

на основе работы доктора Николая Дурова

26 июля 2021 г.

Аннотация

Цель данной работы - предоставить первое описание блокчейн-платформы TON (The Open Network) и связанных с ней технологий одноранговых сетей, распределенного хранилища и хостинга. Чтобы уменьшить размер этого документа до разумного объема, мы сосредоточимся в основном на уникальных и определяющих функциях платформы TON, которые важны для достижения поставленных целей.

Введение

The Open Network (TON) - это быстрая, безопасная и «криптографически запечатанная» блокчейн-платформа. При необходимости этот сетевой проект может обрабатывать миллионы транзакций в секунду, поэтому TON является столь удобным для пользователя и поставщика услуг. Мы стремимся к тому, чтобы на этой блокчейн-платформе можно было разместить все практически применимые приложения, предлагаемые и задуманные в настоящее время. TON можно описать как огромный распределенный суперкомпьютер или, скорее, как огромный «суперсервер», предназначенный для размещения и предоставления различных услуг.

Данный документ не предоставляет исчерпывающую информацию обо всех деталях реализации проекта. Некоторые особенности могут измениться на этапах разработки и тестирования.

Переведено на русский язык:
EQCLk1384-rYMhauBsAT36YABMN5yOTNbgM29THdANTW4-qK

Содержание

1 Краткое описание компонентов TON

The Open Network (TON) представляет собой комбинацию следующих компонентов:

- Гибкая платформа из нескольких блокчейнов (блокчейн TON; см. Главу 2), способная обрабатывать миллионы транзакций в секунду. Благодаря полными по Тьюрингу смарт-контрактам, обновляемым формальным спецификациям блокчейна, транзакциям стоимости для нескольких криптовалют, поддержке каналов микроплатежей и внецепочным платежным платформам, блокчейн TON представляет некоторые новые и уникальные функции, такие как механизм «самовосстановления» вертикального блокчейна (см. 2.1.17) и мгновенная маршрутизация в гиперкубе (Instant Hypercube Routing, см. 2.4.20), которые обеспечивают быстроту, надежность, закрытость и в то же время самосогласованность платформы;
- Одноранговая сеть (сеть TON P2P или просто сеть TON; см. Главу 3), используемая для доступа к блокчейну TON, отправки возможных транзакций и получения обновлений только о тех сегментах блокчейна, которые интересуют клиента (например, сегменты, которые связаны с учетными записями клиента и смарт-контрактами), а также способная поддерживать произвольные распределенные службы, независимо от того, связаны они с блокчейном или нет;
- Технология распределенного хранения файлов (TON Storage; см. 4.1.7) в сети TON, используемая блокчейном TON для хранения архивных копий блоков и данных о состоянии (зафиксированных снимков состояния), а также доступная для хранения произвольных файлов пользователей или других служб, работающих на платформе (технология доступа подобная торрент-сети);
- Уровень сетевого посредника/анонимайзера (TON Proxy; см. 4.1.10 и 3.1.6), аналогичный 12P (Invisible Internet Project), при необходимости используемый для сокрытия идентичности и IP-адресов узлов сети TON (например, узлов, с которых совершаются транзакции из учетных записей с большим количеством криптовалюты, или высокоуровневые узлы валидатора блокчейна, для которых необходимо скрыть их точный IP-адрес и географическое положение в качестве меры против DDoS-атак);
- Распределенная хэш-таблица на основе варианта Kademlia (TON DHT; см. 3.2), используемая в качестве «торрент-трекера» для TON Storage (см. 3.2.10), «локатора входного туннеля» для TON Proxy (см. 3.2.14), а также в качестве средства поиска услуг для сервиса TON Services (см. 3.2.12);
- Платформа для произвольных сервисов (TON Services; см. Главу 4), находящихся и доступных при помощи TON Network и TON Proxy, и имеющих формализованные интерфейсы (см. 4.3.14), которые обеспечивают взаимодействие с приложениями в браузере или смартфоне. Эти формальные интерфейсы и постоянные точки входа в сервисы могут быть опубликованы в блокчейне TON (см. 4.3.17). Фактические узлы, предоставляющие услуги в любой момент времени, можно будет просмотреть через TON DHT, начиная с информации, опубликованной в блокчейне TON (см. 3.2.12). Службы могут создавать смарт-контракты в блокчейне TON и обеспечивать некоторые гарантии для своих клиентов (см. . 4.1.6);
- TON DNS (см. 4.3.1) - сервис для присвоения удобочитаемых имен учетным записям, смарт-контрактам, службам и сетевым узлам;
- TON Payments (см. Главу 5) - платформа для микроплатежей, каналы микроплатежей и сеть каналов микроплатежей. Эту платформу можно использовать для быстрых внецепочных транзакций стоимости, а также для оплаты услуг, предоставляемых TON Services.
- TON будет обеспечивать легкую интеграцию со сторонними приложениями для обмена сообщениями и социальными сетями, что сделает технологию

блокчейна и распределенных сервисов доступными для обычных пользователей (см. 4.3.23), а не только для небольшого количества первых пользователей криптовалюты.

Несмотря на то, что блокчейн TON является ядром проекта TON, а другие компоненты рассматриваются как вспомогательные сервисы для блокчейна, сами по себе они обладают полезными и интересными функциями. В совокупности они позволяют разместить на платформе более универсальные приложения, если сравнивать с использованием обычного блокчейна TON (см. 2.9.13 и 4.1).

2 Блокчейн TON

Начнем с описания блокчейна The Open Network (TON), который является ключевым компонентом проекта. Будет использоваться подход «сверху вниз» - сначала мы дадим общее описание всей структуры, а затем предоставим более подробную информацию о каждом компоненте.

Для простоты мы будем использовать термин «блокчейн TON», хотя в принципе несколько экземпляров этого протокола блокчейна могут работать независимо друг от друга (например, в результате хард-форков). Мы будем рассматривать только один блокчейн TON.

2.1. Блокчейн TON как структура, состоящая из 2D-блокчейнов

Блокчейн TON на самом деле представляет собой несколько блокчейнов (в нем используются два блокчейна - этот момент будет прояснен позже в п. 2.1.17), потому что ни один блокчейн не способен достичь нашей цели по обработке миллионов транзакций в секунду, в отличие от текущего стандарта в несколько десятков транзакций в секунду.

2.1.1. Типы блокчейнов. Концепция блокчейна TON состоит из четырёх уровней:

- Уникальная единая корневая цепь (или сокращенно мастерчейн), содержащая общую информацию о протоколе и текущих значениях его параметров, наборе валидаторов и их стейках, наборе текущих активных воркчейнов
- (воркчейнов) и соответствующих «шардов», и, что наиболее важно, набор хэшей самых последних блоков всех воркчейнов и шардчейнов.
- Несколько (до 2^{32}) воркчейнов (или сокращенно воркчейнов), которые на самом деле являются «рабочими лошадками», содержащими транзакции стоимости и транзакции смарт-контрактов. Разные воркчейны могут иметь разные «правила», то есть разный формат адресов учетной записи, разный формат транзакций, разные криптовалюты, разные виртуальные машины для обработки кода смарт-контрактов и т.д. Однако все они должны удовлетворять определенным основным критериям совместимости, чтобы обеспечить взаимодействие между различными воркчейнами и сделать его относительно простым. В этом отношении воркчейны в TON аналогичны EOS (см. 2.9.7) и PolkaDot (см. 2.9.8), где блокчейн также является гетерогенным (см. 2.8.8).
- В свою очередь, каждый воркчейн подразделяется на шардчейны (которых максимум 2 в 60 степени), имеющие те же правила и формат блока, что и сам воркчейн, но отвечающие только за подмножество учетных записей, в зависимости от нескольких первых (наиболее значимых) битов адреса аккаунта. Другими словами, в систему встроена форма сегментирования (см. 2.8.12). Поскольку во всех этих шардчейнах используется общий формат блоков и правила, блокчейн TON в этом отношении является гомогенным (см. 2.8.8), аналогично к тому, что обсуждалось в одном из предложений по запечатыванию сети Ethereum[1].

- Каждый блок в шардчейне (и в мастерчейне) на самом деле является не просто блоком, а небольшим блокчейном. Обычно этот «блокчейн» или «вертикальный блокчейн» состоит ровно из одного блока, и тогда можно подумать, что это всего лишь соответствующий блок шардчейна (в этой ситуации также называемого «горизонтальной цепью»). Однако, если возникает необходимость исправить некорректные блоки шардчейна, новый блок фиксируется в «вертикальном блокчейне», содержащем либо замену недопустимого блока «горизонтального блокчейна», либо «характеристики различия блоков», содержащие только описание тех частей предыдущей версии этого блока, которые необходимо изменить. Это специфичный для TON механизм замены обнаруженных недопустимых блоков без создания истинного форка всех задействованных шардчейнов (более подробно этот процесс будет объясняться в п. 2.1.17). На данный момент следует просто отметить, что каждый шардчейн (и мастерчейн) - это не обычный блокчейн, а цепь блокчейнов (2D-блокчейн или просто 2-блокчейн).

2.1.2. Парадигма бесконечного шардинга. Практически все решения по шардингу блокчейнов являются «нисходящими»: сначала представляется единый блокчейн, а затем обсуждаются способы его разделения на несколько взаимодействующих шардчейнов для повышения производительности и достижения масштабируемости.
«Восходящий» подход к шардингу в платформе TON объясняется следующим образом.
Представьте, что шардинг доведено до крайности, так что в каждом шардчейне остается ровно одна учетная запись или смарт-контракт. При этом имеется огромное количество «цепочек учетных записей», каждая из которых описывает состояние и переходы между состояниями только одной учетной записи и отправляет на другие цепи сообщения о переносе стоимости и информации.
Конечно, использовать сотни миллионов блокчейнов – это непрактично, причем обновления (т. е. новые блоки) в каждом блокчейне обычно довольно редко появляются. Чтобы обеспечить более высокую эффективность системы, мы группируем эти «цепочки учетных записей» в «шардчейны», так что каждый блок шардчейна по существу представляет собой набор блоков цепочек учетных записей, которые были назначены этому шарду.
Таким образом, «цепочки учетных записей» существуют только исключительно виртуально или логически внутри «шардчейнов».
Мы называем эту перспективу парадигмой бесконечного шардинга, которая объясняет многие проектные решения для блокчейна TON.

2.1.3. Сообщения. Мгновенная маршрутизация в гиперкубе (Instant Hypercube Routing). В парадигме бесконечного шардинга каждая учетная запись (или смарт-контракт) рассматривается так, как если бы она сама находилась в отдельном шардчейне. Тогда единственный способ, при помощи которого одна учетная запись может повлиять на состояние другой учетной записи, - это отправить ей сообщение (это особый случай так называемой модели акторов, причем в роли акторов выступают учетные записи; см. 2.4.2). Следовательно, система передачи сообщений между учетными записями (и шардчейнами, поскольку исходная и конечная учетные записи, как правило, расположены в разных шардчейнах) имеет первостепенное значение для запечатываемой системы, такой как блокчейн TON. Фактически, новая функция блокчейна TON, называемая Instant Hypercube Routing (см. 2.4.20), обеспечивает доставку и обработку сообщений, созданных в блоке одного шардчейна, в следующий блок целевого шардчейна, независимо от общего количества шардчейнов в системе.

2.1.4. Количество мастерчейнов, воркчейнов и шардчейнов. Блокчейн TON содержит ровно один мастерчейн. Однако система потенциально может вместить до 2^{32} воркчейнов, каждый из которых будет разделен на 2^{60} сегментов.

2.1.5. Воркчейны могут быть виртуальными, а не настоящими блокчейнами. Поскольку воркчейн обычно подразделяется на шардчейны, существование воркчейна является «виртуальным» - это не настоящий блокчейн в смысле общего определения, приведенного в 2.2.1 ниже, а просто набор шардчейнов. Если воркчейну соответствует только один шардчейн, этот уникальный шардчейн может быть идентифицирован с воркчейном, который в этом случае становится «настоящим» блокчейном (как минимум на некоторое время), таким образом, приобретая внешнее сходство с обычной структурой, в которой используется один блокчейн. Однако парадигма бесконечного шардинга (см. 2.1.2) говорит нам, что такое сходство является только внешним: это просто совпадение, что потенциально огромное количество «цепочек учетных записей» может быть временно сгруппировано в один блокчейн.

2.1.6. Идентификация воркчейнов. Каждый воркчейн идентифицируется на основании своего номера или идентификатора воркчейна (workchain_id : uint_{32}), который представляет собой 32-битное целое число без знака. Воркчейны создаются специальными транзакциями в мастерчейне, определяющими ранее неиспользованный идентификатор воркчейна и формальное описание воркчейна, которое является достаточным для взаимодействия этой воркчейна с другими воркчейнами, а также для поверхностной проверки блоков этого воркчейна.

2.1.7. Создание и активация новых воркчейнов. Создание нового воркчейна может быть инициировано практически любым членом сообщества, готовым заплатить (высокие) комиссии за транзакции в мастерчейне, необходимые для публикации формальной спецификации нового воркчейна. Однако для того, чтобы новый воркчейн стал активным, требуется консенсус в две трети валидаторов, поскольку им необходимо будет обновить свое программное обеспечение для обработки блоков нового воркчейна и сигнализировать о своей готовности работать с новым воркчейном при помощи специальных транзакций в мастерчейне. Сторона, заинтересованная в активации нового воркчейна, может предоставить валидаторам некоторый стимул для поддержки нового воркчейна посредством вознаграждений, распределяемых при помощи смарт-контракта.

2.1.8. Идентификация шардчейнов. Каждый шардчейн идентифицируется парой (w, s) = (workchain_id, shard_prefix), где workchain_id: uint_{32} идентифицирует соответствующий воркчейн, а shard_prefix: $2^{0\ldots60}$ - это строка битов длиной не более 60, определяющая подмножество учетных записей, за которые отвечает этот шардчейн. А именно, все учетные записи с идентификатором account_id, начинающимся с префикса shard_prefix (т. е. имеющие префикс shard_prefix в качестве наиболее значимых битов), будут назначены этому шардчейну.

2.1.9. Идентификация цепочек учетных записей. Напомним, что цепочки учетных записей существуют только виртуально (см. 2.1.2). Однако у них есть естественный идентификатор, а именно (workchain_id, account_id), поскольку любая цепочка учетных записей содержит информацию о состоянии и обновлениях ровно одной учетной записи (различие между обычной учетной записью или смарт-контрактом здесь не имеет значения).

2.1.10. Динамическое разделение и объединение шардчейнов (см 2.7). В менее сложной системе может использоваться статический шардинг - например, с использованием первых восьми битов идентификатора account_id для выбора одного из 256 предопределенных шардов.
Важной особенностью блокчейна TON является то, что в нем реализуется динамический шардинг – это означает, что количество шардов не является фиксированным. Вместо этого шард (w, s) можно автоматически разделить на несколько новых шардов (w,s.0) и (w, s.1), если выполняются некоторые формальные условия (по сути, если транзакционная нагрузка на исходный шард достаточно высока в течение продолжительного периода времени). И наоборот, если нагрузка остается слишком низкой в течение некоторого периода времени, шарды (w, s.0) и (w, s.1) могут быть автоматически объединены в исходный шард (w, s).
Изначально для воркчейна w создается только один шард (w, θ). Позже при необходимости он может быть разделен на несколько шардов (см. 2.7.6 и 2.7.8).

2.1.11. Исходный воркчейн или Workchain Zero. Несмотря на то, что в системе может быть создано до 2^{32} воркчейнов с соответствующими конкретными правилами и транзакциями, изначально задается только один воркчейн (workchain_ id = 0). Этот воркчейн, называемый Workchain Zero или исходный воркчейн, используется для работы со смарт-контрактами TON и перевода монет TON (см. Приложение). Скорее всего, для большинства приложений потребуется только Workchain Zero. Шардчейны базового воркчейна будут называться исходными шардчейнами.

2.1.12. Интервалы генерации блоков. Ожидается, что новый блок будет генерироваться в каждом шардчейне и мастерчейне примерно раз в пять секунд. Это приведет к разумно малому времени подтверждения транзакции. Новые блоки всех шардчейнов будут генерироваться примерно одновременно; новый блок мастерчейна создается примерно на одну секунду позже, поскольку он должен содержать хэши последних блоков всех шардчейнов.

2.1.13. Использование мастерчейна для создания тесной взаимосвязи между воркчейнами и шардчейнами. После включения хэша блока шардчейна в блок мастерчейна этот блок шардчейна и все предшествующие ему блоки считаются «эталонными». Это означает, что все последующие блоки всех шардчейнов могут на них ссылаться как на нечто фиксированное и неизменяемое. Фактически, каждый новый блок шардчейна содержит хэш самого последнего блока мастерчейна, поэтому новый блок считает все блоки шардчейна, на которые ссылается этот блок мастерчейна, неизменяемыми.
По сути, это означает, что транзакция или сообщение, зафиксированное в блоке шардчейна, можно безопасно использовать в следующих блоках других шардчейнов без необходимости ожидания, к примеру, двадцати подтверждений (т. е. двадцать блоков, сгенерированных после исходного блока в том же блокчейне) передпересылкой сообщения или выполнением других действий на основе предыдущей транзакции, как это часто бывает в большинстве предлагаемых системах «со слабой связью» (см. 2.8.14), таких как EOS. Эта способность использовать транзакции и сообщения в других сегментах сети всего через пять секунд после их фиксации является одной из причин, по которой мы считаем, что наша система «с сильной связью» первой в своем роде сможет обеспечить беспрецедентную производительность (см. 2.8. 12 и 2.8.14).

2.1.14. Хэш блока мастерчейна как глобальное состояние. Согласно п. 2.1.13, хэш последнего блока мастерчейна полностью определяет общее состояние системы с

точки зрения внешнего наблюдателя. Поэтому нет необходимости следить за состоянием всех шардчейнов по отдельности.

2.1.15. Генерация новых блоков валидаторами (см. 2.6). В блокчейне TON используется метод защиты Proof-of-Stake (подтверждение доли, PoS) для создания новых блоков в шардчейнах и мастерчейнах. Это означает, что существует, скажем, до нескольких сотен валидаторов - специальных узлов, которые внесли стейки (большие количества монет TON) с помощью специальной транзакции мастерчейна, чтобы иметь право на создание и проверку новых блоков.
Затем детерминированным псевдослучайным образом каждому сегменту (w, s) назначается меньшее подмножество валидаторов, которое изменяется примерно каждые 1024 блока. Это подмножество валидаторов предлагает и достигает консенсуса в отношении того, каким будет следующий блок шардчейна, путем сбора подходящих предлагаемых транзакций от клиентов в новые допустимые блоки-кандидаты. Для каждого блока существует псевдослучайно выбранный порядок валидаторов, который позволяет определить, чей блок-кандидат имеет наивысший приоритет для фиксации в каждой очереди.
Валидаторы и другие узлы проверяют действительность предложенных блоков-кандидатов; если валидатор подписывает недопустимый блок-кандидата, он может быть автоматически наказан потерей части своей стейка или всего стейка, либо лишением функций валидатора на некоторое время. После этого валидаторы должны прийти к консенсусу по выбору следующего блока. По сути, это достигается с помощью эффективного протокола консенсуса BFT (задача византийских генералов; см. 2.8.4), подобного PBFT [4] или Honey Badger BFT [11]. При достижении
консенсуса создается новый блок, после чего валидаторы делят между собой комиссию за транзакции, включенные в транзакцию, плюс некоторые вновь созданные («отчеканенные») монеты.
Каждый валидатор может быть избран для участия в нескольких подмножествах валидаторов; в этом случае предполагается, что все алгоритмы проверки и согласования будут выполняться параллельно.
После того, как все новые блоки гардчейна были сгенерированы, либо истекло время ожидания, создается новый блок мастерчейна, включая хэши последних блоков всех сегментов шардчейнов. Эта задача реализуется на основе консенсуса BFT всех валидаторов[2].
Более подробная информация о подходе TON PoS и соответствующей экономической модели представлена в разделе 2.6.

2.1.16. Форки мастерчейна. Сложность нашей предлагаемой системы «с сильной связью» заключается в том, что переключение на другой форк в мастерчейне почти обязательно потребует переключения на другой форк как минимум в некоторых из шардчейнов.
С другой стороны, до тех пор, пока в мастерчейне нет форков, форки в шардчейне невозможны, поскольку никакие блоки в альтернативных форках шардчейнов не могут стать «эталонными», если их хэши включены в блок мастерчейна.
Общее правило состоит в том, что если блок B’ мастерчейна является предшественником B, то B’ включает хэш HASH(B’w,s) из B’w,s блока шардчейна (w,s), а B включает хэш HASH(Bw,s), тогда B’w,s должен быть предшественником Bw,s. В противном случае блок B мастерчейна является невалидным.
Ожидается, что форки мастерчейна будут редким, практически невозможным явлением, поскольку в парадигме BFT, принятой в блокчейне TON, они могут возникать только в случае некорректного поведения большинства валидаторов (см. 2.6.1 и 2.6.15), что повлечет за собой значительную потерю нарушителями их стейков. Следовательно, не следует ожидать настоящих форков в шардчейнах. Вместо этого, в

случае обнаружения недопустимого блока шардчейна, он будет исправлен с помощью механизма «вертикального блокчейна» (2-блокчейна) (см. 2.1.17), который может достичь этой цели без создания форков «горизонтального блокчейна» (т. е., шардчейна). Тот же механизм можно использовать и для исправления некритических ошибок в блоках мастерчейна.

2.1.17. Исправление неверных блоков шардчейна. Обычно фиксируются только действительные блоки шардчейна, потому что валидаторы, назначенные для шардчейна, должны достичь консенсуса в две трети при решении задачи византийских генералов, прежде чем может быть зафиксирован новый блок. Однако система должна обеспечивать обнаружение ранее зафиксированных недопустимых блоков и их исправление.

Конечно, при обнаружении невалидного блока шардчейна - либо валидатором (не обязательно назначенным для этого шардчейна), либо «фишерменом» (любым узлом системы, сделавшим определенный депозит, чтобы иметь возможность поднимать вопросы о валидности блока; см. 2.6.4) - заявление о невалидности и соответствующие доказательства передаются в мастерчейн, а валидаторы, подписавшие невалидный блок, наказываются потерей части своего стейка и/или временно лишаются права выполнять функции валидатора (последняя мера важна в том случае, если злоумышленник украл закрытые ключи подписи надежного валидатора).

Однако этого недостаточно, потому что общее состояние системы (блокчейна TON) оказывается недопустимым из-за ранее зафиксированного недопустимого блока в шардчейне. Этот недопустимый блок необходимо заменить более новой действующей версией.

В большинстве систем эта задача достигается путем «отката» к последнему блоку перед недопустимым блоком в данном шардчейне и последним блокам, не затронутым сообщениями, передаваемыми из недопустимого блока в каждый из других блокчейнов, а также созданием нового форка из этих блоков. Недостатком этого подхода является то, что внезапно откатывается большое количество корректных и подтвержденных транзакций, и неясно, будут ли они вообще включены в систему позже.

Блокчейн TON решает эту проблему, превращая каждый «блок» каждого шардчейна и мастерчейна («горизонтальные блокчейны») в небольшой блокчейн («вертикальный блокчейн»), содержащий разные версии этого «блока» или их «различия». Обычно вертикальный блокчейн состоит ровно из одного блока, а шардчейн имеет вид классического блокчейна. Однако, как только невалидность блока будет подтверждена и зафиксирована в блоке мастерчейна, к «вертикальному блокчейну» недопустимого блока разрешается добавить новый блок в вертикальном направлении, после чего будет выполнена замена или изменение недопустимого
блока. Новый блок генерируется текущим подмножеством валидаторов для рассматриваемого шардчейна.

Для нового «вертикального» блока действуют довольно строгие правила. В частности, если виртуальный «блок цепочки учетных записей» (см. 2.1.2), содержащийся в недопустимом блоке, действителен сам по себе, он должен быть оставлен без изменений со стороны нового вертикального блока.

Как только новый «вертикальный» блок фиксируется поверх недопустимого блока, его хэш публикуется в новом блоке мастерчейна (или, скорее, в новом «вертикальном» блоке, находящемся над исходным блоком мастерчейна, где был первоначально опубликован хэш недопустимого блока шардчейна),

[2] На самом деле, двух третей голосов достаточно для достижения консенсуса, однако система старается собрать как можно больше подписей.

а изменения распространяются дальше на любые блоки шардчейна, относящиеся к предыдущей версии этого блока (например, блоки, которые получили сообщения от некорректного блока).
Исправление этой проблемы достигается путем фиксации новых «вертикальных» блоков в вертикальных блокчейнах для всех блоков, ранее ссылающихся на «некорректный» блок. Новые вертикальные блоки будут ссылаться на самые последние (исправленные) версии. Опять же, строгие правила запрещают изменять цепочки учетных записей, которые не были фактически затронуты (т. е. получают те же сообщения, что и в предыдущей версии). Таким образом, исправление некорректного блока создаёт «волны», которые в конечном итоге распространяются к самым недавним блокам всех затронутых шардчейнов; эти изменения также отражены в новых "вертикальных" блока мастерчейна.
Когда «рябь перезаписи истории» достигает самых последних блоков, новые блоки шардчейна генерируются только в одной версии и становятся преемниками только новейших версий блоков. Это означает, что они с самого начала будут содержать ссылки на правильные (самые последние) вертикальные блоки.
Состояние мастерчейна неявно определяет карту, преобразующую хэш первого блока каждого «вертикального» блокчейна в хэш последней версии. Это позволяет клиенту идентифицировать и определять местонахождение любого вертикального блокчейна по хэшу самого первого (и обычно единственного) блока.

2.1.18. Монеты TON и мультивалютные воркчейны. Блокчейн TON поддерживает до 2^{32} различных «криптовалют», «монет» или «токенов», которые различаются 32-битным идентификатором currency_id. Новые криптовалюты добавляются с помощью специальных транзакций в мастерчейне. Каждый воркчейн имеет базовую криптовалюту и может иметь несколько дополнительных криптовалют.
Существует одна специальная криптовалюта с идентификатором currency_id = 0, а именно монета TON (см. Приложение A). Это основная криптовалюта исходного блокчейна. Она также используется в качестве комиссии за транзакции и для определения стейков валидаторов.
В принципе, другие воркчейны могут собирать комиссию за транзакции в других токенах. Для этого необходимо предоставить смарт-контракт для автоматического преобразования этих комиссий за транзакции в монеты TON.

2.1.19. Обмен сообщениями и транзакции стоимости. Шардчейны, принадлежащие к одному воркчейну или разным воркчейнам, могут отправлять сообщения друг другу. Несмотря на то, что точная форма разрешенных сообщений зависит от принимающего воркчейна и принимающей учетной записи (смарт-контракт), существуют некоторые общие поля, которые делают возможным обмен сообщениями между воркчейнами. В частности, к каждому сообщению может быть прикреплена некоторая стоимость в виде определенного количества монет TON и/или других зарегистрированных криптовалют, при условии, что они объявлены принимающим воркчейном как приемлемые криптовалюты.
Самая простая форма такого обмена сообщениями - это перенос стоимости из одной учетной записи (обычно не смарт-контракта) в другую учетную запись.

2.1.20. Виртуальная машина TON. Виртуальная машина TON, также сокращенно TON VM или TVM, - это виртуальная машина, используемая для выполнения кода смарт-контракта в мастерчейне и в исходном воркчейне. В других воркчейнах могут использоваться другие виртуальные машины вместе с TVM или вместо TVM.
Ниже приведены некоторые особенности виртуальной машины TON. Они обсуждаются далее в 2.3.12, 2.3.14 и других разделах.

•

- TVM представляет все данные как набор ячеек (TVM) (см. 2.3.14). Каждая ячейка содержит до 128 байтов данных и до 4 ссылок на другие ячейки. Применение концепции «все есть мешок ячеек» (см. 2.5.14) позволяет TVM работать со всеми данными, относящимися к блокчейну TON, включая блоки и глобальное состояние блокчейна, если это необходимо.
- TVM может работать со значениями произвольных алгебраических типов данных (см. 2.3.12), представленными в виде деревьев или ориентированных ациклических графов из ячеек TVM. Однако виртуальная машина TON не зависит от существования алгебраических типов данных, она просто работает с ячейками.
- TVM имеет встроенную поддержку хэш-карт (см. 2.3.7).
- TVM является стековой вычислительной машиной. В стеке TVM хранятся либо 64-битные целые числа, либо ссылки на ячейки.
- Поддерживаются 64-битные, 128-битные и 256-битные арифметические операции. Все n-битные арифметические операции бывают трех видов: для целых чисел без знака, для целых чисел со знаком и для целых чисел по модулю 2^n (в последнем случае автоматический контроль переполнения отсутствует).
- TVM обеспечивает преобразование целых чисел без знака и со знаком из n-битн в m-бит для всех $0 \leq m, n \leq 256$ с контролем переполнения.
- По умолчанию для всех арифметических операций выполняется контроль переполнения, что значительно упрощает разработку смарт-контрактов.
- TVM выполняет арифметические операции «умножить, затем сдвинуть» и «сдвинуть и разделить» с промежуточными значениями, вычисленными в большем целочисленном типе данных; это упрощает выполнение арифметических операций с фиксированной точкой.
- TVM предлагает поддержку строк битов и строк байтов.
- Присутствует поддержка 256-битное шифрование на основе эллиптических кривых (ECC) для некоторых предопределенных кривых, включая Curve25519.
- Также присутствует поддержка пар Вейля для некоторых эллиптических кривых, необходимая для быстрой реализации zk-SNARK.
- Присутствует поддержка популярных хэш-функций, в том числе SHA256.
- TVM может работать с доказательствами Меркла (см. 5.1.9),
- TVM предлагает поддержку «больших» или «глобальных» смарт-контрактов. В таких смарт-контрактах должна быть информация о шардинге (см. 2.3.18 и 2.3.16). В обычных (локальных) смарт-контрактах может отсутствовать информация о шардинге.
- TVM поддерживает замыкания.
- На TVM может быть легко реализована машина сокращения графов (spineless tagless G-machine) [13].

В дополнение к «сборке TVM» могут быть разработаны несколько высокоуровневых языков программирования. Все эти языки будут статического типа и будут поддерживать алгебраические типы данных. Мы предполагаем следующие возможности:

- Императивный язык, подобный Java, в котором каждый смарт-контракт похож на отдельный класс.
- Функциональный язык для ленивых вычислений (Haskell).
- Функциональный язык для немедленных вычислений (ML).

2.1.21. Настраиваемые параметры. Важной особенностью блокчейна TON является то, что многие его параметры являются конфигурируемыми. Это означает, что они являются частью состояния мастерчейна и могут быть изменены с помощью определенных транзакций предложения/голосования/результатов в мастерчейне без необходимости форков. Для изменения таких параметров в пользу предложения

должны быть собраны две трети голосов валидаторов и более половины голосов всех других участников, которые захотят принять участие в процессе голосования.

2.2. Общие сведения о блокчейнах

2.2.1. Общее определение блокчейна. В целом, любой (настоящий) блокчейн представляет собой последовательность блоков, причем каждый блок B содержит ссылку BLK-PREV(B) на предыдущий блок (обычно хэш предыдущего блока включается в заголовок текущего блока) и список транзакций. Каждая транзакция описывает некоторую трансформацию состояния глобального блокчейна. Транзакции, перечисленные в блоке, применяются последовательно для вычисления нового состояния, начиная со старого состояния, которое является результирующим состоянием после оценки предыдущего блока.

2.2.2. Применимость к блокчейну TON. Напомним, что блокчейн TON - это не настоящий блокчейн, а система из 2 блокчейнов (т. е. это цепочка блокчейнов; см. 2.1.1), поэтому вышесказанное не применимо напрямую к блокчейну TON. Однако мы начнем с общего определения настоящего блокчейна, чтобы использовать его в качестве строительных блоков для наших более сложных конструкций.

2.2.3. Фактический блокчейн и тип блокчейна. Слово «блокчейн» часто используется для обозначения как общего типа блокчейна, так и конкретных экземпляров блокчейна, определяемых как последовательности блоков, удовлетворяющих определенным условиям. Например, в п. 2.2.1 приведено описание экземпляра блокчейна.
Таким образом, тип блокчейна обычно является «подтипом» типа Block* списков (т. е. конечных последовательностей) блоков, состоящих из тех последовательностей блоков, которые удовлетворяют определенным условиям совместимости и действительности:

$$Blockchain \subset Block^* \qquad (1)$$

Лучшим способом определения блокчейна было бы сказать, что блокчейн - это тип зависимой пары, состоящий из пар (B, v), где первый компонент B: Block* имеет тип Block* (т. е. список блоков), а второй компонент v: isValidBc(B) является доказательством или свидетельством действительности B. Таким образом,

$$Blockchain \equiv \Sigma_{(\mathbb{B}:Block^*)} isValidBc(\mathbb{B}) \qquad (2)$$

Здесь мы используем обозначения зависимых сумм типов, взятые из [16].

2.2.4. Теория зависимых типов, Coq и TL. Обратите внимание, что здесь мы используем теорию зависимых типов (теория Мартина-Лёфа), аналогичную теории, которая используется для автоматического доказательства Coq[3]. Упрощенная версия теории зависимых типов также используется в языке TL (Type Language)[4], который будет использоваться в формальной спецификации блокчейна TON для описания сериализации всех структур данных и расположения блоков, транзакций и т.п.
Фактически, теория зависимых типов предоставляет необходимую формализацию того, что такое доказательство, а такие формальные доказательства (или их сериализации) могут оказаться полезными, когда, например, нужно предоставить доказательство невалидности для определенного блока.

2.2.5. Язык TL (Type Language). Поскольку язык TL (Type Language) будет использоваться в формальных спецификациях блоков TON, транзакций и сетевых датаграмм, необходимо краткое обсуждение особенностей этого языка.

TL - это язык, подходящий для описания зависимых алгебраических типов, которым разрешено иметь числовые (натуральные) и типовые параметры. Каждый тип описывается с помощью нескольких конструкторов. У каждого конструктора имеется (человекочитаемый) идентификатор и имя, которое представляет собой битовую строку (по умолчанию 32-битное целое число). Кроме того, определение конструктора содержит список полей с их типами.
Набор определений конструкторов и типов называется TL-схемой. Обычно он хранится в одном или нескольких файлах с расширением .tl.
Важной особенностью TL-схем является то, что они определяют однозначный способ сериализации и десериализации значений (или объектов) определенных алгебраических типов. А именно, когда необходимо выполнить сериализацию значения в поток байтов, сначала выполняется сериализация имени конструктора, используемого для этого значения. Далее следуют рекурсивно вычисляемые сериализации каждого поля.
Описание предыдущей версии TL, подходящей для сериализации произвольных объектов в последовательности 32-битных целых чисел, доступно по адресу *https://core.telegram.org/mtproto/TL*. Новая версия языка TL, названная TL-B, разрабатывается с целью описания сериализации объектов, используемых платформой TON. Эта новая версия может выполнять сериализацию объектов в потоки байтов и даже битов (а не только в 32-битные целые числа) и обеспечивает поддержку сериализации в дерево ячеек TVM (см. 2.3.14). Описание TL-B будет частью формальной спецификации блокчейна TON.

2.2.6. Блоки и транзакции как операторы преобразования состояний. Обычно в любом блокчейне с оператором (type) Blockchain связан оператор глобального состояния (type) State и транзакция (type) Transaction. Семантика блокчейна в значительной степени определяется функцией приложения транзакции:

$$ev\_trans' : Transaction \times State \to State^{?} \qquad (3)$$

Здесь X? обозначает MAYBE X, результат применения монады MAYBE к типу X. Это похоже на использование X * для LiSTX. По существу, значение типа X? является либо значением типа X, либо специальным значением $\perp$, указывающим на отсутствие фактического значения (вспомните о нулевом указателе). В нашем случае мы используем State? вместо State в качестве типа результата, потому что транзакция может быть недействительной, если вызывается из определенных исходных состояний (вспомните о попытке снять со счета больше денег, чем есть на самом деле). Мы могли бы предпочесть чистую версию ev trans':

$$ev\_trans : Transaction \to State \to State^{?} \qquad (4)$$

Поскольку блок - это, по сути, список транзакций, функция оценки блока

$$ev\_block : Block \to State \to State^{?} \qquad (5)$$

может быть получена из ev_trans.
Она принимает блок B: Block и предыдущее состояние блокчейна s: State (которое может включать хэш предыдущего блока) и вычисляет следующее состояние блокчейна s' = ev_block(B)(s): State, которое является истинным состоянием или специальным значением $\perp$, указывающим, что следующее состояние не может быть вычислено (т. е. что блок является невалидным при оценке из заданного начального состояния - например, блок включает транзакцию, которая свидетельствует о попытке списать средства с пустого счета).

2.2.7. Порядковые номера блоков. Ссылка на каждый блок B в блокчейне может осуществляться по его порядковому номеру BLK-SEQNO (B), начиная с нуля для

самого первого блока и в порядке увеличения на единицу при переходе к следующему блоку. Более формально это выглядит следующим образом:

$$\text{BLK-SEQNO}(B) = \text{BLK-SEQNO}\big(\text{BLK-PREV}(B)\big) + 1 \qquad (6)$$

Обратите внимание, что порядковый номер не обеспечивает однозначную идентификацию блока при наличии форков.

