МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
имени Гумарбека Даукеева

Кафедра «Телекоммуникационные сети и системы»
Специальность: 6М071900 «Радиотехника, электроника и телекоммуникации»

ДОПУЩЕН К ЗАЩИТЕ
Зав. кафедрой
PhD, доцент Темырканова Э.К.
(ученая степень, звание, ФИО)

_____
(подпись)
«_____»_____2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
пояснительная записка

на тему: «Исследование технологий для построения платформы смарт-контрактов»

Магистрант: Шарапов Н.А. _____ группа МРЭТн-18-2
(Ф.И.О.) (подпись)
Руководитель: PhD, доцент кафедры ТКСиС_____ Семенякин Н.В.
(ученая степень, звание (подпись) (Ф.И.О.)

Рецензент_____ _____ _____
(ученая степень, звание) (подпись) (Ф.И.О.)

Консультант по ВТ PhD, доцент кафедры ТКСиС _____Семенякин Н.В.
(ученая степень, звание) (подпись) (Ф.И.О.)

Нормоконтроль: PhD, доцент кафедры ТКСиС_____ Семенякин Н.В.
(ученая степень, звание) (подпись) (Ф.И.О.)

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
имени Гумарбека Даукеева

Институт Космической Инженерии и Телекоммуникаций

Специальность: 6М071900 «Радиотехника, электроника и телекоммуникации»

Кафедра: «Телекоммуникационные сети и системы»

ЗАДАНИЕ
на выполнение магистерской диссертации

Магистранту Шарапову Никите Александровичу
(фамилия, имя, отчество)

Тема диссертации «Исследование технологий для построения платформы смарт-контрактов»

Утверждена Ученым советом университета №122 от «25» октября 2018 г.

Срок сдачи законченной диссертации «25»мая 2020г.

Цель исследования состоит в экспериментальном исследовании технологии блокчейн, используемой в построении платформ смарт-контрактов.

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации:

1. Ознакомление с технологией блокчейн и платформами смарт-контрактов
2. Принципы внедрения рассматриваемой платформы в сферу блокчейн
3. Техническое воплощение идеи на определенном языке программирования
4 Создание приложения на основе платформы смарт-контрактов

Перечень графического материала (с точным указанием обязательных чертежей)

Рисунок 1.1 - Примеры входных данных и значений
Рисунок 1.2 – Примеры транзакции
Рисунок 1.4 – Пример древа хэшей
Рисунок 1.10 – Структура смарт-контракта

Рисунок 2.5 – Схема процессов в платформе смарт-контрактов

Рисунок 3.5 – Время обработки транзакции

Рекомендуемая основная литература

1. Zheng Z. et al. An overview of blockchain technology: Architecture, consensus, and future trends //Big Data (BigData Congress), 2017 IEEE International Congress on. – IEEE, 2017. – p. 557-564.

2. Bakre A., Patil N., Gupta S. Implementing Decentralized Digital Identity using Blockchain. – 2017. – p. 3

Г Р А Ф И К

подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1. Информационный обзор согласно теме	05.10.2018	
2. Основные виды и типы распределенных систем, и смарт-контрактов (теоретическая часть)	14.01.2018	
3. Исследование возможности применения технологии блокчейн в платформах смарт-контрактов (исследовательская глава)	02.02.2018	
4. Процесс установки пакетов программы (расчетная часть)	18.10.2019	
5. Анализ полученных экспериментальных и расчетных данных	10.12.2019	

Дата выдачи задания_30 сентября 2018г._____

Заведующий кафедрой _____ (Темырканова Э.К.)
(подпись) (Ф.И.О.)

Научный
руководитель диссертации _____ (Семенякин Н.В.)
(подпись) (Ф.И.О.)

Задание принял к исполнению
магистрант _____ (Шарапов Н.А.)

(подпись) (Ф.И.О.)

Аңдатпа

Ақылды келісімшарт платформалары blockchain технологиясымен үйлесе отырып, қауіпсіздікті қамтамасыз ету және деректерді сенімді сақтау маңызды болып табылатын салаларда қолданудың әртүрлі аспектілері бар. Ақпарат ағыны үлкен жылдамдықпен өзгеріп отырады, мүмкін, әзірленген платформалардың мақсаттарының бірі - үшінші тараптардың екіжақты мәмілелерден шығарылуы. Бұл жағдайда үшінші тарапты алып тастаудың артықшылығы - ең осал деректердің құпиялылығы. Технологияны жетілдіру және оны нақты өмірде қолдану сұранысты арттырып, деректерді сақтаудың дәстүрлі әдістеріне қарсы тұра алады. Сондықтан ақылды келісімшарт платформаларын құруға арналған әртүрлі технологияларды талдауға назар аудару өте маңызды. Әр түрлі платформалар әртүрлі мүмкіндіктер мен идеялармен сипатталады, олар платформалардың жұмысына негізделеді.

Аннотация

Платформы смарт-контрактов в комбинации с технологией блокчейн обладают различными потенциальными аспектами применения в отраслях, в которых вопрос безопасности и надежного хранения данных имеет существенное значение. Потоки информации меняются с большой скоростью и, возможно, одной из целей разработанных платформ является исключение третьих сторон из двусторонних сделок. В данном случае, преимуществами исключения третьих сторон является конфиденциальность наиболее уязвимый данных. Усовершенствование технологий и внедрение в реальную жизнь позволяют повысить спрос и бросить вызов традиционным методам хранения данных. Поэтому крайне важно обратить внимание на анализ различных технологий для построения платформ смарт-контрактов. Различные платформы характеризуются разнообразными возможностями и идеями, на которых основывается работа платформ.

Abstract

Smart contract platforms paired with blockchain technology obtain various potential aspects of implementation in the industries in which the matter of security and safe storage has significant influence. The information is changing in rapid form and probably on the targets is to get rid of intermediate parties in the relation between two parties. The advantage can have some type possibly intangible such things as vulnerable info as well as the people required, might not be well-known to one another. The recent rapid growth and adoption into real life allow to increase the demand and to challenge traditional methods of data storage. Hence, it is extremely important to pay attention on the analysis of different technologies for the smart contract platforms construction. The various platforms indicate different features and the ideas on which the platforms are working.

Введение

В настоящее время технология смарт-контрактов изменяет традиционные понимание бизнес-процессов. Будучи встроенным в блокчейн, технология смарт-контрактов позволяет автоматически выполнять условия договора без вмешательства третьей стороны. В результате, смарт-контраты позволяют сократить расходы на администрирование и услуги, повысить эффективность бизнес-процессов и снизить риски. Несмотря на то, что смарт-контракты обещают стать движущей силой новой волны инноваций в бизнес-процессах, существует ряд проблем, которые необходимо решить. В данном документе представлено исследование технологии для построения платформы смарт-контрактов. В первой части диссертационной работы мы представим обзор архитектуры блоков и сравним специфические алгоритмы слияния, используемые в различных инструментах, использующих технологию блокчейн в платформах смарт-контрактов. Кроме того, будут кратко описаны технические проблемы и последние разработки.

Также мы представим проблемы, связанные с платформами смарт-контрактов, а также последние технические достижения в области блокчейн и смарт-контрактов. Мы также проведем сравнительный анализ платформ смарт-контрактов по нескольким параметрам.

Блокчейн предствляет собой программная система, позволяющая обрабатывать транзакции без необходимости предоставления доверия третьей стороне. В результате бизнес-процессы могут занимать меншее количество времени для осуществления, а также меньшее количество денежных средств для реализации бизнес-проектов. Кроме того, "иммунитет" технологии блокчейн обеспечивается распределенным доверием в сети, так как практически невозможно взломать любые транзакции, хранящиеся в блокчейн, и все исторические транзакции можно проверить и отследить. Технология Blockchain позволяет заключать смарт-контракты, которые были предложены в 1990-х годах Ником Сабо [1]. Положения смарт-контрактов, написанные с использованием языков программирования автоматически выполняться при соблюдении заранее оговоренных условий. Смарт-контракты, состоящие из транзакций, по сути, хранятся, тиражируются и обновляются в распределенных системах. В отличие от смарт-контрактов, обычные контракты применяют принцип доверия третьей стороны, что в свою очередь приводит в централизации всего процесса, а также к длительному времени обрабтки и дополнительным затратам.

С созданием технологии блокчейн (цепочки блоков) в 2008 году удалось применить на практике идею смарт-контрактов. Первой, кто поддержал технологию умных контрактов, была система «Биткоин», ведь благодаря сети биткоина появилась уникальная возможность автоматической и моментальной передачи активов от одной стороны к другой. Во время проведения сделки, ноды (компьютер, подключенный к биткоин, узел сети) проверяли заданные условия сделки, и если все сходилось, то подтверждали легитимность сделки.

Но система биткоин использовала эту технологию только в передаче денег. Безусловное преимущество смарт-контрактов – это автоматическое обеспечение безопасности выполнения сделки, что исключает привлечение третьих лиц к сделке. Получается, что смарт-контракт не только передает информацию, но и одновременно является гарантом выполнения условий сделки всеми сторонами – за несоблюдение условий сделки он может в автоматическом режиме наложить санкции на нарушителя, например, в виде штрафа.

Кроме того, в данной диссертационной работе представлен обзор конкретных приложений и примеров использующие платформы смарт-контрактов, а также рассмотрены вопросы использования системы блокчейн. При использовании технологии блокчейн возникают некоторые проблемы, такие как работоспособность с вредоносными пользователями, реализация контроля над системой и ограничения в использовании. Другими словами, технология блокчейн является важной концепцией и станет основой многих новых решений.

Целью диссертационной работы является разработка прикладной программы на платформе Hyperledger Sawtooth.

Поскольку цель проекта ясна, необходимо составить список задач проекта.

1. Ознакомление с технологией блокчейн и платформой смарт-контрактов;
2. Принципы внедрение рассматриваемой платформы в сферу блокчейн;
3. Техническое воплощение идеи на определенном языке программирования;
4. Создание приложения на основе платформы смарт-контратов.

1 Теоретическая часть

1.1 Структура блокчейн

Блокчейн программы могут показаться сложными, однако их можно быстро понять, изучив технологии каждого компонента в отдельности. На существенном уровне, денежные концепции (например, регистры) фактически используются наряду с компьютерными устройствами (связанные списки перспектив, распределенные сети), а также с зашифрованными примитивами (цифровые подписи, публичные/частные ключи), гибридными финансовыми концепции блокчейн.

На самом деле, важнейшим аспектом технологии блокчейн является использование криптографических хэш-функций для множества операций, таких как содержимое хэш-блоков.

Для определения метода хеширования, можно ввести/использовать любой тип файла (например, текст или изображение). Даже наименьшее изменение во входных данных (например, один бит) может привести к получению различных выходов, но в этом случае будет показан простой пример. Хэш-алгоритм разработан однонаправленно: вычислительно находит любой отрицательный вход любой предварительно заданной выходной карты.

Теоретически, технология применима к любому виду деятельности, где существует риск мошенничества, недоверия или ошибок при передаче данных.

Рассмотрим наиболее перспективные и эффективные способы использования блокировки.

Первый метод - сетевое администрирование. В данном случае, блокчейн играет роль неуязвимого хранилища ключей и списков пользователей, которые имеют право доступа к любым данным - серверам, терминалам, банкоматной сети.

Технология защищает от хакерских атак, серверных ошибок, взлома сетей и устраняет проблему «единого администратора».

Второй способ - хранение цифровых сертификатов. Криптография надежно защищает информацию от несанкционированного чтения, модификации, распространения. Так как сертификаты хранятся не на серверах, а в сети, невозможно получить к ним незаконный доступ, а также перехватить пароли пользователей [2].

Третий способ - доказательство права собственности. Подтверждение и передача прав собственности станет простой, практически мгновенной и безопасной операцией, если в этой сфере применить технологию блокчейн.

Человеку, имеющему доступ к своему блоку, достаточно внести новую информацию в блок, и информация о праве собственности будет распространяться по всей системе [2].

Четвертый способ — это создание системы DNS. С помощью блокчейн распределение имен в доменных сетях станет абсолютно безопасным.

Никакие DDoS-атаки больше не будут пугать ни простых граждан, ни финансовые или правительственные организации.

Пятый метод - идентификация и подтверждение прав доступа. Уже сейчас некоторые передовые компании используют блокирующее оборудование для идентификации своих клиентов, сотрудников и пользователей системы. Применение "цепочки блоков" значительно дешевле и эффективнее любых других методов защиты данных и подтверждения прав доступа.

В предлагаемом документе Bitcoin, Nakamoto предложил блокчейн в качестве решения проблемы двойных расходов путем создания криптографически защищенной не перестраиваемой цифрового реестра, которая хранится в одноранговом режиме всеми вовлеченными сторонами. Поскольку Bitcoin был нацелен на то, чтобы быть децентрализованной одноранговой цифровой наличностью, он должен избежать проблемы двойных расходов и рассмотреть возможность атаки на свою сеть. Сеть защищена от атаки с помощью консенсус-протокола, основанного на рабочем алгоритме, чтобы сделать атаку очень дорогой для выполнения [1].

Она преобразует доступные идентичные технологические инновации, которые лежат в основе Bitcoin, в peer-to-peer электронную наличность, блок-цепочку, которая может быть поставлена во множество дополнительных областей, таких как фискальные провайдеры, здравоохранение, промышленность и бизнес, IoT, а также авторизованных провайдеров. Точно, ряд попыток выполняется на нескольких объектах, таких как публичный нотариус, правильное управление, доказательство наличия документов, процесс аутентификации, пространство для хранения, защита от подделок, а также онлайн-приложения [2].

Однако если мы рассмотрим блокчейн как технологию, нам нужно будет упомянуть об основах, на которых возводится технология блокчейн. Под словом "основы" мы определяем платформы, которые учат пользователей разрабатывать структуру на основе технологии блок-цепочки под названием "умные контракты".

По словам американского ученого-вычислителя Ника Сабо, основная идея "умных" контрактов заключается в том, что многие виды договорных условий (такие как залог, разграничение прав собственности и т.д.) могут быть встроены в аппаратное и программное обеспечение, с которым мы имеем дело, таким образом, чтобы сделать нарушение контракта дорогостоящим для нарушителя. Канонический реальный пример, который мы могли бы считать примитивным предком умных контрактов - это скромный торговый автомат. В пределах ограниченного количества потенциальных потерь (сумма в кассе должна быть меньше, чем стоимость нарушения механизма), автомат принимает монеты, а через простой механизм, который делает новичка проблематичным в конструировании с конечными автоматами, дозировать изменение и продукт справедливо. Умные контракты выходят за рамки торгового автомата, предлагая встраивать контракты во все виды

собственности, которая ценна и контролируется с помощью цифровых средств.

Интеллектуальные контракты ссылаются на это имущество в динамичной, упреждающей форме и обеспечивают гораздо лучшее наблюдение и проверку в тех случаях, когда упреждающие меры должны быть недостаточными. А в тех случаях, когда торговый автомат, как и электронная почта, реализует асинхронный протокол между торговой компанией и клиентом, некоторые смарт-контракты предполагают несколько синхронных шагов между двумя или более сторонами [3].