2.2.8. Блокировка хэшей. Другой способ обращения к блоку B - это его хэш BLK-HASH (B), который на самом деле является хэшем заголовка блока B (однако заголовок блока обычно содержит хэши, которые зависят от всего содержимого блока B). Предполагая, что для используемой хэш-функции нет хэш-конфликтов (либо они как минимум очень маловероятны), блок однозначно идентифицируется по его хэш-функции.

2.2.9. Допущение хэша. Во время формального анализа алгоритмов блокчейна мы предполагаем отсутствие хэш-конфликтов для используемой k-битной хэш-функции HASH: Bytes * → 2^k:

$$\text{HASH}(s) = \text{HASH}(s') \Rightarrow s = s' \quad \text{for any } s, s' \in Bytes^* \qquad (7)$$

Здесь Bytes = {0 ... 255} = 2^8 - это тип байтов или набор всех байтовых значений, а Bytes * - тип или набор произвольных (конечных) списков байтов; в то время как 2 = {0,1} - это битовый тип, а 2^k - это набор (или фактически тип) всех k-битовых последовательностей (т. е. k-битных чисел).
Конечно, формула (7) невозможна с математической точки зрения, потому что отображение бесконечного множества в конечное множество не может быть инъективным. Более строгое предположение было бы следующим:

$$\forall s, s' : s \neq s', P\big(\text{HASH}(s) = \text{HASH}(s')\big) = 2^{-k} \qquad (8)$$

Однако для доказательств это не так удобно. Если (8) используется не более N раз в доказательстве с $2^{-k}N < \in$ для некоторого малого ∈ (скажем, ∈ = 10^{-18}), мы можем предположить, что (7) является истинным, при условии, что мы принимаем вероятность отказа ∈ (т. е. окончательные выводы будут верными с вероятностью не менее 1 - ∈).
Заключительное замечание: для того, чтобы утверждение о вероятности (8) было действительно строгим, необходимо ввести распределение вероятностей на множестве Bytes* всех последовательностей байтов. Один из способов сделать это - предположить, что все последовательности байтов одинаковой длины l равновероятны, и установить вероятность наблюдения за последовательностью длины l равной $p^l - p^{l+1}$ для некоторого p → 1–. Тогда (8) следует понимать как

2.2. Общие сведения о блокчейнах

предел условной вероятности P(HASH (S) = HASH(S') | s ≠ s'), когда *p* стремится к единице снизу.

2.2.10. Хэш, используемый для блокчейна TON. В настоящее время мы используем 256-битный хэш SHA256 для блокчейна TON. Если он окажется слабее, чем ожидалось, в будущем его можно заменить другой хэш-функцией. Выбор хэш-функции - это настраиваемый параметр протокола, поэтому его можно изменить без необходимости создания хард-форков, как описано в 2.1.21.

[3] https://coq.inria.fr
[4] https://core.telegram.org/mtproto/T

2.3. Состояние блокчейна, учетные записи и хэш-карты

Выше мы отметили, что любой блокчейн определяет заданное глобальное состояние, а каждый блок и каждая транзакция определяют преобразование этого глобального состояния. Ниже описывается глобальное состояние, используемое блокчейнами TON.

2.3.1. Идентификаторы аккаунтов. Базовые идентификаторы учетных записей, используемые блокчейнами TON (или как минимум его мастерчейном и исходным воркчейном) представляют собой 256-битные целые числа, которые считаются открытыми ключами для 256-битного шифрования на основе эллиптических кривых (ECC) для конкретной эллиптической кривой. Таким образом,

$$account\_id : Account = uint_{256} = \mathbf{2}^{256} \qquad (9)$$

Где Account - это тип учетной записи, а account_id: Account - это конкретная переменная типа Account.
Другие воркчейны могут использовать другие форматы идентификаторов учетных записей (256-битные или другие). Например, можно использовать подобные биткоину идентификаторы учетной записи, равные SHA256 открытого ключа ECC.
Однако длина l в битах идентификатора учетной записи должна быть зафиксирована во время создания воркчейна (в мастерчейне), и она должна быть не менее 64, поскольку первые 64 бита идентификатора account_id используются для сегментирования и маршрутизации сообщений.

2.3.2. Основной компонент: хэш-карты. Главный компонент состояния блокчейна TON - это хэш-карта. В некоторых случаях мы рассматриваем (частично определенные) «карты» h: 2^n -⟶ + 2^m. В более общем плане нас могут заинтересовать хэш-карты h: 2^n -⟶ X для составного типа X. Однако тип источника (или индекса) почти всегда 2^n. Иногда «значение по умолчанию» является пустым: X, а хэш-карта h: 2^n ⟶ X «инициализируется» своим пустым «значением по умолчанию» i ⟶.

2.3.3. Пример: остатки на счетах TON. Важный пример - остатки на счетах TON. Это хэш-карта,

$$balance : Account \rightarrow uint_{128} \qquad (10)$$

которая сопоставляет Account = 2^{256} с балансом монет TON типа $uint_{128} = 2^{128}$. Эта хэш-карта имеет значение по умолчанию, равное нулю, то есть изначально (до обработки первого блока) баланс всех учетных записей равен нулю.

2.3.4. Пример: постоянное хранилище смарт-контракта. Другой пример - постоянное хранилище смарт-контрактов, которое можно (очень приблизительно) представить в виде хэш-карты.

$$storage : \mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256} \qquad (11)$$

Эта хэш-карта также имеет значение по умолчанию, равное нулю, то есть неинициализированные ячейки постоянного хранилища считаются равными нулю.

2.3.5. Пример: постоянное хранилище всех смарт-контрактов. Поскольку у нас есть несколько смарт-контрактов, различающихся идентификатором account_ id, каждый из которых имеет отдельное постоянное хранилище, у нас должна быть фактическая хэш-карта,

$$Storage : Account \dashrightarrow (\mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256}) \qquad (12)$$

которая соотносит идентификатор account_id смарт-контракта с его постоянным хранилищем.

2.3.6. Тип хэш-карты. Хэш-карта - это не просто абстрактная (частично определенная) функция 2^n --⟶ X; у нее есть конкретное представление. Поэтому мы предполагаем, что у нас есть специальный тип хэш-карты,

$$Hashmap(n, X) : Type \tag{13}$$

соответствующий структуре данных, кодирующей (частичное) отображение 2^n -⟶ X. Мы также можем записать

$$Hashmap(n : nat)(X : Type) : Type \tag{14}$$

или

$$Hashmap : nat \to Type \to Type \tag{15}$$

Мы всегда можем преобразовать h: Hashmap (n, X) в карту hget (h): $2^n \to X^?$, С этого момента мы обычно пишем h[i] вместо hget(h)(i):

$$h[i] :\equiv hget(h)(i) : X^? \quad \text{for any } i : \mathbf{2}^n,\ h : Hashmap(n, X) \tag{16}$$

2.3.7. Определение типа хэш-карты как дерева Patricia. Логически можно определить Hashmap(n, X) как (неполное) двоичное дерево глубины n с метками ребер 0 и 1 и со значениями типа X в листьях. Другим способом описания той же структуры было бы (побитовое) префиксное дерево для двоичных строк длиной, равной n.
На практике мы предпочитаем использовать компактное представление этого дерева, сжимая каждую вершину, имеющую только один дочерний элемент со своим родителем. Результирующее представление известно как дерево Patricia или двоичное основание системы счисления. Каждая промежуточная вершина теперь имеет ровно два дочерних элемента, помеченных двумя непустыми двоичными строками, начиная с нуля для левого дочернего элемента и с одного для правого дочернего элемента.
Другими словами, в дереве Patricia имеется два типа (некорневых) узлов:
• LEAF(x), содержащий значение x типа X.
• NODE (д, s_l, r, s_r), где l - (ссылка на) левый дочерний элемент или поддерево, s_l - это цепочка битов, обозначающая ребро, соединяющее эту вершину с ее левым дочерним элементом (всегда начинается с 0), r - правое поддерево, а sr - это цепочка битов, обозначающая край правого дочернего элемента (всегда начинается с 1).
Также необходим третий тип узла, который будет использоваться только один раз в корне дерева Patricia:
• ROOT (n, s_0, t), где n - общая длина индексных битовых строк Hashmap (n, X), s_0 - общий префикс всех индексных битовых строк, а t - ссылка на LEAF или NODE .
Если мы хотим, чтобы дерево Patricia было пустым, необходимо использовать четвертый тип (корневого) узла:
• EMPTYROOT (n), где n - общая длина всех битовых строк индекса.
Мы определяем высоту дерева Patricia следующим образом:

$$\text{HEIGHT}(\text{LEAF}(x)) = 0 \tag{17}$$

$$\text{HEIGHT}(\text{NODE}(l, s_l, r, s_r)) = \text{HEIGHT}(l) + \text{LEN}(s_l) = \text{HEIGHT}(r) + \text{LEN}(s_r) \tag{18}$$

$$\text{HEIGHT}(\text{ROOT}(n, s_0, t)) = \text{LEN}(s_0) + \text{HEIGHT}(t) = n \tag{19}$$

Последние два выражения в каждой из двух последних формул должны быть равны. Мы используем деревья Patricia высоты n для представления значений типа Hashmap(n, X),

Если в дереве N листьев (т. е. наша хэш-карта содержит N значений), то имеется ровно N - 1 промежуточных вершин. Вставка нового значения всегда включает в себя разделение существующего ребра путем вставки новой вершины в середину и добавления нового листа в качестве другого дочернего элемента этой новой вершины. Удаление значения из хэш-карты приводит к обратному эффекту: лист и его родитель удаляются, а родитель родителя и другой его дочерний элемент становятся напрямую связанными.

2.3.8. Деревья Merkle Patricia. При работе с блокчейнами мы хотим иметь возможность сравнивать деревья Patricia (то есть хэш-карты) и их поддеревья, сводя их к одному хэш-значению. Классический способ достижения этой задачи – использование дерева Merkle. По сути, мы хотим описать способ хэширования объектов h типа Hashmap(n, X) с помощью хэш-функции HASH, определенной для двоичных строк, при условии, что мы знаем, как вычислять хэш-значения HASH(X) объектов x: X (например, путем применения хэш-функции HASH к двоичной сериализации объекта x).
Можно определить HASH(h) рекурсивно следующим образом:

$$\textsc{Hash}\big(\textsc{Leaf}(x)\big) := \textsc{Hash}(x) \tag{20}$$

$$\textsc{Hash}\big(\textsc{Node}(l, s_l, r, s_r)\big) := \textsc{Hash}\big(\textsc{Hash}(l).\,\textsc{Hash}(r).\,\textsc{code}(s_l).\,\textsc{code}(s_r)\big) \tag{21}$$

$$\textsc{Hash}\big(\textsc{Root}(n, s_0, t)\big) := \textsc{Hash}\big(\textsc{code}(n).\,\textsc{code}(s_0).\,\textsc{Hash}(t)\big) \tag{22}$$

Здесь s.t обозначает конкатенацию (битовых) строк s и t, а CODE (S) - это префиксный код для всех битовых строк s. Например, можно закодировать 0 на 10, 1 на 11 и конец строки на 0.[5]
Позже мы увидим (см. 2.3.12 и 2.3.14), что это (слегка измененная) версия рекурсивно определенных хэшей для значений произвольных (зависимых) алгебраических типов.

2.3.9. Пересчет хэшей дерева Merkle. Этот способ рекурсивного разделения HASH (h), называемый хэшем дерева Merkle, обладает следующим преимуществом: если один явно хранит HASH(h') вместе с каждым узлом h' (в результате получается структура, называемая деревом Merkle, или, в нашем случае, деревом Merkle Patricia), необходимо пересчитывать не более n хэшей при добавлении, удалении или изменении элемента в хэш-карте.
Таким образом, если кто-то представляет глобальное состояние блокчейна подходящим хэшем дерева Merkle, его можно легко пересчитывать после каждой транзакции.

2.3.10. Доказательства Меркла. На основании предположения (7) «инъективности» выбранной хэш-функции HASH, можно построить доказательство того, что для данного значения z HASH(h), h: Hashmap(n, X) имеет место hget(h)(i) = x для некоторых i: 2^n и x: X. Такое доказательство будет состоять из пути в дереве Merkle Patricia от листа, соответствующего i, до корня, дополненного хэшами всех братьев и сестер всех встречающихся узлов на этом пути.
Таким образом, неполная нода[6], знающая только значение HASH(h) для некоторой хэш-карты h (например, постоянное хранилище смарт-контракта или глобальное состояние блокчейна), может запросить у полной ноды[7] не только значение x = h[i] = hget(h)(i), но такое значение вместе с доказательством Меркла, начиная с уже известного значения HASH(h).
Затем, в предположении (7), неполная нода может сама проверить, что x действительно является правильным значением h [i].

В некоторых случаях клиент может захотеть получить вместо этого значение y = HASH(X) = HASH(h[i]) - например, в случае очень большого значения x (например, самой хэш-карты). Тогда вместо этого можно предоставить доказательство Меркла для (i, y). Если x также является хэш-картой, то второе доказательство Меркла, начиная с y = HASH(X), может быть получено из полной ноды, чтобы предоставить значение x[j] = h[i][j] или только его хэш.

2.3.11. Важность доказательств Меркла для многозвенной системы, такой как TON. Обратите внимание, что нода обычно не может быть полной нодой для всех шардчейнов, существующих в среде TON. Обычно эта нода является полной только для некоторых шардчейнов – например, шардчейнов, которые содержат ее собственную учетную запись, смарт-контракт, с которым она работает, либо шардчейнов, для которых эта нода была назначена валидатором. Для других шардчейнов это должна быть неполная нода, иначе требования к хранилищу, вычислениям и пропускной способности сети будут непомерно высокими. Это означает, что такая нода не может напрямую проверять утверждения о состоянии других шардчейнов; она должна полагаться на доказательства Меркла, полученные из полных нод для этих шардчейнов. Этот процесс является таким же безопасным, как и собственно проверка, если только (7) он не завершится неудачей (т. е. не будет обнаружен хэш-конфликт).

2.3.12. Особенности виртуальной машины TON. Виртуальная машина TON или TVM, используемая для запуска смарт-контрактов в мастерчейне и исходном воркчейне, значительно отличается от обычных систем, вдохновленных виртуальной машиной Ethereum (EVM). TVM работает не только с 256-битными целыми числами, но и фактически с (почти) произвольными «записями», «структурами» или «типами суммы-произведения», что делает ее более подходящей для выполнения кода, написанного на высокоуровневых языках программирования (особенно функциональных языках). По сути, TVM использует тегированные типы данных, аналогичные тем, которые используются в реализациях Prolog или Erlang.
Сначала можно подумать, что состояние смарт-контракта TVM - это не просто хэш-карта $2^{256} \rightarrow 2^{256}$, от Hashmap (256, 2^{256}), но (в качестве первого шага) Hashmap (256, X), где X - это тип с несколькими конструкторами, позволяющий хранить, помимо 256-битных целых чисел, другие структуры данных, включая, в частности, другие хэш-карты Hashmap (256, X). Это означало бы, что ячейка хранилища TVM (постоянного или временного) - переменная или элемент массива в коде смарт-контракта TVM - может содержать не только целое число, но и целую новую хэш-карту. Конечно, это будет означать, что ячейка содержит не только 256 бит, но также, скажем, 8-битный тег, описывающий, как эти 256 бит должны интерпретироваться.

[5] Можно показать, что это кодирование оптимально примерно для половины всех меток ребер дерева Patricia со случайными или последовательными индексами. Остальные граничные метки, вероятно, будут длинными (т. е. длиной почти 256 бит). Следовательно, почти оптимальным кодированием для граничных меток является использование приведенного выше кода с префиксом 0 для «коротких» битовых строк и кодирование 1, а затем девять битов, содержащих длину l = |s| битовой строки s, а затем l битов s для «длинных» битовых строк ($l \leq 10$).

[6] Неполная нода - это узел, который не отслеживает полное состояние шардчейна; вместо этого он сохраняет минимальную информацию, такую как хэши нескольких самых последних блоков, и полагается на информацию, полученную от полных нод, когда возникает необходимость проверить некоторые части полного состояния.

[7] Полная нода - это узел, отслеживающий полное актуальное состояние рассматриваемого шардчейна.

Фактически, значения не обязательно должны быть точно 256-битными. Формат значения, используемый TVM, состоит из последовательности необработанных байтов и ссылок на другие структуры, смешанных в произвольном порядке, причем некоторые байты дескриптора вставлены в подходящие места, что дает возможность отличать указатели от необработанных данных (например, строки или целые числа) (см. 2.3.14)
Этот формат необработанных значений может использоваться для реализации произвольных алгебраических типов суммы-произведения. В этом случае значение будет сначала содержать необработанный байт, описывающий используемый «конструктор» (с точки зрения высокоуровневого языка программирования), а затем другие «поля» или «аргументы конструктора», состоящие из необработанных байтов и ссылок на другие структуры в зависимости от выбранного конструктора (см. 2.2.5). Однако TVM ничего не знает о соответствии конструкторов и их аргументов; смесь байтов и ссылок явно описывается определенными байтами дескриптора[8].
Хэширование дерева Merkle распространяется на произвольные структуры - для вычисления хэша такой структуры все ссылки рекурсивно заменяются хэшами объектов, на которые имеется ссылка, а затем вычисляется хэш результирующей байтовой строки (включая байты дескриптора).
Таким образом, хэширование дерева Merkle для хэш-карт, описанное в п. 2.3.8, представляет собой просто простой способ хэширования для произвольных (зависимых) алгебраических типов данных, применяемый к типу Hashmap (n, X) с двумя конструкторами[9].

2.3.13. Постоянное хранилище смарт-контрактов TON. Постоянное хранилище смарт-контрактов TON по существу состоит из его «глобальных переменных», сохраняемых между вызовами смарт-контракта. По сути, это просто тип «произведение», «кортеж» или «запись», состоящий из полей правильных типов, каждое из которых соответствует одной глобальной переменной. Если существует слишком много глобальных переменных, они не могут поместиться в одну ячейку TON из-за глобального ограничения на размер ячейки TON. Для простоты они разделены на несколько записей и организованы в дерево, по сути становясь типом «произведение произведений» или «произведение произведений произведений», а не просто типом-произведением.

2.3.14. Ячейки TVM. В конечном итоге виртуальная машина TON хранит все данные в виде набора ячеек (TVM). Каждая ячейка сначала содержит два байта дескриптора, указывающие, сколько байтов необработанных данных присутствует в этой ячейке (до 128) и сколько имеется ссылок на другие ячейки (до 4). Затем следуют байты необработанных данных и ссылки. На каждую ячейку имеется ровно одна ссылка, поэтому мы могли бы включить в каждую ячейку ссылку на ее «родительский элемент» (единственную ячейку, ссылающуюся на эту ячейку). Однако эта ссылка не обязательно должна быть явной.
Таким образом, ячейки постоянного хранилища смарт-контракта TON организованы в дерево со ссылкой на корень этого дерева[10], хранящейся в описании смарт-контракта. Если необходимо, рекурсивно вычисляется хэш дерева Merkle для всего этого постоянного хранилища, начиная с листьев, а затем просто заменяются все ссылки в ячейке рекурсивно вычисленными хэшами ячеек, на которые имеются ссылки, и впоследствии вычисляется хэш полученной таким образом строки байтов.

2.3.15. Обобщенные доказательства Меркла для значений произвольных алгебраических типов. Поскольку виртуальная машина TON представляет значение произвольного алгебраического типа посредством дерева, состоящего из (TVM)

ячеек, а каждая ячейка имеет четко определенный (рекурсивно вычисленный) хэш Меркла,
фактически зависящий от всего поддерева, имеющего корень в этой ячейке, мы можем предоставить «обобщенные доказательства Меркла» для (частей) значений произвольных алгебраических типов, предназначенные для доказательства того, что определенное поддерево дерева с известным хэшем Меркла принимает определенное значение или значение с определенным хэшем. Это обобщает подход в п. 2.3.10, где были рассмотрены только доказательства Меркла для x[i] = y.

2.3.16. Поддержка сегментирования в структурах данных TON VM. Только что мы в общих чертах обрисовали, как виртуальная машина TON, не будучи чрезмерно сложной, поддерживает произвольные (зависимые) алгебраические типы данных на высокоуровневых языках смарт-контрактов. Однако для шардинг крупных (или глобальных) смарт-контрактов требуется специальная поддержка на уровне BM TON. С этой целью в систему была добавлена специальная версия типа hashmap, равная «карте» Account -->X . Эта «карта» может показаться эквивалентной Hashmap (m, X), где Account = 2^m, однако, когда шард разделяется на два шарда, либо два шарда объединяются, такие хэш-карты автоматически разделяются на две хэш-карты или объединяются обратно, что позволяет сохранить только те ключи, которые принадлежат соответствующему шарду.

2.3.17. Плата за постоянное хранилище. Примечательной особенностью блокчейна TON является плата, взимаемая со смарт-контрактов за хранение их постоянных данных (то есть за увеличение состояния блокчейна в целом). Этот процесс выглядит следующим образом:
В каждом блоке декларируются два стейка, номинированные в основной валюте блокчейна (обычно это монета TON): цена за хранение одной ячейки в постоянном хранилище и цена за хранение одного необработанного байта в какой-либо ячейке постоянного хранилища. Статистика по общему количеству ячеек и байтов, используемых каждой учетной записью, сохраняется как часть ее состояния, поэтому, умножив эти числа на два стейка, указанные в заголовке блока, мы можем вычислить платеж, который будет вычтен из баланса счета для хранения его данных между предыдущим блоком и текущим.
Однако плата за использование постоянного хранилища взимается не для каждой учетной записи и смарт-контракта в каждом блоке. Вместо этого порядковый номер блока, в котором этот платеж был взыскан в последний раз, сохраняется в данных учетной записи, и когда с учетной записью выполняется какое-либо действие (например, перенос стоимости, либо получение или обработка сообщения смарт-контрактом), то плата за использование хранилища для всех блоков с момента предыдущего такого платежа вычитается из баланса аккаунта перед выполнением каких-либо дальнейших действий. Если после этого баланс учетной записи станет отрицательным, учетная запись будет уничтожена.

[8] Эти два байта дескриптора, присутствующие в любой ячейке TVM, описывают только общее количество ссылок и общее количество необработанных байтов; ссылки хранятся вместе либо до, либо после всех необработанных байтов.
[9] Фактически LEAF и NODE являются конструкторами вспомогательного типа HashmapAux (n, X). Тип Hashmap (n, X) имеет конструкторы ROOT и EMPTYROOT, причем ROOT содержит значение типа HashmapAux(n, X).
[10] Логически; представление «набор ячеек», описанное в 2.5.5, идентифицирует все повторяющиеся ячейки, преобразовывая это дерево в ориентированный ациклический граф при сериализации.

Воркчейн может объявить некоторое количество байтов необработанных данных для каждой учетной записи «бесплатными» (т. е. не участвующими в платежах за постоянное хранилище) для создания «простых» учетных записей, которые сохраняют только свой баланс в одной или двух криптовалютах, освобожденных от этих постоянных платежей.
Обратите внимание: если никто не отправляет сообщения в учетную запись, плата за постоянное хранение не взимается, и учетная запись может существовать бесконечно. Однако любой пользователь может отправить, например, пустое сообщение для уничтожения такой учетной записи. Отправителю такого сообщения может быть предоставлено небольшое вознаграждение, взимаемое с части первоначального баланса учетной записи, подлежащей уничтожению. Однако мы ожидаем, что валидаторы будут уничтожать такие неплатежеспособные учетные записи бесплатно, просто чтобы уменьшить размер состояния блокчейна и избежать хранения больших объемов данных без компенсации.
Платежи, собранные за хранение постоянных данных, распределяются между валидаторами шардчейна или мастерчейна (во втором случае пропорционально их стейкам).

2.3.18. Локальные и глобальные смарт-контракты; экземпляры смарт-контрактов. Смарт-контракт обычно находится только в одном шарде, выбранном в соответствии с идентификатором account_ id смарт-контракта (аналогично «обычной» учетной записи). Обычно этого достаточно для большинства приложений. Однако для некоторых смарт-контрактов с высокой нагрузкой может потребоваться наличие «экземпляра» в каждой шардчейне какого-либо воркчейна. Для этого они должны распространить свою создаваемую транзакцию на все шардчейны, например, зафиксировав эту транзакцию в «корневом» шардчейне (w, θ)[11] воркчейна w и установив большую комиссию[12].
Это позволяет эффективно создавать экземпляры смарт-контракта в каждом шарде с отдельными балансами. Первоначально баланс, переданный в создаваемой транзакции, распределяется просто путем присвоения экземпляру в шарде (w, s) значений $2^{-|s|}$ от части общего баланса. Когда шард разделяется на два дочерних шарда, балансы всех экземпляров глобальных смарт-контрактов делятся пополам; когда два шарда объединяются, балансы складываются.
В некоторых случаях разделение/объединение экземпляров глобальных смарт-контрактов может включать (отложенное) выполнение специальных методов этих смарт-контрактов. По умолчанию балансы разделяются и объединяются, как описано выше, а некоторые специальные хэш-карты, индексированные «по учетным записям», также автоматически разделяются и объединяются (см. 2.3.16).

2.3.19. Ограничение разделения смарт-контрактов. Глобальный смарт-контракт может ограничивать глубину разделения d при его создании, что позволяет сделать расходы на постоянное хранение более предсказуемыми. Это означает, что если шардчейн (w, s) вместе с $|s| > d$ делится на две части, только один из двух новых сегментов наследует экземпляр смарт-контракта. Эта шардчейн выбирается детерминировано: каждый глобальный смарт-контракт имеет некоторый «account_id», который по сути является хэшем создаваемой транзакции, а его экземпляры имеют тот же идентификатор account_id с заменой первых <d битов подходящими значениями, необходимыми для попадания в правильный шард. Этот идентификатор account_id выбирает, какой шард унаследует экземпляр смарт-контракта после разделения.

2.3.20. Состояние учетной записи/смарт-контракта. На основании всего вышесказанного мы можем сделать вывод, что состояние учетной записи или смарт-контракта включает следующее:

- Баланс в основной валюте блокчейна
- Баланс в других валютах блокчейна
- Код смарт-контракта (или его хэш)
- Постоянные данные смарт-контракта (или соответствующий хэш Меркла)
- Статистика количества используемых ячеек постоянного хранения и необработанных байтов
- Последний раз (собственно, номер блока мастерчейна), когда была получена оплата за постоянное хранение смарт-контракта
- Открытый ключ, необходимый для перевода валюты и отправки сообщений с этой учетной записи (необязательно; по умолчанию равен самому идентификатору account_id). В некоторых случаях здесь может быть расположен более сложный код проверки подписи, аналогично коду, который используется для выходных данных транзакции биткоина; тогда идентификатор account_id будет равен хэшу этого кода.
- Также в состоянии учетной записи или в какой-либо другой хэш-карте, индексированной учетной записью, необходимо хранить следующие данные:
- Очередь выходных сообщений учетной записи (см. 2.4.17)
- Набор (хэшей) недавно доставленных сообщений (см. 2.4.23)

Не все эти функции действительно необходимы для каждой учетной записи; например, код смарт-контракта нужен только для смарт-контрактов, но не для «простых» учетных записей. Кроме того, хотя на любой учетной записи должен быть ненулевой баланс в основной валюте (например, монеты TON для мастерчейна и щардчейнов основного воркчейна), на ней могут быть нулевые балансы в других валютах. Чтобы избежать хранения неиспользуемых данных, (во время создания воркчейна) определяется тип суммы-произведения (в зависимости от воркчейна), который использует разные байты тегов (например, конструкторы TL; см. 2.2.5), чтобы различать разные « конструкторы ». В конечном итоге, состояние учетной записи сохраняется как набор ячеек постоянного хранилища TVM.

2.4. Сообщения между шардчейнами

Важным компонентом блокчейна TON является система обмена сообщениями между блокчейнами. Эти блокчейны могут быть шардчейнами одного или разных воркчейнов.

2.4.1. Сообщения, счета и транзакции: общая структура системы. Сообщения отправляются из одной учетной записи в другую. Каждая транзакция включает следующие этапы: учетная запись получает одно сообщение, меняет свое состояние в соответствии с определенными правилами и генерирует несколько (возможно, одно или ноль) новых сообщений для других учетных записей. Каждое сообщение создается и принимается (доставляется) ровно один раз.

Это означает, что сообщения играют в системе фундаментальную роль, сравнимую с ролью учетных записей (смарт-контрактов). С точки зрения парадигмы бесконечного шардинга (см. 2.1.2), каждая учетная запись находится в своей отдельной «цепочке учетных записей», и единственный способ повлиять на состояние какой-либо другой учетной записи - это отправить сообщение.

[11] Более дорогая альтернатива - опубликовать такой «глобальный» смарт-контракт в мастерчейне.

[12] Это своего рода «широковещательная» функция для всех шардов, и поэтому она должна быть довольно дорогой

2.4.2. Учетные записи как процессы или субъекты; модель акторов. Можно воспринимать учетные записи (и смарт-контракты) как «процессы» или «акторы», которые могут обрабатывать входящие сообщения, изменять свое внутреннее состояние и в результате генерировать некоторые исходящие сообщения. Это тесно связано с так называемыми моделями акторов, которые используются в таких языках, как Erlang (однако акторы в Erlang обычно называются «процессами»). Поскольку новые акторы (то есть смарт-контракты) также могут быть созданы существующими акторами в результате обработки входящего сообщения, наблюдается практически полное соответствие с моделью акторов.

2.4.3. Получатель сообщения. У любого сообщения есть получатель, который характеризуется идентификатором целевого воркчейна w (предполагается, что по умолчанию он такой же, как идентификатор исходного шардчейна) и учетной записью получателя account_id. Точный формат (т. е. количество битов) account_id зависит от w; однако шард всегда определяется его первыми (наиболее значимыми) 64 битами.

2.4.4. Отправитель сообщения. В большинстве случаев у сообщения есть отправитель, который характеризуется парой (w', account_id'). При наличии такого идентификатора он находится после получателя сообщения и ценности сообщения. Иногда отправитель не важен, либо это кто-то за пределами блокчейна (то есть не смарт-контракт), поэтому это поле отсутствует.
Обратите внимание, что в модели акторов не требуется, чтобы сообщения имели неявного отправителя. Вместо этого сообщения могут содержать ссылку на актора, которому должен быть отправлен ответ на запрос (обычно совпадает с отправителем). Однако полезно иметь поле явного не поддающегося подделке отправителя в сообщении в криптовалютной среде, в которой применяется задача византийских генералов.

2.4.5. Стоимость сообщения. Другой важной характеристикой сообщения является его прикрепленная стоимость в одной или нескольких криптовалютах, поддерживаемых как исходным, так и целевой воркчейном. Стоимость указывается в самом начале сообщения сразу после получателя сообщения; по сути, это список пар (currency_id, value).
Обратите внимание, что «простые» транзакции стоимости между «простыми» учетными записями - это пустые сообщения с некоторой прикрепленной к ним стоимостью. С другой стороны, немного более сложное тело сообщения может содержать простой текст или двоичный комментарий (например, о цели платежа).

2.4.6. Внешние сообщения или «сообщения из ниоткуда». Некоторые сообщения поступают в систему «из ниоткуда», то есть они не генерируются учетной записью (смарт-контрактом), находящейся в блокчейне. Наиболее типичный пример - когда пользователь хочет перевести часть средств с контролируемой им учетной записи на другую учетную запись.
В этом случае пользователь отправляет «сообщение из ниоткуда» в свою учетную запись, запрашивая создание сообщения для принимающей учетной записи, содержащего указанное значение. Если это сообщение правильно подписано, учетная запись получает его и генерирует необходимые исходящие сообщения.
Фактически, можно рассматривать «простую» учетную запись как частный случай смарт-контракта с предопределенным кодом. Этот смарт-контракт получает только один тип сообщения. Такое входящее сообщение должно содержать список исходящих сообщений, которые должны быть сгенерированы в результате доставки (обработки) входящего сообщения, вместе с подписью. Смарт-контракт проверяет подпись и, если она верна, генерирует необходимые сообщения.

Конечно, есть разница между «сообщениями из ниоткуда» и обычными сообщениями, поскольку «сообщения из ниоткуда» не могут иметь стоимости, и соответственно не могут сами платить за «газ» (то есть за их обработку). Вместо этого они предварительно выполняются с небольшим лимитом газа, прежде чем будет предложено их включение в новый блок шардчейна; если выполнение не удается (подпись неверна), «сообщение из ниоткуда» считается неверным и отбрасывается. Если выполнение будет завершено в пределах небольшого лимита газа, сообщение может быть включено в новый блок шардчейна и полностью обработано, при этом оплата за потребленный газ (мощность обработки) будет взиматься со счета получателя. Для «сообщений из ниоткуда» также может быть определена некоторая комиссия за транзакцию, которая снимается со счета получателя сверх платы за газ и перераспределяется между валидаторами.
В этом смысле «сообщения из ниоткуда» или «внешние сообщения» выступают в роли кандидатов на транзакции, используемых в других системах блокчейнов (например, Биткоин и Эфириум).

2.4.7. Сообщения журнала или «сообщения в никуда». Точно так же иногда может быть сгенерировано специальное сообщение, которое направляется в конкретный шардчейн не для доставки его получателю, а для регистрации в журнале, чтобы это сообщение мог легко заметить любой пользователь, который получает обновления о рассматриваемом шарде. Такие зарегистрированные сообщения могут выводиться на консоль пользователя или запускать выполнение некоторого сценария на сервере вне сети. В этом смысле они представляют собой внешний «выход» «суперкомпьютера с блокчейном», так же как «сообщения из ниоткуда» представляют собой внешний «вход» «суперкомпьютера с блокчейном».

2.4.8. Взаимодействие с сервисами вне блокчейна и внешними блокчейнами. Эти внешние входные и выходные сообщения могут использоваться для взаимодействия с сервисами вне блокчейна и другими (внешними) блокчейнами, такими как Биткоин или Эфириум.
В блокчейне TON можно создавать токены или криптовалюты, привязанные к биткойну, эфиру или любым токенам ERC-20, определенным в блокчейне Ethereum, и использовать «сообщения из ниоткуда» и «сообщения в никуда», генерируемые и обрабатываемые скриптами, находящимися на сторонних серверах вне блокчейна, чтобы реализовать необходимое взаимодействие между блокчейном TON и этими внешними блокчейнами.

2.4.9. Тело сообщения. Тело сообщения — это просто последовательность байтов, значение которой определяется только принимающим воркчейном и/или смарт-контрактом. Для блокчейнов, в которых используется виртуальная машина TON, это может быть сериализация любой ячейки TVM, автоматически сгенерированная с помощью операции Send O. Такая сериализация достигается путем простой рекурсивной замены всех ссылок в ячейке BM TON на указанные ячейки. В конечном итоге появляется строка необработанных байтов, перед которой обычно идет 4-байтовый «тип сообщения» или «конструктор сообщения», используемый для выбора правильного метода принимающего смарт-контракта.
Другим вариантом может быть использование объектов, сериализированных с помощью языка TL (см. 2.2.5), в качестве тел сообщения. Это может быть особенно полезно для связи между различными воркчейнами, в одном из которых или во всех не обязательно используется виртуальная машина TON.

2.4.10. Лимит газа и другие параметры, специфичные для воркчейна/VM. Иногда сообщение должно нести информацию о лимите газа, цене на газ, комиссии за

транзакции и аналогичных значениях, которые зависят от принимающего воркчейна и актуальны только для него, но не обязательно для исходного воркчейна. Такие параметры включаются в тело сообщения или перед ним, иногда (в зависимости от воркчейна) со специальными 4-байтовыми префиксами, указывающими на их присутствие (которые могут быть определены с помощью TL-схемы; см. 2.2.5).

2.4.11. Создание сообщений: смарт-контракты и транзакции. Существуют два источника новых сообщений. Большинство сообщений создается во время выполнения смарт-контракта (посредством операции Send() в VM TON), когда для обработки входящего сообщения вызывается какой-либо смарт-контракт. Как вариант, сообщения могут приходить извне как «внешние сообщения» или «сообщения из ниоткуда» (см. 2.4.6)[13].

2.4.12. Доставка сообщений. Когда сообщение достигает шардчейна, содержащего целевую учетную запись, оно «доставляется» в целевую учетную запись[14]. Дальнейшие процессы зависят от воркчейна; со стороны важно, чтобы такое сообщение никогда не могло быть переадресовано дальше этого шардчейна.
Для шардчейнов основного воркчейна доставка заключается в добавлении стоимости сообщения (за вычетом любых платежей за газ) к балансу принимающей учетной записи и, возможно, в последующем вызове зависимого от сообщения метода получающего смарт-контракта, если получающая учетная запись является смарт-контрактом. Фактически, смарт-контракт имеет только одну точку входа для обработки всех входящих сообщений, поэтому он должен различать разные типы сообщений путем анализа первых нескольких байтов (например, первые четыре байта, содержащие конструктор TL; см. 2.2.5).

2.4.13. Доставка сообщения как транзакция. Поскольку доставка сообщения изменяет состояние учетной записи или смарт-контракта, этот процесс является особой транзакцией в принимающем шардчейне, которая явно регистрируется как таковая. По сути, без учета некоторых незначительных технических деталей, все транзакции в блокчейне TON представляют собой доставку одного входящего сообщения в принимающую учетную запись (смарт-контракт).

2.4.14. Сообщения между экземплярами одного и того же смарт-контракта. Напомним, что смарт-контракт может быть локальным (т. е. может находиться в одной шардчейне, как и любая обычная учетная запись) или глобальным (т. е. иметь экземпляры во всех шардах или, по крайней мере, во всех шардах до некоторой известной глубины d; см. 2.3.18). При необходимости экземпляры глобального смарт-контракта могут обмениваться специальными сообщениями между собой для передачи информации и стоимости. В этом случае особенно важным становится (не поддающийся подделке) идентификатор account_id (см. 2.4.4).

2.4.15. Сообщения на любой экземпляр смарт-контракта; адреса с подстановочными знаками. Иногда сообщение (например, запрос клиента) необходимо доставить в любой экземпляр глобального смарт-контракта, обычно ближайший (если он находится в том же шардчейне, что и отправитель, это очевидный кандидат).

[13] Вышеупомянутое должно быть верным только для исходного воркчейна и соответствующих шардчейнов. Другие воркчейны могут предоставлять другие способы создания сообщений.

Один из способов сделать это - использовать «адрес получателя с подстановочными знаками», при этом первым d битам целевого идентификатора account_id разрешено принимать произвольные значения. На практике для этих битов d обычно устанавливаются те же значения, что и в идентификаторе отправителя account_id.

2.4.16. Отсутствие очереди ввода. Все сообщения, полученные блокчейном (обычно шардчейном, иногда мастерчейном) - или, по сути, «цепочкой учетных записей», находящейся внутри определенного шардчейна, - немедленно доставляются (т. е. обрабатываются принимающей учетной записью).
Следовательно, «очередь ввода» как таковая отсутствует. Вместо этого, если из-за ограничений на общий размер блоков и использование газа могут быть обработаны не все сообщения, предназначенные для определенного шардчейна, некоторые сообщения просто накапливаются в очередях вывода исходных шардчейнов.

2.4.17. Очереди вывода. С точки зрения парадигмы бесконечного шардинга (см. 2.1.2), каждая цепочка учетных записей (т. е. каждая учетная запись) имеет свою собственную очередь вывода, состоящую из всех сообщений, которые были в ней сгенерированы, но еще не доставлены получателям. Конечно, цепочки учетных записей существуют только виртуально; они сгруппированы в шардчейны, а шардчейн имеет выходную «очередь», состоящую из очередей вывода всех учетных записей, относящихся к шардчейну.
Эта выходная «очередь» шардчейна обеспечивает только частичный порядок в сообщениях участников шардчейна. А именно, сообщение, сгенерированное в предыдущем блоке, должно быть доставлено до любого сообщения, сгенерированного в последующем блоке, а любые сообщения, сгенерированные той же учетной записью и имеющие то же место назначения, должны быть доставлены в порядке их генерации.

2.4.18. Надежный и быстрый обмен сообщениями между цепями. В масштабируемом проекте с несколькими блокчейнами (таком как TON) чрезвычайно важно иметь возможность пересылать и доставлять сообщения между разными шардчейнами (см. 2.1.3), даже если в системе их миллионы. Необходимо обеспечить надежную (т. е. сообщения не должны теряться или доставляться более одного раза) и быструю доставку сообщений. В блокчейне TON эта цель достигается путем использования комбинации двух механизмов «маршрутизации сообщений».

2.4.19. Маршрутизация в гиперкубе: «медленный путь» для сообщений с гарантированной доставкой. В блокчейне TON применяется технология «маршрутизации в гиперкубе» в качестве медленного и одновременно безопасного и надежного способа доставки сообщений из одного шардчейна в другой, причем при необходимости для передачи используется несколько промежуточных шардчейнов. В противном случае валидаторам любого данного шардчейна потребуется отслеживать состояние (очередей вывода) всех других шардчейнов, что потребует непомерно больших вычислительных мощностей и пропускной способности сети по мере роста общего количества шардчейнов и приведет к ограничению масштабируемости. системы. Следовательно, возможность доставлять сообщения напрямую из одного шарда в другой отсутствует. Вместо этого каждый шард «подключен» только к шардам, которые отличаются ровно одной шестнадцатеричной цифрой в идентификаторах (w, s) (см. 2.1.8).

[14] В качестве вырожденного случая этот шардчейн может совпадать с исходным шардчейном, например, если операция выполняется внутри воркчейна, который еще не была разделен.

Таким образом, все шардчейны составляют граф типа «гиперкуб», а сообщения перемещаются по краям этого гиперкуба.
Если сообщение отправляется на шард, отличный от текущего шарда, одна из детерминировано выбранных шестнадцатеричных цифр текущего идентификатора шарда заменяется соответствующей цифрой целевого шарда, а полученный идентификатор используется в качестве ближайшей цели для пересылки сообщения.
Основное преимущество маршрутизации в гиперкубе заключается в том, что условия допустимости блока подразумевают, что валидаторы, создающие блоки шардчейна, должны собирать и обрабатывать сообщения из выходных очередей «соседних» шардчейнов, опасаясь потери своих долей.
Таким образом, можно ожидать, что любое сообщение рано или поздно достигнет своего конечного пункта назначения; при этом сообщение не может быть потеряно в пути или доставлено дважды.
Обратите внимание, что маршрутизация в гиперкубе приводит к некоторым дополнительным задержкам и расходам из-за необходимости пересылать сообщения через несколько промежуточных шардчейнов. Однако количество этих промежуточных шардчейнов увеличивается очень медленно по мере роста логарифма $\log N$ (точнее, $[\log_{16} N] - 1$) от общего количества шардчейнов N. Например, если $N = 250$, то будет наблюдаться максимум один промежуточный переход; а для $N = 4000$ шардчейнов — максимум два. С четырьмя промежуточными переходами мы можем поддерживать до одного миллиона шардчейнов. Мы считаем, что это очень небольшая цена за практически неограниченную масштабируемость системы (фактически, не обязательно платить даже ее).

2.4.20. Мгновенная маршрутизация в гиперкубе (Instant Hypercube Routing): «быстрый путь» для сообщений. Новой особенностью блокчейна TON является то, что в нем используется «быстрый путь» для пересылки сообщений от одного шардчейна к любому другому шардчейну, что позволяет в большинстве случаев полностью обойти «медленную» маршрутизацию в гиперкубе (2.4.19) и доставлять сообщение в следующий блок конечного шардчейна.
Идея этой технологии заключается в следующем. Во время «медленной» маршрутизации в гиперкубе сообщение перемещается (в сети) по краям гиперкуба, но задерживается (примерно на пять секунд) в каждой промежуточной вершине и фиксируется в соответствующем шардчейне, прежде чем продолжить свой путь.
Чтобы избежать ненужных задержек, вместо этого можно передавать сообщение вместе с подходящим доказательством Меркла по краям гиперкуба, не дожидаясь его фиксации в промежуточных шардчейнах. Фактически, сетевое сообщение должно быть переадресовано от валидаторов «целевой группы» (см. 2.6.8) исходного шарда указанному создателю блоков (см. 2.6.9) «целевой группы» конечного шарда; это можно сделать напрямую, без остановок сообщения на краях гиперкуба.
Когда это сообщение с доказательством Меркла достигает валидаторов (точнее, сверщиков - см. 2.6.5) целевого шардчейна, они могут немедленно зафиксировать его в новом блоке, не дожидаясь, пока сообщение завершит продвижение по «медленному пути».

[15] Это не обязательно окончательная версия алгоритма, используемого для вычисления следующего перехода при маршрутизации в гиперкубе. В частности, шестнадцатеричные цифры могут быть заменены r-битовыми группами с r конфигурируемым параметром, не обязательно равным четырем.
[16] Однако у валидаторов есть некоторый стимул сделать это как можно скорее, потому что они смогут собрать все платежи за пересылку, связанные с сообщением, которое еще не было использовано на «медленном» пути.
[17] На самом деле, можно было временно или навсегда отключить механизм мгновенной доставки, и система продолжала бы работать, хотя и медленнее.

Затем подтверждение доставки вместе с подходящим доказательством Меркла отправляется обратно по краям гиперкуба, и его можно использовать, чтобы остановить движение сообщения по «медленному пути» путем выполнения специальной транзакции.
Обратите внимание, что этот механизм «мгновенной доставки» не заменяет «медленный», но отказоустойчивый механизм, описанный в 2.4.19. «Медленный путь» по-прежнему необходим, поскольку валидаторы не должны наказываться за потерю сообщения или просто решение не фиксировать сообщения в процессе «быстрого пути» в новых блоках своих блокчейнов[16].
Следовательно, оба метода пересылки сообщений выполняются параллельно, и «медленный» механизм прерывается только в том случае, если доказательство успеха «быстрого» механизма фиксируется в промежуточном шардчейне[17].

2.4.21. Сбор входных сообщений из очередей вывода соседних шардчейнов. Когда предлагается создание нового блока в шардчейне, некоторые из выходных сообщений соседних (в смысле гиперкуба маршрутизации 2.4.19) шардчейнов включаются в новый блок как «входные» сообщения и немедленно доставляются (т. е. обрабатываются). Существуют определенные правила относительно порядка, в котором выходные сообщения этих соседних шардчейнов должны обрабатываться. По сути, «старое» сообщение (исходящее из блока шардчейна, относящегося к более старому блоку мастерчейна) должно быть доставлено до любого «нового» сообщения; а для сообщений, поступающих из того же соседнего шардчейна, должен соблюдаться частичный порядок очереди вывода, описанный в 2.4.17.

2.4.22. Удаление сообщений из очередей вывода. После того, как сообщение очереди вывода помечается как доставленное соседним шардчейном сообщение, оно явно удаляется из очереди вывода специальной транзакцией.

2.4.23. Предотвращение двойной доставки сообщений. Чтобы предотвратить двойную доставку сообщений, взятых из очередей вывода соседних шардчейнов, каждый шардчейн (точнее, каждая цепочка учетных записей в шардчейне) сохраняет набор недавно доставленных сообщений (или только их хэшей)
как часть своего состояния. Когда обнаруживается, что доставленное сообщение было удалено из очереди вывода соответствующим исходным соседним шардчейном (см. 2.4.22), оно также удаляется из набора недавно доставленных сообщений.

2.4.24. Пересылка сообщений, предназначенных для других шардчейнов. Маршрутизация в гиперкубе (см. 2.4.19) означает, что иногда исходящие сообщения доставляются не в шардчейн, содержащий предполагаемого получателя, а в соседний шардчейн, находящийся в гиперкубе на пути к месту назначения. В этом случае «доставка» заключается в перемещении входящего сообщения в очередь исходящих сообщений. Этот процесс явно отражается в блоке как специальная транзакция пересылки, содержащая само сообщение. По сути, это выглядит так, как если бы сообщение было получено кем-то внутри шардчейна, и в результате было сгенерировано одно идентичное сообщение.

2.4.25. Оплата пересылки и хранения сообщения. Транзакция пересылки фактически расходует некоторое количество газа (в зависимости от размера пересылаемого сообщения), поэтому оплата газа вычитается из стоимости сообщения, пересылаемого от имени валидаторов этого шардчейна. Этот платеж за пересылку обычно значительно меньше, чем платеж за газ, который взимается, когда сообщение окончательно доставляется получателю, даже если сообщение было переадресовано несколько раз в процессе маршрутизации в гиперкубе. Кроме того, пока сообщение

хранится в очереди вывода шардчейна, оно является частью глобального состояния шардчейна, поэтому за хранение глобальных данных в течение длительного времени также может взиматься плата с помощью специальных транзакций.

2.4.26. Сообщения, передаваемые в мастерчейн и из него. Сообщения могут быть отправлены напрямую из любого шардчейна в мастерчейн и наоборот. Однако оплата за газ для отправки и обработки сообщений в мастерчейне довольно высокая, поэтому эта возможность будет использоваться только в случае необходимости - например, валидаторами для внесения своих долей. В некоторых случаях может быть определен минимальный депозит (прикрепленное значение) для отправляемых в мастерчейн сообщений, который будет возвращаться только в том случае, если сообщение считается «действительным» принимающей стороной.
Автоматическая маршрутизация сообщений в мастерчейне отсутствует. Сообщение с идентификатором workchain_id ≠ —1 (—1 — специальный workchain_id, указывающий на мастерчейн) не может быть доставлено в мастерчейн.
В принципе, можно создать смарт-контракт для пересылки сообщений внутри мастерчейна, однако цена за его использование будет непомерно высокой.

2.4.27. Сообщения между учетными записями в одном шардчейне. В некоторых случаях учетной записью, принадлежащей определенному шардчейну, создается сообщение, предназначенное для другой учетной записи в том же шардчейне. Например, это происходит в новом воркчейне, который еще не разделен на несколько шардчейнов благодаря нормальному уровню нагрузки.
Такие сообщения могут накапливаться в выходной очереди шардчейна, а затем обрабатываться как входящие сообщения в последующих блоках (для этой цели любой шард считается своим собственным соседним элементом). Однако в большинстве случаев эти сообщения можно доставить в самом исходном блоке.
Для этого для всех транзакций в блоке шардчейна вводится частичный порядок, а транзакции (каждая из которых заключается в доставке сообщения какой-либо учетной записи) обрабатываются с соблюдением этого частичного порядка. В частности, транзакции разрешено обрабатывать некоторое выходное сообщение предыдущей транзакции по отношению к этому частичному порядку.
В этом случае тело сообщения не копируется дважды. Вместо этого исходящая и обрабатывающая транзакции ссылаются на общую копию сообщения.

2.5. Глобальное состояние шардчейна. Философия «мешка ячеек».

Теперь мы готовы описать глобальное состояние блокчейна TON или как минимум шардчейна базового воркчейна.
Мы начинаем с «высокоуровневого» или «логического» описания, которое заключается в том, что глобальное состояние является значением алгебраического типа данных ShardchainState.

2.5.1. Состояние шардчейна как совокупность состояний цепочки учетных записей. Согласно парадигме бесконечного шардинга (см. 2.1.2), любой шардчейн представляет собой (временную) совокупность виртуальных «цепочек учетных записей», каждая из которых содержит ровно одну учетную запись. Это означает, что, по сути, состояние глобальной шардчейна представлено в виде хэш-карты.

$$ShardchainState := (Account \dashrightarrow AccountState) \qquad (23)$$

где все идентификаторы account_ id, появляющиеся как индексы этой хэш-карты, должны начинаться с префикса s, если имеется в виду состояние шарда (w, s) (см. 2.1.8).

На практике нам может потребоваться разделить AccountState на несколько частей (например, сохранить отдельную очередь выходных сообщений учетной записи, чтобы упростить ее проверку соседними шардчейнами) и использовать несколько хэш-карт (Account --> * $\text{AccountStatePart}_i$) внутри ShardchainState. Мы также можем добавить небольшое количество «глобальных» или «интегральных» параметров в ShardchainState (например, общий баланс всех учетных записей, принадлежащих этому шарду, или общее количество сообщений во всех очередях вывода).
Однако (23) – это наглядное приближение того, как выглядит глобальное состояние шардчейна, по крайней мере, с «логической» («высокоуровневой») точки зрения. Формальное описание алгебраических типов Accountstate и ShardchainState может быть выполнено с помощью TL-схемы (см. 2.2.5), которая будет предоставлена где-либо еще.
2.5.2. Разделение и объединение состояний шардчейна. Обратите внимание, что описание состояния шардчейна (23) в парадигме бесконечного шардинга показывает, как это состояние должно обрабатываться при разделении или слиянии шардчейнов. Фактически, эти преобразования состояний являются очень простыми операциями с хэш-картами.

2.5.3. Состояние цепочки учетных записей. Состояние (виртуальной) цепочки учетных записей - это просто состояние одной учетной записи, описываемое типом AccountState. Обычно в нем есть все или некоторые из полей, перечисленных в 2.3.20, в зависимости от конкретного используемого конструктора.

2.5.4. Состояние глобального воркчейна. Подобно (23), мы можем определить состояние глобального воркчейна по той же формуле, но с использованием идентификатора account_id, s, которому разрешено принимать любые значения, а не только значения, которые принадлежат одному шарду. В этом случае применимы замечания, подобные замечаниям в 2.5.1: мы можем разделить эту хэш-карту на несколько хэш-карт и добавить некоторые «интегральные» параметры, такие как общий баланс.
По сути, глобальное состояние воркчейна должно быть задано тем же типом ShardchainState, что и состояние шардчейна, потому что это состояние шардчейна, которое мы получили бы, если бы все существующие шардчейны данного воркчейна внезапно слились бы в один шардчейн.

2.5.5. Низкоуровневая перспектива: «мешок ячеек». Существует также «низкоуровневое» описание цепочки учетных записей или состояния шардчейна, дополняющее приведенное выше «высокоуровневое» описание. Это описание очень важно, потому что оно является довольно универсальным и обеспечивает общую основу для представления, хранения, сериализации и передачи по сети почти всех данных, используемых блокчейном TON (блоки, состояния шардчейна, хранилище смарт-контрактов, доказательства Меркла и т.д.). В то же время такое понятное универсальное «низкоуровневое» описание позволяет нам сосредоточить внимание только на соображениях «высокого уровня».
Напомним, что TVM представляет значения произвольных алгебраических типов (включая, например, ShardchainState из (23)) посредством дерева ячеек TVM, или для краткости «ячеек» (см. 2.3.14 и 2.2.5).
Любая такая ячейка состоит из двух байтов дескриптора, определяющих определенные флаги и значения $0 \le b \le 128$, количество необработанных байтов, и $0 \le c \le 4$ - количество ссылок на другие ячейки. Затем следуют b необработанных байтов и c ссылок на ячейки[18].
Точный формат ссылок на ячейки зависит от реализации и от того, находится ли ячейка в ОЗУ, на диске, в сетевом пакете, в блоке и т. д. Полезная абстрактная модель

заключается в том, что все ячейки хранятся в памяти с адресацией по содержимому с адресом ячейки, равным ее хэш-функции (SHA256). Напомним, что хеш (Меркла) ячейки вычисляется путем замены ссылок на ее дочерние ячейки их (рекурсивно вычисляемыми) хешами и хеширования полученной байтовой строки.
Таким образом, если мы используем хэши ячеек для ссылки на ячейки (например, внутри описаний других ячеек), система несколько упрощается, и хэш ячейки начинает совпадать с хешем представляющей ее байтовой строки.
Теперь мы видим, что любой представляемый TVM объект, включая глобальное состояние шардчейна, может быть представлен как «мешок ячеек» — т. е. набор ячеек вместе с «корневой» ссылкой на одну из них (например, хешем). Обратите внимание, что повторяющиеся ячейки отсутствуют в этом описании («набор ячеек» — это набор ячеек, а не мультимножество ячеек), поэтому представление абстрактного дерева фактически может стать представлением ориентированного ациклического графа (dag).
Это состояние можно сохранить на диске в дереве B- или B+-, содержащем все рассматриваемые ячейки (возможно, с некоторыми дополнительными данными, такими как высота поддерева или счетчик ссылок) и проиндексированном хэшем ячеек. Однако примитивная реализация этой идеи может привести к тому, что состояние одного смарт-контракта будет разбросано по удаленным частям файла на диске, чего мы предпочли бы избежать[19].
Теперь мы собираемся более подробно объяснить, как почти все объекты, используемые блокчейном TON, могут быть представлены в виде «мешков ячеек», демонстрируя тем самым универсальность этого подхода.

2.5.6. Блок шардчейна как «мешок ячеек». Блок шардчейна может быть описан алгебраическим типом и храниться как «мешок ячеек». Тогда простое двоичное представление блока может быть получено простым объединением строк байтов, представляющих каждую из ячеек в «мешке ячеек», в произвольном порядке. Это представление может быть улучшено и оптимизировано, например, путем предоставления списка смещений всех ячеек в начале блока и замены хеш-ссылок на другие ячейки 32-битными индексами в этом списке, когда это возможно.
Однако следует учитывать, что блок — это, по сути, «мешок ячеек», а все остальные технические детали — лишь мелкие компоненты оптимизации и реализации.

2.5.7. Обновление объекта как «мешка ячеек». Представьте, что у нас есть старая версия некоторого объекта, представленного в виде «мешка ячеек», и мы хотим представить новую версию того же объекта, предположительно не слишком отличающуюся от предыдущей. Новое состояние может быть представлено просто как еще один «мешок ячеек» с собственным корнем, из которого будут удалены все ячейки, встречающиеся в старой версии. По сути, оставшийся «мешок ячеек» является обновлением объекта. Каждый пользователь, у которого есть старая версия этого объекта и обновление, может вычислить новую версию, просто объединив два пакета ячеек и удалив старый корень (уменьшив его счетчик ссылок и освободив ячейку, если счетчик ссылок становится равным нулю).

[18] Можно показать, что если доказательства Меркла для всех данных, хранящихся в дереве ячеек, требуются одинаково часто, следует использовать ячейки с $b+ch \approx 2(h+r)$, чтобы минимизировать средний размер доказательства Меркла, где $h = 32$ — это размер хеша в байтах, а $r \approx 4$ — размер ссылки на ячейку в байтах. Другими словами, ячейка должна содержать либо две ссылки и несколько необработанных байтов, либо одну ссылку и около 36 необработанных байтов, либо вообще не иметь ссылок с 72 необработанными байтами.
[19] Наилучшей реализацией было бы сохранение состояния смарт-контракта в виде сериализированной строки, если оно маленькое, или в отдельном B-дереве, если оно большое. Тогда структура верхнего уровня, представляющая состояние блокчейна, будет B-деревом, листья которого могут содержать ссылки на другие B-деревья.

2.5.8. Обновления состояния учетной записи. В частности, обновления состояния учетной записи или глобального состояния шардчейна или любой хэш-карты могут быть представлены с использованием идеи, описанной в 2.5.7. Это означает, что когда мы получаем новый блок шардчейна (который представляет собой «мешок ячеек»), мы интерпретируем не только собственно этот «мешок ячеек», но сначала объединяем его с «мешком ячеек», представляющим предыдущее состояние шардчейна. В этом смысле каждый блок может «содержать» все состояние блокчейна.

2.5.9. Обновления в блоке. Напомним, что сам блок представляет собой «мешок ячеек», поэтому, если возникает необходимость отредактировать блок, можно аналогичным образом определить «обновление блока» как «мешок ячеек», интерпретируемый при наличии «мешка ячеек», который является предыдущей версией этого блока. Этот процесс является примерной идеей «вертикальных блоков», описываемых в 2.1.17.

2.5.10. Доказательство Меркла как «мешок ячеек». Обратите внимание, что (обобщенное) доказательство Меркла - например, утверждение, что x[i] = y, начиная с известного значения HASH(x) = h (см. 2.3.10 и 2.3.15), - также может быть представлено как «мешок ячеек». А именно, нужно просто предоставить подмножество ячеек, соответствующее пути от корня x: Hashmap(n, X) к целевому листу с индексом i: 2^n и значением y: X. Ссылки на дочерние элементы этих ячеек, которые не лежат на этом пути, останутся «нерешенными» в этом доказательстве, представленном хешами ячеек. Можно также обеспечить одновременное доказательство Меркла, скажем, x[i] = y и x[i'] = y', включив в «мешок ячеек» ячейки, лежащие на пути объединения двух путей от корня x в листы, соответствующие индексам i и i'.

2.5.11. Доказательства Меркла в виде ответов на запросы от полных нод. По сути, полная нода с полной копией состояния шардчейна (или цепочки учетных записей) может предоставить доказательство Меркла по запросу неполного узла (например, сетевого узла, на котором запущена облегченная версия клиента блокчейна TON), что позволяет получателю выполнять некоторые простые запросы без внешней помощи, используя только ячейки в этом доказательстве Меркла. Неполная нода может отправлять свои запросы в сериализированном формате на полную ноду и получать правильные ответы с доказательствами Меркла (или просто доказательства Меркла), потому что запрашивающая сторона должна иметь возможность вычислять ответы, используя только ячейки, включенные в доказательство Меркла. Это доказательство Меркла будет состоять из «мешка ячеек», содержащего только ячейки, принадлежащие к состоянию шардчейна, к которым получила доступ полная нода при выполнении запроса неполной ноды. Такой подход может использоваться, в частности, для выполнения «запросов на получение» смарт-контрактов (см. 4.3.12).

2.5.12. Дополненное обновление или обновление состояния с доказательством Меркла. Напомним (см. 2.5.7), что мы можем описать изменения в состоянии объекта от старого значения x: X к новому значению x': X с помощью «обновления», которое представляет собой просто «мешок ячеек» с ячейками, которые лежат в поддереве, представляющем новое значение x', но не в поддереве, представляющем старое значение x, поскольку предполагается, что получатель имеет копию старого значения x и всех его ячеек.
Однако если получатель не имеет полной копии x, но знает только ее хэш (Меркла) h = HASH(X), он не сможет проверить правильность обновления (т. е. все

«подвешенные» ссылки на ячейки в обновлении действительно относятся к ячейкам, присутствующим в дереве x). Хотелось бы иметь «проверяемые» обновления, дополненные доказательствами Меркла для существования всех упомянутых ячеек в старом состоянии. Тогда любой пользователь, который знает только h = HASH(X), сможет проверить правильность обновления и вычислить новый h' = HASH (X') самостоятельно.
Поскольку наши доказательства Меркла сами по себе являются «мешками ячеек» (см. 2.5.10), можно построить такое расширенное обновление, как «мешок ячеек», содержащий старый корень x, некоторые из его потомков вместе с путями из корня x, а также новый корень x' и всех его потомков, которые не являются частью x.

2.5.13. Обновление состояния учетной записи в блоке шардчейна. В частности, обновления состояния учетной записи в блоке шардчейна должны быть дополнены, как обсуждалось в 2.5.12. В противном случае кто-то может зафиксировать блок, содержащий недопустимое обновление состояния, ссылаясь на ячейку, отсутствующую в старом состоянии; доказать невалидность такого блока будет проблематично (как претенденту доказать, что ячейка не является частью предыдущего состояния?).
Если дополняются все обновления состояния, включенные в блок, их достоверность можно легко проверить, а их невалидность также отображается как нарушение рекурсивного определяющего свойства (обобщенных) хэшей Меркла.

2.5.14. Философия «все есть мешок ячеек». Предыдущие соображения показывают, что все данные, которые необходимо хранить или передавать, будь то в блокчейне TON или в сети, можно представить в виде «мешка ячеек». Это важная часть философии проектного решения блокчейна TON. После объяснения подхода «мешок ячеек» и определения некоторых «низкоуровневых» сериализаций «мешков ячеек» можно просто определить все (формат блока, шардчейн и состояние учетной записи и т. д.) с помощью высокого уровня абстрактных (зависимых) алгебраических типов данных.
Объединяющий эффект философии «все есть мешок ячеек» значительно упрощает реализацию, казалось бы, несвязанных сервисов (в п. 5.1.9 приведен пример с платежными каналами).

2.5.15. «Заголовки» блоков для блокчейнов TON. Обычно блок в блокчейне начинается с небольшого заголовка, содержащего хэш предыдущего блока, время его создания, хэш Меркла дерева всех транзакций, содержащихся в блоке, и так далее. Затем хеш блока определяется как хеш этого небольшого заголовка блока. Поскольку заголовок блока в конечном итоге зависит от всех данных, включенных в блок, нельзя невозможно изменить блок, не изменив его хэш.
При использовании подхода «мешок ячеек», который используется блокчейнами TON, назначенный заголовок блока отсутствует. Вместо этого хеш блока определяется как хеш (Меркла) корневой ячейки блока. Следовательно, верхняя (корневая) ячейка блока может считаться небольшим «заголовком» этого блока.
Однако корневая ячейка может не содержать всех данных, обычно ожидаемых от такого заголовка. По сути, нужно, чтобы заголовок содержал некоторые поля, определенные в типе данных Block. Обычно эти поля содержатся в нескольких ячейках, включая корневую ячейку. Эти ячейки вместе составляют «доказательство Меркла» для значений рассматриваемых полей. Можно было бы настаивать на том, чтобы эти «ячейки заголовка» содержались в самом начале блока, перед любыми другими ячейками. Тогда нужно будет загрузить только первые несколько байтов сериализации блока, чтобы получить все «ячейки заголовка» и изучить все необходимые поля.

2.6. Создание и проверка новых блоков

По сути, блокчейн TON состоит из блоков шардчейна и мастерчейна. Чтобы обеспечить бесперебойную и правильную работу системы, эти блоки должны создаваться, проверяться и распространяться по сети всем заинтересованным сторонам.

2.6.1. Валидаторы. Новые блоки создаются и проверяются специальными назначенными узлами, называемыми валидаторами. В сущности, любая желающая нода может стать валидатором, при условии внесения достаточно большого стейка (в монетах TON, см. Приложение A) в мастерчейн. Валидаторы получают "вознаграждения" за хорошую работу, а именно, комиссии за транзакции, хранение и газ со всех транзакций (сообщений), включенных в только что созданные блоки, и некоторое количество только что "отчеканенных" монет, что отражает "благодарность" всего сообщества валидаторам за поддержание работы блокчейна TON. Этот доход распределяется среди всех участвующих валидаторов пропорционально их стейкам.

Однако быть валидатором - это большая ответственность. Если валидатор подписывает невалидный блок, он может быть наказан потерей части или всего своего стейка, и временным или постоянным исключением из набора валидаторов. Если валидатор не участвует в создании блока, он не получает свою долю награды за этот блок. Если валидатор воздерживается от создания новых блоков длительное время, он может потерять часть стейка и быть временно или навсегда исключен из набора валидаторов.

Всё это значит, что валидатор не получает свои деньги "ни за что". Действительно, валидатор должен следить за состояниями всех или некоторых шардчейнов (каждый валидатор отвечает за валидацию и создание новых блоков в конкретном наборе шардчейнов), выполнять все вычисления для смарт-контрактов в этих шардчейнах, получать обновления о других шардчейнах и так далее. Эта деятельность требует значительного дискового пространства, вычислительной мощности и ширины интернет-канала.

2.6.2. Валидаторы вместо майнеров. Напомним, что блокчейн TON использует подход Proof-of-Stake, вместо подхода Proof-of-Work, используемого Биткоином, текущей версией Эфириума, и большинством других криптовалют. Это значит, что нельзя "майнить" новые блоки, предоставляя некое доказательство проделанной работы (вычисления большого количества в ином случае бесполезных хэшей) и получать в результате новые монеты. Вместо этого, нужно стать валидатором и тратить вычислительные мощности на хранение и обработку реквестов и данных блокчейна TON. Коротко говоря, нужно быть валидатором, чтобы "майнить" новые монеты. В этом смысле, валидаторы являются новыми майнерами.

Тем не менее, есть другие способы заработать монеты помимо становления валидатором.

2.6.3. Номинаторы и «майнинг-пулы». Чтобы стать валидатором, обычно требуется приобретать и устанавливать несколько высокопроизводительных серверов и обеспечивать хорошее интернет-соединение для них. Это не так дорого как оборудование ASIC, на данный момент требуемое для майнинга Биткоинов.

Однако, майнить новые монеты TON определенно не получится на домашнем компьютере, не говоря о смартфоне.

В майнинг-сообществах Биткоина, Эфириума и других Proof-of-Work криптовалютах есть определение «майнинг-пулов» с большим количеством нод, которые сами по себе не имеют достаточно мощности для майнинга новых блоков, но объединяют усилия и впоследствии разделяют награду.

Соответствующим определением в мире Proof-of-Stake является номинатор. По сути, номинатор - это нода, одалживающая свои деньги, чтобы помочь валидатору увеличить его стейк. Впоследствии валидатор отдает соответствующую часть своей награды (или ранее согласованную её долю, скажем, 50%) номинатору.
В этом смысле номинатор также может принять участие в «майнинге» и получить часть награды пропорционально количеству денег, которую он желает одолжить для этой цели. Он получает только часть доли от награды валидатора, поскольку он предоставляет только «капитал», но не имеет нужды приобретать вычислительную мощность, дисковое пространство и интернет-канал.
Однако если валидатор теряет свой стейк из-за невалидного поведения, номинатор также теряет свою долю стейка. В этом понимании номинатор разделяет риск. Он должен выбирать номинированного валидатора мудро, иначе он может потерять деньги. В этом смысле номинаторы принимают взвешенное решение и «голосуют» за конкретных валидаторов своими деньгами.
С другой стороны, эта система номинирования или одалживания позволяет стать валидатором без изначального инвестирования большого количества денег в монеты TON. Другими словами, это не позволяет обладателям большого количества монет TON стать монополистом в среде валидаторов.

2.6.4. Фишермен: получение денег за указание на чужие ошибки. Другой способ получить награду без необходимости становиться валидатором это возможность стать фишерменом. По сути, любая нода может стать фишерменом, сделав небольшой депозит в мастерчейн. После этого она может через специальные транзакции в мастерчейне публиковать доказательства (Меркла) невалидности каких-либо блоков (обычно в шардчейнах), ранее подписанных и опубликованных валидаторами. Если другие валидаторы соглашаются с доказательством невалидности, обвиняемые валидаторы наказываются (потерей части их стейка), и фишермен получает награду (часть монет, конфискованных у обвиняемых валидаторов). После чего невалидный блок (шардчейна) должен быть исправлен, как описано в 2.1.17. Исправление невалидных блоков мастерчейна может вовлекать создание "вертикальных" блоков поверх ранее включенных блоков мастерчейна (см. 2.1.17); необходимость создавать форк мастерчейна отсутствует.
Обычно фишермену может потребоваться стать полной нодой для хотя бы некоторых шардчейнов и тратить вычислительные ресурсы на выполнение кода хотя бы некоторых смарт-контрактов. Хотя фишермену не нужно иметь столько же вычислительной мощности как валидатору, мы считаем обычной кандидатурой в фишермены тех, кто мог бы быть валидатором, но не был выбран в качестве валидатора (например, из-за невозможности внесения достаточно большого стейка).

2.6.5. Коллаторы: получение денег за предложение новых блоков валидаторам. Еще один способ получать награду без становления валидатором - это становление коллатором. Это нода, которая готовит и предлагает валидаторам кандидаты в блоки шардчейна, дополненные (collated) данными, взятыми из состояния этого шардчейна и из других (обычно соседних) шардчейнов, вместе с приемлемыми доказательствами Меркла. (Это необходимо, например, когда некоторые сообщения нужно переслать из соседних шардчейнов.) Затем валидатор может легко проверить предложенный блок-кандидат на валидность без необходимости скачивать всё состояние этого или других шардчейнов.
Поскольку валидатору нужно отправлять новые (collated) кандидаты в блоки для получения награды, есть смысл платить часть награды коллатору, желающему предоставить подходящих блоков-кандидатов. Таким образом, валидатор может освободить себя от необходимости следить за состоянием соседних шардчейнов, отдавая это на аутсорс коллатору.

Тем не менее, мы ожидаем, что в фазе изначального развертывания системы в ней не будет отдельно назначенных коллаторов, поскольку все валидаторы будут иметь возможность являться коллаторами для самих себя.

2.6.6. Коллататоры или валидаторы: получение денег за включение пользовательских транзакций. Пользователи могут открывать каналы микроплатежей для некоторых коллекторов или валидаторов и платить небольшие суммы монет в обмен на включение своих транзакций в шардчейн.

2.6.7. Отбор «глобального» набора валидаторов. «Глобальный» набор валидаторов выбирается один раз в месяц (фактически, каждые 2^{19} блоков мастерчейна). Этот набор определяется и становится общеизвестным за месяц вперед.
Чтобы стать валидатором, узел должен передать несколько монет TON в мастерчейн, а затем отправить их на специальный смарт-контракт в качестве предлагаемого стейка. Другой параметр, отправляемый вместе со стейком, - это $l \geq 1$, максимальная проверочная нагрузка, которую этот узел готов принять, относительно минимально возможной. Также существует глобальная верхняя граница (еще один настраиваемый параметр) L для l, равная, скажем, 10.
Затем этот смарт-контракт выбирает глобальный набор валидаторов, просто выбирая до T кандидатов с максимальными предлагаемыми ставками и публикуя их описание. Изначально общее количество валидаторов T = 100; мы ожидаем, что оно вырастет до 1000 по мере увеличения нагрузки. Этот параметр является настраиваемым (см. 2.1.21),
Фактический стейк каждого валидатора рассчитывается следующим образом: если верхние значения T предложенных стейков составляют $s_1 \geq s_2 \geq \bullet \bullet \bullet \geq s_T$, то фактический стейк i-го валидатора устанавливается на $s'_i := \min(s_i, l_i \bullet s_T)$. Таким образом, $s'_i/s'_T \leq l_i$, поэтому i-й валидатор не получает больше чем $l_i \leq L$ раз больше нагрузки самого слабого валидатора (поскольку нагрузка в конечном итоге пропорциональна стейку).
Затем избранные валидаторы могут отозвать неиспользованную часть своего стейка, $s_i - s'_i$. Кандидаты, которые не стали валидаторами, могут отозвать всю свою предложенную долю.
Каждый валидатор публикует свой открытый ключ подписи, не обязательно равный открытому ключу учетной записи, из которой был получен стейк[20].
Ставки валидаторов замораживаются до окончания срока, на который они были избраны, а также еще на один месяц на случай возникновения новых споров (т.е. обнаружения невалидного блока, подписанного одним из этих валидаторов). После этого стейк возвращается вместе с долей валидатора в монетах и комиссиями от транзакций, обработанных за это время.