1.1.1 Хэши. Алгоритмом хэширования, используемым во многих технологиях blockchain, является алгоритм Secure Hash Algorithm (SHA) с размером вывода 256 бит (SHA-256). Многие компьютеры поддерживают этот алгоритм в аппаратном обеспечении, что ускоряет его вычисление. Этот алгоритм имеет вывод 32 (8-битных) символов (показано ниже, в таблице 1, как шестнадцатеричная строка с 64 символами), что означает, что существует $2^{256} \approx 10^{77}$ возможных значений дайджеста. Алгоритм для SHA-256, как и других, указан в Федеральном стандарте обработки информации (FIPS) 180-4 [12]. Веб-сайт NIST Secure Hashing [12] содержит спецификации FIPS для всех алгоритмов хеширования, одобренных NIST.

Входные данные	SHA-256
1	0x6b86b27ff55dfkd95860wfkdsf0434849dsfj
2	0xd4735efgp634werkw32ekfos095skdfp2342
Hello, world!	0xdffd6021bb2bd5b0af6762wef9342fkfdos23

Рисунок 1.1 - Примеры входных данных и значений

Все данные в системе защищены. Цепь блоков надежно зашифрована, что открывает путь к получению достоверной и открытой информации. С помощью блоков можно узнать у кого сколько лежит денежных средств на счету, но невозможно узнать кому они принадлежат. Для подтверждения используется специальный ключ. От него зависит, будет ли пользователь идентифицирован системой или нет.

Безопасности и надежности технологии «блокчейн» построена на специальных ключах, с помощью которых упрощается процесс проверки правильности и правдивости информации. Сам криптографический ключ представляет собой группу букв и цифр, вычисление которых производится с использованием специально созданного алгоритма, называемого хэш-функцией [4]. В этом случае у пользователя есть только один ключ, обладающий двумя разными свойствами:

Во-первых, имея на руках ключ, вы не сможете узнать первичную (исходную) информацию;

Во-вторых, невозможно выбрать другой пакет данных, позволяющий создать тот же ключ.

Пользователь, у которого есть ключ, не может причинить вред системе или другому пользователю. Кроме того, даже при небольшой корректировке данных, ключ также будет изменен [3].

На основе рассматриваемых технологий могут быть запущены системы для проведения различных операций, независимо от того, знакомы ли участники сделки или нет, присутствуют ли во взаимоотношении участников элементы предварительной лояльности. Поскольку количество потенциальных входных значений и потенциальных выходов фактически представляет собой конечное количество возбуждаемых значений, хэш(x) = хэш (y) (т.е. хэш двух разных входов фактически передается одновременно). Тем не менее, практически для любого такого входа Y и X можно разработать тот же самый маршрут, что и для технологии «блокчейн» (в данном конкретном случае обе области имеют законные операции блокировки). Считается, что время использования другого алгоритма хэширования (SHA-256) на самом деле является сопротивлением столкновению, поскольку алгоритм был реализован для обнаружения столкновения между SHA-256, в среднем в 2^{128} случаях.

Технология «блокчейн» использует список и транзакции "отпечатков пальцев" (дайджест отпечатков пальцев). Любой человек с точно таким же списком транзакций способен дать точный отпечаток пальца. В случае изменения данных в учетной системе, данной изменение шифруется как с пометкой "изменения" в блоке, что значительно облегчает поиск незначительных изменений (т.е. присутствует возможность почти моментального отслеживания изменений).

1.1.2 Транзакции. Сделка представляет собой запись передачи активов между сторонами (цифровая валюта, единицы инвентаря и т. д.). В каждый момент времени, когда фактически вносится депозит или, возможно, снимается депозит, он фиксируется на имитируемом счете банка. В таблице показан пример, связанный с фиктивной сделкой. Каждый блок внутри "блокчейна" имеет множество транзакций. Для транзакции требуется не менее одного информационного поля, однако может быть и несколько информационных полей:

- сумма - общая сумма цифрового актива для передачи (общее количество цифровых активов, используемых для передачи).

- Входы - список цифровых активов, подлежащих передаче (их общая стоимость равна сумме). Обратите внимание, что каждый отдельный цифровой ресурс уникально идентифицирован. Однако активы не могут быть добавлены или удалены из существующих цифровых активов. Вместо этого цифровые активы можно разделить на несколько новых цифровых активов (каждый с меньшей стоимостью) или объединить, чтобы сформировать меньше новых цифровых активов (каждый из которых имеет соответственно большее значение) [12].

- Выходы. учетные записи, которые будут получателями цифровых активов. Каждый вывод указывает значение, которое должно быть передано

новому владельцу (владельцам), личность нового владельца и набор условий для идентификации нового пользователя. Если предоставленные цифровые активы более чем необходимы, дополнительные средства возвращаются отправителю (это механизм для «внесения изменений»)

- Идентификатор транзакции / Хэш - уникальный идентификатор для каждой транзакции. Некоторые блоксхемы используют идентификатор, а другие принимают хеш конкретной транзакции как уникальный идентификатор.

	Вход	Выход	Сумма	Итог
Транзакция ID: 0xa1b2c3	Аккаунт A	Аккаунт B	0.0321	
		Аккаунт C	2.5000	
				2.5321

Рисунок 1.2 – Пример транзакции

1.1.3 Криптография с асимметричным ключом. Основополагающей технологией, используемой технологиями blockchain, является криптография с асимметричным ключом (также называемая криптографией с открытым / закрытым ключом). Криптография с асимметричным ключом использует пару ключей: открытый ключ и закрытый ключ, которые математически связаны друг с другом. Открытый ключ можно обнародовать без снижения безопасности процесса, но закрытый ключ должен оставаться секретным, если данные должны сохранять криптографическую защиту [13].

Несмотря на то, что между двумя ключами существует связь, частный ключ не может быть эффективно определен на основе знания открытого ключа. Асимметричная криптография ключей использует разные ключи пары ключей для определенных функций, в зависимости от того, какая служба должна быть предоставлена. Например, при цифровом подписи данных криптографический алгоритм использует закрытый ключ для подписывания. Затем подпись может быть проверена с использованием соответствующего открытого ключа.

Использование криптографии с асимметричным ключом в системах Blockchain:

- Частные ключи используются для цифровой подписи транзакций.

- Открытые ключи используются для получения адресов, что позволяет использовать подход «один-ко-многим» для псевдонимов (одна пара открытого ключа может давать несколько адресов, в некоторых случаях несколько пар открытых ключей используются для создания нескольких адресов).

- Открытые ключи используются для проверки подписей, сгенерированных с помощью закрытых ключей.

- Криптография с асимметричным ключом обеспечивает возможность проверки того, что пользователь, передающий значение другому пользователю, имеет закрытый ключ, способный подписать значение.

1.1.4 Адреса и вывод адресов. Адрес пользователя представляет собой короткую буквенно-цифровую строку, полученную из открытого ключа пользователя с использованием хэш-функции, а также некоторые дополнительные данные (используемые для обнаружения ошибок). Адреса используются для отправки и получения цифровых активов [14]. Адреса короче открытых ключей и не являются секретными [14]. Чтобы сгенерировать адрес, необходимо использовать открытый ключ, хэширование и преобразование хэша в текст:

открытый ключ> хэш-функция> адрес

Пользователь может создать несколько пар закрытых/открытых ключей, для того, чтобы гарантировать анонимность. Решить проблему "идентичности" пользователя при блокировке публичного доступа, и, как правило, адрес будет конвертирован в QR-код для удобства использования. Когда блок выделяет цифровые активы, они реализуются путем назначения их по адресу. Для того чтобы разместить цифровые активы, пользователям необходимо иметь частный ключ, связанный с адресом. Цифровая подпись сделки, имеющей элемент частного ключа, может быть подтверждена открытым ключом.