2.6.8. Выбор «групп задач» валидаторов. Весь глобальный набор валидаторов (где каждый валидатор считается присутствующим с множественностью, равной его стейку - в противном случае у валидатора может возникнуть соблазн использовать несколько идентификаторов и разделить свой стейк между ними) используется только для проверки новых блоков мастерчейна. Блоки шардчейна проверяются только специально выбранными подмножествами валидаторов из глобального набора валидаторов, выбранных в соответствии с п. 2.6.7.

[20] Имеет смысл генерировать и использовать новую пару ключей при каждом подборе валидаторов.

Эти «подмножества» или «группы задач» валидаторов, определенные для каждого шарда, меняются каждый час (фактически, каждые 2^{10} блоков мастерчейна), причем они становятся известны за час вперед, так что каждый валидатор знает, какие шарды ему необходимо будет проверить, и может подготовиться к этому (например, загрузив недостающие данные шардчейна).
Для выбора групп задач валидаторов для каждого шарда (w, s) применяется детерминированный псевдослучайный алгоритм.
Он использует псевдослучайные числа, встроенные валидаторами в каждый блок мастерчейна (сгенерированные путем получения консенсуса с использованием пороговых сигнатур) для создания случайного начального числа, а затем вычисляет, например, Hash(code(w): code(s):validator_id:rand_seed) для каждого валидатора. Затем валидаторы сортируются по значению этого хэша, и выбираются первые несколько валидаторов, имеющих не менее 20/T от общих стеков валидаторов (группа состоит как минимум из 5 валидаторов).
Этот выбор может быть сделан с помощью специального смарт-контракта. В этом случае алгоритм выбора можно было бы легко обновить без хард-форков с помощью механизма голосования, упомянутого в 2.1.21. Все другие упомянутые до сих пор «константы» (такие как 2^{19}, 2^{10}, T, 20 и 5) также являются настраиваемыми параметрами.

2.6.9. Смена приоритета в каждой группе задач. Для членов группы задач шарда вводится определенный «приоритетный» порядок, зависящий от хэша предыдущего блока мастерчейна и порядкового номера блока (шардчейна). Этот порядок определяется путем генерации и сортировки хэшей, как описано выше.
Когда необходимо сгенерировать новый блок шардчейна, валидатор группы задач шарда, выбранный для создания этого блока, обычно является первым в отношении этого чередующегося «приоритетного» порядка. Если создать блок не удается, это может сделать второй или третий валидатор. По сути, все они могут предлагать своих блоки-кандидаты, однако кандидат, предложенный валидатором с наивысшим приоритетом, должен победить в результате применения протокола консенсуса (задача византийских генералов, BFT).

2.6.10. Распространение блоков-кандидатов шардчейна. Поскольку членство в группе задач шардчейна известно за час вперед, их участники могут использовать это время для создания выделенной «многоадресной оверлейной сети для валидаторов шардчейна», используя общие механизмы сети TON (см. 3.3). Когда необходимо сгенерировать новый блок шардчейна - обычно через одну или две секунды после создания самого последнего блок мастерчейна – все валидаторы знают, у кого наивысший приоритет для генерации следующего блока (см. 2.6.9). Этот валидатор создает новый блок-кандидат либо сам, либо с помощью коллатора (см. 2.6.5). Валидатор должен проверить (подтвердить) этот блок-кандидат (особенно, если он был подготовлен коллатором) и подписать его своим закрытым ключом (валидатора). Затем блок-кандидат распространяется на оставшуюся часть группы задач с использованием заранее организованной оверлейной сети многоадресной рассылки (группа задач создает свою собственную частную оверлейную сеть, как описано в п. 3.3, а затем использует версию протокола многоадресной потоковой передачи, описанную в п. 3.3.15, для распространения блоков-кандидатов).
Наиболее правильный метод – это использование BFT протокола многоадресной рассылки, такого, который используется в Honey Badger BFT [11]: закодировать блока-кандидата с помощью кода стирания (N, 2N/3), отправить 1/N полученных данных непосредственно каждому члену группы и ожидать, что они будут передавать свою часть данных в рамках многоадресной рассылки напрямую всем другим членам группы.

Однако более быстрый и простой способ сделать это (см. также 3.3.15) заключается в том, чтобы разбить блок-кандидат на последовательность подписанных однокилобайтных блоков («фрагментов»), дополнить их последовательность с помощью кода Рида-Соломона или исходного кода (такой как RaptorQ [9] [14]) и начать передачу фрагментов соседним узлам в «сетке многоадресной рассылки» (т. е. оверлейной сети), ожидая, что они будут распространять эти фрагменты дальше.
Как только валидатор получает достаточно фрагментов для восстановления на их основе блока-кандидата, он подписывает валидацию и распространяет ее по всем соседям в своей группе. Затем соседи валидатора перестают посылать ему новые фрагменты, но могут продолжать отправлять (исходные) подписи этих фрагментов, полагая, что этот узел может генерировать последующие фрагменты, применяя код Рида-Соломона или фонтанный код (имея все необходимые данные), объединить их с подписями и передать их соседним узлам, которые еще не закончили процесс валидации.
Если «многоадресная сетка» (оверлейная сеть) остается подключенной после удаления всех «плохих» узлов (напомним, что до одной трети узлов могут вести себя злонамеренно), этот алгоритм распространяет блок-кандидат как можно быстрее.
Не только назначенный создатель высокоприоритетного блока может выполнять многоадресную рассылку своего блока-кандидата на всю группу. Валидаторы со вторым и третьим приоритетом могут начать многоадресную рассылку своих блоков-кандидатов либо сразу, либо после того, как не получили блок-кандидат от валидатора с наивысшим приоритетом. Однако обычно только блок-кандидат с максимальным приоритетом будет подписан всеми валидаторами (фактически как минимум двумя третями группы задач) и зафиксирован как новый блок шардчейна.

2.6.11. Проверка блоков-кандидатов. Сразу после получения блока-кандидата валидатором и проверки его исходной подписи принимающий валидатор проверяет валидность этого блока-кандидата, выполняя все транзакции и контролируя, чтобы их результат совпадал с заявленным результатом. Все сообщения, импортированные из других блокчейнов, должны поддерживаться подходящими доказательствами Меркла в сопоставленных данных, в противном случае блок-кандидат считается невалидным (и, если соответствующее доказательство было передано в мастерчейн, уже подписавшие этот блок-кандидат валидаторы могут быть наказаны). С другой стороны, если блок-кандидат признан валидным, принимающий валидатор подписывает его и распространяет свою подпись другим валидаторам в группе либо через «ячеистую многоадресную сеть», либо посредством прямых сетевых сообщений.
Следует подчеркнуть, что валидатору не нужен доступ к состояниям текущего или соседних шардчейнов для проверки валидности (сопоставленного) блока-кандидата[21]. Это позволяет проводить валидацию очень быстро (без доступа к диску) и снижает вычислительную нагрузку и нагрузку на хранилище для валидаторов (особенно если они готовы принять услуги внешних коллаторов для создания блоков-кандидатов).

2.6.12. Отбор следующего блока кандидата. Как только блок-кандидат собирает не менее двух третей (по стейкам) подписей валидаторов в группе задач, он может быть зафиксирован в качестве следующего блока шардчейна.

[21] Возможным исключением является состояние выходных очередей соседних шардчейнов, необходимое для обеспечения требований к порядку сообщений, описанных в 2.4.21, поскольку в этом случае размер доказательств Меркла может стать непомерно высоким.

Протокол BFT используется для достижения консенсуса по выбранному блоку-кандидату (может быть предложено несколько блоков), при этом все «хорошие» валидаторы предпочитают блок-кандидат с наивысшим приоритетом для этого раунда. В результате использования этого протокола блок дополняется подписями не менее двух третей валидаторов (по стекам).

Эти подписи свидетельствуют не только о валидности рассматриваемого блока, но и о его отборе в соответствии с протоколом BFT. После этого блок (без сопоставленных данных) объединяется с этими подписями, сериализуется детерминированным способом и передается по сети всем заинтересованным сторонам.

2.6.13. Валидаторы должны сохранять подписанные ими блоки. Ожидается, что во время своего членства в группе задач и в течение как минимум одного часа (или, скорее, 2^{10} блоков) после этого валидаторы будут сохранять блоки, которые они подписали и зафиксировали. Непредставление подписанного блока другим валидаторам может повлечь за собой штрафные санкции.

2.6.14. Передача заголовков и подписей новых блоков шардчейна всем валидаторам. Валидаторы передают заголовки и подписи вновь созданных блоков шардчейна глобальному набору валидаторов, используя многоадресную ячеистую сеть, аналогичную той, которая создается для каждой группы задач.

2.6.15. Генерация новых блоков мастерчейна. После того, как все (или почти все) новые блоков шардчейна были сгенерированы, может быть сгенерирован новый блоков мастерчейна. Процедура, по существу, такая же, как и для блоков шардчейна (см. 2.6.12), с той лишь разницей, что все валидаторы (или как минимум две трети из них) должны участвовать в этом процессе. Поскольку заголовки и подписи новых блоков шардчейна передаются всем валидаторам, хэши самых новых блоков в каждом шардчейне могут и должны быть включены в новый блок мастерчейна. После того, как эти хэши будут зафиксированы в блоке мастерчейна, внешние наблюдатели и другие шардчейны могут считать новые блоки шардчейна зафиксированными и неизменными (см. 2.1.13).

2.6.16. Валидаторы должны сохранять состояние мастерчейна. Заслуживающая внимания разница между мастерчейном и шардчейном заключается в том, что все валидаторы должны отслеживать состояние мастерчейна, не полагаясь на сопоставленные данные. Это важно, потому что информация, получаемая группой задач валидаторов, основана на состоянии мастерчейна.

2.6.17. Блоки шардчейна генерируются и распространяются параллельно. Обычно каждый валидатор является членом нескольких групп задач шардчейна; их количество (и, следовательно, нагрузка на валидатора) примерно пропорционально стейку валидатора. Это означает, что валидатор параллельно запускает несколько экземпляров нового протокола генерации блоков шардчейна.

2.6.18. Предупреждение атак с удержанием блоков. Поскольку полный набор валидаторов вставляет хэш нового блока шардчейна в мастерчейн после просмотра только его заголовка и подписей, существует небольшая вероятность того, что валидаторы, которые сгенерировали этот блок, вступят в сговор и попытаются избежать публикации нового блока целиком. Это приведет к неспособности валидаторов соседних шардчейнов создавать новые блоки, потому что они должны

знать как минимум очередь выходных сообщений нового блока после того, как его хэш будет зафиксирован в мастерчейне.
Чтобы предупредить эту проблему, новый блок должен собирать подписи от некоторых других валидаторов (например, двух третей объединения групп задач соседних шардчейнов), свидетельствующих о том, что эти валидаторы действительно имеют копии этого блока и готовы отправить их любым другим валидаторам при необходимости. Только после предоставления этих подписей хэш нового блока может быть включен в мастерчейн.

2.6.19. Блоки мастерчейна генерируются позже, чем блоки шардчейна. Блоки мастерчейна генерируются примерно раз в пять секунд, как и блоки шардчейна. Однако в то время как генерация новых блоков во всех шардчейнах выполняется по существу одновременно (обычно этот процесс запускается после выпуска нового блока мастерчейна), генерация новых блоков мастерчейна намеренно задерживается, чтобы обеспечить включение хэшей вновь сгенерированных блоков шардчейна в мастерчейне.

2.6.20. Более медленные валидаторы могут получать меньшее вознаграждение. Если валидатор работает «медленно», он может не успевать проверить новые блоки-кандидаты, и две трети подписей, необходимых для фиксации нового блока, могут быть собраны без его участия. В этом случае он получит меньшую долю вознаграждения, связанного с этим блоком.
Это дает валидаторам стимул оптимизировать свое оборудование, программное обеспечение и сетевое соединение, чтобы обрабатывать транзакции пользователей как можно быстрее.
Однако если валидатору не удается подписать блок до его фиксации, его подпись может быть включена в один из следующих блоков, а затем часть вознаграждения (экспоненциально уменьшается в зависимости от того, сколько блоков было сгенерировано с тех пор - например, 0.9^k, если валидатор опаздывает на k блоков) будет по-прежнему передана этому валидатору.

2.6.21. «Глубина» подписей валидатора. Обычно, когда валидатор подписывает блок, подпись свидетельствует только об относительной валидности блока: этот блок действителен при условии, что действительны все предыдущие блоки в этом и других сегментах. Валидатор не может быть наказан за то, что принял недействительные данные, переданные в предыдущие блоки.
Однако подпись валидатора блока имеет целочисленный параметр, называемый «глубиной». Если он не равен нулю, это означает, что валидатор также подтверждает (относительную) валидность указанного количества предыдущих блоков. Этот способ позволяет «медленным» или «временно отключенным» валидаторам догнать процесс и подписать некоторые из блоков, которые были зафиксированы без их подписей. Тогда валидаторам все равно будет передана некоторая часть награды за блок (см. 2.6.20).

2.6.22. Валидаторы несут ответственность за относительную валидность подписанных блоков шардчейна. После этого следует абсолютная валидность. Мы хотели бы еще раз подчеркнуть, что подпись валидатора на блоке шардчейна B свидетельствует только об относительной валидности этого блока (или также d предыдущих блоков, если подпись имеет «глубину» d, см. 2.6.21; но это не сильно влияет на последующее обсуждение). Другими словами, валидатор утверждает, что следующее состояние s' шардчейна выводится из предыдущего состояния s путем применения функции оценки блока ev_block, описанной в 2.2.6:

$$s' = ev\_block(B)(s) \qquad (24)$$

Таким образом, валидатор, подписавший блок B, не может быть наказан, если исходное состояние s оказывается «некорректным» (например, из-за невалидности одного из предыдущих блоков). Фишермен (см. 2.6.4) может указать на ошибку, только если он находит относительно невалидный блок. Система PoS в целом стремится сделать каждый блок относительно валидным, а не рекурсивно (или абсолютно) валидным. Однако следует обратить внимание, что если все блоки в блокчейне являются относительно валидными, то все они и блокчейн в целом являются абсолютно валидными; это утверждение легко проиллюстрировать с помощью математической индукции по длине блокчейна. Таким образом, легко проверяемые утверждения об относительной валидности блоков вместе демонстрируют гораздо большую абсолютную валидность всего блокчейна.

Обратите внимание: подписывая блок B, валидатор утверждает, что блок является валидным в исходном состоянии s (т. е. результат (24) не является значением ?, указывающим, что следующее состояние не может быть вычислено). Таким образом, валидатор должен выполнять минимальные формальные проверки ячеек исходного состояния, к которым осуществляется доступ во время оценки (24).

Например, представим, что ячейка, которая, как ожидается, будет содержать исходный баланс учетной записи, к которой осуществляется доступ из зафиксированной в блоке транзакции, имеет нулевые необработанные байты вместо ожидаемых 8 или 16 байтов. Тогда исходный баланс просто не может быть получен из ячейки, и при попытке обработать блок происходит «необработанное исключение». В этом случае валидатор не должен подписывать такой блок под угрозой наказания.

2.6.23. Подписание блоков мастерчейна. Ситуация с блоками мастерчейна несколько иная: подписывая блок мастерчейна, валидатор подтверждает не только его относительную валидность, но и относительную валидность всех предшествующих блоков до самого первого блока, с которого этот валидатор начал проверять блоки (но не дальше).

2.6.24. Общее количество валидаторов. Верхний лимит T для общего числа валидаторов, которых можно выбрать (см. 2.6.7), не может быть в описанной системе больше чем, скажем, несколько сотен или тысяча, потому что все валидаторы должны принимать участие в протоколе консенсуса BFT для создания каждого нового блока мастерчейна, и неясно, могут ли такие протоколы масштабироваться до тысяч участников. Ещё более важно, что блоки мастерчейна должны собирать подписи как минимум двух третей валидаторов (по стейку), и эти подписи должны быть включены в новый блок (иначе все другие ноды в системе не будут иметь причины доверять новому блоку до собственноручной его валидации). Если, скажем, одна тысяча подписей валидаторов будет включаться в каждый блок мастерчейна, это будет подразумевать больше информации в каждом блоке мастерчейна, которую нужно хранить всем полным нодам и распространять через сеть, и больше вычислительной мощности будет тратиться на проверку этих подписей (в PoS-системе полные ноды не должны валидировать блоки сами по себе, но должны проверять подписи валидаторов вместо этого).

Хотя ограничение T до тысячи валидаторов выглядит более эффективным для первой фазы развертывания блокчейна TON, должен быть запас для будущего роста, когда общее число шардчейнов станет настолько большим, что нескольких сотен валидаторов не будет достаточно для обработки их всех. С этой целью мы представляем дополнительный конфигурируемый параметр $T' \leq T$ (изначально равный T), и только верхние T' выбранных валидаторов (по стейку) будут создавать и подписывать новые блоки мастерчейна.

2.6.25. Децентрализация системы. Можно заподозрить, что Proof-of-Stake система, такая как блокчейн TON, полагающаяся на T≈1000 валидаторов для создания всех блоков шардчейнов и мастерчейна, обязательно станет «слишком централизованной», в противовес общепринятым Proof-of-Work блокчейнам, таким как Биткоин или Эфириум, где все (в принципе) могут майнить новые блоки, без явно выраженного верхнего лимита на число майнеров.
Однако популярные Proof-of-Work блокчейны, такие как Биткоин и Эфириум, на текущий момент требуют огромного количества вычислительной мощности (высокие «хэш-рейты») для майнинга новых блоков с приемлемой вероятностью успеха. Таким образом, майнинг новых блоков имеет тенденцию концентрироваться в руках нескольких крупных игроков: одни инвестируют большие деньги в датацентры и специальное программное обеспечение, оптимизированное для майнинга, другие же концентрируют и координируют усилия больших групп людей, неспособных самостоятельно предоставить достаточный «хэш-рейт», тем самым образуя крупные майнинг-пулы.
Таким образом, по состоянию на 2017 год больше чем 75% новых блоков Эфириума и Биткоина создаются менее чем десятью майнерами. Фактически, два крупнейших майнинг-пула Эфириума производят вместе больше половины всех новых блоков! Очевидно, такая система намного более централизована чем та, которая полагается на T≈1000 нод для производства новых блоков.
Также следует отметить, что вложения, необходимые для того, чтобы стать валидатором блокчейна TON - то есть, для приобретения оборудования (скажем, несколько высокопроизводительных серверов) и стейка (который при необходимости может быть легко собран через пул номинаторов; см. 2.6.3) - намного меньше, чем требуемые для успешного самостоятельного майнинга Биткоина или Эфириума. По сути, параметр L будет заставлять номинаторов не присоединяться к крупнейшему "майнинг-пулу" (то есть, к валидатору, который собрал крупнейший стейк), но, скорее, искать меньших валидаторов, на данный момент собирающих средства от номинаторов, или даже создавать новых валидаторов, потому что это обеспечит более высокую пропорцию s'_i/s_i стейка валидатора — и как следствие номинатора, откуда следует более высокая награда за майнинг. Таким образом, Proof-of-Stake система TON на самом деле поощряет децентрализацию (создание и использование большего числа валидаторов) и наказывает централизацию.

2.6.26. Относительная надежность блока. (Относительная) надежность блока - это просто сумма стейков всех валидаторов, подписавших данный блок. Другими словами, это сумма денег, которую некоторые акторы потеряют, если этот блок окажется невалидным. Если передаваемая транзакциями стоимость ниже уровня надежности блока, их можно считать достаточно безопасными. В этом смысле относительная надежность - это мера доверия внешнего наблюдателя по отношению к определенному блоку.
Обратите внимание, что показатель относительной надежности блока является гарантией валидности блока, при условии, что предыдущий блок и все упомянутые блоки других шардчейнов являются валидными (см. 2.6.22).
Относительная надежность блока может возрастать после его фиксации — например, при добавлении подписей «медленных» валидаторов (см. 2.6.21). С другой стороны, если один из этих валидаторов теряет часть или весь свой стейк из-за его некорректности, связанной с некоторыми другими блоками, относительная надежность блока может снизиться.

2.6.27. «Усиление» блокчейна. Важно создать стимулы для валидаторов, чтобы максимально повысить относительную надежность блоков. Один из способов сделать - это выделить небольшое вознаграждение валидаторам за добавление подписей к

блокам других шардчейнов. Даже «потенциальные» валидаторы, которые внесли сумму, недостаточную для того, чтобы попасть в ТОП T валидаторов по размеру стейка и быть включенными в глобальный набор валидаторов (см. 2.6.7), могут участвовать в этой деятельности (если они согласны оставить свой стейк вместо того, чтобы снимать его после неудачных результатов отбора). Такие потенциальные валидаторы могут выступать в роли фишерменов (см. 2.6.4): если им все равно нужно проверять валидность определенных блоков, они могут также сообщать о невалидных блоках и получать связанные с ними вознаграждения.

2.6.28. Рекурсивная надежность блока. Рекурсивную надежность блока можно определить как минимум его относительной надежности и рекурсивной надежности всех блоков, на которые он ссылается (т. е. блок мастерчейна, предыдущего блока шардчейна и некоторых блоков соседних шардчейнов). Другими словами, если блок окажется невалидным сам по себе, либо потому, что недействителен один из блоков, от которых он зависит, то кем-то будет потеряна как минимум эта сумма денег. Если пользователь действительно не уверен, следует ли доверять конкретной транзакции в блоке, следует вычислить рекурсивную надежность этого блока, а не только относительную.

Нет смысла возвращаться слишком далеко назад при вычислении рекурсивной надежности – в этом случае мы увидим блоки, подписанные валидаторами, чьи ставки уже были разморожены и сняты. Как бы то ни было, мы не позволяем валидаторам автоматически пересматривать слишком старые блоки (т. е. блоки, созданные более двух месяцев назад, если используются текущие значения настраиваемых параметров) и создавать на их основании форки, либо исправлять их с помощью «вертикальных блокчейнов» (см. 2.1.17), даже если они окажутся невалидными. Мы предполагаем, что два месяца – это достаточный период для того, чтобы обнаружить невалидные блоки и сообщить о них, поэтому, если валидность блока не будет оспариваться в течение этого периода, она вряд ли будет оспариваться когда-либо.

2.6.29. Последствия подхода Proof-of-Stake для неполных нод. Важным следствием подхода Proof-of-Stake, используемого в блокчейне TON, является то, что неполной ноде (запускающей «облегченное» клиентское программное обеспечение) блокчейна TON не нужно загружать «заголовки» всех шардчейнов или даже блоков мастерчейна, чтобы иметь возможность самостоятельно проверить достоверность доказательств Меркла, предоставленных неполной ноде полными нодами в качестве ответов на ее запросы.

В самом деле, поскольку самые последние хэши блоков шардчейна включены в блоки мастерчейна, полная нода может легко предоставить доказательство Меркла, что данный блок шардчейна является валидным, начиная с известного хэша блока мастерчейна. Кроме того, неполная нода должна «знать» только самый первый блок мастерчейна (где объявляется самый первый набор валидаторов), который (или, по крайней мере, хэш которого) может быть встроен в клиентское программное обеспечение, и только один блок мастерчейна примерно каждый месяц после этого, когда объявляются новые наборы валидаторов, потому что этот блок будет подписан предыдущим набором валидаторов. Начиная с этого момента, нода может получить несколько самых последних блоков мастерчейна, либо как минимум их заголовки и подписи валидаторов, и использовать их в качестве основы для проверки доказательств Меркла, предоставленных полными нодами.

2.7. Разделение и объединение шардчейнов

Одной из наиболее характерных и уникальных особенностей блокчейна TON является его способность автоматически разделять шардчейн на две части, когда нагрузка становится слишком высокой, и объединять их обратно, если нагрузка снижается (см.

2.1.10). Эту особенность следует обсудить более подробно из-за ее уникальности и важности для масштабируемости всего проекта.

2.7.1. Конфигурация шарда. Напомним, что в любой момент времени каждый воркчейн w разбивается на один или несколько шардчейнов (w, s) (см. 2.1.8). Эти шардчейны могут быть представлены листьями двоичного дерева, корнем (w, 0) и каждым не-листовым узлом (w, s), имеющим дочерние элементы (w, s.0) и (w, s.1). Таким образом, каждая учетная запись в воркчейне w назначается ровно одному шарду, и все, кто знают текущую конфигурацию шардчейна, могут определить сегмент (w, s), содержащий учетную запись account_id: это единственный сегмент с двоичной строкой s, являющейся префиксом account_id.
Конфигурация шарда - то есть это двоичное дерево шарда или совокупность всех активных (w, s) для данного w (соответствующего листьям двоичного дерева шарда) - является частью состояния мастерчейна и доступна всем, кто отслеживает мастерчейн[22].

2.7.2. Самая последняя конфигурация и состояние шарда. Напомним, что хэши самых последних блоков шардчейна включены в каждый блоков мастерчейна. Эти хэши организованы в двоичное дерево шардов (на самом деле это набор деревьев, по одному на каждый воркчейн). Таким образом, каждый блок мастерчейна содержит самую последнюю конфигурацию шарда.

2.7.3. Объявление и внесение изменений в конфигурацию шарда. Конфигурацию шарда можно изменить двумя способами: шард (w, s) можно разделить на два шарда (w, s.0) и (w, s.1), либо два «родственных» шарда (w, s .0) и (w, s.1) можно объединить в один шард (w, s).
Эти операции разделения/слияния объявляются несколькими блоками заранее (например, 2^6 - это настраиваемый параметр), сначала в «заголовках» соответствующих блоков шардчейна, а затем в блоке мастерчейна, который относится к этим блокам шардчейна. Такое заблаговременное объявление позволяет всем заинтересованным сторонам подготовиться к запланированному изменению (например, создать оверлейную многоадресную сеть для распределения новых блоков вновь созданных шардчейнов, как описано в 3.3). Затем изменение фиксируется сначала в блоке (или заголовке) шардчейна (в случае разделения; при слиянии блоки обоих шардчейнов должны зафиксировать изменение), а затем передается на блоки мастерчейна. Таким образом, блок мастерчейна определяет не только самую последнюю конфигурацию шарда перед его созданием, но также и следующую конфигурацию шарда.

2.7.4. Группы задач валидаторов для новых шардчейнов. Напомним, что для каждого шардчейна обычно назначается подмножество валидаторов (группа задач валидаторов), предназначенных для создания и проверки новых блоков в соответствующем шардчейне (см. 2.6.8). Эти группы задач избираются на некоторый период времени (приблизительно один час), являются известными заранее (также приблизительно за час) и остаются неизменными в течение этого периода времени[23].
Однако фактическая конфигурация шардчейнов может измениться в течение этого периода времени из-за операций разделения/слияния. Необходимо назначить группы задач для вновь созданных шардов. Это делается следующим образом:

[22] На самом деле конфигурация шарда полностью определяется последним блоком мастерчейна; это упрощает получение доступа к конфигурации шарда.
[23] Если некоторые валидаторы будут временно или навсегда забанены из-за подписания невалидных блоков, то они автоматически исключаются из всех групп задач.

Обратите внимание, что любой активный шард (w, s) будет либо наследником некоторого однозначно определенного исходного шарда (w, s'), то есть, s' является префиксом s, либо он будет корнем поддерева исходных шардов (w, s'), где s будет префиксом каждого s'. В первом случае просто берется и удваивается группа задач исходного шарда (w, s'), в результате чего получается группа задач нового шарда (w, s). Во втором случае группа задач нового шарда (w, s) будет объединением групп задач всех исходных шардов (w, s '), которые являются наследниками (w, s) в дереве шардов. Таким образом, для каждого активного шарда (w, s) назначается четко определенное подмножество валидаторов (группа задач). При разделении оба результирующих шарда наследуют всю группу задач от исходного шарда. При объединении двух шардчейнов соответствующие группы задач также объединяются.
Любой, кто отслеживает состояние мастерчейна, может вычислить группы задач валидаторов для каждого из активных шардов.

2.7.5. Ограничение операций разделения/объединения в период ответственности исходных групп задач. В конечном итоге будет учтена новая конфигурация шарда, и для каждого шарда будут автоматически назначены новые выделенные подмножества (группы задач) валидаторов. Прежде чем это произойдет, необходимо наложить определенный лимит на операции разделения/слияния; в противном случае исходная группа задач может закончить проверку 2^k шардчейнов для большого k одновременно, если исходный шард быстро разделится на 2^k новых шардчейнов.
Это достигается путем наложения ограничений на то, насколько активная конфигурация шарда может быть удалена от исходной конфигурации шарда (той, которая используется для выбора групп задач валидаторов, отвечающих за проверку блоков в настоящее время). Например, может быть задано требование, чтобы расстояние в дереве шардов от активного шарда (w, s) до исходного шарда (w, s') не превышало 3, если s' является предшественником s (т. е. S' является префиксом двоичной строки s), и не должно превышать 2, если s' является преемником s (т. е. s является префиксом s'). В противном случае операция разделения или слияния не будет разрешена.
Грубо говоря, в данном случае вводится ограничение на количество процессов разделения (например, три) или объединения (например, два) шарда в течение периода ответственности данного набора валидаторов. Кроме того, после создания шарда в результате слияния или разделения его нельзя перенастроить в течение некоторого периода времени (некоторого количества блоков).

2.7.6. Определение необходимости операций разделения. Операция разделения шардчейна запускается при определенных формальных условиях (например, если 64 последовательных блока шардчейна заполнены не менее чем на 90%). Эти условия отслеживаются группой задач шардчейна.
Если они выполняются, сначала в заголовок нового блока шардчейна включается флаг «подготовки к разделению» (он также распространяется на блок мастерчейна, относящийся к данному блоку шардчейна). Через несколько блоков в заголовок блока шардчейна включается флаг «выполнить разделение» (он распространяется на следующий блок мастерчейна).

2.7.7. Выполнение операций разделения. После добавления флага «выполнить разделение» в блок B шардчейна (w, s) в этом шардчейне не может быть последующего блока B'. Вместо этого будут созданы блоки B'_0 и B'_1 шардчейнов (w, s.0) и (w, s.1), соответственно, причем оба они будут ссылаться на блок B как на предыдущий блок (и у обоих шардчейнов в заголовке будет флаг с указанием, что

шард только что был разделен). Следующий блок мастерчейна будет содержать хэши блоков B'$_0$ и B'$_1$ новых шардчейнов; в блоке не может содержаться хэш нового блока шардчейна B' (w, s), потому что событие «выполнить разделение» уже было зафиксировано в предыдущем блоке мастерчейна.
Обратите внимание, что оба новых шардчейна будут проверяться той же группой задач валидаторов, что и исходный шардчейн, поэтому они автоматически получат копию своего состояния. Сама операция разделения состояний довольно проста с точки зрения парадигмы бесконечного шардинга (см. 2.5.2)

2.7.8. Определение необходимости операций слияния. Необходимость операций слияния шардчейнов также определяется некоторыми формальными условиями (например, если в 64 последовательных блоках сумма размеров двух блоков родственных шардчейнов не превышает 60% от максимального размера блока). Формальные условия также должны учитывать общий объем газа, израсходованный этими блоками, и сравнивать его с текущим лимитом газа на блок, в противном случае блоки могут оказаться небольшими из-за наличия некоторых транзакций, требующих большого объема вычислений, которые препятствуют включению большего количества транзакций.
Эти условия отслеживаются группами задач валидаторов обоих родственных шардов (w, s.0) и (w, s.1). Обратите внимание, что родственные шардчейны обязательно являются соседями с точки зрения маршрутизации в гиперкубе (см. 2.4.19), поэтому валидаторы из группы задач любого шарда в любом случае будут в какой-то степени контролировать родственный шард.
Если эти условия выполнены, одна из подгрупп валидаторов может предложить другой объединиться, отправив специальное сообщение. Затем они объединяются во временную «объединенную группу задач» с объединенным членством, способную запускать согласованные алгоритмы BFT и при необходимости распространять обновления блоков и блоков-кандидатов.
Если валидаторы достигают консенсуса относительно необходимости и готовности слияния, флаги «подготовки к слиянию» фиксируются в заголовках некоторых блоков каждом шардчейне вместе с подписями не менее двух третей валидаторов группы задач родственного шардчейна (и распространяются на следующие блоки мастерчейна, чтобы все могли подготовиться к неминуемой реконфигурации). Однако они продолжают создавать отдельные блоки шардчейна для некоторого заранее определенного количества блоков.

2.7.9. Выполнение операций слияния. Когда валидаторы из двух объединенных исходных групп задач готовы стать валидаторами объединенного шардчейна (это может включать в себя передачу состояния из родственного шардчейна и операцию слияния), они фиксируют флаг «выполнить слияние» в заголовках блоков соответствующего шардчейна (это событие распространяется на следующие блоки мастерчейна) и прекращают создание новых блоков в отдельных шардчейнах (после появления флага «выполнить слияние» создание блоков в отдельных шардчейнах запрещено). Вместо этого создается объединенный блок шардчейна (путем объединения двух исходных групп задач) со ссылкой на оба его «предыдущих блока» в «заголовке». Это отражается в следующем блоке мастерчейна, который будет содержать хэш вновь созданного блока объединенного шардчейна. После этого объединенная группа задач продолжает создавать блоки в объединенном шардчейне.

2.8. Классификация блокчейн-проектов

Мы завершим наше краткое обсуждение блокчейна TON сравнением его с существующими и предлагаемыми блокчейн-проектами. Однако перед этим

необходимо представить обобщенную классификацию блокчейн-проектов. Сравнение конкретных блокчейн-проектов на основе этой классификации будет приведено в п. 2.9.

2.8.1. Классификация блокчейн-проектов. В качестве первого шага мы предлагаем некоторые критерии классификации блокчейнов (т. е. блокчейн-проектов). Любая такая классификация является в некоторой степени неполной и поверхностной, поскольку она игнорирует некоторые из наиболее специфических и уникальных особенностей рассматриваемых проектов. Однако мы считаем, что это необходимый первый шаг для предоставления хотя бы очень приблизительной схемы блокчейн-проектов.

Список критериев, которые мы будем рассматривать:

- Сингл-блокчейн vs. мульти-блокчейн архитектура (см. 2.8.2)
- Алгоритм консенсуса: Proof-of-Stake vs. Proof-of-Work (см. 2.8.3)
- Для систем Proof-of-Stake: конкретное поколение блокчейна, валидация и алгоритм консенсуса (двумя основными опциями являются DPOS vs. BFT; см. 2.8.4)
- Поддержка «произвольных» (Тьюринг-полных) смарт-контрактов (см. 2.8.6)

Мульти-блокчейн системы имеют дополнительные критерии классификации (см. 2.8.7):

- Типы и правила для блокчейнов: гомогенные, гетерогенные (см. 2.8.8), смешанные (см. 2.8.9). Конфедерации (см. 2.8.10).
- Отсутствие или наличие мастерчейна, внутреннего или внешнего (см. 2.8.11)
- Нативная поддержка шардинга (см. 2.8.12), статический или динамический шардинг (см. 2.8.13).
- Взаимодействие между блокчейнами: слабо-связанные и тесно-связанные системы (см. 2.8.14)

2.8.2. Сингл-блокчейн vs. мульти-блокчейн проекты. Первым критерием классификации является количество блокчейнов в системе. Старейшие и простейшие проекты состоят из одного блокчейна («синглчейн-проекты» для краткости); в более сложных проектах используется (или, скорее, планируется использование) несколько блокчейнов («мультичейн-проекты»).

Сингл-блокчейн проекты обычно проще и лучше тестируются; они выдержали испытание временем. Их главный недостаток в низкой производительности, или по крайней мере пропускной способности для транзакций, которая находится на уровне от десяти (Биткоин) до менее чем сотни[24] (Эфириум) транзакций в секунду для многоцелевых систем. Некоторые специализированные системы (такие как Bitshares) способны обрабатывать десятки тысяч специализированных транзакций в секунду, взамен требуя хранить состояние блокчейна в памяти, и ограничивая обработку до предопределенного специального набора транзакций, которые затем выполняются высокооптимизированным кодом на языках вроде C++ (никаких виртуальных машин).

Мультичейн-проекты могут поддерживать большее общее число состояний и больше транзакций в секунду, взамен делая проект и его имплементацию намного более сложными. В результате, есть всего несколько уже работающих мультичейн-проектов, но большинство предлагаемых проектов являются мультичейн-проектами. Мы верим, что будущее именно за мультичейн-проектами.

[24] Скорее, на данный момент это значение составляет 15. Тем не менее, планируются некоторые обновления, чтобы увеличить пропускную способность транзакций Ethereum в несколько раз

2.8.3. Создание и валидация блоков: Proof-of-Work vs. Proof-of-Stake. Другое важное отличие – это алгоритм и протокол, которые используются для создания и распространения новых блоков, проверки их валидности и выбора одного из нескольких ответвлений, если таковые появляются.

Две наиболее распространенные парадигмы - это Proof-of-Work (PoW) и Proof-of-Stake (PoS). Подход Proof-of-Work обычно позволяет любой ноде создавать («майнить») новые блоки (и получать некоторую награду, связанную с майнингом блока), если ей повезёт решить в ином случае бесполезную вычислительную задачу (обычно включающую вычисление большого количества хэшей) раньше, чем это успеют сделать конкуренты. В случае форков (например, если две ноды публикуют два валидных, но разных блока, следующих за предыдущим) побеждает самый длинный форк. Таким образом, гарантия неизменности блокчейна основана на объеме работы (вычислительных ресурсов), затрачиваемых на создание блокчейна: любому, кто хотел бы создать форк этого блокчейна, необходимо было бы повторно выполнить эту работу, чтобы создать альтернативные версии уже включенных блоков. Для этого нужно контролировать более 50% общей вычислительной мощности, затрачиваемой на создание новых блоков, в противном случае у альтернативного форка будут экспоненциально низкие шансы стать самым длинным.

Подход Proof-of-Stake основан на больших стейках (в криптовалюте), сделанных некоторыми специальными нодами (валидаторами), чтобы подтвердить, что они проверили (валидировали) некоторые блоки и нашли их правильными. Валидаторы подписывают блоки и получают за это небольшие награды; однако, если валидатор когда-либо был пойман на подписании неправильного блока, и было представлено доказательство этого, часть стейка или весь его стейк аннулируется. Таким образом, гарантия валидности и неизменности блокчейна обеспечивается общим объемом стейков, поставленных валидаторами на валидность блокчейна.

Подход Proof-of-Stake более естественен в том смысле, что он стимулирует валидаторов (которые заменяют майнеров PoW) выполнять полезные вычисления (необходимые для проверки или создания новых блоков, в частности, путем выполнения всех транзакций, перечисленных в блоке) вместо вычисления бесполезных хэшей. Таким образом, валидаторы будут приобретать оборудование, которое лучше приспособлено для обработки пользовательских транзакций, чтобы получать вознаграждения, связанные с этими транзакциями, что кажется весьма полезным вложением с точки зрения системы в целом.

Однако системы Proof-of-Stake несколько сложнее реализовать, поскольку необходимо предусмотреть множество редких, но возможных условий. Например, некоторые злонамеренные валидаторы могут вступить в сговор, чтобы нарушить работу системы и извлечь прибыль (например, путем изменения своих собственных балансов криптовалюты). Это приводит к необходимости решать некоторые нетривиальные задачи теории игр.

В целом, более естественен и более перспективен, особенно для мультичейн-проектов (потому что Proof-of-Work потребует непомерно больших вычислительных ресурсов, если блокчейнов много), но его необходимо более тщательно продумать и реализовать. Большинство работающих в настоящее время блокчейн-проектов, особенно самых старых (таких как Биткоин и как минимум оригинальный Эфириум), используют Proof-of-Work.

2.8.4. Варианты Proof-of-Stake. DPOS vs. BFT. Хотя алгоритмы Proof-of-Work очень похожи друг на друга и различаются в основном хэш-функциями, которые необходимо вычислять для добычи новых блоков, для алгоритмов Proof-of-Stake существует больше возможностей и они заслуживают отдельной подклассификации.

По сути, нужно ответить на следующие вопросы об алгоритме Proof-of-Stake:

- Кто может произвести («добыть») новый блок - любая полная нода или только член (относительно) небольшого подмножества валидаторов? (Большинство систем PoS требуют, чтобы новые блоки создавались и подписывались одним из нескольких назначенных валидаторов.)
- Гарантируют ли валидаторы валидность блоков своими подписями или все полные ноды должны проверять все блоки самостоятельно? (Масштабируемые системы PoS должны полагаться на подписи валидаторов вместо того, чтобы требовать, чтобы все ноды проверяли все блоки всех блокчейнов.)
- Есть ли назначенный и заранее известный создатель следующего блока блокчейна, вместо которого никто не сможет произвести этот блок?
- Является ли вновь созданный блок изначально подписанным только одним валидатором (его создателем), или он должен собирать большинство подписей валидаторов с самого начала?

Может показаться, что в зависимости от ответов на эти вопросы алгоритмы PoS можно разделить на 2^4 возможных класса, на практике различие сводится к двум основным подходам к PoS. Фактически, для большинства современных алгоритмов PoS, разработанных для использования в масштабируемых мультичейн-системах, ответы на первые два вопроса являются одинаковыми: только валидаторы могут создавать новые блоки, и именно они гарантируют валидность блока (при этом не требуется, чтобы все полные ноды проверяли валидность всех блоков по отдельности).

Что касается двух последних вопросов, ответы на них сильно зависят от свойств конкретной системы, оставляя по существу только два основных варианта:

- *Delegated Proof-of-Stake (DPOS)*: для каждого блока есть общеизвестный назначенный производитель; никто другой не может произвести этот блок; новый блок изначально подписывается только производящим его валидатором.
- *Byzantine Fault Tolerant (BFT)* алгоритмы PoS: существует известное подмножество валидаторов, любой из которых может предложить новый блок; выбор следующего блока среди нескольких предложенных кандидатов, который должен быть проверен и подписан большинством валидаторов перед передачей другим нодам, достигается версией протокола консенсуса Byzantine Fault Tolerant.

2.8.5. Сравнение DPOS и BFT PoS. Подход BFT обладает тем преимуществом, что вновь созданный блок с самого начала имеет подписи большинства валидаторов, подтверждающие его валидность. Еще одно преимущество заключается в том, что, если большинство валидаторов правильно выполняет протокол консенсуса BFT, форки системы невозможны. С другой стороны, алгоритмы BFT обычно довольно запутанны и при их использовании необходимо больше времени для того, чтобы подмножество валидаторов достигло консенсуса. Следовательно, блоки нельзя генерировать слишком часто. Вот почему мы ожидаем, что блокчейн TON (который является проектом BFT с точки зрения этой классификации) будет создавать блок не более одного раза в пять секунд. На практике этот интервал может быть уменьшен до 2-3 секунд (хотя мы этого не обещаем), но не больше, так как валидаторы разбросаны по всему миру.

Преимущество алгоритма DPOS состоит в том, что он довольно прост и понятен. Он может генерировать новые блоки довольно часто - скажем, раз в две секунды или даже раз в секунду[25] - благодаря тому, что он полагается на заранее известных производителей блоков.

Однако DPOS требует, чтобы все ноды – или как минимум все валидаторы - проверяли все полученные блоки, потому что валидатор, создающий и подписывающий новый блок, подтверждает не только относительную валидность этого блока, но также валидность предыдущего блока, на который он ссылается, а также все блоки назад в цепочке (возможно, до начала периода ответственности текущего подмножества

валидаторов). В текущем подмножестве валидаторов существует предопределенный порядок, поэтому каждому блоку назначается производитель (т. е. валидатор, который, как ожидается, сгенерирует этот блок); ротация этих назначенных производителей осуществляется по круговой схеме.

Таким образом, блок сначала подписывается только валидатором. Затем, когда генерируется следующий блок, и его производитель предпочитает ссылаться на этот блок, а не на одного из его предшественников (в противном случае его блок будет находиться в более короткой цепочке, которая может проиграть конкуренцию «самому длинному форку» в будущем), подпись следующего блока, по сути, также является дополнительной подписью предыдущего блока. Таким образом, новый блок постепенно собирает подписи большего количества валидаторов - скажем, двадцать подписей за время, необходимое для генерации следующих двадцати блоков. Полная нода должна будет либо дождаться этих двадцати подписей, либо подтвердить блок самостоятельно, начиная с достаточно валидного блока (скажем, двадцать блоков назад), что может быть не так просто.

Очевидным недостатком алгоритма DPOS по сравнению с алгоритмами BFT является то, что новый блок (и транзакции, зафиксированные в нем) достигает того же уровня доверия («рекурсивная надежность», как описано в п. 2.6.28) только после добычи следующих двадцати дополнительных блоков, которые сразу же обладают таким же уровнем доверия (скажем, двадцать подписей). Другой недостаток заключается в том, что DPOS использует подход «побеждает самый длинный форк» для переключения на другие форки; это делает форки весьма вероятными, если некоторые производители не смогут сгенерировать последующие блоки после того блока, который нас интересует (или мы не сможем наблюдать эти блоки из-за нарушения связности сети или хакерских атак).

Мы полагаем, что хотя подход BFT и является более сложным для реализации и требует более длительных интервалов времени между блоками, чем DPOS, BFT лучше приспособлен к «сильно-связанным» (см. 2.8.14) мультичейн-системам, потому что другие блокчейны могут начать работу почти сразу после фиксации транзакции (например, генерировать сообщение, предназначенное для них) в новом блоке, не дожидаясь двадцати подтверждений валидности (т. е. следующих двадцати блоков) или следующих шести блоков, чтобы убедиться в отсутствии форков и проверить новый блок самостоятельно (проверка блоков других блокчейнов может стать недопустимой в масштабируемой мультичейн-системе). Таким образом, обеспечивается масштабируемость системы с одновременным сохранением высокой надежности и доступности (см. 2.8.12).

С другой стороны, DPOS может быть хорошим выбором для «слабо-связанной» мультичейн-системы, где быстрое взаимодействие между блокчейнами не требуется - например, если каждый блокчейн («воркчейн») представляет собой отдельную распределенную систему обменов, а взаимодействие между блокчейнами ограничивается редкими передачами токенов из одного воркчейна в другой (или, скорее, обменом одного альткойна, находящегося в одном воркчейне, на другой со скоростью, приближающейся к 1: 1).

Именно это и реализуется в проекте BitShares, который довольно успешно использует подход DPOS.

Подводя итог, хотя DPOS может генерировать новые блоки и включать транзакции в них быстрее (с меньшими интервалами между блоками), эти транзакции достигают уровня доверия, необходимого для их использования в других блокчейнах и офф-чейн приложениях как «включенных» и «неизменяемых» гораздо медленнее, чем в системах BFT - скажем, за тридцать[26] секунд вместо пяти.

[25] Некоторые пользователи даже заявляют, что время генерации блока DPOS составляет полсекунды, что не кажется реалистичным, если валидаторы разбросаны по нескольким континентам

Более быстрое включение транзакции не означает более быстрое выполнение транзакции. Это может стать огромной проблемой, если требуется быстрое взаимодействие между блокчейнами. В этом случае следует отказаться от DPOS и вместо этого выбрать BFT PoS..

2.8.6. Поддержка Тьюринг-полного кода в транзакциях, то есть, произвольных смарт-контрактов. Проекты блокчейнов обычно собирают некоторые транзакции в своих блоках, которые изменяют состояние блокчейна необходимым образом (например, переводят некоторое количество криптовалюты с одной учетной записи на другую). Некоторые блокчейн-проекты могут разрешать только заранее определенные типы транзакций (например, транзакции стоимости из одной учетной записи в другую при условии наличия правильных подписей). Другие могут поддерживать определенную ограниченную форму написания сценариев в транзакциях. Наконец, некоторые блокчейны поддерживают выполнение произвольно сложного кода в транзакциях, позволяя системе (по крайней мере, в принципе) поддерживать произвольные приложения, если позволяет производительность системы. Обычно это связано с «Тьюринг-полными виртуальными машинами и языками сценариев» (это означает, что любая программа, которая может быть написана на любом другом языке программирования вычислений, может быть переписана для выполнения внутри блокчейна) и «смарт-контрактами» (которые представляют собой программы, находящиеся в блокчейне).
Поддержка произвольных смарт-контрактов делает систему по-настоящему гибкой. С другой стороны, такая гибкость обходится дорого: код этих смарт-контрактов должен выполняться на какой-то виртуальной машине, и это нужно делать каждый раз для каждой транзакции в блоке, когда кто-то хочет создать или валидировать блок. Это снижает производительность системы по сравнению со случаем предопределенного и неизменяемого набора типов простых транзакций, которые можно оптимизировать, реализовав их обработку на языке вроде C++ (вместо какой-либо виртуальной машины).
В конечном счете поддержка Тьюринг-полных смарт-контрактов представляется желательной в любом универсальном блокчейн-проекте; в противном случае разработчики блокчейн-проекта должны заранее решить, для каких приложений будет использоваться их блокчейн. Фактически, отсутствие поддержки смарт-контрактов в блокчейне Bitcoin было основной причиной создания нового блокчейн-проекта Ethereum.
В (гетерогенной; см. 2.8.8) мультичейн-системе можно получить «все самое лучшее», используя Тьюринг-полные смарт-контракты в некоторых блокчейнах (воркчейнах) и небольшой предопределенный набор высокоэффективных оптимизированные транзакций в других блокчейнах.

2.8.7. Классификация мультичейн-систем. До сих пор классификация действовала как для синглчейн, так и для мультичейн-систем.
Однако мультичейн-системы допускают еще несколько критериев классификации, отражающих взаимосвязь между различными блокчейнами в системе. Давайте обсудим эти критерии.

2.8.8. Типы блокчейнов: гомогенные и гетерогенные системы. В мультичейн-системе все блокчейны могут быть одного типа и иметь одинаковые правила (то есть использовать один и тот же формат транзакций, одну и ту же виртуальную машину

[26] Например, EOS, один из лучших проектов на основе DPOS, предложенных на сегодняшний день, обещает 45-секундное подтверждение и задержку взаимодействия между блокчейнами (см. [5], разделы «Подтверждение транзакции и задержка взаимодействия внутри сети»).

для выполнения кода смарт-контрактов, одну и ту же криптовалюту и так далее), и это сходство явно используется, но с разными данными в каждом блокчейне. В этом случае мы говорим, что система гомогенная. В ином случае разные блокчейны (которые в этом случае обычно называются воркчейнами) могут иметь разные «правила». Тогда мы говорим, что система гетерогенная.

2.8.9. Смешанные гетерогенные-гомогенные системы. Система может быть смешанной – в ней существует несколько наборов типов или правил для блокчейнов, но присутствует множество блокчейнов с одинаковыми правилами, и этот факт явно используется. Тогда это смешанная гетерогенная-гомогенная система. Насколько нам известно, блокчейн TON является единственным примером такой системы.

2.8.10. Гетерогенные системы с несколькими воркчейнами с одинаковыми правилами, или конфедерации. В некоторых случаях несколько блокчейнов (воркчейнов) с одинаковыми правилами могут присутствовать в гетерогенной системе, но взаимодействие между ними такое же, как и между блокчейнами с разными правилами (то есть их сходство не используется явно). Даже если они используют "одну и ту же" криптовалюту, на самом деле они используют разные "альткоины" (независимые воплощения криптовалюты). Иногда можно даже иметь определенные механизмы для конвертации этих альткоинов со скоростью, близкой к 1 : 1. Однако, на наш взгляд, это не делает систему гомогенной; она остается гетерогенной. Мы называем такую гетерогенную коллекцию воркчейнов с одинаковыми правилами конфедерацией.
Хотя создание гетерогенной системы, позволяющей создавать несколько воркчейнов с одинаковыми правилами (например, конфедерацию), может показаться дешевым способом построения масштабируемой системы, этот подход также имеет множество недостатков. По сути, если кто-то размещает большой проект в нескольких воркчейнах с одинаковыми правилами, он получает не большой проект, а множество небольших экземпляров этого проекта. Это похоже на приложение для общения (или игру), в котором может быть не более 50 участников в любой комнате чата (или игры), но которое при необходимости «масштабируется» путем создания новых комнат для общения большего количества пользователей. В результате многие пользователи могут участвовать в чатах или в игре, но можно ли сказать, что такая система действительно является масштабируемой?

2.8.11. Наличие мастерчейна, внешнего или внутреннего. Иногда в мультичейн-проекте есть выделенный «мастерчейн» («управляющий блокчейн»), который используется, например, для хранения общей конфигурации системы (набора всех активных блокчейнов, или, скорее, воркчейнов), текущего набора валидаторов (для системы Proof-of-Stake) и так далее. Иногда другие блокчейны «привязаны» к мастерчейну, например, путем включения в него хэшей своих последних блоков (это то, что делает блокчейн TON).
В некоторых случаях мастерчейн является внешним, что означает, что он не является частью проекта, а является каким-то другим существующим блокчейном, изначально совершенно не связанным с новым проектом и не зависящим от него. Например, можно попробовать использовать блокчейн Эфириум в качестве мастерчейна для внешнего проекта и для этой цели опубликовать специальные смарт-контракты в блокчейне Эфириум (например, для выбора и наказания валидаторов).

2.8.12. Поддержка шардинга. Некоторые проекты (или системы) на блокчейне имеют встроенную поддержку шардинга, что означает, что несколько (обязательно гомогенных; см. 2.8.8) блокчейнов рассматриваются как шарды единого (если смотреть сверху) виртуального блокчейна. Например, можно создать 256 шардчейнов

с одними и теми же правилами и сохранить состояние учетной записи точно в одном выбранном шарде в зависимости от первого байта его идентификатора account_id.
Шардинг — это естественный подход к масштабированию блокчейн-систем, потому что, если он правильно реализован, пользователям и смарт-контрактам в системе вообще не нужно знать о существовании шардинга. На самом деле, когда нагрузка становится слишком высокой, часто имеет смысл добавить шардинг к существующему синглчейн-проекту (например, Эфириум).
Альтернативный подход к масштабированию мог бы заключаться в использовании "конфедерации" гетерогенных воркчейнов, позволяющей каждому пользователю держать свой аккаунт в одном или нескольких воркчейнах по своему выбору и при необходимости переводить средства со своего аккаунта в одном воркчейне в другой воркчейн, по сути выполняя обмен альткоинов 1 : 1. Недостатки этого подхода уже обсуждались в п. 2.8.10.
Однако шардинг не так просто реализовать быстрым и надежным способом, поскольку он подразумевает пересылку множества сообщений между различными шардчейнами. Например, если учетные записи равномерно распределены между N шардами, а единственными транзакциями являются простые переводы средств с одной учетной записи на другую, то только небольшая часть (1/N) всех транзакций будет выполняться в одном блокчейне. Почти все транзакции (1–1/N) будут включать в себя два блокчейна, что потребует взаимодействия между блокчейнами. Если мы хотим, чтобы эти транзакции были быстрыми, нам нужна быстрая система для передачи сообщений между шардчейнами. Другими словами, блокчейн-проект должен быть «сильно-связанным» в смысле, описанном в п. 2.8.14

2.8.13. Динамический и статический шардинг. Шардинг может быть динамическим (если при необходимости автоматически создаются дополнительные шарды) или статическим (когда есть предопределенное количество шардов, которое можно изменить только с помощью хард-форка в лучшем случае). В большинстве проектов предлагается статичный шардинг; в блокчейне TON используется динамический шардинг (см. 2.7).

2.8.14. Взаимодействие между блокчейнами: слабо-связанные и тесно-связанные системы. Мультиблокчейн-проекты можно классифицировать в соответствии с поддерживаемым уровнем взаимодействия между составляющими их блокчейнами.
Наименьший уровень поддержки — это отсутствие какого-либо взаимодействия между различными блокчейнами вообще. Эти блокчейны являются не частями одной блокчейн-системы, а просто отдельными примерами одного и того же протокола блокчейна.
Следующий уровень — это отсутствие какой-либо конкретной поддержки обмена сообщениями между блокчейнами, что делает взаимодействие в принципе возможным, но неудобным. Мы называем такие системы "слабо-связанными"; в них нужно отправлять сообщения и переводить ценности между блокчейнами, как если бы они были блокчейнами, принадлежащими к полностью отдельным блокчейн-проектам (например, Биткоин и Эфириум; представьте, что две стороны хотят обменять биткоины, хранящиеся в блокчейне Биткоин, на эфиры, хранящиеся в блокчейне Эфириум). Другими словами, необходимо включить исходящее сообщение (или его генерирующую транзакцию) в блок исходного блокчейна. Затем нужно дождаться достаточного количества подтверждений (например, заданного количества последующих блоков), чтобы считать исходящую транзакцию «включённой» и «неизменной», чтобы иметь возможность выполнять внешние действия на основе ее существования. Только после этого может быть включена транзакция, передающая сообщение в целевой блокчейн (возможно, вместе с референсом и доказательством Меркла для существования исходящей транзакции).

Если не ждать достаточно долго, прежде чем передать сообщение, или если форк все равно произойдет по какой-либо другой причине, объединенное состояние двух блокчейнов окажется противоречивым: во второй блокчейн будет доставлено сообщение, которое никогда не было сгенерировано в (окончательно выбранном форке) первом блокчейне.
Иногда добавляется частичная поддержка обмена сообщениями путем стандартизации формата сообщений и расположения очередей входных и выходных сообщений в блоках всех воркчейнов (это особенно полезно в гетерогенных системах). Хотя это в определенной степени облегчает обмен сообщениями, концептуально он не слишком отличается от предыдущего случая, поэтому такие системы все еще являются «слабо-связанными».
Напротив, "тесно-связанные" системы включают специальные механизмы для обеспечения быстрого обмена сообщениями между всеми блокчейнами. Желаемое поведение - иметь возможность доставить сообщение в другой воркчейн сразу после того, как оно было сгенерировано в блоке исходного блокчейна. Также ожидается, что "тесно-связанные" системы будут в целом оставаться непротиворечивыми в случае форков. Хотя на первый взгляд эти два требования кажутся противоречащими друг другу, мы полагаем, что механизмы, использующиеся в блокчейне TON (включение хэшей блоков шардчейна в блоки мастерчейна; использование «вертикальных» блокчейнов для исправления невалидных блоков, см. 2.1.17; маршрутизация в гиперкубе, см. 2.4.19; мгновенная маршрутизация в гиперкубе, см. 2.4.20) делают ее «тесно-связанной» системой (возможно, пока единственной в своем роде).
Конечно, построить «слабо-связанную» систему намного проще; однако быстрый и эффективный шардинг (см. 2.8.12) требует, чтобы система была «тесно-связанной».

2.8.15. Упрощенная классификация. Поколения блокчейн-проектов. Предложенная нами классификация разбивает все блокчейн-проекты на большое количество классов. Однако на практике критерии классификации, которые мы используем, довольно сильно коррелируют между собой. Это позволяет нам предложить упрощенный «поколенческий» подход к классификации блокчейн-проектов как очень грубое приближение к реальности с некоторыми примерами. Проекты, которые еще не реализованы и не развернуты, выделены курсивом; жирным выделены наиболее важные характеристики поколения.

- Первое поколение: Сингл-чейн, PoW, нет поддержки смарт-контрактов. Примеры: Bitcoin (2009) и множество других не интересующих нас подражателей (Litecoin, Monero, ...).
- Второе поколение: Сингл-чейн, PoW, поддержка смарт-контрактов. Пример: Ethereum (2013; развернут в 2015), по крайней мере в его оригинальной форме.
- Третье поколение: Сингл-чейн, PoS, поддержка смарт-контрактов. Пример: будущий Ethereum (2018 или позднее).
- Альтернативное третье (3′) поколение: Мульти-чейн, PoS, нет поддержки смарт-контрактов, слабо-связан. Пример: Bitshares (2013–2014; использует DPOS).
- Четвертое поколение: Мульти-чейн, PoS, поддержка смарт-контрактов, слабо-связан. Примеры: EOS (2017; использует DPOS), PolkaDot (2016; использует BFT).
- Пятое поколение: Мульти-чейн, PoS и BFT, поддержка смарт-контрактов, тесно-связан, с шардингом. Примеры: TON (2017).

Хотя не все блокчейн-проекты попадают в одну из этих категорий, большинство из них относится к одной из этих категорий.

2.8.16. Сложности изменения «генома» блокчейн-проекта. Приведенная выше классификация определяет «геном» блокчейн-проекта. Этот геном достаточно

«жесткий»: его практически невозможно изменить, как только проект развернут и используется большим количеством людей. Потребуется серия хард-форков (что потребует одобрения большинства сообщества), и даже в этом случае изменения должны быть очень консервативными, чтобы сохранить обратную совместимость (например, изменение семантики виртуальной машины может сломать существующие смарт-контракты). Альтернативой было бы создание новых «сайдчейнов» с разными правилами и их привязка каким-либо образом к блокчейну (или блокчейнам) исходного проекта. Можно использовать блокчейн существующего синглчейн-проекта в качестве внешнего мастерчейна для принципиально нового и отдельного проекта[27].

Мы пришли к выводу, что геном проекта очень сложно изменить после его развертывания. Даже начать с PoW и планировать его замену на PoS в будущем – это довольно сложная задача[28]. Добавление шардов в проект, изначально созданный без их поддержки, кажется практически невозможным[29]. Фактически, добавление поддержки смарт-контрактов в проект (а именно, Биткоин), изначально разработанный без поддержки таких функций, было сочтено невозможным (или, по крайней мере, нежелательным для большей части сообщества Биткоин) и в конечном итоге привело к созданию нового блокчейн-проекта - Эфириума.

2.8.17. Геном блокчейна TON. Следовательно, если кто-то хочет создать масштабируемую блокчейн-систему, нужно тщательно выбирать ее геном с самого начала. Если система предназначена для поддержки некоторых дополнительных конкретных функций в будущем, неизвестных на момент ее развертывания, она должна изначально поддерживать «гетерогенные» воркчейны (с потенциально разными правилами). Чтобы система была действительно масштабируемой, она должна поддерживать шардинг с самого начала; шардинг имеет смысл только в том случае, если система «тесно-связана» (см. 2.8.14), а это, в свою очередь, подразумевает наличие мастерчейна, быстрой системы обмена сообщениями между блокчейнами, использование BFT PoS и так далее.

Если принять всё это во внимание, большинство дизайнерских решений, сделанных для блокчейн-проекта TON, кажутся естественными и почти единственно возможными.

[27] Например, в проекте Plasma планируется использование блокчейна Ethereum в качестве (внешнего) мастерчейна; в противном случае он мало взаимодействует с Ethereum, и он мог быть предложен и реализован командой, не связанной с проектом Ethereum.

[28] По состоянию на 2017 год Ethereum все еще пытается перейти от PoW к комбинированной системе PoW+PoS; мы надеемся, что когда-нибудь Ethereum станет настоящей системой PoS.

[29] Предложения по шардингу в Ethereum появились еще в 2015 году; неясно, как их можно реализовать и развернуть, не нарушая работы Ethereum и не создавая по существу независимого параллельного проекта.

Проект	Год	П.	Конс.	Ск.	Ч.	R.	Ш.	Int.
Bitcoin	2009	1	PoW	нет	1			
Ethereum	2013, 2015	2	PoW	да	1			
NXT	2014	2+	PoS	нет	1			
Tezos	2017, ?	2+	PoS	да	1			
Casper	2015, (2017)	3	PoW/PoS	да	1			
BitShares	2013, 2014	30	DPoS	нет	m	ht.	нет	L
EOS	2016, (2018)	4	DPoS	да	m	ht.	нет	L
PolkaDot	2016, (2019)	4	PoS BFT	да	m	ht.	нет	L
Cosmos	2017, ?	4	PoS BFT	да	m	ht.	нет	L
TON	2017, (2018)	5	PoS BFT	да	m	mix	dyn.	T

Таблица 1: Краткое описание некоторых известных блокчейн-проектов. В столбцах представлены следующие параметры: Проект - название проекта; Год - год анонсирования и год развертывания; П. - поколение (ср. 2.8.15); Конс. - алгоритм консенсуса (см. 2.8.3 и 2.8.4); Ск. - поддержка произвольного кода (смарт-контракты; см. 2.8.6); Ч. – сингл-/мульти-блокчейн-система (см. 2.8.2); R. - гетерогенные/гомогенные мультичейн системы (см. 2.8.8); Ш. - поддержка шардинга (см. 2.8.12); Int. - взаимодействие между блокчейнами, (L)oose или (T)ight – слабо-связанные или тесно-связанные системы (см. 2.8.14).

2.9. Сравнение с другими блокчейн-проектами

Мы завершаем наше краткое обсуждение блокчейна TON и его наиболее важных и уникальных функций в попытке найти для него место среди существующих и предлагаемых блокчейн-проектов. Мы используем критерии классификации, описанные в п. 2.8, для обсуждения различных проектов блокчейнов и построения своеобразной «карты блокчейн-проектов». Мы представляем эту карту в виде Таблицы 1, а затем кратко обсуждаем несколько проектов по отдельности, указывая на их особенности, которые могут не вписываться в общую схему.

2.9.1. Биткоин [12]; https://bitcoin.org/. Биткоин (2009) - первый и самый известный блокчейн-проект. Это типичный блокчейн-проект первого поколения: в нём один блокчейн, используется Proof-of-Work с алгоритмом выбора «побеждает самый длинный форк» и нет Тьюринг-полного скриптового языка (однако поддерживаются простые скрипты без циклов). Блокчейн Bitcoin не имеет понятия аккаунтов; вместо этого он использует модель UTXO (Unspent Transaction Output).

2.9.2. Ethereum [2]; https://ethereum.org/. Ethereum (2015) - первый блокчейн с поддержкой Тьюринг-полных смарт-контрактов. Таким образом, это типичный проект второго поколения и самый популярный среди них. Он использует Proof-of-Work на одном блокчейне, но имеет смарт-контракты и аккаунты.

2.9.3. NXT: https://nxtplatform.org/. NXT (2014) - это первый блокчейн и валюта на основе PoS. Это по-прежнему синглчейн-проект и он не поддерживает смарт-контракты.

2.9.4. Tezos; https://www.tezos.com/. Tezos (2018 или позже) - то предлагаемый синглчейн-проект на основе PoS. Мы упоминаем его здесь из-за его уникальной особенности: его функция интерпретации блока ev_block (см. 2.2.6) не фиксирована, но определяется модулем OCaml, который можно обновить, отправив новую версию в блокчейн (и собрав голоса для предлагаемого изменения). Таким образом, можно создавать собственные синглчейн-проекты, сначала развернув "ванильный" блокчейн Tezos, а затем постепенно изменяя функцию интерпретации блоков в желаемом направлении, без каких-либо хард-форков.
Эта идея хоть и является интригующей, но имеет очевидный недостаток, заключающийся в том, что она запрещает любые оптимизированные реализации на других языках, таких как C++, поэтому блокчейн на основе Tezos обречен на низкую производительность. Мы думаем, что аналогичный результат можно было бы получить, опубликовав формальную спецификацию предлагаемой функции интерпретации блока ev_trans без закрепления конкретной имплементации.

2.9.5. Casper.[30] Casper - новый алгоритм PoS для Ethereum; его постепенное развертывание в 2017 (или 2018), в случае успеха, превратит Ethereum в синглчейн PoS или смешанную систему PoW + PoS с поддержкой смарт-контрактов, сделав из Ethereum проект третьего поколения.

2.9.6. BitShares [8]; https://bitshares.org. BitShares (2014) — это платформа для распределенных бирж на основе блокчейна. Это гетерогенная мультиблокчейн-система DPoS без смарт-контрактов; она достигает своей высокой производительности, позволяя использовать только небольшой набор предопределенных специализированных типов транзакций, которые могут быть эффективно реализованы на C++ при условии, что состояние блокчейна умещается в памяти. Это также первый блокчейн-проект, в котором используется Delegated Proof-of-Stake (DPoS), демонстрирующий его жизнеспособность, по крайней мере, для некоторых специализированных целей.

2.9.7. EOS [5]; https://eos.io. EOS (2018 или позже) - это предлагаемая гетерогенная мультиблокчейн-система DPoS с поддержкой смарт-контрактов и некоторой минимальной поддержкой обмена сообщениями (система всё ещё слабо-связана в смысле, описанном в п. 2.8.14). Это разработка той же команды, которая ранее создала проекты BitShares и SteemIt, демонстрирующая сильные стороны алгоритма консенсуса DPoS. Масштабируемость будет достигаться за счет создания специализированных воркчейнов для проектов, которые в этом нуждаются (например, распределенная биржа может использовать воркчейн, поддерживающий специальный набор оптимизированных транзакций, аналогично тому, что сделал BitShares) и путем создания нескольких воркчейнов с одинаковыми правилами (конфедерации, см. п. 2.8.10). Недостатки и ограничения этого подхода к масштабируемости обсуждались в loc. cit. Более подробное описание DPoS, шардинга, взаимодействия между воркчейнами и их значение для масштабируемости системы блокчейнов приведено в 2.8.5, 2.8.12 и 2.8.14.
В то же время, даже если никто не сможет «создать Facebook внутри блокчейна» (см. 2.9.13), с помощью EOS или иным способом, мы думаем, что EOS может стать удобной платформой для некоторых узкоспециализированных слабо взаимодействующих распределенных приложений, подобных BitShares (децентрализованная биржа) и SteemIt (децентрализованная платформа для блогов).

[30] https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/

2.9.8. PolkaDot [17]; https://polkadot.io/. PolkaDot (2019 или позже) - один из самых продуманных и детально проработанных мультичейн-проектов Proof-of-Stake; его разработкой руководит один из соучредителей Ethereum. Этот проект является одним из самых близких к TON Blockchain на нашей карте. (Фактически, мы обязаны своей терминологией для «фишерменов» и «номинаторов» проекту PolkaDot.)
PolkaDot - это гетерогенный слабо-связанный мультичейн-проект Proof-of-Stake с консенсусом Byzantine Fault Tolerant (BFT) для генерации новых блоков и мастерчейном (который может быть внешним - например, блокчейн Ethereum). Он также использует гиперкубовую маршрутизацию, похожую на (медленную версию) используемую в TON, как описано в п. 2.4.19.
Его уникальная особенность - это возможность создавать не только публичные, но также приватные блокчейны. Эти приватные блокчейны также смогут взаимодействовать с другими публичными блокчейнами, PolkaDot или применять другие способы взаимодействия.
Таким образом, PolkaDot может стать платформой для крупномасштабных приватных блокчейнов, которые могут использоваться, например, банковскими консорциумами для быстрого перевода средств друг другу или для любых других целей, которые крупная корпорация может достичь с технологией приватного блокчейна.
Однако PolkaDot не поддерживает шардинг и не является тесно-связанной системой. Это несколько затрудняет её масштабируемость, которая примерно аналогична EOS. (Возможно, в случае PolkaDot ситуация немного лучше, потому что PolkaDot использует BFT PoS вместо DPoS.)

2.9.9. Universa; https://universa.io. Единственная причина, по которой мы упоминаем здесь этот необычный блокчейн-проект, заключается в том, что пока это единственный проект, в котором есть нечто похожее на нашу парадигму бесконечного шардинга (см. 2.1.2).
Другой его особенностью является то, что он обходит все сложности, связанные с Byzantine Fault Tolerance, обещая, что только доверенные и лицензированные партнеры проекта будут допущены в качестве валидаторов, поэтому они никогда не будут включать невалидные блоки. Это интересное решение; тем не менее, оно, по сути, делает блокчейн-проект намеренно централизованным, чего блокчейн-проекты обычно стараются избегать (зачем вообще нужен блокчейн для работы в доверенной централизованной среде?).

2.9.10. Plasma; https://plasma.io). Plasma (2019?) - отличный от других блокчейн-проект от другого соучредителя Ethereum. Предполагается, что он смягчит некоторые ограничения Ethereum без введения шардинга. По сути, это отдельный от Ethereum проект, представляющий иерархию (гетерогенных) воркчейнов, привязанных к блокчейну Ethereum (для использования в качестве внешнего мастерчейна) на верхнем уровне. Средства могут быть переведены из любого блокчейна вверх по иерархии (начиная с блокчейна Ethereum в качестве корня) вместе с описанием задания, которое необходимо выполнить. Затем в дочернем воркчейне выполняются необходимые вычисления (возможно, требующие пересылки частей исходного задания дальше вниз по дереву), их результаты передаются вверх, и собирается вознаграждение. Проблема достижения согласованности и проверки этих воркчейнов Проблема достижения непротиворечивости и валидации этих воркчейнов обходится с помощью (основанного на платежном канале) механизма, позволяющего пользователям в одностороннем порядке выводить свои средства из некорректно функционирующего воркчейна в его родительский воркчейн (хотя и медленно) и перераспределять свои средства и свои задания в другой воркчейн.
Таким образом, Plasma может стать платформой для распределенных вычислений, связанных с блокчейном Ethereum, чем-то вроде «математического сопроцессора».

Однако это не похоже на способ достигнуть истинной масштабируемости общего назначения.

2.9.11. Специализированные блокчейн-проекты. Существуют также некоторые специализированные блокчейн-проекты, такие как FileCoin (система, которая стимулирует пользователей предлагать свое дисковое пространство для хранения файлов других пользователей, которые готовы за это платить), Golem (платформа на основе блокчейна для аренды и предоставления вычислительной мощности для специализированных приложений, таких как 3D-рендеринг) или SONM (еще один аналогичный проект по предоставлению вычислительной мощности). Такие проекты не привносят ничего концептуально нового на уровне организации блокчейна; скорее, это конкретные приложения блокчейна, которые могут быть реализованы с помощью смарт-контрактов, работающих в блокчейне общего назначения, при условии, что он может обеспечить требуемую производительность.
Таким образом, проекты такого типа, вероятно, будут использовать в качестве своей основы один из существующих или планируемых блокчейн-проектов, например EOS, PolkaDot или TON. Если проекту нужна «настоящая» масштабируемость (на основе шардинга), лучше использовать TON; если проекту подходит функционирование в контексте «конфедерации» посредством определения семейства собственных воркчейнов, явно оптимизированных для его целей, он может выбрать EOS или PolkaDot.

2.9.12. Блокчейн TON. Блокчейн TON (The Open Network) (планируется в 2018 г.) - это проект, который мы описываем в данном документе. Он разработан как первый блокчейн-проект пятого поколения, то есть BFT PoS-мультичейн, смешанный гомогенный/гетерогенный, с поддержкой (шардируемых) настраиваемых воркчейнов, с нативной поддержкой шардинга и тесно-связанный блокчейн-проект (в частности, способный пересылать сообщения между шардами почти мгновенно, сохраняя при этом непротиворечивое состояние всех шардчейнов). Таким образом, это действительно масштабируемый блокчейн-проект общего назначения, способный вместить практически любые приложения, которые вообще могут быть реализованы в блокчейне. При добавлении других компонентов проекта TON его возможности расширяются еще больше.