Большинство клиентов, блокирующих систему, не обрабатывают свои собственные частные ключи. Кошелек способен хранить закрытые ключи, открытые ключи, а также связанные с ними адреса. Программное обеспечение электронного кошелька также может вычислить общие активы пользователя [14].

Частный ключ часто завершается с использованием безопасной произвольной функции, невозможно пересобрать M 454, то есть пользователь не может быть приватно подключен к потере любого имущества, связанного с блокировкой. В ситуации приватной цепочки злоумышленник не может иметь полный доступ к каждому из активов, управляемых закрытым ключом.

Безопасность частного ключа чрезвычайно важна, большинство клиентов используют специальное аппаратное обеспечение безопасности для его хранения, на самом деле блокировка представляет собой набор частных инженерных средств. Это открытие или, возможно, новостной канал, что означает, что частный ключ был обнаружен, чтобы использовать наличные деньги для перевода наличных на совершенно новый счет, но метод не был уничтожен. Обратите внимание, что данные не могут быть изменены должным образом и не могут быть отозваны, как только украденный личный элемент преступника, а также связанные с общественностью финансы были переведены на следующий аккаунт.

1.1.5 Реестр. Владелец счета — это совокупность операций. На протяжении всей истории бумажные документы использовались для наблюдения за обменом услугами и товарами. В последнее время документы сохранили большинство крупномасштабных централизованных баз данных в цифровом виде, а также внедрили лидера сообщества "надежных" третьих лиц

для представления интересов владельцев (т.е. владельцев депозитов третьих лиц).

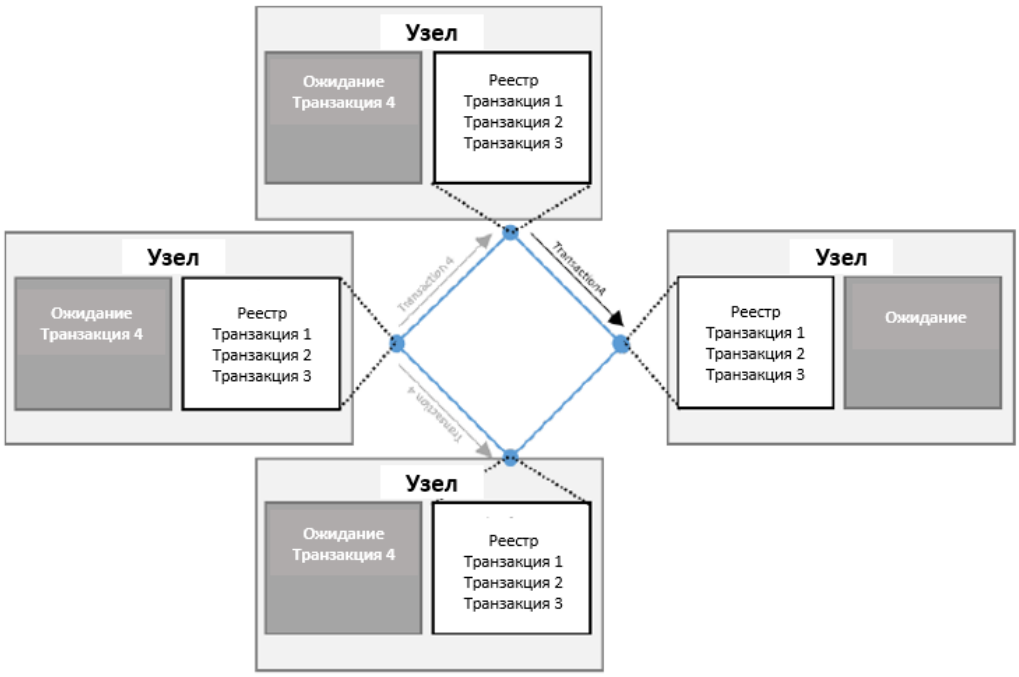

Рисунок 1.3 - Простая сеть, поддерживающая 486 копию реестра по узлам

Центральные реестры могут столкнуться со следующей проблемами:

- Они могут быть потеряны или уничтожены. Пользователи должны верить, что владелец правильно выполняет резервное копирование системы.
- Сделка недействительна; пользователь должен верить, что владелец проверяет каждую полученную транзакцию.
- Список транзакций не может быть завершен; пользователь считает, что владелец включает в себя все действительные транзакции.
- Данные могли измениться; пользователь должен верить, что владелец не изменит предыдущую транзакцию.

История транзакций, а также резервные данные которые принимают транзакцию, в наибольших интересах любого централизованного счета, включая все действительные транзакции не меняется.

Реестры, используемые технологией "блокчейн", может уменьшить эти проблемы за счет использования распределенных систем. Одной из причин этого является то, что блок-канал будет скопирован и распределен в каждом узле системы.

1.1.6 Блоки. Владельцы аккаунтов могут размещать свои учетные записи, отправляя пользователей на некоторые узлы, участвующие в канале блокировки. Представленная активность продвигается на другие узлы сети (но не для того, чтобы уничтожить транзакции) и не ждет распределенных транзакций в очереди или пуле транзакций, если только они не подключены к

узлу "добычи" в блокирующих ситуациях. Узлы добычи - это подмножества узлов блокировки, которые выпускают новые блоки. При публикации блока узла добычи [2] добавляются транзакции в блоке.

В одном блоке имеется набор верифицированных транзакций. Эмитент фонда определяет "действительность", убедившись, что каждый пароль подписан сделкой (как указано в "входном" значении фонда).

Он проверяет использование средств частных ключей для торговли, чтобы подписать имеющиеся средства. Другие добывающие узлы будут проверять действительность всех сделок в опубликованном блоке, и при наличии каких-либо незаконных сделок блок не будет принят. Реальное построение этого блока несколько сложнее (рис. 1.4).

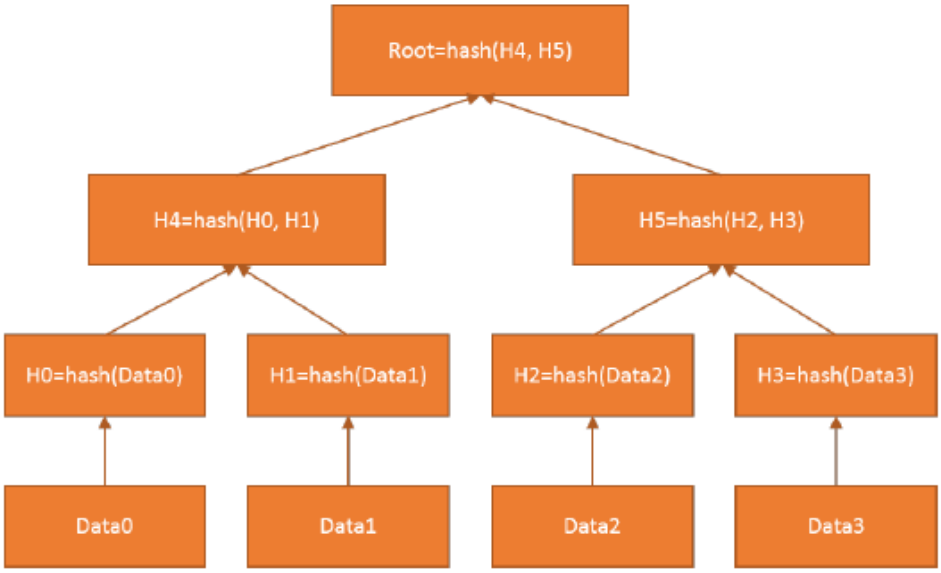

Рисунок 1.4 – Пример древа хэшей

Поле данных содержит следующее:

- текущий блок хэшей
- предыдущий блок хэшей
- оригинальное древо хэшей определяется ниже)
- временная метка
- размер блока
- декомпрессированные значения, которые являются некоторыми из инструментов, с которыми приходится работать добывающим узлам для решения проблем хэширования. Это дает им право публиковать блоки.
- список транзакций, включенных в этот блок

Пока существует необычный корень (хэш корня дерева хэшей), хэш-значение данных дерева хэшей комбинируется. Корень является эффективным механизмом суммирования транзакций в блоках и подтверждения транзакций в блоках. Эта комбинация гарантирует, что данные, отправленные в распределенной сети, являются действительными, так как любые изменения в

исходных данных будут обнаружены и отброшены. На рисунке 5 показано дерево хэшей:

- В нижней строке показана компрессия данных, т.е. данные транзакций в блочной цепи
- Из второй строки видно, что был создан хэш данных.
- Вторая строка собирает строгие данные, а третья - хэш-данные.