2.9.13. Возможно ли «загрузить Facebook в блокчейн»? Иногда люди утверждают, что можно будет реализовать социальную сеть в масштабе Facebook как распределенное приложение, размещенное в блокчейне. Обычно в качестве возможного «хоста» для такого приложения указывается любимый блокчейн-проект.
Нельзя сказать, что это технически невозможно. Конечно, нужен тесно-связанный блокчейн-проект с настоящим шардингом (например, TON), чтобы такое большое приложение не работало слишком медленно (например, доставляло сообщения и обновления от пользователей, находящихся в одном шардчейне, их друзьям, находящимся в другом, с приемлемой задержкой). Однако мы думаем, что нет необходимости это делать и что это никогда не будет сделано, потому что цена будет непомерно высокой.
Давайте рассмотрим «загрузку Facebook в блокчейн» как мысленный эксперимент; примером может служить любой другой проект подобного масштаба. После загрузки Facebook в блокчейн все операции, выполняемые в настоящее время серверами Facebook, будут сериализированы как транзакции в определенных блокчейнах (например, в шардчейнах TON) и будут выполняться всеми валидаторами этих блокчейнов. Каждую операцию нужно будет выполнить, скажем, не менее двадцати раз, если мы ожидаем, что каждый блок соберет не менее двадцати подписей валидатора (мгновенно или со временем, как в системах DPOS). Точно так же все

данные, которые серверы Facebook хранят на своих дисках, будут храниться на дисках всех валидаторов для соответствующего шардчейна (то есть как минимум в двадцати копиях).
Поскольку валидаторы по сути являются теми же серверами (или, скорее, кластерами серверов, но это не влияет на достоверность этого аргумента), что и те, которые в настоящее время используются Facebook, мы видим, что общие затраты на оборудование, связанные с запуском Facebook в блокчейне, как минимум в двадцать раз больше, чем если бы это было реализовано обычным способом.
Фактически, расходы будут еще намного более высокими, потому что виртуальная машина блокчейна работает медленнее, чем "голый процессор", на котором работает оптимизированный скомпилированный код, а ее хранилище не оптимизировано для конкретных проблем Facebook. Можно частично смягчить эту проблему, создав специфический воркчейн с некими специальными транзакциями, адаптированными для Facebook; это подход BitShares и EOS к достижению высокой производительности, также доступный в блокчейне TON. Однако общий дизайн блокчейна сам по себе налагает некоторые дополнительные ограничения, такие как необходимость регистрировать все операции как транзакции в блоке, организовывать эти транзакции в дереве Меркла, вычислять и проверять их хэши Меркла, распространять этот блок дальше и так далее.
Таким образом, по самым скромным подсчетам, потребуется в 100 раз больше серверов с той же производительностью, что и те, которые сейчас используются Facebook, чтобы валидировать блокчейн-проект, на котором размещена социальная сеть такого масштаба. Кто-то должен будет заплатить за эти серверы, будь то компания, владеющая распределенным приложением (представьте, что вы видите 700 рекламных объявлений на каждой странице Facebook вместо 7), или ее пользователи. В любом случае это не выглядит экономически жизнеспособным.
Мы считаем неверным, что всё должно быть загружено в блокчейн. Например, нет необходимости хранить фотографии пользователей в блокчейне; регистрация хэшей этих фотографий в блокчейне и хранение фотографий в распределенном офф-чейн хранилище (таком как FileCoin или TON Storage) было бы лучшей идеей. Это причина, по которой TON - это не просто блокчейн-проект, а совокупность нескольких компонентов (TON P2P Network, TON Storage, TON Services), сосредоточенных вокруг блокчейна TON, как описано в главах 1 и 4.

3 Система TON Networking

Для любого блокчейн-проекта необходимы не только спецификации формата блока и правил проверки блокчейна, но и сетевого протокола, используемого для распространения новых блоков, отправки и сбора транзакций-кандидатов и т. д. Другими словами, в каждом проекте блокчейн должна быть создана специализированная одноранговая сеть. Эта сеть должна быть одноранговой, поскольку обычно предполагается, что блокчейн-проекты будут децентрализованными, поэтому нельзя полагаться на централизованную группу серверов и использовать традиционную архитектуру «клиент-сервер», как, например, в классических приложениях для онлайн-банкинга. Даже тонкие клиенты (например, приложения для смартфонов с кошельком для криптовалюты), которые должны подключаться к полным нодам по типу «клиент-сервер», на самом деле могут свободно подключаться к другой полной ноде, если соответствующая предыдущая одноранговая нода выходит из строя, при условии, что протокол, используемый для подключения для полных нод в достаточной степени стандартизирован.
В то время как сетевые потребности синглчейн-проектов, таких как Биткоин или Эфириум, могут быть удовлетворены довольно легко (по сути, необходимо построить «случайную» одноранговую оверлейную сеть и распространять все новые блоки и транзакции-кандидаты с помощью протокола сплетен), мультичейн-проекты с

несколькими блокчейнами, такие как TON Blockchain, гораздо более требовательны (например, нужно иметь возможность подписаться на обновления только некоторых шардчейнов, не обязательно всех из них). Поэтому сетевая часть блокчейна TON и проекта TON в целом заслуживает хотя бы краткого обсуждения.
С другой стороны, как только будут созданы более сложные сетевые протоколы, необходимые для поддержки блокчейна TON, их можно будет легко использовать для целей, не обязательно связанных с непосредственными потребностями блокчейна TON, что позволит обеспечить больше возможностей и гибкость для создания новых сервисов в экосистеме TON.

3.1. Сетевой уровень абстрактной датаграммы

Краеугольным камнем в построении сетевых протоколов TON является абстрактный сетевой уровень (датаграммы) (TON). Он позволяет всем узлам принимать определенные «сетевые идентификаторы», представленные 256-битными «абстрактными сетевыми адресами», и обмениваться данными (отправлять датаграммы друг другу в качестве первого шага), используя только эти 256-битные сетевые адреса для идентификации отправителя и получателя. В частности, не нужно беспокоиться об адресах IPv4 или IPv6, номерах портов UDP и т.п. - они скрыты в абстрактном сетевым уровне.

3.1.1. Абстрактные сетевые адреса. Абстрактный сетевой адрес, или просто адрес для краткости, является 256-битным целым числом равным 256-битному публичному ключу ECC. Этот публичный ключ может быть сгенерирован произвольно, таким образом создавая столько сетевых идентификаторов, сколько нравится узлу. Однако для получения (и расшифровки) сообщений, предназначенных для такого адреса, необходимо знать соответствующий закрытый ключ.
Фактически, адрес не является открытым ключом; вместо этого это 256-битный хэш (Hash = sha256) сериализированного TL-объекта, который может описывать несколько типов открытых ключей и адресов в зависимости от его конструктора (первые четыре байта). В простейшем случае этот сериализированный TL-объект состоит из 4-байтового магического числа и 256-битного открытого ключа криптографии с эллиптической кривой (ECC); в этом случае адрес будет равен хешу этой 36-байтовой структуры. Однако можно вместо этого использовать 2048-битные ключи RSA или любую другую схему криптографии с открытыми ключами. Когда узел узнает абстрактный адрес другого узла, он также должен получить свой «образ» (т.е. сериализированный TL-объект, хэш которого равен этому абстрактному адресу), иначе он не сможет шифровать и отправлять датаграммы на этот адрес.

3.1.2. Сети более низкого уровня. Реализация UDP. С точки зрения почти всех сетевых компонентов TON, единственное, что существует, - это сеть (сетевой уровень абстрактных датаграмм), способная отправлять датаграммы с одного абстрактного адреса на другой. В принципе, абстрактный сетевой уровень датаграм (ADNL) может быть реализован в различных существующих сетевых технологиях. Однако мы собираемся реализовать его по протоколу UDP в сетях IPv4 / IPv6 (таких как Интернет или интрасети), с необязательным запасным вариантом TCP, если UDP недоступен.
3.1.3. Простой вариант использования ADNL по UDP. Простейший случай отправки датаграмм с абстрактного адреса отправителя на любой другой абстрактный адрес (с известным изображением) может быть реализован следующим образом. Предположим, что отправитель каким-то образом знает IP-адрес и UDP-порт получателя, которому принадлежит абстрактный адрес назначения, и что и получатель, и отправитель используют абстрактные адреса, полученные из 256-битных открытых ключей ECC.

В этом случае отправитель просто добавляет датаграмм для отправки своей подписью ECC (выполненной с помощью своего закрытого ключа) и адресом своего источника. Результат шифруется открытым ключом получателя, встраивается в датаграмму UDP и отправляется на известный IP-адрес и порт получателя. Поскольку первые 256 бит датаграммы UDP содержат абстрактный адрес получателя, получатель может определить, какой закрытый ключ следует использовать для расшифровки оставшейся части датаграммы. Только после этого раскрывается личность отправителя.

3.1.4. Менее безопасный способ, с адресом отправителя в тексте. Иногда недостаточно безопасной схемы, когда адреса получателя и отправителя хранятся в незашифрованном виде в дейтаграмме UDP; закрытый ключ отправителя и открытый ключ получателя объединяются вместе с помощью ECDH (эллиптическая кривая Диффи-Хеллмана) для генерации 256-битного общего секрета, который используется впоследствии, наряду со случайным 256-битным одноразовым номером, также включенным в незашифрованную часть, получить ключи AES, используемые для шифрования. Целостность может быть обеспечена, например, путем объединения хеша исходных данных открытого текста в открытый текст перед шифрованием.
Преимущество этого подхода состоит в том, что если ожидается обмен более чем двумя дейтаграммами между двумя адресами, общий секрет может быть вычислен только один раз, а затем кэширован, тогда более медленные операции эллиптической кривой больше не будут требоваться для шифрования или дешифрования следующих датаграмм.

3.1.5. Каналы и идентификаторы каналов. В простейшем случае первые 256 бит датаграммы UDP, несущей встроенную дейтаграмму TON ADNL, будут равны адресу получателя. Однако в целом они составляют идентификатор канала. Существуют разные типы каналов. Некоторые из них являются двухточечными; они создаются двумя сторонами, которые желают обмениваться большим количеством данных в будущем и генерировать общий секрет путем обмена несколькими пакетами, зашифрованными в соответствии с 3.1.3 или 3.1.4 , путем запуска классической или эллиптической кривой Диффи-Хеллмана (если необходима дополнительная безопасность), или просто одна сторона генерирует случайный общий секрет и отправляет его другой стороне.
После этого идентификатор канала извлекается из общего секрета в сочетании с некоторыми дополнительными данными (такими как адреса отправителя и получателя), например, путем хеширования, и этот идентификатор используется в качестве первых 256 битов датаграмм UDP, переносящих данные, зашифрованные с помощью помощь этого общего секрета.

3.1.6. Канал как идентификатор туннеля. В общем, «канал» или «идентификатор канала» просто выбирает способ обработки входящей датаграммы UDP, известный получателю.
Если канал является абстрактным адресом получателя, обработка выполняется, как описано в п. 3.1.3 или 3.1.4; если канал является установленным двухточечным каналом, описанным в п. 3.1.5, обработка состоит в расшифровке датаграммы с помощью общего секрета, как описано в */цитата/* и т. д.
В частности, идентификатор канала может фактически выбрать «туннель», когда непосредственный получатель просто пересылает полученное сообщение кому-то другому - фактическому получателю или другому прокси. Некоторые этапы шифрования или дешифрования (напоминающие «луковую маршрутизацию» [6] или даже «чесночную маршрутизацию») могут выполняться по пути, и другой идентификатор канала может быть использован для перешифрованных

переадресованных пакетов (например, одноранговая маршрутизация). Равноправный канал может использоваться для пересылки пакета следующему получателю на пути. Таким образом, некоторая поддержка «туннелирования» и «прокси» - что-то вроде того, что обеспечивается проектами TOR или I2P - может быть добавлена на уровне сетевого уровня абстрактных дейтаграмм TON, не затрагивая функциональность всех высших сетевые протоколы уровня TON, которые были бы независимы от такого дополнения. Эта возможность используется службой TON Proxy (см. 4.1.10).

3.1.7. Нулевой канал и проблема начальной загрузки. Обычно TON ADNL будет иметь некоторую «таблицу соседних узлов», содержащую информацию о других известных узлах, таких как их абстрактные адреса и их прообразы (то есть открытые ключи), а также их IP-адреса и порты UDP. Затем он будет постепенно расширять эту таблицу, используя информацию, полученную из этих известных узлов, в качестве ответов на специальные запросы, а иногда и удалять устаревшие записи.
Однако когда узел TON ADNL только запускается, может случиться так, что он не знает ни одного другого узла и может узнать только IP-адрес и UDP-порт узла, но не его абстрактный адрес. Это происходит, например, если легкий клиент не может получить доступ ни к одному из ранее кэшированных узлов и к любым узлам, жестко закодированным в программное обеспечение, и должен попросить пользователя ввести IP-адрес или DNS-домен узла, который необходимо разрешить через DNS.
В этом случае узел будет отправлять пакеты на специальный «нулевой канал» рассматриваемого узла. Это не требует знания открытого ключа получателя (но сообщение все равно должно содержать личность и подпись отправителя), поэтому сообщение передается без шифрования. Обычно его следует использовать только для получения идентификатора (возможно, единовременного идентификатора, созданного специально для этой цели) получателя и затем для начала более безопасной связи.
Как только становится известен хотя бы один узел, можно легко заполнить «таблицу соседних узлов» и «таблицу маршрутизации» большим количеством записей, изучая их на основе ответов на специальные запросы, отправленные на уже известные узлы. Не все узлы требуются для обработки дейтаграмм, отправленных на нулевой канал, но те, которые используются для начальной загрузки легких клиентов, должны поддерживать эту функцию.

3.1.8. TCP-подобный стриминговый протокол поверх ADNL. ADNL, являющийся малым протоколом датаграмм на основе 256-битных абстрактных адресов может использоваться в качестве основы для более сложных сетевых протоколов. Можно создать, например, TCP-подобный потоковый протокол, используя ADNL в качестве абстрактной замены IP. Однако большинству компонентов проекта TON такой потоковый протокол не нужен.

3.1.9. RLDP или надежный протокол больших датаграмм поверх ADNL. Надежный протокол датаграмм произвольного размера, построенный на ADNL, называемый RLDP, используется вместо протокола, подобного TCP. Этот надежный протокол датаграмм может использоваться, например, для отправки запросов RPC удаленным хостам и получения от них ответов (см. 4.1.5).

3.2. TON DHT: Распределенная хеш-таблица, подобная Kademlia

Распределенная хеш-таблица TON (DHT) играет решающую роль в сетевой части проекта TON и используется для определения местоположения других узлов в сети.

[31] https://geti2p.net/en/docs/how/garlic-routing

Например, клиент, желающий зафиксировать транзакцию в шардчейне, может захотеть найти валидатора или коллатора этого шардчейна или хотя бы какой-нибудь узел, который мог бы передать транзакцию клиента коллатору. Это можно сделать, посмотрев специальный ключ в TON DHT. Еще одно важное применение TON DHT состоит в том, что его можно использовать для быстрого заполнения таблицы соседних элементов нового узла (см. 3.1.7) просто путем поиска случайного ключа или адреса нового узла. Если узел использует проксирование и туннелирование для своих входящих датаграмм, он публикует идентификатор туннеля и его точку входа (например, IP-адрес и порт UDP) в TON DHT; тогда все узлы, желающие отправить датаграммы на этот узел, сначала получат эту контактную информацию от DHT.
TON DHT является членом семейства распределенных хеш-таблиц, подобных Kademlia [10].

3.2.1. Ключи TON DHT. Ключи TON DHT - это просто 256-битные целые числа. В большинстве случаев они вычисляются как SHA256 TL-сериализированного объекта (см. 2.2.5), называемого прообразом ключа или описанием ключа. В некоторых случаях абстрактные адреса узлов сети TON (см. 3.1.1) также могут использоваться в качестве ключей TON DHT, потому что они также являются 256-битными, и они также являются хэшами TL-сериализированных объектов. Например, если узел не боится публиковать свой IP-адрес, его может найти любой, кто знает его абстрактный адрес, просто просмотрев этот адрес в виде ключа в DHT.

3.2.2. Значения DHT. Значения, присвоенные этим 256-битным ключам, по сути, представляют собой произвольные байтовые строки ограниченной длины. Интерпретация таких байтовых строк определяется прообразом соответствующего ключа; обычно это известно как узлу, который ищет ключ, так и узлу, который хранит ключ.

3.2.3. Узлы DHT. Полупостоянные сетевые идентификаторы. Сопоставление ключа и значения TON DHT хранится на узлах DHT, по сути, всех участников сети TON. С этой целью любой узел Сеть TON (возможно, за исключением некоторых очень легких узлов), помимо любого количества эфемерных и постоянных абстрактных адресов, описанных в 3.1.1, имеет как минимум один полупостоянный адрес, который идентифицирует ее как члена TON DHT. Этот полупостоянный или DHT-адрес не их следует менять слишком часто, иначе другие узлы не смогли бы найти
ключи, которые они ищут. Если узел не хочет раскрывать свою истинную личность, он генерирует отдельный абстрактный адрес, который будет использоваться только с целью участия в DHT. Однако этот абстрактный адрес должен быть общедоступным, поскольку он будет связан с IP-адресом и портом узла.

3.2.4. Расстояние Kademlia. Теперь у нас есть как 256-битные ключи, так и 256-битные (полупостоянные) адреса узлов. Мы вводим так называемое расстояние XOR или расстояние Kademlia dK на наборе 256-битных последовательностей, заданное формулой

$$d_K(x,y) := (x \oplus y) \quad \text{interpreted as an unsigned 256-bit integer} \qquad (25)$$

которое интерпретируется как 256-битное целое число без знака.
Здесь $x \oplus y$ обозначает побитовое исключающее ИЛИ (XOR) двух битовых последовательностей одинаковой длины.
Расстояние Kademlia представляет собой метрику набора 2^{256} всех 256-битных последовательностей. В частности, $d_K(x, y) = 0$ тогда и только тогда, когда $x = y$, $d_K(x, y) = d_K(y, x)$ и $d_K(x, z) \le dK(x, y) + dK(y, z)$. Другое важное свойство состоит в

том, что на любом заданном расстоянии от x есть только одна точка: $d_K(x, y) = d_K(x, y')$ подразумевает $y = y'$.

3.2.5. DHT, подобные Kademlia, и DHT TON. Мы говорим, что распределенная хеш-таблица (DHT) с 256-битными ключами и 256-битными адресами узлов является распределенной хеш-таблицей, подобной Kademlia, если ожидается, что она сохранит значение ключа K на s Kademlia - ближайших к K узлах (т. е. s узлов с наименьшим расстоянием Kademlia от их адресов до K).
Здесь s - небольшой параметр, скажем s = 7, необходимый для повышения надежности DHT (если мы будем хранить ключ только на одном узле, ближайшем к K, значение этого ключа будет потеряно, если этот единственный узел переходит в автономный режим).
Согласно этому определению, TON DHT - это таблица DHT, подобная Kademlia. Она будет реализована по протоколу ADNL, описанному в п. 3.1.

3.2.6. Таблица маршрутизации Kademlia. Любой узел, участвующий в DHT, подобном Kademlia, обычно поддерживает таблицу маршрутизации Kademlia. В случае TON DHT он состоит из n = 256 шардчейнов, пронумерованных от 0 до n - 1. i-й сегмент будет содержать информацию о некоторых известных узлах (фиксированное количество t «лучших» узлов и, возможно, некоторые дополнительные кандидаты), которые лежат на расстоянии Kademlia от 2^i до $2^{i+1} - 1$ от адреса узла a[32]. Эта информация включает их (полупостоянные) адреса, IP-адреса и порты UDP, а также некоторую информацию о доступности, такую как время и задержка последнего пинга.
Когда узел Kademlia узнает о любом другом узле Kademlia в результате некоторого запроса, он включает его в подходящую корзину своей таблицы маршрутизации, сначала в качестве кандидата. Затем, если некоторые из «лучших» узлов в этом сегменте выходят из строя (например, не отвечают на запросы проверки связи в течение длительного времени), они могут быть заменены некоторыми из кандидатов. Таким образом, таблица маршрутизации Kademlia остается всегда заполненной.
Новые узлы из таблицы маршрутизации Kademlia также включаются в таблицу соседних узлов ADNL, описанную в 3.1.7. Если часто используется «лучший» узел из группы таблицы маршрутизации Kademlia, для облегчения шифрования датаграмм может быть установлен канал, описанный в 3.1.5.
Особенностью TON DHT является то, что таблица пытается выбрать узлы с наименьшими задержками приема-передачи в качестве «лучших» узлов для шардчейнов таблицы маршрутизации Kademlia.

3.2.7. (Сетевые запросы Kademlia). Узел Kademlia обычно поддерживает следующие сетевые запросы:

- PING - проверяет доступность узла.
- STORE (ключ, значение) — просит узел сохранить значение в качестве значения для ключа key. Для TON DHT запросы STORE немного сложнее (см. 3.2.9).
- FIND_NODE(key, l) - просит узел вернуть l ближайших к Kademlia известных узлов (из его таблицы маршрутизации Kademlia) ключу.
- FIND_VALUE(key, l) — То же, что и выше, но если узел знает значение, соответствующее ключу key, он просто возвращает это значение.

Когда какой-либо узел хочет найти значение ключа K, он сначала создает набор S из s' узлов (для некоторого небольшого значения s', скажем, s'= 5), ближайших к K

[32] Если в сегменте достаточно много узлов, его можно дополнительно разделить, скажем, на восемь подсегментов в зависимости от четырех старших битов расстояния Kademlia. Это ускорит поиск DHT.

относительно расстояния Kademlia среди всех известных узлов (т. е. они взяты из таблицы маршрутизации Kademlia). Затем каждому из них отправляется запрос FIND_VALUE, и узлы, упомянутые в ответах, включаются в S. Затем узлам s из S, ближайшим к K, также отправляется запрос FIND_VALUE, если это не было сделано ранее, и процесс продолжается до тех пор, пока не будет найдено значение или пока множество S не перестанет расти. Это своего рода «направленный поиск» узла, ближайшего к K по отношению к расстоянию Kademlia.
Если необходимо установить значение некоторого ключа K, та же процедура выполняется для $s' \geq s$ с запросами FIND_NODE вместо FIND_VALUE, чтобы найти s ближайших узлов к K. После этого всем этим узлам отправляются запросы STORE.
В реализации подобной Kademlia таблицы DHT есть некоторые менее важные детали (например, любой узел должен искать ближайшие к себе узлы, скажем, один раз в час, и повторно публиковать все сохраненные ключи к ним с помощью запросов STORE). Мы пока будем игнорировать эти детали.

3.2.8. Загрузка узла Kademlia. Когда узел Kademlia подключается к сети, он сначала заполняет свою таблицу маршрутизации Kademlia, просматривая свой собственный адрес. Во время этого процесса он определяет s ближайших к себе узлов. Он может загружать из них все известные им пары (ключ, значение) для заполнения своей части DHT.

3.2.9. Сохранение значений в TON DHT. Хранение значений в TON DHT немного отличается от обычной Kademlia-подобной таблицы DHT. Если необходимо сохранить значение, должен быть предоставлен не только сам ключ K для запроса STORE, но и его прообраз, то есть TL-сериализированная строка (с одним из нескольких предопределенных TL-конструкторов в начале), содержащую «описание» ключа. Это описание ключа позже сохраняется в узле вместе с ключом и значением.
Описание ключа описывает «тип» сохраняемого объекта, его «владельца» и соответствующие «правила обновления» в случае будущих обновлений. Владелец обычно идентифицируется открытым ключом, включенным в описание ключа. Если он включен, обычно будут приниматься только обновления, подписанные соответствующим закрытым ключом. «Тип» хранимого объекта - это обычно просто байтовая строка. Однако в некоторых случаях он может быть более сложным - например, описание входного туннеля (см. 3.1.6) или набор адресов узлов.
«Правила обновления» тоже могут быть разными. В некоторых случаях они просто разрешают замену старого значения новым значением, при условии, что новое значение подписано владельцем (подпись должна быть сохранена как часть значения, которое позже проверяется любыми другими узлами после того, как они получат значение этого ключа). В других случаях старое значение так или иначе влияет на новое значение. Например, оно может содержать порядковый номер, а старое значение перезаписывается только в том случае, если новый порядковый номер больше (для предотвращения атак повторного воспроизведения).

3.2.10. Распространение «торрент-трекеров» и «сетевых специализированных групп» в TON DHT. Еще один интересный случай, когда значение содержит список узлов - возможно, с их IP-адресами и портами или просто с соответствующими абстрактными адресами, - а «правило обновления» заключается во включении запрашивающей стороны в этот список при условии, что она может подтвердить свою идентичность.
Этот механизм можно использовать для создания распределенного «торрент-трекера», где все узлы, заинтересованные в определенном «торренте» (т. е. в определенном узле), могут найти другие узлы, которые заинтересованы в этом же торренте или уже имеют соответствующую копию.

TON Storage (см. 4.1.7) использует эту технологию для поиска узлов, у которых есть копия требуемого файла (например, зафиксированный снимок состояния шардчейна или старого блока). Однако более важная задача - создание «оверлейных подсетей многоадресной рассылки» и «сетевых специализированных групп» (см. 3.3). Идея состоит в том, что только некоторые узлы заинтересованы в обновлениях определенного шардчейна. Если количество шардчейнов становится очень большим, поиск даже одного узла, заинтересованного в одном и том же шарде, может стать сложным. Этот «распределенный торрент-трекер» предоставляет удобный способ поиска некоторых из этих узлов. Другой вариант - запросить их у валидатора, однако этот подход исключает масштабирование, и валидаторы могут решить не отвечать на такие запросы, поступающие от произвольных неизвестных узлов.

3.2.11. Ключи отката. Большинство описанных до сих пор «типов ключей» имеют дополнительное 32-битное целое число, содержащееся в соответствующем TL-описании, обычно равное нулю.
Однако если ключ был получен путем хеширования, это описание не может быть получено или обновлено в TON DHT, значение в этом удерживаемом ключе увеличивается, и делается новая попытка получения. Таким образом, невозможно «захватить» и «подвергнуть цензуре» ключ (то есть выполнить атаку с удержанием ключа) путем создания множества абстрактных адресов, находящихся рядом с атакуемым ключом, и управления соответствующими узлами DHT.

3.2.12. Услуги по определению местоположения. Для некоторых сервисов, расположенных в сети TON и доступных по (протоколам более высокого уровня, основанным на) TON ADNL, описанным в п. 3.1, может понадобиться опубликовать их абстрактные адреса, чтобы их клиенты знали, где их найти.
Однако публикация абстрактного адреса сервиса в блокчейне TON может быть не самым лучшим подходом, поскольку может потребоваться довольно часто менять абстрактный адрес, и поэтому может иметь смысл предоставить несколько адресов в целях надежности или балансировки нагрузки.
Альтернативой является публикация открытого ключа в блокчейне TON и использование специального ключа DHT, указывающего этот открытый ключ в качестве его «владельца» в строке описания TL (см. 2.2.5), что позволяет публиковать обновленный список абстрактных адресов сервиса. Это один из подходов, используемых в TON Services.

3.2.13. Поиск владельцев учетных записей блокчейна TON. В большинстве случаев владельцы учетных записей блокчейна TON не хотят, чтобы их ассоциировали с абстрактными сетевыми адресами, особенно с IP-адресами, поскольку это может нарушить их конфиденциальность. Однако в некоторых случаях владелец учетной записи блокчейна TON может захотеть опубликовать один или несколько абстрактных адресов, по которым с ним можно связаться.
Типичный случай - это узел в «сети моментальных платежей» TON Payments (см. 5.2), платформе для мгновенных переводов криптовалюты. Общедоступный узел TON Payments может захотеть не только установить платежные каналы с другими одноранговыми узлами, но также опубликовать абстрактный сетевой адрес, который можно использовать для последующей связи с ним для передачи платежей по уже установленным каналам.
Одним из вариантов может быть включение абстрактного сетевого адреса в смарт-контракт, создающий платежный канал. Более гибкий вариант — включить открытый ключ в смарт-контракт, а затем использовать DHT, как описано в 3.2.12.

Наиболее естественным способом было бы использовать тот же закрытый ключ, который управляет учетной записью в блокчейне TON, для подписания и публикации обновлений в TON DHT об абстрактных адресах, связанных с этой учетной записью. Этот процесс практически совпадает с п. 3.2.12; однако для используемого ключа DHT потребуется специальное описание ключа, содержащее только идентификатор account_id, равный SHA256 «описания учетной записи», который содержит открытый ключ учетной записи. Подпись, включенная в значение этого ключа DHT, также будет содержать описание учетной записи.
Таким образом, становится доступным механизм определения абстрактных сетевых адресов некоторых владельцев учетных записей блокчейна TON.

3.2.14. Поиск абстрактных адресов. Обратите внимание, что хотя таблица TON DHT и реализуется поверх TON ADNL, TON DHT сама используется TON ADNL для нескольких целей.
Самая важная задача - найти узел или его контактные данные, начиная с 256-битного абстрактного адреса. Это необходимо, потому что TON ADNL должен иметь возможность отправлять датаграммы на произвольные 256-битные абстрактные адреса, даже если не предоставляется никакой дополнительной информации.
Для этого 256-битный абстрактный адрес просто просматривается как ключ в DHT. При этом либо находится узел с этим адресом (т. е. с использованием этого адреса в качестве общедоступного полупостоянного адреса DHT), и в этом случае можно узнать его IP-адрес и порт; либо может быть получено описание входного туннеля как значение рассматриваемого ключа, подписанное правильным закрытым ключом, и в этом случае это описание туннеля будет использоваться для отправки датаграмм ADNL предполагаемому получателю.
Обратите внимание: для того, чтобы сделать абстрактный адрес «общедоступным» (доступным с любых узлов в сети), его владелец должен либо использовать его как полупостоянный адрес DHT, либо опубликовать (в ключе DHT, равном рассматриваемому абстрактному адресу) описание входного туннеля с другим его общедоступным абстрактным адресом (например, полупостоянным адресом) в качестве точки входа в туннель. Другой вариант - просто опубликовать IP-адрес и порт UDP.

3.3. Оверлейные сети и многоадресные сообщения

В мультичейн-системе, такой как блокчейн TON, даже полные ноды обычно заинтересованы в получении обновлений (т. е. новых блоков) только для некоторых шардчейнов. С этой целью внутри сети TON для каждого шардчейна должна быть построена специальная оверлейная (под) сеть поверх протокола ADNL, описанного в п. 3.1.
Следовательно, возникает необходимость в создании произвольных оверлейных подсетей, открытых для любых узлов, желающих в них участвовать. В этих оверлейных сетях будут работать специальные протоколы сплетен, основанные на ADNL. В частности, эти протоколы сплетен могут использоваться для распространения (передачи) произвольных данных внутри такой подсети.

3.3.1. Оверлейные сети. Оверлейная (под) сеть - это просто (виртуальная) сеть, реализованная внутри более крупной сети. Обычно только некоторые узлы более крупной сети участвуют в оверлейной подсети, и только некоторые «связи» между этими узлами, физическими или виртуальными, являются частью оверлейной подсети.
Таким образом, если представить сеть в виде графа (возможно, полного графа в случае сети датаграмм, такой как ADNL, где любой узел может легко связываться с любым другим), оверлейная подсеть является подграфом этого графа.

В большинстве случаев оверлейная сеть реализуется с использованием какого-либо протокола, построенного на сетевом протоколе более крупной сети. Она может использовать те же адреса, что и более крупная сеть, либо использовать собственные адреса.

3.3.2. Оверлейные сети в TON. Оверлейные сети в TON построены на протоколе ADNL, описанном в п. 3.1; в них также используются 256-битные абстрактные адреса ADNL в качестве адресов. Каждый узел обычно выбирает один из своих абстрактных адресов, который удваивается в оверлейной сети.
В отличие от ADNL, оверлейные сети TON обычно не поддерживают отправку датаграмм на другие произвольные узлы. Вместо этого между некоторыми узлами (называемыми «соседями» по отношению к рассматриваемой оверлейной сети) устанавливается «полупостоянная связь», и сообщения обычно пересылаются по этой связи (т. е. от узла к одному из его соседей). Таким образом, оверлейная сеть TON представляет собой (обычно не полный) подграф внутри (полного) графа сети ADNL. Связи с соседями в оверлейных сетях TON могут быть реализованы с использованием выделенных одноранговых каналов ADNL (см. 3.1.5).
Каждый узел оверлейной сети поддерживает список соседей (по отношению к оверлейной сети), содержащий их абстрактные адреса (которые они используют для собственной идентификации в оверлейной сети) и некоторые данные связи (например, канал ADNL, используемый для связи).

3.3.3. Частные и общедоступные оверлейные сети. Некоторые оверлейные сети являются общедоступными, что означает, что любой узел может присоединиться к ним по своему желанию. Другие являются частными – в них могут быть допущены только определенные узлы (например, узлы, которые могут подтвердить, что они являются валидаторами). Некоторые частные оверлейные сети могут быть даже неизвестны «широкой публике». Информация о таких оверлейных сетях доступна только определенным доверенным узлам; например, она может быть зашифрована открытым ключом, и расшифровать эту информацию смогут только узлы, имеющие копию соответствующего закрытого ключа.

3.3.4. Централизованно управляемые оверлейные сети. Некоторые оверлейные сети управляются централизованно одним или несколькими узлами или владельцем широко известного открытого ключа. Другие являются децентрализованными – это означает, что за них не отвечают какие-либо конкретные узлы.

3.3.5. Присоединение к оверлейной сети. Когда узел хочет присоединиться к оверлейной сети, он сначала должен узнать свой 256-битный сетевой идентификатор, обычно равный SHA256 описания оверлейной сети - TL-сериализированный объект (см. 2.2.5), который может содержать, например, центр оверлейной сети (то есть его открытый ключ и, возможно, его абстрактный адрес[33]), строка с именем оверлейной сети, идентификатор сегмента блокчейна TON, если это оверлейная сеть, связанная с этим шардом, и т. д.
Иногда возможно восстановить описание оверлейной сети, начиная с идентификатора сети, просто просмотрев его в TON DHT. В других случаях (например, для частных оверлейных сетей) необходимо получить описание сети вместе с идентификатором сети.

3.3.6. Поиск одного члена оверлейной сети. После того, как узел узнает идентификатор сети и описание оверлейной сети, к которой он хочет присоединиться, он должен найти хотя бы один узел, принадлежащий этой сети.

Это также необходимо для узлов, которые не хотят присоединяться к оверлейной сети, а хотят просто связаться с ее участниками. Например, может существовать оверлейная сеть, предназначенная для сбора и распространения транзакций-кандидатов для определенного шардчейна, и клиент может захотеть подключиться к любому узлу этой сети, чтобы предложить транзакцию.
Метод, используемый для обнаружения элементов оверлейной сети, определяется в описании этой сети. Иногда (особенно для частных сетей) нужно уже знать принадлежащий этой сети узел, чтобы иметь возможность присоединиться. В других случаях в описании сети содержатся абстрактные адреса некоторых узлов. Более гибкий подход состоит в том, чтобы указать в описании сети только центральный узел, ответственный за сеть, и тогда абстрактные адреса будут доступны через значения определенных ключей DHT, подписанных этим центральным узлом.
Наконец, действительно децентрализованные общедоступные оверлейные сети могут использовать механизм «распределенного трекера», описанный в 3.2.10, также реализованный с помощью TON DHT.

3.3.7. Поиск дополнительных участников оверлейной сети. Создание ссылок. Как только один узел оверлейной сети будет найден, этому узлу может быть отправлен специальный запрос, запрашивающий список других членов, например, соседей запрашиваемого узла, или список случайных узлов.
Это позволяет присоединяющемуся узлу заполнить «список своих соседей» по отношению к оверлейной сети, выбрав некоторые недавно изученные сетевые узлы и установив с ними связи (т. е. выделенные двухточечные каналы ADNL, как описано в общих чертах в п. 3.3.2). После этого всем соседям отправляются специальные сообщения, указывающие, что новый участник готов работать в оверлейной сети. Соседи включают свои ссылки на нового участника в свои списки соседей.

3.3.8. Ведение списка соседей. Узел оверлейной сети должен время от времени обновлять свой список соседей. Некоторые соседи или хотя бы ссылки (каналы) на них могут перестать отвечать; в этом случае эти ссылки должны быть помечены как «приостановленные», должны быть предприняты некоторые попытки повторного подключения к таким соседям, и, если эти попытки не удались, ссылки должны быть уничтожены.
С другой стороны, каждый узел иногда запрашивает у случайно выбранного соседа свой список соседей (или некоторый случайный выбор из них) и использует его для частичного обновления своего собственного списка соседей, добавляя к нему несколько недавно обнаруженных узлов и удаляя некоторые старые узлы (случайным образом или в зависимости от времени отклика и статистики потери датаграмм).

3.3.9. Оверлейная сеть как случайный подграф. Таким образом, оверлейная сеть становится случайным подграфом внутри сети ADNL. Если степень каждой вершины не менее 3 (т. е. если каждый узел связан, по крайней мере, с тремя соседями), этот случайный граф связан с вероятностью, почти равной единице. Точнее, вероятность того, что случайный граф с n вершинами будет отключен, экспоненциально мала, и этой вероятностью можно полностью пренебречь, если, скажем, $n \geq 20$. (Конечно, это не применимо в случае разделения глобальной сети, когда узлы на разных сторонах раздела не имеют возможности узнать друг о друге.) С другой стороны, если n меньше 20, достаточно потребовать, чтобы каждая вершина имела, скажем, как минимум десять соседей.

3.3.10. Оптимизация оверлейных сетей TON для уменьшения задержки.

[33] Как вариант, абстрактный адрес может храниться в DHT, как описано в 3.2.12.

Оверлейные сети TON оптимизируют «случайный» сетевой граф, сгенерированный предыдущим методом, следующим образом. Каждый узел пытается сохранить как минимум трех соседей с минимальным периодом кругового обращения, причем этот список «быстрых соседей» очень редко меняется. В то же время у каждого узла есть как минимум три других «медленных соседа», которые выбираются совершенно случайным образом, поэтому граф оверлейной сети всегда будет содержать случайный подграф. Это необходимо для поддержания связи и предотвращения разделения оверлейной сети на несколько несвязанных местных подсетей. Также выбираются и сохраняются как минимум три «промежуточных соседа», которые имеют промежуточный период кругового обращения, ограниченный определенной константой (фактически, функцией периода кругового обращения для быстрых и медленных соседей).
Таким образом, в графе оверлейной сети сохраняется достаточный уровень произвольности подключения, однако он оптимизирован для более низкой задержки и более высокой пропускной способности.

3.3.11. Протоколы сплетен в оверлейной сети. Оверлейные сети часто используются для запуска одного из так называемых протоколов сплетен, которые достигают какой-то глобальной цели, позволяя каждому узлу взаимодействовать только со своими соседями. Например, существуют протоколы сплетен для построения приблизительного списка всех членов (не слишком большой) оверлейной сети или для вычисления оценки числа членов (произвольно большой) оверлейной сети, используя только ограниченное количество памяти на каждом узле (более подробная информация приведена в [15, 4.4.3] или [1]).

3.3.12. Оверлейная сеть как домен широковещательной рассылки. Самый важный протокол сплетен, работающий в оверлейной сети, - это протокол широковещательной передачи, предназначенный для распространения широковещательных сообщений, генерируемых любым узлом сети или одним из назначенных узлов-отправителей, всем другим узлам.
На самом деле существует несколько протоколов широковещательной передачи, оптимизированных для разных случаев применения. Самый простой из них принимает новые широковещательные сообщения и ретранслирует их всем соседям, которые сами еще не отправили копию этого сообщения.

3.3.13. Более сложные протоколы широковещательной передачи. Для некоторых приложений могут потребоваться более сложные протоколы широковещательной передачи. Например, при широковещательной рассылке сообщений большого размера имеет смысл отправлять соседям не само вновь полученное сообщение, а его хэш (или набор хэшей новых сообщений). Соседний узел может запросить само сообщение после изучения ранее невидимого хэша передаваемого сообщения, скажем, с использованием надежного протокола больших датаграмм (RLDP), описанного в п. 3.1.9. Таким образом, новое сообщение будет загружено только от одного соседнего узла.

3.3.14. Проверка возможности подключения оверлейной сети. Связность оверлейной сети можно проверить с использованием известного узла (например, «владельца» или «создателя» оверлейной сети), который должен находиться в этой оверлейной сети. Затем рассматриваемый узел просто время от времени передает короткие сообщения, содержащие текущее время, порядковый номер и подпись. Любой другой узел может быть уверен, что он все еще подключен к оверлейной сети, если он недавно получил такое широковещательное сообщение. Этот протокол может быть расширен на нескольких хорошо известных узлов; например, все они будут

отправлять такие широковещательные рассылки, а все остальные узлы будут ожидать широковещательные рассылки от более чем половины хорошо известных узлов.
В случае оверлейной сети, используемой для распространения новых блоков (или только новых заголовков блоков) определенного шардчейна, хороший способ для узла проверить возможность подключения - это отслеживать самый последний полученный блок. Поскольку блок обычно генерируется каждые пять секунд, если новый блок не принимается в течение, скажем, тридцати секунд, то узел, вероятно, был отключен от оверлейной сети.

3.3.15. Протокол потоковой трансляции. Существует протокол потоковой трансляции для оверлейных сетей TON, который используется, например, для распространения блоков-кандидатов среди валидаторов некоторого шардчейна («группа задач шардчейна»), которые, конечно же, создают для этой цели частную оверлейную сеть. Тот же протокол можно использовать для распространения новых блоков шардчейна на все полные ноды этого шардчейна.
Этот протокол уже был описан в 2.6.10: новое (большое) широковещательное сообщение разбивается, скажем, на N блоков по одному килобайту; последовательность этих фрагментов увеличивается до $M \geq N$ фрагментов с помощью стирающего кода, такого как код Рида-Соломона или исходный код (например, код RaptorQ [9] [14]), и эти M фрагментов передаются всем соседям в порядке возрастания номеров блоков. Участвующие узлы собирают эти фрагменты до тех пор, пока они не смогут восстановить исходное большое сообщение (для этого необходимо успешно получить не менее N фрагментов), а затем проинструктировать своих соседей прекратить отправку новых фрагментов потока, потому что теперь эти узлы могут генерировать последующие блоки самостоятельно, имея копию исходного сообщения. Такие узлы продолжают генерировать последующие фрагменты потока и отправлять их своим соседям, если соседи, в свою очередь, не укажут, что в этом больше нет необходимости.
Таким образом, узлу не нужно полностью загружать большое сообщение перед его дальнейшим распространением. Это позволяет минимизировать задержку потоковой трансляции, особенно в сочетании с оптимизацией, описанной в 3.3.10.

3.3.16. Построение новых оверлейных сетей на основе существующих сетей. Иногда строить оверлейную сеть с нуля нет смысла. Вместо этого известна одна или несколько ранее существовавших оверлейных сетей, и ожидается, что членство в новой оверлейной сети будет значительно перекрываться с объединенным членством этих оверлейных сетей.
Важный пример – разделение шардчейна TON на два шардчейна, либо объединение двух родственных шардчейнов в один (см. 2.7). В первом случае оверлейные сети, используемые для распространения новых блоков на полные ноды, должны быть построены для каждого нового шардчейна. Однако можно ожидать, что каждая из этих новых оверлейных сетей будет содержаться в сети распространения блоков исходного шардчейна (и включать примерно половину ее членов). Во втором случае оверлейная сеть для распространения новых блоков объединенного шардчейна будет состоять примерно из объединения членов двух оверлейных сетей, связанных с двумя объединяемыми родственными шардчейнами.
В таких случаях описание новой оверлейной сети может содержать явную или неявную ссылку на список связанных существующих оверлейных сетей. Узел, желающий присоединиться к новой оверлейной сети, может проверить, является ли он уже членом одной из этих существующих сетей, и запросить своих соседей в этих сетях, заинтересованы ли они также в новой сети. В случае положительного ответа для таких соседей могут быть установлены новые двухточечные каналы, и они могут быть включены в список соседей для новой оверлейной сети.

Этот механизм не заменяет полностью общий механизм, описанный в 3.3.6 и 3.3.7; скорее, оба механизма выполняются параллельно и используются для заполнения списка соседей. Это необходимо для предотвращения случайного разделения новой оверлейной сети на несколько несвязанных подсетей.

3.3.17. Оверлейные сети внутри оверлейных сетей. Другой интересный случай возникает при реализации сервиса TON Payments («сеть моментальных платежей» для мгновенных переводов стоимости вне сети; см. 5.2). В этом случае сначала строится оверлейная сеть, содержащая все транзитные узлы «сети моментальных платежей». Однако некоторые из этих узлов имеют платежные каналы в блокчейне; они всегда должны быть соседями в этой оверлейной сети, в дополнение к любым «случайным» соседям, выбранным общими алгоритмами оверлейной сети, описанными в 3.3.6, 3.3.7 и 3.3.8, эти «постоянные соединения» с соседями с установленными платежными каналами используются для запуска определенных сетевых протоколов Lightning. Таким образом, обеспечивается эффективное создание оверлейной подсети (не обязательно подключенной, если что-то пойдет не так) внутри охватывающей (почти всегда подключенной) оверлейной сети.

4 Сервисы и приложения TON

Мы подробно обсудили технологии блокчейна TON и TON Networking. Далее будут описаны некоторые способы, при помощи которых эти технологии могут быть объединены для создания широкого спектра услуг и приложений, а также некоторые из сервисов, которые будут предоставляться самим проектом TON с самого начала, либо позже.

4.1. Стратегии внедрения сервисов TON

Мы начнем с обсуждения того, как различные приложения и сервисы, связанные с блокчейном и сетью, могут быть реализованы внутри экосистемы TON. Прежде всего, следует привести простую классификацию:

4.1.1. Приложения и сервисы. Мы будем использовать слова «приложение» и «сервисы» в качестве синонимов. Однако есть небольшое и несколько расплывчатое различие: приложение обычно предоставляет некоторые услуги напрямую пользователям-людям, в то время как сервис обычно используется другими приложениями и сервисами. Например, TON Storage - это сервис, потому что она предназначена для хранения информации от имени других приложений и сервисов, даже если пользователь-человек также может использовать ее напрямую. Гипотетический «Facebook в блокчейне» (см. 2.9.13), если он будет доступен через сеть TON (т. е. реализован как «сервис TON»), скорее будет приложением, даже если некоторые «боты» могут получить к нему доступ автоматически без вмешательства человека.

4.1.2. Расположение приложения: ончейн, оффчейн или смешанный тип. Сервис или приложение, разработанное для экосистемы TON, должно где-то хранить свои данные и обрабатывать их. В результате получается следующая классификация приложений (и сервисов):

- Сетевые приложения (см. 4.1.4): все данные и обработка находятся в блокчейне TON.
- Оффчейн-приложения (см. 4.1.5): все данные и обработка находятся за пределами блокчейна TON, на серверах, доступных через сеть TON.

- Смешанные приложения (см. 4.1.6): некоторые (но не все) данные и обработка находятся в блокчейне TON; остальные находятся на оффчейн-серверах, доступных через сеть TON.

4.1.3. Централизация: централизованные и децентрализованные или распределенные приложения. Другой критерий классификации заключается в том, полагается ли приложение (или сервис) на централизованный кластер серверов или оно действительно является «распределенным» (см. 4.1.8). Все сетевые приложения автоматически являются децентрализованными и распределенными. Оффчейн и смешанные приложения могут иметь разную степень централизации.
Теперь рассмотрим вышеупомянутые возможности более подробно.

4.1.4. Чистые «ончейн» приложения: распределенные приложения или «децентрализованные приложения», находящиеся в блокчейне. Одним из возможных подходов, упомянутых в 4.1.2, является развертывание «распределенного приложения» (сокращенно «dapp») полностью в блокчейне TON в виде одного смарт-контракта или набора смарт-контрактов. Все данные будут храниться как часть постоянного состояния этих смарт-контрактов, а все взаимодействие с проектом будет осуществляться с помощью сообщений (блокчейна TON), отправленных на эти смарт-контракты или полученных от них.
Мы уже обсуждали в 2.9.13, что этот подход имеет свои недостатки и ограничения. У него также есть свои преимущества: такому распределенному приложению не нужны серверы для работы или хранения данных (оно работает «в блокчейне», то есть на оборудовании валидаторов), и обладает чрезвычайно высокой (основанной на протоколе византийского соглашения) надежностью и доступностью блокчейна. Разработчику такого распределенного приложения не нужно покупать или арендовать какое-либо оборудование. Все, что необходимо сделать, это разработать какое-то программное обеспечение (например, код для смарт-контрактов). После этого разработчик фактически арендует вычислительную мощность у валидаторов и платит за нее TON монетами сам, либо переложив это бремя на плечи своих пользователей.

4.1.5. Чисто сетевые услуги: «TON-сайты» и «TON-сервисы». Другой крайний вариант - развернуть службу на некоторых серверах и сделать ее доступной для пользователей через протокол ADNL, описанный в п. 3.1, и, возможно, какой-либо протокол более высокого уровня (например, описанный в п. 3.1.9 протокол RLDP), который можно использовать для отправки запросов RPC на сервис в любом настраиваемом формате и получить ответы на эти запросы. Таким образом, сервис будет полностью отключен от сети и будет находиться в сети TON почти без использования блокчейна TON.
Блокчейн TON может использоваться только для определения местоположения абстрактного адреса или адресов службы, как указано в 3.2.12, например, с помощью службы вроде TON DNS (см. 4.3.1), для облегчения перевода понятных человеку доменных имен в абстрактные адреса.
В той степени, в которой сеть ADNL (то есть сеть TON) похожа на проект Invisible Internet Project (I^2P), такие (почти) чисто сетевые услуги аналогичны так называемым «еер-сервисам» (т. е. сервисам, которые имеют 1^2P-адрес в качестве точки входа и доступен клиентам через сеть 1^2P). Можно сказать, что такие чисто сетевые сервисы, реализуемые в сети TON, являются «TON-сервисами».
«Еер-сервис» может реализовать HTTP в качестве своего клиент-серверного протокола. В контексте сети TON «TON-сервис» может просто использовать датаграммы RLDP (см. 3.1.9) для передачи HTTP-запросов и ответов на них. Если он использует TON DNS, чтобы разрешить поиск своего абстрактного адреса по удобочитаемому доменному имени, можно провести почту точную аналогию с веб-

сайтом. Можно даже написать специализированный браузер или специальный прокси-сервер («ton-proxy»), который запускается локально на машине пользователя и принимает произвольные HTTP-запросы от обычного веб-браузера, который использует пользователь (как только локальный IP-адрес и TCP-порт прокси-сервера вводятся в конфигурацию браузера), и пересылает эти запросы через сеть TON на абстрактный адрес службы. Тогда пользователь сможет просматривать страницы аналогично работе во всемирной паутине (WWW).
В экосистеме 1^2P такие «еер-сервисы» называются «еер-сайтами». В экосистеме TON можно легко создавать «TON-сайты». Этому отчасти способствует существование таких служб, как TON DNS, которые используют блокчейн TON и TON DHT для преобразования доменных имен (TON) в абстрактные адреса.'

4.1.6. Смешанные сервисы: частично оффчейн, частично ончейн сервисы. Некоторые сервисы могут использовать смешанный подход: выполнять большую часть обработки вне сети, но также иметь некоторую часть в блокчейне (например, для регистрации своих обязательств перед пользователями и наоборот). Таким образом, часть состояния по-прежнему будет храниться в блокчейне TON (то есть в неизменяемом публичном реестре), а любое неправильное поведение сервиса или его пользователей может быть наказано при помощи смарт-контрактов.

4.1.7. Пример: хранение файлов вне сети; TON Storage. Примером такого сервиса является TON Storage. В своей простейшей форме он позволяет пользователям хранить информацию вне сети, сохраняя в блокчейне только хэш информации, которая будет храниться, и, возможно, смарт-контракт, в котором некоторые другие стороны соглашаются хранить данную информацию в течение определенного периода времени за заранее оговоренную плату. Фактически, он может быть разделен на фрагменты небольшого размера (например, 1 килобайт), дополнен стирающим кодом (например, кодом Рида-Соломона или исходным кодом), после чего может быть построен хэш дерева Меркла для расширенной последовательности фрагментов, который может быть опубликован в смарт-контракте вместо обычного хеша файла или вместе с ним. Это чем-то напоминает способ хранения файлов в виде торрент-файлов.
Еще более простая форма хранения файлов - полностью вне сети: можно по существу создать «торрент» для нового файла и использовать TON DHT в качестве «распределенного торрент-трекера» для этого торрента (см. 3.2.10). Такая модель может действительно хорошо сработать для популярных файлов. Однако не дается никаких гарантий доступности. Например, для гипотетического сервиса «Facebook в блокчейне» (см. 2.9.13), который будет хранить фотографии своих пользователей полностью вне сети в таких «торрентах», существует риск потери фотографий обычных (не особенно популярных) пользователей, либо риск отсутствия возможности предоставлять эти фотографии в течение длительного времени. Технология TON Storage, которая в основном является автономной, но использует ончейн смарт-контракт для обеспечения доступности хранимых данных, может быть наилучшим решением для этой задачи.

4.1.8. Децентрализованные смешанные услуги или «сервисы туманных вычислений». До сих пор мы обсуждали централизованные смешанные сервисы и приложения. В то время как их сетевой компонент в блокчейне обрабатывается децентрализованным и распределенным образом, соответствующий автономный компонент полагается на некоторые серверы, контролируемые поставщиком услуг обычным централизованным образом. Вместо использования некоторых выделенных серверов вычислительные мощности могут быть арендованы у службы облачных

вычислений, предлагаемой одной из крупных компаний. Однако это не приведет к децентрализации автономного компонента сервиса.
Децентрализованный подход к реализации оффчейн компонента сервиса заключается в создании рынка, на котором пользователи с необходимым оборудованием, которые готовы сдать в аренду свои вычислительные мощности или дисковое пространство, будут предлагать свои услуги тем, кто в них нуждается.
Например, может существовать реестр (который также может называться «рынком» или «биржей»), в котором все узлы, заинтересованные в сохранении информации о других пользователях, будут публиковать свои контактные данные, а также информацию о доступной емкости хранилища, политике доступности и ценах. Пользователи, которым нужны эти услуги, могут найти поставщиков в этом реестре, и если другая сторона согласится, создать смарт-контракты в блокчейне и загрузить файлы для хранения вне сети. Таким образом, такой сервис, как TON Storage, становится действительно децентрализованным, поскольку ему не нужно полагаться на какой-либо централизованный кластер серверов для хранения информации.

4.1.9. Пример: платформы «туманных вычислений» как децентрализованные смешанные сервисы.
Еще один пример такого децентрализованного смешанного приложения возникает, когда кто-то хочет выполнить некоторые конкретные вычисления (например, 3D-рендеринг или обучение нейронных сетей), что зачастую требует специального и дорогостоящего оборудования. Пользователи, у которых имеется такое оборудование, могут предлагать свои услуги через аналогичный «обмен», а те, кто нуждается в таких услугах, будут их арендовать, причем обязательства сторон будут регистрироваться с помощью смарт-контрактов. Эта модель похожа на то, что обещают предоставить платформы «туманных вычислений», такие как Golem (https://golem.network/) или SONM (https://sonm.io/).

4.1.10. Пример: TON Proxy как сервис «туманных вычислений». Сервис TON Proxy является еще одним примером сервиса «туманных вычислений». В нем могут регистрироваться нем узлы, предлагающие свои услуги (с компенсацией или без нее) в качестве туннелей для сетевого трафика ADNL, а пользователи, которые нуждаются в таких услугах, могут выбрать один из этих узлов в зависимости от цены, задержки сигнала и предлагаемой ширины канала. После этого можно использовать платежные каналы, предоставляемые TON Payments, для обработки микроплатежей за услуги этих прокси-серверов, при этом платежи собираются, например, за каждые 128 KiB переведенных данных.

4.1.11. Пример: TON Payments как сервис «туманных вычислений». Платформа TON Payments (см. 5) также является примером такого децентрализованного смешанного приложения.

4.2. Подключение пользователей и поставщиков услуг

Мы увидели в 4.1.8, что «сервисы туманных вычислений» (т. е. смешанные децентрализованные сервисы) обычно нуждаются в рынках, биржах или реестрах, где происходит взаимодействие пользователей, которым необходимы определенные услуги, с пользователями, которые эти услуги предоставляют.
Вероятно, такие рынки будут реализованы как внутрисетевые, автономные или смешанные сервисы, централизованные или распределенные.

4.2.1. Пример: подключение к TON Payments. Например, если кто-то хочет использовать сервис TON Payments (см. 5), первым шагом будет поиск по как минимум нескольких существующих транзитных узлов «сети мгновенных платежей»

(см. 5.2) и установление с ними каналов оплаты в случае готовности этих узлов. Некоторые узлы могут быть найдены с помощью «охватывающей» оверлейной сети, которая должна содержать все узлы транзитной сети мгновенных платежей (см. 3.3.17). Однако неясно, будут ли эти узлы готовы создавать новые платежные каналы. Следовательно, необходим реестр, в котором узлы, готовые к созданию новых ссылок, могут публиковать свою контактную информацию (например, свои абстрактные адреса).

4.2.2. Пример: загрузка файла в TON Storage. Точно так же, если кто-то хочет загрузить файл в хранилище TON, он должен найти несколько узлов, желающих подписать смарт-контракт, обязывающий их хранить копию этого файла (или любого другого файла ниже определенного предела размера). Следовательно, необходим реестр узлов, предлагающих свои услуги по хранению информации.

4.2.3. Сетевые, смешанные и автономные реестры. Такой реестр поставщиков услуг может быть реализован полностью ончейн с помощью смарт-контракта, который будет хранить реестр в постоянном хранилище. Однако это довольно медленно и дорого. Смешанный подход более эффективен, когда относительно небольшой и редко изменяемый ончейн реестр используется только для указания некоторых узлов (по их абстрактным адресам или по их открытым ключам, которые могут использоваться для определения фактических абстрактных адресов, как описано в 3.2.12), которые предоставляют услуги реестра вне сети (централизованные).
Наконец, при децентрализованном, чисто автономном подходе может использоваться общедоступная оверлейная сеть (см. 3.3), в которой пользователи, которые хотят предложить свои услуги, или пользователи, которые хотят купить чьи-то услуги, просто транслируют свои предложения, подписанные закрытыми ключами. Если предоставляемая услуга очень проста, даже широковещательная передача предложений может не потребоваться: примерное членство в самой оверлейной сети может использоваться в качестве «реестра» тех, кто желает предоставить конкретную услугу. Клиент, которому требуется эта услуга, может найти (см. 3.3.7) и запросить некоторые узлы этой оверлейной сети, а затем запросить их соседей, если уже известные узлы не готовы предоставить необходимую услугу.

4.2.4. Регистрация или обмен в сайдчейне. Другой подход к реализации децентрализованных смешанных реестров заключается в создании независимого специализированного блокчейна («сайдчейна»), поддерживаемого собственным набором самопровозглашенных валидаторов, которые публикуют свои идентификаторы в ончейн смарт-контракте и предоставляют всем заинтересованным сторонам доступ в этот специализированный блокчейн, собирая транзакции-кандидаты и транслируя обновления блоков через выделенные оверлейные сети (см. 3.3). Затем любая полная нода для этого сайдчейна может поддерживать свою собственную копию общего реестра (по существу, равную глобальному состоянию этого сайдчейна) и обрабатывать произвольные запросы, связанные с этим реестром.

4.2.5. Реестр или обмен в воркчейне. Другой вариант - создать отдельный воркчейн внутри блокчейна TON, специализирующийся на создании реестров, рынков и бирж. Этот способ может быть более эффективным и менее затратным, чем использование смарт-контрактов в основном воркчейне (см. 2.1.11), однако это все равно будет дороже, чем поддержание реестров в сайдчейнах (см. 4.2.4).

4.3. Доступ к сервисам TON

В разделе 4.1 мы обсудили различные подходы, которые можно использовать для создания новых сервисов и приложений в экосистеме TON. Теперь мы обсудим, как можно получить доступ к этим сервисам, а также некоторые из «вспомогательных сервисов», которые будут предоставляться TON, включая TON DNS и TON Storage.

4.3.1. TON DNS: иерархическая служба доменных имен в сети (в основном ончейн). TON DNS - это предопределенная служба, которая использует набор смарт-контрактов для сохранения карты от понятных человеку доменных имен до (256-битных) адресов сетевых узлов ADNL, учетных записей и смарт-контрактов блокчейна TON.
Хотя в принципе любой может реализовать такой сервис с использованием блокчейна TON, полезно иметь подобный предопределенный сервис с хорошо известным интерфейсом, который будет использоваться по умолчанию всякий раз, когда приложение или сервис захочет преобразовать понятные человеку идентификаторы в адреса.

4.3.2. Примеры использования TON DNS. Например, пользователь, желающий передать некоторую криптовалюту другому пользователю или продавцу, может предпочесть запомнить доменное имя TON DNS учетной записи этого пользователя или продавца, вместо того, чтобы держать свои 256-битные идентификаторы учетных записей под рукой и копировать и вставлять их в поля информации о получателе, хранящейся в легком кошельке клиента.
Аналогичным образом, TON DNS может использоваться для определения идентификаторов учетных записей смарт-контрактов или точек входа в TON-сервисы и TON-сайты (см. 4.1.5), включая специализированный клиент («TON-браузер») или обычный интернет-браузер в сочетании со специализированным расширением ton-proxy или автономным приложением, чтобы предоставить пользователю возможность просмотра веб-страниц в стиле WWW.

4.3.3. Смарт-контракты TON DNS. TON DNS реализуется с помощью дерева специальных смарт-контрактов (DNS). Каждый смарт-контракт DNS отвечает за регистрацию поддоменов некоторого фиксированного домена. «Корневой» смарт-контракт DNS, в котором будут храниться домены первого уровня системы TON DNS, находится в мастерчейне. Соответствующий идентификатор учетной записи должен быть жестко запрограммирован во всем программном обеспечении, которое будет напрямую обращаться к базе данных TON DNS.
Любой смарт-контракт DNS содержит хэш-карту, отображающую строки UTF-8 переменной длины с завершающим нулем в их «значениях». Эта хэш-карта реализована в виде двоичного дерева Patricia, подобного дереву, описанному в 2.3.7, но также поддерживает битовые строки переменной длины в качестве ключей.

4.3.4. Значения хэш-карты DNS или записей TON DNS. Значения представлены в виде «DNS-записей TON», описываемых TL-схемой (см. 2.2.5). Они состоят из «магического числа», выбора одной из поддерживаемых опций, а также одного из следующих идентификаторов: идентификатора учетной записи, идентификатора смарт-контракта, абстрактного сетевого адреса (см. 3.1), открытого ключа, используемого для определения местоположения абстрактных адресов сервиса (см. 3.2.12) или описания оверлейной сети и т. д. Важным является использование другого смарт-контракта DNS: в таком случае этот смарт-контракт используется для преобразования поддоменов собственного домена. Таким образом, можно создавать отдельные реестры для разных доменов, контролируемые владельцами этих доменов.

Эти записи также могут содержать время истечения срока действия, время кеширования (обычно очень большое, поскольку обновление значений в блокчейне слишком часто обходится дорого) и в большинстве случаев ссылку на владельца рассматриваемого поддомена. Владелец имеет право изменить эту запись (в частности, владелец может передать домен под чей-то контроль) и продлить ее.

4.3.5. Регистрация новых поддоменов существующих доменов. Чтобы зарегистрировать новый поддомен существующего домена, нужно просто отправить сообщение смарт-контракту, который является регистратором этого домена, содержащим поддомен (то есть ключ), который должен быть зарегистрирован, значение в одном из нескольких предопределенных форматов, личность владельца, срок действия и некоторое количество криптовалюты, определяемое владельцем домена.
Поддомены регистрируются по принципу «в порядке очереди».

4.3.6. Получение данных из смарт-контракта DNS. В принципе, любая полная нода мастерчейна или шардчейна, содержащая смарт-контракт DNS, может иметь возможность поиска любого поддомена в базе данных этого смарт-контракта, если известна структура и расположение хэш-карты в постоянном хранилище смарт-контракта.
Однако этот подход будет работать только для определенных смарт-контрактов DNS. В случае использования нестандартного смарт-контракта DNS такой подход будет неуспешным.
Вместо этого используется подход, основанный на общих интерфейсах смарт-контрактов и методах получения (см. 4.3.11). Любой смарт-контракт DNS должен определять «метод получения» с «известной подписью», который вызывается для поиска ключа. Поскольку этот подход имеет смысл и для других смарт-контрактов, особенно для тех, которые обеспечивают сетевые и смешанные сервисы, он будет более подробно описан в п. 4.3.11.

4.3.7. Перевод домена TON DNS. Если любой полной ноде, действующей самой по себе или от имени какого-либо тонкого клиента, необходимо найти записи в базе данных любого смарт-контракта DNS, произвольные доменные имена TON DNS могут быть рекурсивно преобразованы, начиная с хорошо известного и фиксированного идентификатора корневого смарт-контракта DNS (учетной записи).
Например, если необходимо перевести A. B. C, следует найти ключи .C, .B.C и A.B.C в базе данных корневого домена. Если первый из них не был найден, а второй был найден, и его значение является ссылкой на другой смарт-контракт DNS, тогда A ищется в базе данных этого смарт-контракта, после чего извлекается окончательное значение.

4.3.8. Перевод доменов TON DNS для неполных нод. Таким образом, полная нода мастерчейна, а также всех шардчейнов, участвующих в процессе поиска домена, может преобразовывать любое доменное имя в его текущее значение без внешней помощи. Неполная нода может запросить полную ноду сделать это от своего имени и вернуть значение вместе с доказательством Меркла (см. 2.5.11). Это доказательство Меркла позволит неполной ноде проверить правильность ответа, поэтому такие ответы TON DNS не могут быть «подделаны» злонамеренным перехватчиком, в отличие от обычного протокола DNS.
Поскольку нельзя ожидать, что нода будет полной нодой по отношению ко всем шардчейнам, фактическая трансляция домена TON DNS будет включать комбинацию этих двух стратегий.

4.3.9. Выделенные «серверы TON DNS». Можно предоставить простой «сервер TON DNS», который будет получать RPC-запросы «DNS» (например, через протоколы

ADNL или RLDP, описанные в 3.1), запрашивая, чтобы сервер переводил данный домен, обрабатывал эти запросы, при необходимости перенаправляя некоторые подзапросы на другие (полные) ноды, и возвращать ответы на исходные запросы, дополненные доказательствами Меркла, если необходимо.
Такие «DNS-серверы» могут предлагать свои услуги (бесплатно или платно) любым другим узлам и особенно легким клиентам, используя один из методов, описанных в п. 4.2. Обратите внимание, что эти серверы, если их рассматривать как часть службы TON DNS, эффективно преобразовали бы ее из распределенного сервиса в распределенный смешанный сервис (т. е. «сервис туманных вычислений»).
На этом мы завершаем краткий обзор службы TON DNS, масштабируемого сетевого реестра для понятных человеку доменных имен объектов блокчейна TON и сети TON Network.

4.3.10. Доступ к данным, хранящимся в смарт-контрактах. Мы уже увидели, что иногда необходимо получить доступ к данным, хранящимся в смарт-контракте, без изменения его состояния.
Если пользователю известны детали реализации смарт-контракта, можно извлечь всю необходимую информацию из постоянного хранилища смарт-контракта, доступного для всех полных нод шардчейна, в котором находится смарт-контракт. Однако это довольно грубый способ, очень сильно зависящий от реализации смарт-контракта.

4.3.11. «Методы получения» смарт-контрактов. Лучшим способом было бы определить некоторые методы получения в смарт-контракте, то есть некоторые типы входящих сообщений, которые не влияют на состояние смарт-контракта при доставке, но генерируют одно или несколько выходных сообщений, содержащих «результат» метода получения. Таким образом, можно получить данные из смарт-контракта, зная только, что он реализует метод получения с известной подписью (т. е. известный формат входящего сообщения, которое должно быть отправлено, и исходящего сообщения, которое должно быть в результате получено).
Этот способ намного более изящный и соответствует объектно-ориентированному программированию (ООП). Однако у него имеется очевидный дефект: нужно фактически зафиксировать транзакцию в блокчейне (отправить сообщение get в смарт-контракт), дождаться, пока она будет зафиксирована и обработана валидаторами, извлечь ответ из нового блока и внести оплату за газ (то есть за выполнение метода получения на оборудовании валидаторов). Это пустая трата ресурсов: методы получения в любом случае не изменяют состояние смарт-контракта, поэтому их не нужно выполнять в блокчейне.

4.3.12. Предварительное выполнение методов получения смарт-контрактов. Мы уже отмечали (см. 2.4.6), что любая полная нода может предварительно выполнить любой метод любого смарт-контракта (то есть доставить любое сообщение в смарт-контракт), начиная с заданного состояния смарт-контракта, без фактического совершения соответствующей сделки. Полная нода может просто загрузить код рассматриваемого смарт-контракта в виртуальную машину TON, инициализировать свое постоянное хранилище из глобального состояния шардчейна (известного всем полным нодам шардчейна) и выполнить код смарт-контракта с входящим сообщением в качестве входного параметра. Созданные выходные сообщения будут иметь результат этого вычисления.
Таким образом, любая полная нода может оценивать произвольные методы получения произвольных смарт-контрактов, если известна их подпись (т. е. формат входящих и исходящих сообщений). Нода может отслеживать ячейки состояния шардчейна, к которым осуществляется доступ во время этой оценки, и создавать доказательство

Меркла валидности выполненных вычислений в пользу неполной ноды, которая отправила бы соответствующий запрос на полную ноду (см. 2.5.11).

4.3.13. Интерфейсы смарт-контрактов в TL-схемах. Напомним, что методы, реализованные в смарт-контракте (то есть принимаемые им входные сообщения), по сути, являются некоторыми TL-сериализированными объектами, которые могут быть описаны TL-схемой (см. 2.2.5). Полученные выходные сообщения также можно описать той же TL-схемой. Таким образом, интерфейс смарт-контракта с другими учетными записями и смарт-контрактами может быть формализован с помощью TL-схемы.
В частности, (подмножество) методов получения, поддерживаемых смарт-контрактом, можно описать с помощью такого формализованного интерфейса смарт-контракта.

4.3.14. Публичные интерфейсы смарт-контракта. Обратите внимание, что формализованный интерфейс смарт-контракта, в форме TL-схемы (представленной как исходный файл TL; см. 2.2.5), либо в сериализированной форме[34], может быть опубликован, например, в специальной форме, хранящейся в описании учетной записи смарт-контракта, хранящемся в блокчейне или отдельно, если к этому интерфейсу будут часто обращаться. В последнем случае хэш поддерживаемого общедоступного интерфейса может быть включен в описание смарт-контракта вместо самого описания интерфейса.
Примером такого общедоступного интерфейса является смарт-контракт DNS, который должен реализовывать как минимум один стандартный метод получения для поиска поддоменов (см. 4.3.6). Стандартный метод регистрации новых поддоменов также может быть включен в стандартный общедоступный интерфейс смарт-контрактов DNS.

4.3.15. Пользовательский интерфейс смарт-контракта. Существование публичного интерфейса для смарт-контракта имеет и другие преимущества. Например, клиентское приложение кошелька может загружать такой интерфейс при изучении смарт-контракта по запросу пользователя и отображать список общедоступных методов (то есть доступных действий),
поддерживаемых смарт-контрактом, возможно, с некоторыми удобочитаемыми комментариями, если таковые имеются, в формальном интерфейсе. После того, как пользователь выберет один из этих методов, форма может быть автоматически сгенерирована в соответствии с TL-схемой, где пользователю будет предложено ввести все поля, необходимые для выбранного метода, и желаемое количество криптовалюты (например, монет TON) которые должны быть прикрепленным к этому запросу. Отправка этой формы создаст новую транзакцию блокчейна, содержащую только что составленное сообщение, отправленное из учетной записи блокчейна пользователя.
Таким образом, пользователь сможет взаимодействовать с произвольными смарт-контрактами из клиентского приложения кошелька удобным для пользователя способом, заполняя и отправляя определенные формы, при условии, что эти смарт-контракты опубликовали свои интерфейсы.

4.3.16. Пользовательский интерфейс «TON-сервиса». «TON-сервисы» (т. е. сервисы, находящиеся в сети TON и принимающие запросы через протоколы ADNL и RLDP из 3; см. 4.1.5) также могут получить прибыль от наличия общедоступных

[34] TL-схемы сами могут быть TL-сериализированы; см. https://core.telegram.org/mtproto/TL-tl

интерфейсов, описываемых TL- схемами (см. 2.2.5). Клиентское приложение, такое как легкий кошелек или «тонкий браузер», может предлагать пользователю выбрать один из методов и заполнить форму с параметрами, определенными интерфейсом, аналогично той схеме, которая только что обсуждалась в 4.3. 15. Единственное отличие состоит в том, что результирующее TL-сериализированное сообщение не отправляется как транзакция в блокчейне; вместо этого оно отправляется как запрос RPC на абстрактный адрес рассматриваемого «TON-сервиса», и ответ на этот запрос анализируется и отображается в соответствии с формальным интерфейсом (т. е. TL-схемой).

4.3.17. Поиск пользовательских интерфейсов через TON DNS. Запись TON DNS, содержащая абстрактный адрес TON-сервиса или идентификатор учетной записи смарт-контракта, также может содержать необязательное поле, описывающее общедоступный (пользовательский) интерфейс этого объекта или несколько поддерживаемых интерфейсов. Затем клиентское приложение (будь то кошелек, TON-браузер или TON-прокси) сможет загружать интерфейс и взаимодействовать с рассматриваемым объектом (будь то смарт-контракт или TON-сервис).

4.3.18. Стирание различий между ончейн и оффчейн сервисами. Таким образом, различие между ончейн, оффчейн и смешанными сервисами (см. 4.1.2) стирается для конечного пользователя: он просто вводит доменное имя необходимого сервиса в адресную строку своего TON-браузера или кошелек, а остальное легко обрабатывается клиентским приложением.