Наконец, верхняя строка представляет собой корневой хэш, который соответствует H4 и H5. Корневой хэш имеет все предыдущие комбинации и хэш-хэши [24].

1.1.7 Цепочные блоки. Добавляя голову первого блока, блоки складываются в каждый блок. Если ранее освобожденный блок изменился, то его хэш-значения будут отличаться. Далее, все последующие блоки будут иметь разные хэши, потому что в них будет добавлен хэш предыдущего блока. Это позволяет легко найти и отклонить любые изменения в ранее опубликованных блоках. На рисунке 1.5 представлена общую цепочка блоков.

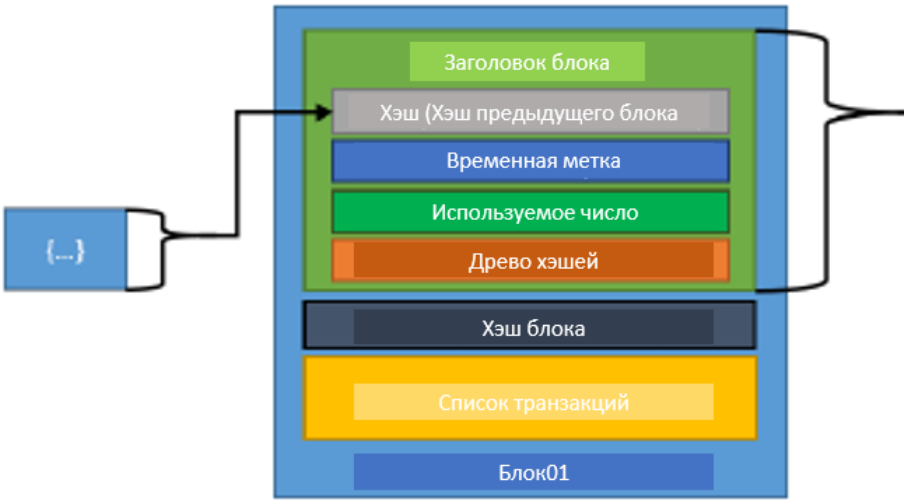

Рисунок 1.5 – Часть блока в цепи

1.2 Основные криптовалюты

1.2.1 Биткоин. Данная система представляет собой цифровую «монетарную» система известную как первопроходец в системе «блокчейн». Биткоин работает по принципу создания новый блоков, используя систему криптографических алгоритмов SHA-256, которая создает новые блоки каждые 10 минут.

Однако, возникают определенные сложности для поддержания 10-минутного интервала при создании новых блоков. Поэтому, было принято ввести дополнительную плату за обработку блоков, т.е. пользователи, заплатившие за обработку блоков, получают доступ в обработке быстрее, чем обычные пользователи. Также, стоимости обработки может варьироваться в зависимости от загруженности [28].

Добавление дополнительных транзакционных издержек означает добавление дополнительных кредитов в канал блокировки. Изначально, узел добычи получает 50 биткойн за каждый блок, и только некоторые из них являются блоками. Например, в июле 2016 года бонус за добычу блока составил 12,5 Bitcoin. С каждым соглашением о блочной цепочке эта прибыль будет снижена на 210 000 блоков (примерно за четыре года) и после 21 миллиарда блоков уже не будет выплачиваться прибыль от добычи биткоинов [13].

Добыча биткоинов будет продолжать находиться в этих местах, но бонус будет заработан за счёт полных транзакционных сборов. Последняя интересная техническая заметка заключается в том, что каждая транзакция в блокчейне имеет кусок кода, написанный в скрипте, который указывает на простую процедуру указания транзакции. Он не имеет циклов и сильно ограничен с точки зрения функциональности. Транзакции Bitcoin сегодня используют лишь малую часть возможностей скрипта. Фактически, в большинстве транзакций Bitcoin, только несколько шаблонных кодов используются для финансирования деятельности сторон.

Для работы Биткои требуется два основных устройства: блокировка и добыча.

Эти транзакции находятся в блоках, что является самым безопасным способом шифрования информации.

Блокчейн доступен в любое время для всех, и большинство сетей можно изменить, но все зависит от вычислительных мощностей устройств.

Рисунок1.6 – Пошаговая инструкция по применению «блокчейн»

1. Разработчики присоединяются в сделке.

2. Блок защищается шифрованием и интрегрируется в последующий блок.

3. Разработчики заработали вознаграждение, а также у них появляется возможность подгрузить свой блок обратно в цепочку.

Добыча на самом деле является процедурой, необходимой для того, чтобы продолжать быть в безопасности в каждом блоке, во время выпуска новых криптовалютных единиц. Данная процедура назвается "ценой блока". В случае с bitcoin, цена блока в настоящее время составляет 12,5 Bitcoin.

Роль разработчика на самом деле заключается в решении этого сложного алгоритма [3]. В связи с важной ролью их в сети, разработчики

имеют контроль над добычей, который важен над bitcoin, т.е. на данный момент добыча является прогрессирующим бизнесом [11].

Как только токены будут выпущены, они могут быть проданы/куплены (т.е. выпущены в цифровой обмен), а также могут быть сохранены в цифровом кошельке.

Что такое форк для биткойна?

Прерывание происходит, когда блокчейн фактически разделяется на 2 части, а затем производятся две отдельные записи.

Программа добычи групп вил на самом деле блокирует необходимые обновления приложения. Два основных типа - это жёсткие вилки и мягкие вилки.

Рисунок 1.7 – Софтфорк

Софтфорк: Софтфорк обычно включает в себя незначительные изменения кода. Это могут быть изменения, которые не меняют способ работы заданного блок-цепочки. Софтфорки часто ненавязчивы и могут сосуществовать вместе с версией без форков.

С помощью софтфорков "добытчики" могут переключаться на новый код. Поскольку никаких серьезных изменений нет, любые узлы, работающие под старым программным обеспечением, все равно должны быть в состоянии принимать блоки, сгенерированные новыми узлами - как бы то ни было, новые узлы будут отвергать блоки, сгенерированные старыми узлами. Если достаточное количество шахтеров переключится на вилочную версию, на старом коде не будет достаточного количества узлов для обработки транзакций. Новые узлы будут отвергать блоки, сгенерированные старыми узлами, заставляя их переключаться.

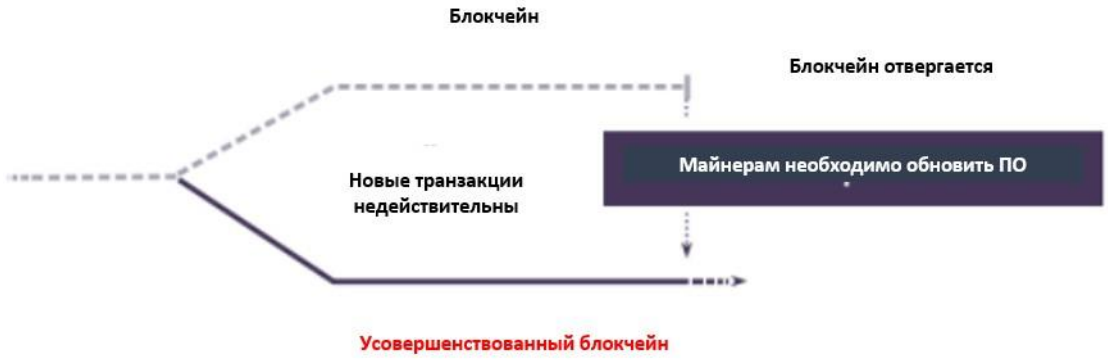

Рисунок 1.8 – Хардфорк

Хардфорки. Хардфорки часто вызывает гораздо больше проблем. Для генерации хардфорков обычно требуется ресурс всей сети, чтобы перейти на новый код. Если некоторые узлы должны будут выполнять неразборчивый код, то узлы могут генерировать данные различными способами и выпадать из синхронизации. Несмотря на то, что на одной и той же блок-схеме можно запускать как хардфорки, так и незавершенные узлы, это делается редко. Каждый набор узлов будет отвергать друг-друга, и могут возникнуть проблемы с дальнейшей реализацией.

Проблемы с хардфорками приводят к тому, что разработчики стараются делать так, чтобы каждый узел обновлялся до форка.

Если в коде происходит хардфорк, но некоторые разработчики не согласятся с изменением, то это может привести к путаннице. Сами шахтеры могут оказаться в центре ожесточенной битвы и быть вынуждены выбрать сторону. Если все станет действительно плохо, может создаться совершенно новый проект, основанный на оригинальном коде, с изменениями, внесенными хардфорками.

1.2.2 Эфир. Что такое эфир - мало кто знает, о данной криптовалюте несмотря на то, что криптовалюта довольно хорошо известна в виртуальной среде. Проект "эфир" на самом деле является платформой для создания децентрализованного онлайн-сервиса, который работает на основе блокчейн и "умных" контрактов. Хороший пример проекта системы этериума и новых крипто-валют был представлен Виталиком Бутериным в конце 2013 года. Но принцип, тем не менее, не является готовым проектом, так как он был запущен только 30 июля 2015 года.

Бутерин на самом деле является отцом-основателем Bitcoin Magazine, потому что интерес его к развитию своей криптовалюты на самом деле понятен. В результате, мысль пришла к воплощению в жизнь как единое, децентрализованное виртуальное устройство. Правда в том, что "эфир" на самом деле является более простым вариантом, используемым технологию "блокчейн" по сравнению с отсальными криптовалютами. И эта крипто-валюта может быть выгодна для добычи, в дополнение к биткойну, особенно

потому, что она намного дешевле, и процесс добычи проще. Что касается покупки криптовалюты, то это можно сделать на самых известных мировых биржах, что также говорит в пользу данной монеты [18]. Конечно, крипто-валюты, разрабатываемые после bitcoin, появляются довольно часто, но они не отличаются ни стабильностью, ни перспективностью. Когда появился эфир, многие обратили на него внимание, так как данная криптовалюта определенно отличается от технологии "блокчейн" - так называемого "умного контракта", который значительно защищает сделки и в то же время упрощает их [19].

Конечно, многих пользователей интересует максимальное количество монет "эфира". Их количество также ограничено, как и в биткойне, но в данном случае максимальное количество монет составляет 1.800.000.000 ETH.

Ethereum является открытой платформой, так как данная криптовалюта создавалась с целью привлечения интереса со стороны большего колличества пользователей. Например, такие крупные разработчики программного обеспечения как Microsoft, IBM и Acronis заинтересованы в системе.

Среди заинтересованных лиц есть также международная благотворительная организация UNICEF.

Таким образом, такая криптовалюта, как этерий, имеет большие шансы стать, если не второй криптовалютой после биткойна, то именно догнать её по цене.

Кстати, выясняя, что такое эфир, нельзя не обратить внимание на историю создания эфира, которую разработали российские специалисты. Как уже упоминалось, история создания началась в 2013 году, но нынешнее образование произошло чуть позже. Первым пунктом образования валюты стала публикация "Бутерина".

Далее Этереум описал в своей работе Гевин Вуд. Кроме того, в то же время эта конкретная техника была описана как следующая версия bitcoin и была известна как Bitcoin 2.0. Итак, уточняя, кто именно является основателем "эфира", гнеобходимо упомянуть имя Виталия Бутерина [21].

В 2014 году они начали привлекать ресурсы для развития краудфайдинга.

Экономическое развитие "эфира" началось после первого публичного представления Затем 31 591 биткойн (на данный момент 18 439 086 долларов) были обменяны на 60 102 216 эфиров. Это привлекло внимание банков.

Платформа блоков Ethereum была официально запущена 30 июля 2015 года.

14 марта 2016 года начался второй этап развития, так как Этериум вышел из первой альфа-модели Frontier. Правда, до этого разработчики не могли гарантировать полную безопасность сети. Очевидно, что новый вариант протокола не был особенно стабильным, однако гарантировал более или, возможно, менее безопасную работу.

Так что датой, когда появился «эфир», можно считать 2015 год, а в 2016 году эту крипто-валюту и систему стало возможно использовать без опасений,

что все рухнет в любой момент. Кроме того, другие крипто-валюты можно использовать в первую очередь для финансовых операций, в то время как эфир активно используется как средство обмена ресурсами и т.д., что позволяет ему завоевывать все больше и больше сторонников.

Рисунок 1.9 – Умный контракт «эфир»

Этериум поддерживается "умными" контрактами. Умный контракт - это необратимое действие, которое будет выполнено, как только для этого будут созданы соответствующие условия. Если простыми словами объяснить, что такое "умный" контракт, то "умный" контракт - это программа, которая одновременно является автоматической и автономной. В нее нельзя вмешиваться извне, поэтому она абсолютно безопасна.

Эта технология работает следующим образом:

- Создатель контракта ставит эфир на две позиции, планируя сделку.
- Умный контракт создается, в него уже невозможно внести изменения;
- Система проверяет, может ли клиент выполнить запланированное действие с помощью "эфира";
- Если действие осуществимо, система переводит трансляцию на счет, который был изначально указан в контракте;
- Сделка завершена;

Все довольно просто, а главное, эта процедура абсолютно прозрачна, что привлекает пользователей. Таким образом, система "умных" контрактов претендует на звание идеальной системы договорных отношений. Ее развитие не ограничено, так как постоянно совершенствуется, делая различные сделки максимально доступными для пользователей.

1.3 Разновидности платформ смарт-контрактов

Hyperledger - это набор проектов, направленных на создание распределенных реестров корпоративного класса с открытым исходным кодом. Проект Hyperledger поддерживается и размещается в Linux Foundation. Несмотря на то, что он размещается на Linux Foundation, каждый проект разрабатывается и поставляется из различных источников. В проекте Hyperledger есть много проектов, которые предоставляют платформу для решения каждой конкретной проблемы.

Некоторым из исполнительных органов управления Hyperledger является руководящий комитет. В нем работают более 10 должностных лиц, большинство из которых имеют десятилетний опыт работы с открытыми исходными кодами и имеют связи с большинством отраслей промышленности. Для своих членов Hyperledger не только предоставляет технические знания и программные данные, но и обеспечивает промышленность и разработчиков [15].