4.3.19. «TON-сайты» как TON-сервисы, поддерживающие HTTP-интерфейс. TON-сайт - это просто TON-сервис, который поддерживает интерфейс HTTP, возможно, вместе с некоторыми другими интерфейсами. Об этой поддержке можно объявить в соответствующей записи TON DNS.

4.3.20. Гиперссылки. Обратите внимание, что HTML-страницы, возвращаемые TON-сайтами, могут содержать TON-гиперссылки (то есть ссылки на другие TON-сайты, смарт-контракты и учетные записи посредством специально созданных схем URI (см. 4.3.21)), содержащие абстрактные сетевые адреса, идентификаторы учетных записей или человекочитаемые домены TON DNS. Затем «TON-браузер» может следовать по такой гиперссылке, когда пользователь выбирает ее,
обнаруживать интерфейс, который будет использоваться, и отображать форму пользовательского интерфейса, как описано в 4.3.15 и 4.3.16.

4.3.21. URL-адреса гиперссылок могут указывать некоторые параметры. URL-адреса гиперссылок могут содержать не только DNS-домен (TON) или абстрактный адрес рассматриваемого сервиса, но также имя вызываемого метода и некоторые или все его параметры. Возможная схема URI для этого может выглядеть следующим образом:

ton://<*domain*>/<*method*>?<*field1*>=<*value1*>&<*field2*>=...

Когда пользователь выбирает такую ссылку в TON-браузере, то действие либо выполняется немедленно (особенно если это метод получения смарт-контракта, вызываемый анонимно), либо отображается частично заполненная форма, которая должна быть явно подтверждена и отправлена пользователем (это может потребоваться для форм оплаты).

4.3.22. POST-действия. TON-сайт может встраиваться в HTML-страницы, он возвращает некоторые обычные POST-формы, при этом POST-действия ссылаются на

TON-сайты, TON-сервисы или смарт-контракты с помощью подходящих (TON) URL-адресов. В этом случае, как только пользователь заполняет и отправляет эту настраиваемую форму, соответствующее действие предпринимается либо сразу, либо после явного подтверждения.

4.3.23. Сеть TON WWW. Все вышеперечисленное приведет к созданию целой сети объектов с перекрестными ссылками, находящихся в сети TON, которые будут доступны конечному пользователю через тонкий браузер, предоставляя пользователю возможность просмотра веб-страниц в стиле WWW. Для конечных пользователей это, наконец, сделает приложения блокчейна принципиально похожими на веб-сайты, к которым они уже привыкли.

4.3.24. Преимущества TON WWW. Этот «TON WWW» сетевых и автономных сервисов имеет некоторые преимущества перед своим традиционным аналогом. Например, платежи по своей сути интегрированы в систему. Идентификационные данные пользователя всегда могут быть предоставлены сервисам (посредством автоматически сгенерированных подписей на транзакциях и сгенерированных запросов RPC) или скрыты по желанию. Сервисам не нужно будет перепроверять учетные данные пользователя; эти учетные данные могут быть опубликованы в блокчейне раз и навсегда. Анонимность пользователей в сети может быть легко сохранена с помощью TON Proxy, а все сервисы будут всегда доступными. Также будут доступными микроплатежи, потому что TON-браузеры могут быть интегрированы с системой TON Payments.

5 Платежная система TON Payments

Последний компонент проекта TON, который мы кратко обсудим в этом тексте, - это TON Payments, платформа для (микро) платежных каналов и передачи значений «сети мгновенных платежей». Это позволит осуществлять «мгновенные» платежи без необходимости фиксировать все транзакции в блокчейне, оплачивать соответствующие комиссии за транзакции (например, за потребленный газ) и ждать пять секунд, пока блок, содержащий рассматриваемые транзакции, не будет подтвержден.

Общие накладные расходы на такие мгновенные платежи настолько малы, что их можно использовать для микроплатежей. Например, служба хранения файлов TON может взимать с пользователя плату за каждые 128 Кбайт загруженных данных, а платный прокси-сервер TON может потребовать небольшую микроплату за каждые 128 Кбайт ретранслируемого трафика.

Хотя платформа TON Payments, вероятно, будет выпущена позже, чем основные компоненты проекта TON, некоторые соображения необходимо учесть в самом начале. Например, виртуальная машина TON (VM TON; см. 2.1.20), используемая для выполнения кода смарт-контрактов блокчейна TON, должна поддерживать некоторые специальные операции с доказательствами Меркла. Если такая поддержка отсутствует в исходном проекте, добавление ее на более позднем этапе может стать проблематичным (см. 2.8.16). Однако мы увидим, что VM TON изначально поддерживает «умные» каналы оплаты (см. 5.1.9).

5.1. Каналы оплаты

Мы начнем с обсуждения каналов оплаты «напрямую» и способов их реализации в блокчейне TON.

5.1.1. Идея платежного канала. Предположим, две стороны (A и B) знают, что в будущем им нужно будет совершать множество платежей друг другу. Вместо того

чтобы фиксировать каждый платеж как транзакцию в блокчейне, они создают общий «денежный пул» (или, возможно, небольшой частный банк с ровно двумя счетами) и вносят в него некоторые средства: A вносит a монет, B вносит b монет. Это достигается путем создания специального смарт-контракта в блокчейне и отправкой на него средств.
Перед созданием «денежного пула» стороны соглашаются на определенный протокол. Они будут отслеживать состояние пула, то есть свои балансы в общем пуле. Первоначально состоянием является (a, b), что означает, что монеты на самом деле принадлежат A, а b монет принадлежат B. Затем, если A хочет заплатить d монет в пользу B, они могут просто согласиться с тем, что новое состояние - (a', b') = (a - d, b + d). Впоследствии, если, скажем, B хочет заплатить d' монет в пользу A, состояние станет (a'', b'') = (a' + d', b' - d') и так далее.
Все это обновление балансов внутри пула происходит полностью вне сети. Когда две стороны решают снять причитающиеся им средства из пула, они делают это в соответствии с окончательным состоянием пула. Это достигается путем отправки специального сообщения в смарт-контракт, содержащего согласованное конечное состояние (a *, b *) вместе с подписями как A, так и B. Затем смарт-контракт отправляет a * монет в A, b * монеты в B и самоуничтожается.
Этот смарт-контракт вместе с сетевым протоколом, используемым A и B для обновления состояния пула, представляет собой простой платежный канал между A и B. Согласно классификации, описанной в 4.1.2, это смешанный сервис: часть его состояние находится в блокчейне (смарт-контракте), но большая часть соответствующих обновлений состояния выполняется вне сети (по сетевому протоколу). Если все идет хорошо, две стороны смогут выполнить столько платежей друг другу, сколько захотят (с единственным ограничением, что не должна быть превышена «пропускная способность» канала, т. е. их остатки на платежном канале не будут отрицательными), совершая только две транзакции в блокчейне: одна для открытия (создания) платежного канала (смарт-контракт), а другая для его закрытия (уничтожения).

5.1.2. Не требующие доверия каналы оплаты. Предыдущий пример был несколько нереалистичным, так как он предполагает, что обе стороны готовы сотрудничать и никогда не обманывают друг друга, чтобы получить какое-либо преимущество. Представьте, например, что A решит не подписывать окончательный баланс (a', b') с a' <a. Такой вариант может поставить B в затруднительное положение.
Чтобы защитить пользователей от таких сценариев, обычно разрабатываются «не требующие доверия» протоколы каналов оплаты, которые не требуют от сторон доверия друг к другу, и предусматривают наказание любой стороны, которая попытается обмануть другую сторону.
Обычно это достигается с помощью подписей. Смарт-контракт платежного канала знает открытые ключи A и B и при необходимости может проверять их подписи. Протокол платежного канала требует, чтобы стороны подписали промежуточные состояния и отправили подписи друг другу. Затем, если одна из сторон обманывает - например, делает вид, что какое-то состояние платежного канала никогда не существовало, - ее некорректное поведение можно доказать, поставив свою подпись на этом состоянии. Смарт-контракт платежного канала выступает в роли «сетевого арбитра», способного обрабатывать жалобы двух сторон друг на друга и наказывать виновную сторону, конфисковав все ее средства и передав их другой стороне.

5.1.3. Простой двунаправленный синхронный канал оплаты, не требующий доверия. Рассмотрим следующий, более реалистичный пример: Пусть состояние платежного канала описывается тройкой ($б_i$, i, o_i, где i - порядковый номер состояния (изначально он равен нулю, а затем увеличивается на единицу при последующем

появляется состояние), $б_i$ - дисбаланс канала (это означает, что A и B владеют монетами a + $б_i$ и b - $б_i$ соответственно), а o_i - сторона, которой разрешено генерировать следующее состояние (либо A, либо B). Каждое состояние должно быть подписано как A, так и B, прежде чем будет достигнут какой-либо дальнейший прогресс.

Теперь, если A хочет передать d монет B внутри платежного канала, а текущее состояние - $S_i = (б_i, i, o_i)$ с o_i = A, то он просто создает новое состояние $S_i + 1 = (б_i - d , i + 1, o_i + 1)$, подписывает его и отправляет B вместе со своей подписью. Затем B подтверждает это, подписывая и отправляя копию своей подписи A. После этого обе стороны получают копию нового состояния с обеими своими подписями, и может произойти новая передача.

Если A хочет передать монеты B в состоянии S_i с o_i = B, то сначала он просит B зафиксировать последующее состояние S_{i+1} с таким же дисбалансом $б_{i+1} = б_i$, но с o_{i+1} = A После этого, A сможет Выполнить передачу.

Когда две стороны соглашаются закрыть платежный канал, они ставят свои специальные окончательные подписи на состояние S_k, которое они считают окончательным, и вызывают чистый или двусторонний метод финализации смарт-контракта платежного канала, отправив ему окончательное состояние вместе с обеими окончательными подписями.

Если другая сторона не соглашается предоставить свою окончательную подпись или просто перестает отвечать, есть возможность закрыть канал в одностороннем порядке. Для этого сторона, желающая сделать это, вызывает метод односторонней финализации, отправив смарт-контракту свою версию конечного состояния, свою окончательную подпись и самое последнее состояние, имеющее подпись другой стороны. После этого смарт-контракт не сразу воздействует на полученное конечное состояние. Вместо этого он ждет в течение определенного периода времени (например, одного дня), пока другая сторона не представит свою версию окончательного состояния. Когда другая сторона отправляет свою версию, и она оказывается совместимой с уже представленной версией, «истинное» конечное состояние вычисляется смарт-контрактом и используется для соответствующего распределения средств. Если другая сторона не может представить свою версию конечного состояния смарт-контракту, то деньги перераспределяются в соответствии с единственной представленной копией конечного состояния.

Если одна из двух сторон обманывает - например, подписывая два разных состояния как окончательные, подписывая два разных следующих состояния S_{i+1} и S'_{i+1}, либо подписывая недопустимое новое состояние S_{i+1} (например, с дисбалансом $б_{i+1}$ < -a или > b) - тогда другая сторона может предоставить доказательство этого неправомерного поведения третьему методу смарт-контракта. Виновная сторона немедленно наказывается полной потерей своей доли в платежном канале.

Этот простой протокол платежного канала справедлив в том смысле, что любая сторона всегда может получить должное, при сотрудничестве или без сотрудничества с другой стороной, и, вероятно, потеряет все свои средства, переданные в платежный канал, если попытается обмануть.

5.1.4. Синхронный платежный канал как простой виртуальный блокчейн с двумя валидаторами. Приведенный выше пример простого синхронного платежного канала можно изменить следующим образом. Представьте, что последовательность состояний $S_0, S_1,...,S_n$ на самом деле является последовательностью блоков очень простого блокчейна. Каждый блок этого блокчейна содержит по существу только текущее состояние блокчейна и, возможно, ссылку на предыдущий блок (то есть его хэш). Обе стороны A и B выступают в роли валидаторов для этого блокчейна, поэтому каждый блок должен собирать обе подписи. Состояние S_i блокчейна определяет назначенного производителя с ^ для

следующего блока, поэтому между A и B нет гонки за создание следующего блока. Производителю A разрешено создавать блоки, которые переводят средства от A к B (т. е. уменьшают дисбаланс: $б_{i+1} \leq б_i$), а B может переводить средства только от B к A (т. е. увеличивать б).
Если два валидатора согласовывают окончательный блок (и окончательное состояние) блокчейна, он завершается сбором специальных «окончательных» подписей двух сторон и отправкой их вместе с последним блоком в смарт-контракт канала для обработки и соответствующее перераспределение средств.
Если валидатор подписывает недействительный блок, создает форк или подписывает два разных финальных блока, он может быть наказан. При этом доказательство некорректного поведения валидатора предоставляется смарт-контракту, который действует как «ончейн-арбитр» для двух валидаторов; тогда нарушившая сторона теряет все свои деньги, хранящиеся в платежном канале, что аналогично тому, как валидатор теряет свою долю.

5.1.5. Асинхронный платежный канал как виртуальный блокчейн с двумя воркчейнами. Синхронный платежный канал, описанный в 5.1.3, имеет определенный недостаток: нельзя начать следующую транзакцию (перевод денег средств внутри платежного канала) до того, как предыдущая будет подтверждена другой стороной. Это можно исправить, заменив единый виртуальный блокчейн, описанный в 5.1.4, системой из двух взаимодействующих виртуальных воркчейнов (или, скорее, шардчейнов).
Первый из этих воркчейнов содержит только транзакции A, а его блоки могут быть сгенерированы только A. Его состояния: $S_i = (i, ф_i, j, ψ_j)$, где i - порядковый номер блока (т. е. количество транзакций или денежных переводов, выполненных A на данный момент), $ф_i$ - это общая сумма, переведенная от A к B на данный момент , j - порядковый номер самого последнего действительного блока в блокчейне B's котором известно A, $ψ_j$ - сумма денег, переданная от B к A в его j транзакциях. Подпись B, помещенная в его j-й блок, также должна быть частью этого состояния. Также могут быть включены хэши предыдущего блока этой воркчейна и j-го блока другой воркчейна. Условия применимости для S_i включают $ф_i \geq 0$ $ф_i \geq ф_{i-1}$, если $i > 0$, $ψ_j \geq 0$, и $-a \leq ψ_j - ф_i \leq b$.
Точно так же второй воркчейн содержит только транзакции B, а его блоки генерируются только B. Его состояния: $T_j = (j, ψ_j , i, φ_i)$ с аналогичными условиями выполнения.
Если A хочет перевести средства B, он просто создает новый блок в своем воркчейне, подписывает его и отправляет B, не дожидаясь подтверждения.
Платежный канал завершается подписанием A (его версией) конечного состояния своего блокчейна (со специальной «окончательной подписью»), подписанием B конечного состояния своего блокчейна и представлением этих двух конечных состояний методу чистой финализации смарт-контракта платежного канала. Одностороннее завершение также возможно, однако в этом случае смарт-контракт должен будет дождаться, пока другая сторона представит свою версию окончательного состояния, по крайней мере, в течение некоторого периода отсрочки.

5.1.6. Однонаправленные платежные каналы. Если только A необходимо произвести платежи для B (например, B является поставщиком услуг, а A - его клиентом), то может быть создан канал односторонних платежей. По сути, это просто первый воркчейн, описанный в 5.1.5, без второго воркчейна. И наоборот, можно сказать, что асинхронный платежный канал, описанный в 5.1.5, состоит из двух однонаправленных платежных каналов или «полуканалов», управляемых одним и тем же смарт-контрактом.

5.1.7. Более сложные каналы оплаты. «Обещания». Позже в 5.2.4 мы увидим, что «сеть мгновенных платежей» (см. 5.2), которая обеспечивает мгновенные денежные переводы через цепочки из нескольких платежных каналов, требует более высокой степени сложности задействованных платежных каналов.

В частности, мы хотим иметь возможность давать «обещания» или совершать «условные денежные переводы»: A соглашается отправить *c* монет B, но B получит деньги только при выполнении определенного условия, например, если B может представить некоторую строку u с HASH(U) = v для известного значения v. В противном случае A может вернуть деньги через определенный период времени.

Такое обещание может быть легко реализовано в сети с помощью простого смарт-контракта. Однако мы хотим, чтобы обещания и другие виды условных денежных переводов были возможны оффчейн, в платежном канале, потому что они значительно упрощают денежные переводы по цепочке платежных каналов, существующих в «сети мгновенных платежей» (см. 5.2. 4).

Схема «платежный канал как простой блокчейн», описанная в 5.1.4 и 5.1.5, здесь становится очень удобной. Рассмотрим более сложный виртуальный блокчейн, состояние которого содержит набор таких невыполненных «обещаний» и количество средств, заблокированных в таких обещаниях. Этот блокчейн - или два воркчейна в асинхронном случае - должны будут явно ссылаться на предыдущие блоки по их хэшам. Тем не менее, общий механизм остается прежним.

5.1.8. Проблемы при использовании сложных смарт-контрактов платежных каналов. Обратите внимание, что, хотя конечное состояние сложного платежного канала все еще невелико, а «чистое» завершение является простым (если две стороны согласовали свои причитающиеся суммы, и обе подписали соглашение, больше ничего не нужно делать), метод одностороннего завершения и метод наказания мошенничества должны быть более сложными. Действительно, они должны иметь возможность принимать доказательства Меркла при неправомерном поведении и проверять, правильно ли были обработаны более сложные транзакции блокчейна платежного канала.

Другими словами, смарт-контракт платежного канала должен иметь возможность работать с доказательствами Меркла, проверять их «хеш-валидность» и должен содержать реализацию функций ev_trans и ev_block (см. 2.2.6) для платежного канала (виртуального) блокчейна.

5.1.9. VM TON поддерживает «умные» каналы оплаты. Виртуальная машина TON, используемая для запуска кода смарт-контрактов блокчейна TON, справляется с задачей выполнения смарт-контрактов, необходимых для «умных» или сложных каналов оплаты (см. 5.1.8),

На этом этапе парадигма «все есть мешок ячеек» (см. 2.5.14) становится чрезвычайно удобной. Поскольку все блоки (включая блоки блокчейна канала мгновенного платежа) представлены как мешки ячеек (и описываются некоторыми алгебраическими типами данных), и то же самое верно для сообщений и доказательств Меркла, доказательство Меркла может быть легко встроено во входящее сообщение, отправленное на смарт-контракт платежного канала.

«Условие хеширования» доказательства Меркла будет проверяться автоматически, и когда смарт-контракт получит доступ к предоставленному «доказательству Меркла», он будет работать с ним, как если бы это было значение соответствующего алгебраического типа данных - хотя и неполное, в котором некоторые поддеревья дерева заменены специальными узлами, содержащими хэш Меркла пропущенного поддерева. Затем смарт-контракт будет работать с этим значением, которое может предоставлять, например, блок (виртуального) блокчейна платежного канала вместе с его состоянием, и будет оценивать функцию ev_block (см. 2.2.6) этого блокчейна в

этом блоке и предыдущем состоянии. Затем либо вычисление завершается, и конечное состояние может быть сравнено с тем, что заявлено в блоке, либо при попытке доступа к отсутствующему поддереву выдается исключение «отсутствующий узел», указывающее, что доказательство Меркла недействительно.
Таким образом, реализация кода проверки для блокчейнов «умных» платежных каналов оказывается довольно простой при использовании смарт-контрактов блокчейна TON. Можно сказать, что виртуальная машина TON имеет встроенную поддержку для проверки валидности других простых блокчейнов. Единственным ограничивающим фактором является размер доказательства Меркла, которое должно быть включено во входящее сообщение для смарт-контракта (т. е. в транзакцию).

5.1.10. Простой платежный канал в «умном» платежном канале. Мы хотели бы обсудить возможность создания простого (синхронного или асинхронного) платежного канала внутри существующего платежного канала.
Хотя это может показаться несколько запутанным, эту схему не намного сложнее понять и реализовать, чем «обещания», описанные в 5.1.7. По сути, вместо обещания заплатить c монет другой стороне, если присутствует некоторое решение задачи с хешем, A обещает выплатить B до c монет в соответствии с окончательным расчетом некоторого другого (виртуального) блокчейна платежного канала. Вообще говоря, этот блокчейн другого платежного канала даже не обязательно должен находиться между A и B; в нем могут быть задействованы другие стороны, например C и D, желающие внести монеты c и d в свой простой платежный канал, соответственно (эта возможность используется позже в 5.2.5.).
Если охватывающий платежный канал асимметричен, два обещания должны быть зафиксированы в двух воркчейнах: A обещает выплатить -б монет B, если окончательный расчет «внутреннего» простого платежного канала дает отрицательный окончательный дисбаланс 6 с $0 \leq -б \leq c$; и B должен будет пообещать заплатить б в пользу A, если б положителен. С другой стороны, если охватывающий платежный канал является симметричным, это может быть выполнено путем фиксации одной транзакции «создания простого платежного канала» с параметрами (c, d) в блокчейне одного платежного канала с помощью A (который заморозит c монет, принадлежащих A), а затем фиксации специальной «транзакции подтверждения» от B (которая заморозит d монет B).
Мы ожидаем, что внутренний платежный канал будет чрезвычайно простым (например, простой синхронный платежный канал, описанный в 5.1.3), что позволит минимизировать объем предоставляемых доказательств Меркла. Внешний платежный канал должен быть «умным» в смысле, описанном в п. 5.1.7.

5.2. Сеть платежных каналов или «сеть мгновенных платежей» (Lightning)

Теперь мы можем обсудить «сеть мгновенных платежей» системы TON Payments, которая обеспечивает мгновенные денежные переводы между любыми двумя участвующими узлами.

5.2.1. Ограничения платежных каналов. Платежный канал полезен для сторон, которые собираются выполнять большое количество денежных переводов между своими учетными записями. Однако если нужно перевести средства только один или два раза конкретному получателю, создание платежного канала с ним будет нецелесообразно. Среди прочего, это повлечет за собой замораживание значительной суммы денег в платежном канале и в любом случае потребует как минимум двух транзакций блокчейна.

5.2.2. Сети платежных каналов или «сети мгновенных платежей». Сети платежных каналов преодолевают ограничения платежных каналов, обеспечивая денежные переводы по цепочкам платежных каналов. Если A хочет перевести деньги в E, ему не нужно устанавливать платежный канал с E. Достаточно иметь цепочку платежных каналов, связывающую A с E через несколько промежуточных узлов - скажем, четыре платежных канала: от A до B, от B до C, от C до D и от D до E.

5.2.3. Обзор сетей платежных каналов. Напомним, что сеть платежных каналов, известная также как «сеть мгновенных платежей», состоит из набора участвующих узлов, некоторые из которых установили между собой долговременные платежные каналы. Мы скоро увидим, что эти платежные каналы должны быть «умными» в смысле 5.1.7. Когда участвующий узел A хочет перевести деньги любому другому участвующему узлу E, он пытается найти путь, связывающий A с E внутри сети платежных каналов. Когда такой путь находится, выполняется «цепной перевод денег» по этому пути.

5.2.4. Цепные денежные переводы. Предположим, что существует цепочка платежных каналов от A до B, от B до C к D и от D к E. Предположим также, что A хочет передать x монет E.

Упрощенный подход состоял бы в том, чтобы передать x монет B по существующему платежному каналу и попросить его переслать деньги дальше C. Однако не очевидно, почему B просто не возьмет деньги себе. Следовательно, необходимо использовать более изощренный подход, не требующий от всех вовлеченных сторон доверять друг другу.

Этого можно добиться следующим образом. A генерирует большое случайное число u и вычисляет его хэш v = HASH(u). Затем он создает обещание выплатить x монет B, если в его платежном канале с B присутствует число u с хешем v (см. 5.1.7). Это обещание содержит v, но не u, а значение пока держится в секрете.

После этого B создает аналогичное обещание для C в своем платежном канале. Он не боится давать такое обещание, потому что знает о существовании подобного обещания, данного ему A. Если C когда-либо предоставит решение хеш-задачи по сбору x монет, обещанное B, то B немедленно представит это решение задачи A для сбора x монет у A.

Затем создаются аналогичные обещания от C к D и от D к E. Когда все обещания будут выполнены, A запускает передачу, сообщая решение u всем вовлеченным сторонам или только E.

Некоторые мелкие детали опущены в этом описании. Например, эти обещания должны иметь разное время истечения, а обещанная сумма может немного отличаться по цепочке (B может обещать C только x - e монет, где e - небольшая заранее согласованная плата за транзит). Мы пока игнорируем такие детали, потому что они не слишком актуальны для понимания того, как работают платежные каналы и как их можно реализовать в TON.

5.2.5. Виртуальные платежные каналы внутри цепочки платежных каналов. Теперь предположим, что A и E рассчитывают выполнять большое количество платежей друг другу. Они могут создать новый платежный канал между собой в блокчейне, но это все равно будет довольно дорого, потому что некоторые средства будут заблокированы в этом платежном канале. Другой вариант - использовать цепные денежные переводы, описанные в 5.2.4, для каждого платежа. Однако это потребует большой сетевой активности и большого количества транзакций в виртуальных блокчейнах всех задействованных платежных каналов.

Альтернативой является создание виртуального платежного канала внутри блокчейна, связывающего A и E в сети платежных каналов. Для этого A и E создают

(виртуальный) блокчейн для своих платежей, как если бы они собирались создать платежный канал в блокчейне. Однако вместо создания смарт-контракта платежного канала в блокчейне они просят все промежуточные платежные каналы - те, которые связывают A с B, B с C и т. д. - создать внутри них простые платежные каналы, привязанные к виртуальному блокчейну, созданному A и E (см. 5.1.10). Другими словами, теперь обещание перевести деньги в соответствии с окончательным расчетом между A и E существует внутри каждого промежуточного платежного канала.

Если виртуальный платежный канал является однонаправленным, такие обещания могут быть реализованы довольно легко, потому что окончательный дисбаланс 5 будет неположительным, поэтому простые платежные каналы могут быть созданы внутри промежуточных платежных каналов в том же порядке, как описано в 5.2.4. , Таким же образом можно установить срок их действия.

Если виртуальный платежный канал является двунаправленным, ситуация становится несколько сложнее. В этом случае следует разделить обещание передать б монет в соответствии с окончательным расчетом на два «полуобещания», как описано в 5.1.10: передать $б^{-} = \max(0, -б)$ монет в прямом направлении и допередать $б^{+} = \max(0, б)$ в обратном направлении. Эти полуобещания могут быть созданы в промежуточных платежных каналах независимо, одна цепочка полуобещаний в направлении от A к E, а другая цепочка - в противоположном направлении.

5.2.6. Поиск путей в сети мгновенных платежей. Пока не обсуждается один вопрос: как A и E найдут путь, соединяющий их в платежной сети? Если платежная сеть не слишком большая, можно использовать протокол типа OSPF: все узлы платежной сети создают оверлейную сеть (см. 3.3.17), а затем каждый узел передает всю доступную информацию (т. е. участвующие платежные каналы) своим соседям по протоколу сплетен. В итоге все узлы будут иметь полный список всех платежных каналов, участвующих в платежной сети, и смогут сами найти кратчайшие пути, например, применив версию алгоритма Дейкстры, измененную с учетом «возможностей» задействованных платежных каналов (т. е. максимальные суммы, которые могут быть переведены по ним). После того, как путь-кандидат будет найден, его можно исследовать с помощью специальной датаграммы ADNL, содержащей полный путь и запрашивающей у каждого промежуточного узла подтверждение существования рассматриваемого платежного канала, и пересылать эту датаграмму дальше в соответствии с путем. После этого можно построить цепочку и запустить протокол для цепных переводов (см. 5.2.4) или для создания виртуального платежного канала внутри цепочки платежных каналов (см. 5.2.5).

5.2.7. Оптимизация. Здесь можно сделать некоторые оптимизации. Например, только транзитные узлы сети мгновенных платежей должны участвовать в протоколе типа OSPF, описанном в 5.2.6. Два «листовых» узла, желающие подключиться через сеть мгновенных платежей, будут передавать друг другу списки транзитных узлов, к которым они подключены (т. е. с которыми они установили платежные каналы, участвующие в платежной сети). Затем можно проверить пути, соединяющие транзитные узлы из одного списка с транзитными узлами из другого списка, как описано выше в п. 5.2.6.

5.2.8. Вывод. Мы обрисовали в общих чертах, насколько блокчейн и сетевые технологии проекта TON соответствуют задаче создания TON Payments - платформы для мгновенных денежных переводов и микроплатежей вне сети. Эта платформа может быть чрезвычайно полезна для сервисов, находящихся в экосистеме TON, позволяя им легко собирать микроплатежи, когда и где это необходимо.

Вывод

Мы предложили масштабируемую многоблочную архитектуру, способную поддерживать широко популярную криптовалюту и децентрализованные приложения с удобными интерфейсами.

Для достижения необходимой масштабируемости мы предложили использовать блокчейн TON, «тесно-связанную» систему с несколькими блокчейнами (см. 2.8.14) с восходящим подходом к сегментированию (см. 2.8.12 и 2.1.2). Чтобы повысить потенциальную производительность, мы ввели механизм с двумя блокчейнами для замены недопустимых блоков (см. 2.1.17), а также мгновенную маршрутизацию в гиперкубе (Instant Hypercube Routing) для более быстрой связи между шардами (см. 2.4.20). Краткое сравнение блокчейна TON с существующими и предлагаемыми проектами блокчейнов (см. 2.8 и 2.9) подчеркивает преимущества этого подхода для систем, которые должны будут обрабатывать миллионы транзакций в секунду.

Сеть TON, описанная в главе 3, покрывает сетевые потребности предлагаемой мультиблокчейн-инфраструктуры. Этот сетевой компонент также может использоваться в сочетании с блокчейном для создания широкого спектра приложений и сервисов, что невозможно при использовании только блокчейна (см. 2.9.13). Эти сервисы, обсуждаемые в главе 4, включают TON DNS, службу для преобразования человекочитаемых идентификаторов объектов в их адреса; TON Storage, распределенную платформу для хранения произвольных файлов; TON Proxy, сервис для анонимизации доступа к сети и доступа к сервисам на базе TON; а также TON Payments (см. главу 5), платформу для мгновенных денежных переводов вне сети через экосистему TON, которую приложения могут использовать для микроплатежей. Инфраструктура TON позволяет создавать специализированные легкие клиентские кошельки и приложения для настольных компьютеров и смартфонов с «TON-браузером», которые позволят конечному пользователю работать по аналогии с работой в браузере (см. 4.3.23), совершать платежи в криптовалюте и взаимодействовать со смарт-контрактами и другими сервисами на платформе TON, доступной массовому пользователю.

Список литературы

[1] K. BIRMAN, Reliable Distributed Systems: Technologies, Web Services and Applications, Springer, 2005.
[2] V. BUTERIN, Ethereum: A next-generation smart contract and decentralized application platform, https : //github. com/ethereum/wiki/ wiki/White-Paper, 2013.
[3] M. BEN-OR, B. KELMER, T. RABIN, Asynchronous secure computations with optimal resilience, in Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, p. 183-192. ACM, 1994.
[4] M. CASTRO, B. LISKOV, ET AL., Practical byzantine fault tolerance, Proceedings of the Third Symposium on Operating Systems Design and Implementation (1999), p. 173-186, available at http://pmg.csail.mit. edu/papers/osdi99.pdf.
[5] EOS.IO, EOS.IO technical white paper, https://github.com/EOSIO/ Documentation/blob/master/TechnicalWhitePaper.md, 2017.
[6] D. GOLDSCHLAG, M. REED, P. SYVERSON, Onion Routing for Anonymous and Private Internet Connections, Communications of the ACM, 42, num. 2 (1999), http://www.onion-router.net/Publications/ CACM-1999.pdf.
[7] L. LAMPORT, R. SHOSTAK, M. PEASE, The byzantine generals problem, ACM Transactions on Programming Languages and Systems, 4/3 (1982), p. 382-401.
[8] S. LARIMER, The history of BitShares, https://docs.bitshares.org/ bitshares/history.html,2013.
[9] M. LUBY, A. SHOKROLLAHI, ET AL., RaptorQ forward error correction scheme for object delivery, IETF RFC 6330, https: //tools. ietf . org/ html/rfc6330, 2011.
[10] P. MAYMOUNKOV, D. MAZIERES, Kademlia: A peer-to-peer information system based on the XOR metric, in IPTPS '01 revised papers from the First International Workshop on Peer-to-Peer Systems, p. 53-65, available at http://pdos.csail.mit.edu/~petar/papers/ maymounkov-kademlia-lncs.pdf, 2002.
[11] A. MILLER, YU XIA, ET AL., The honey badger of BFT protocols, Cryptology e-print archive 2016/99, https://eprint.iacr.org/2016/ 199.pdf, 2016.
[12] S. NAKAMOTO, Bitcoin: A peer-to-peer electronic cash system, https: //bitcoin.org/bitcoin.pdf, 2008.
[13] S. PEYTON JONES, Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine, Journal of Functional Pro-gramming 2 (2), p. 127-202, 1992.
[14] A. SHOKROLLAHI, M. LUBY, Raptor Codes, IEEE Transactions on Information Theory 6, no. 3-4 (2006), p. 212-322.
[15] M. VAN STEEN, A. TANENBAUM, Distributed Systems, 3rd ed., 2017.
[16] THE UNIVALENT FOUNDATIONS PROGRAM, Homotopy Type Theory: Univalent Foundations of Mathematics, Institute for Advanced Study, 2013, available at https://homotopytypetheory.org/book.
[17] G. WOOD, PolkaDot: vision for a heterogeneous multi-chain framework, draft 1, https://github.com/w3f/polkadot-white-paper/raw/ master/PolkaDotPaper.pdf, 2016.

Монета TON Coin

Основной криптовалютой блокчейна TON и, в частности, его мастерчейна и основного воркчейна, является TON Coin. TON Coin используется для внесения депозитов, необходимых для того, чтобы стать валидатором; комиссии за транзакции, платежи за газ (т. е. сборы за обработку сообщений смарт-контрактов) и платежи за постоянное хранение также обычно собираются в TON Coin.

A.1. Подгруппы и соответствующая терминология. TON Coin подразделяется на один миллиард (10^9) единиц меньшего размера, называемых нанотонами, нтонами или просто нано. Все переводы и остатки на счетах выражаются в виде неотрицательных целых чисел, кратных нано. Другие единицы включают в себя:

- Нано, нтон или нанотона — наименьшая единица, равная 10^{-9} монетам TON.
- Микро или микротон равняется одной тысяче (10^3) нано.
- Милли — это один миллион (10^6) нано или одна тысячная часть (10^{-3}) монеты TON.
- Монета TON равняется одному миллиарду (10^9) нано.
- Килотон, или kTon, равняется одной тысяче (10^3) монет TON.
- Мегатон, или MTon, равняется одному миллиону (10^6) монет TON, или 10^{15} нано.
- Наконец, гигатон, или GTon, равняется одному миллиарду (10^9) монет TON, или 10^{18} нано.

Необходимость в более крупных единицах будет отсутствовать, поскольку первоначальный запас монет TON будет ограничен пятью миллиардами ($5 \cdot 10^9$) монет TON (то есть 5 гигатон).

A.2. Меньшие единицы для выражения цен на газ. Если возникнет необходимость в более мелких единицах, будут использоваться «спеки» размером от 2^{-16} нанотон. Например, цены на газ могут быть указаны в спеках. Однако фактическая плата, подлежащая уплате, вычисляемая как произведение цены на газ и количества потребленного газа, всегда будет округляться до ближайшего числа, кратного 2^{16}, и выражаться целым числом нано.

A.3. Первоначальное предложение, вознаграждение за майнинг и инфляция. Общее количество монет TON Coin изначально ограничено 5 гигатонами (то есть пятью миллиардами монет TON Coin или $5 \cdot 10^{18}$ нано).

Это предложение будет увеличиваться очень медленно по мере накопления вознаграждений валидаторам за майнинг новых блоков мастерчейна и шардчейна. Эти вознаграждения будут составлять примерно 20% (точное число может быть изменено в будущем) от доли валидатора в год при условии, что валидатор старательно выполняет свои обязанности, подписывает все блоки, никогда не переходит в автономный режим и никогда не подписывает недействительные блоки. Таким образом, валидаторы получат достаточно прибыли, чтобы инвестировать в лучшее и более быстрое оборудование, необходимое для обработки постоянно растущего количества транзакций пользователей.

Мы ожидаем, что не более 10%[35] от общего количества монет TON Coin, в среднем, будет привязано к стейкам валидаторов в любой момент времени. Это приведет к росту инфляции в 2% в год и, как следствие, удвоит общее предложение монет TON Coin (до десяти гигатон) через 35 лет.

[35] Максимальная общая сумма стейков валидаторов является настраиваемым параметром блокчейна, поэтому при необходимости это ограничение может быть применено в рамках протокола.

По сути, эта инфляция представляет собой платеж, производимый всеми членами сообщества валидаторам за поддержание системы в рабочем состоянии.
С другой стороны, если валидатор будет уличен в некорректном поведении, часть или весь его стейк будет снят в качестве наказания, а большая часть впоследствии будет «сожжена», что приведет к уменьшению общего количества монет TON Coin. Это приведет к дефляции. Меньшая часть штрафа может быть передана валидатору или «фишермену», который предоставил доказательство некорректного поведения виновного валидатора.