Это решение отделено от стратегических целей супер лыжника, и обычно исходит от богатого решения, разработанного в механизме валютных блоков, используемого в индустрии технологии Blockchain.

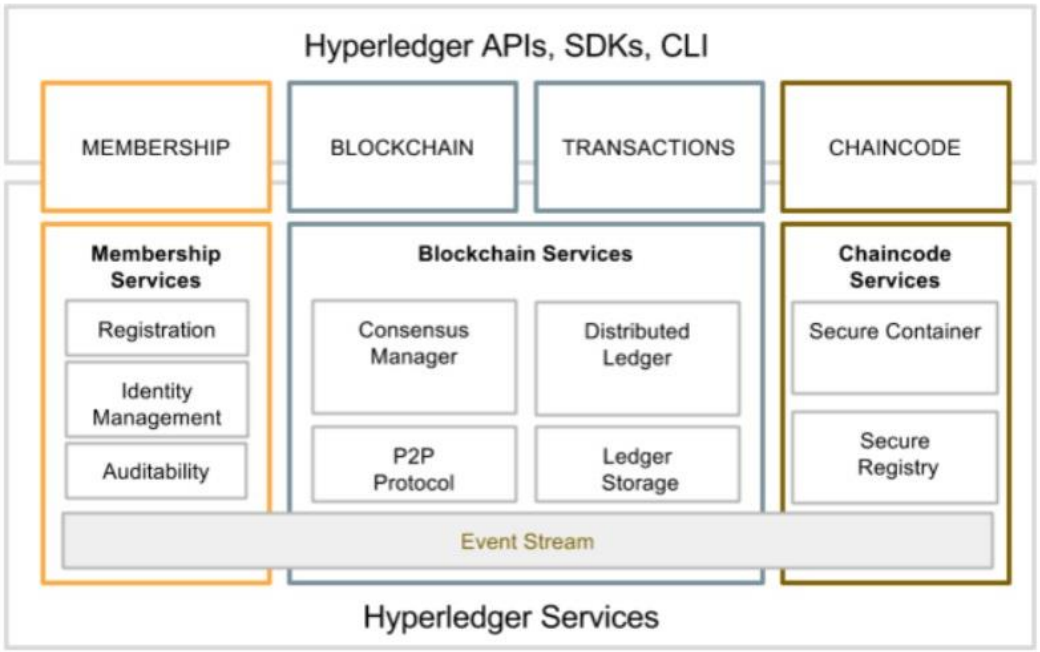

Рисунок 1.10 –Структура Hyperlegder

Основным требованием Hyperledger на самом деле является модульная комбинация. Различные сервисы должны быть подключены и пользователи смогут без усилий удалять и устанавливать модули в зависимости от вашего бизнес-портфолио.

Кроме того, ценность конфиденциальности и торговли "умными" контрактами также важна. Hyperledger имеет множество протоколов шифрования, а также алгоритмов, которые вы можете выбирать и выполнять без ущерба для производительности.

Также существует спрос на гибкую модель PKI (Public Key Infrastructure - Инфраструктура открытого ключа), которую можно было бы использовать для регулирования функций контроля доступа. Предполагается, что возможности и типы криптографии будут различаться в зависимости от потребностей пользователей.

Еще одним требованием является возможность аудита. Предполагается, что все идентификационные данные, связанные с ними транзакции и любые изменения будут и далее подвергаться аудиту.

Чтобы убедиться в том, что между различными задачами существует взаимосвязь, все больше программ должны иметь набор условий. Предполагается, что HyperPluger не только легковесен, но и дополнительно в степени кода, библиотек и API [10].

В настоящее время Hyperledger на самом деле является владельцем следующих элементов:

1. Hyperledger Sawtooth: Это модульный набор блок-цепей, разработанный Intel, совершенно новый алгоритм синтеза под названием Proof of Work (POW).

2. Hyperledger Iroha: Iroha на самом деле является проектом компании в Японии, которая может легко увеличить фреймворки в блокчейн.

3. Hyperledger Fabric: Основанный в 2015 году блокчейн-консорциум возглавляет некоммерческая организация Linux Foundation. Hyperledger поддерживает различные блокчейн и DLT-проекты, ориентированные на повышение производительности и надежности существующих систем. Hyperledger Fabric — один из самых популярных проектов консорциума. Fabric — это программный фреймворк для разработки приложений и специализированных бизнес-решений на основе блокчейна. Фреймворк имеет модульную архитектуру — различные компоненты можно использовать по принципу plug-and-play. Для выполнения смарт-контрактов Fabric использует технологию контейнеров, которая позволяет запускать их в изолированной среде.

4. Hyperledger Burrow: Это предприятие разработало утвержденный интеллектуальный контрактный механизм, который содержит конкретные детали цепи "блокчейн".

1.3.1 Hyperledger Fabric. Модульный, допустимый блок способен выполнять разумные контракты (называемые цепными кодами). Блок-цепочка Fabric заранее внесла свой вклад в активы отдела проектов Hyperledger Project Department, а также в компанию IBM.

На самом деле это не фреймворк "блокчейн", а модульная архитектура, поддерживающая усовершенствование решений на основе блоков. BlackBerrys может иметь plug-and-play плейлисты, включая параллельные и абонентские услуги, а также различные волоконно-оптические компоненты. Архитектура предназначена для составления плана интеграции самой деятельности прямо в персональную сеть блокировки, которая может управлять более чем 1000 транзакций в секунду [11].

Фреймворк разработан на языке программирования Go. Это сделано для того, чтобы сделать консорциумную цепь эффективной на различных уровнях лицензий. Цепной код Fabric в значительной степени опирается на интеллектуальную систему контрактов, которая является партнером в сети, запущенной Docker Container.

Структура меньше узлов общей структуры, а данные вычисляются параллельно, что делает Hyperledger Fabric лучше, чем публичная сеть. Кроме того, инфраструктура поддерживает конфиденциальные данные, поэтому,

если они будут найдены в публичных блоках, их члены будут более конфиденциальными [27].

Умные контракты в Fabric обозначены как цепной код. Сеть Hyperledger Fabric состоит из "одноранговых узлов", которые выполняют код блокировки, доступ к данным реестра, транзакции помощи, а также интерфейс приложения.

Фактически, Fabric создается для задач, для которых требуется распределенная инженерия регистров (DLT). Поддерживает цепной код в Go (Golang), Javascript и Java (через Hyperledger Composer) и, следовательно, возможно, гораздо более адаптируемый по сравнению с языком других умных контрактов.

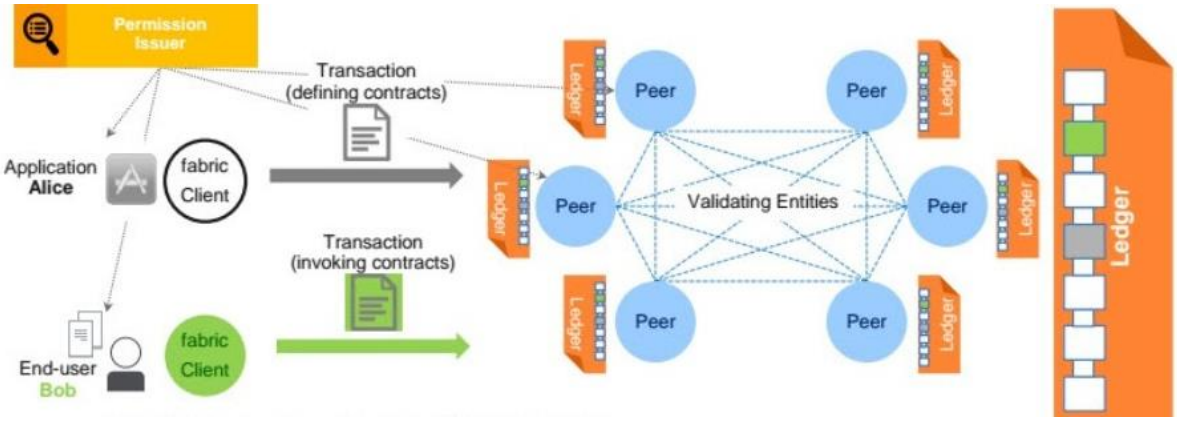

Рисунок 1.11 – Модель Hyperledger Fabric

В отсутствие базовой валюты, эта конкретная структура позволяет пользователям определять активы с клиентами и использовать их. Цепной код этой системы фактически аналогичен бизнес-логике атрибута "умной" контрактной структуры, так называемого состояния активов, обозначенного правилами как для чтения, так и для изменения [10].

Помимо структурного блока публичных криптографических валют, участники имеют возможность создавать отдельные каналы для своих активов, а значит, отделять и отделять транзакции от владельцев счетов. Таким образом, для участников в конкретном бизнес-модели будет установлен цепной код, необходимый для прочтения и изменения состояния актива. [11] По аналогии с хорошей чат-программой, программа блокировки структуры позволяет пользователю открывать и участвовать в 2 видах частных взаимодействий [28].

Основополагающий принцип:

- система разрешений, мощное управление идентификацией
- разграничение между законными дилерами и пользователями
- пользователи внедряют совершенно новый код (коды цепочки), а также распространяют и организовывают транзакции;
- проверяющий оценивает влияние подразделения и операции на новую версию владельца счета;

- общий объем операций «хэш» (глобальный статус).

- подключаемый протокол консенсуса; лицензии Fabric, а также конфиденциальность являются очень гибкими, поэтому более высокий уровень работы акционеров сети может обеспечить высокий уровень ликвидности.

Архитектура Hyperledger называется веб-платформой, и все участники имеют привычную идентичность. При рассмотрении разрешенных сетей следует учитывать, должны ли проблемы, используемые в вашем блочном цехе, соответствовать правилам защиты данных. В большинстве случаев, особенно в финансовом секторе и индустрии здравоохранения, они подчиняются законам о защите данных, которые принадлежат членам сети и должны знать, кто получает доступ к определенным данным.

Например, рассмотрим частные компании. По определению, частные акции не котируются на фондовых биржах, и их инвесторами обычно являются фирмы венчурного капитала, частные инвестиционные компании или ангелы-инвесторы. Участники этой сети должны быть идентифицированы и иметь доверие к капиталу, чтобы они могли участвовать в блок-цепочки [30].

Hyperledger Fabric построена на модульной архитектуре, которая разделяет обработку транзакций на три этапа: обработка распределенной логики и протоколов ("контрольная сумма"), последовательности обработки транзакций, а также аутентификацию и обязательства по обработке транзакций.

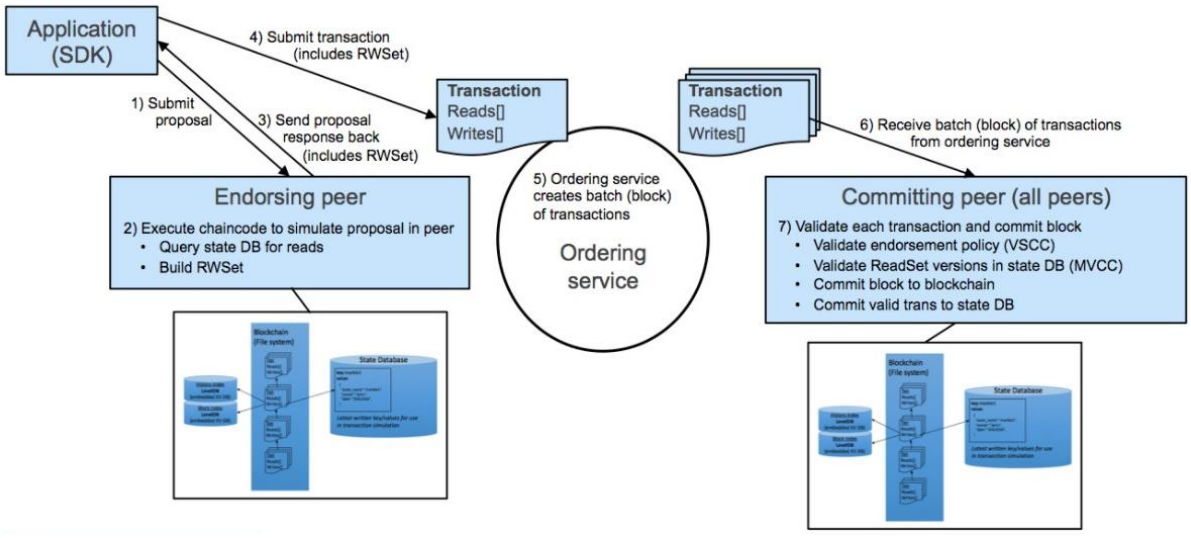

Рисунок 1.12 – Hyperledger Fabric

С левой стороны вышеуказанного рисунка:

- Предоставление контрагентами доказательств по сделке через заявку.

- В политике поддержки указано, сколько представителей должны зарегистрироваться в предложении. Поддержка активирует сканирование для

создания конфигураций чтения/записи в сетевых системах для имитации инициализации,

- Позже сторонники снова посылают рекомендации к предложениям, которые подписывают заявку.

- Программное обеспечение подает подписи и транзакции в системе подписки - Доставляет партии или, возможно, блоки капельницы и отправляет их вместе.

- Когда партнер по инвестициям получает пакет транзакций, за каждую транзакцию

- Он проверяет, был ли проверен набор для верификации/чтения, а также были ли найдены противоречащие друг другу операции. В случае прохождения каждого из этих проверок блоком должен быть аккаунт, а также статус каждой транзакции реально отображается в обновленной базе данных состояния.

В результате конкурентной борьбы бизнеса, контроль этой конфиденциальности закона о безопасности и индивидуальной информации определяет конфиденциальность конкретных реквизитов веществ, которые могут быть получены путем разбиения данных по каналу инфоблока. Каналы, поддерживаемые структурой Hyperledger, просто позволяют этим сторонам узнать, какую информацию они должны знать.

Hyperledger Fabric поддерживает использование одной или, возможно, гораздо большего количества сетей, каждая из которых управляет транзакциями, соглашениями и различными активами между различными наборами узлов сети. Ключевыми функциями Hyperledger Fabric на самом деле является регистрация обновлений и запросов, поиск по поисковым фразам, диапазонам запросов, а также комбинация основных запросов, запросы истории транзакций, которые на самом деле только читаются, транзакции обладают подписями каждого авторизованного партнера, пользователи проверяют транзакции с точки зрения политики применения и утверждения этой политики, реестр каналов состоит из блока конфигурации, который описывает политику, команду контрольного списка доступа, а также другую связанную информацию.

Каналы позволяют создавать криптографические материалы от нескольких центров сертификации.

Механизм консенсуса. Консенсус достигается тогда, когда порядок и результаты блочных транзакций совпадают с явным рассмотрением требований политики.

Владельцы учетных записей фактически являются экспресс-транспортными записями для блочных приложений. В результате каждой операции получается набор пар стоимости активов, составленных из счетов, обновленных или, возможно, удаленных. Неизменный фактический источник в V1.0 был введен в структуру файла-аналога, в котором уровни БД фактически встроены.

Одним из преимуществ модульной архитектуры Hyperledger Fabric является сетевым дизайнерам вставлять элементы на основе их приоритетов, что является преимуществом [10]. Одной из областей, где модульность является наиболее требовательной, на самом деле является "получение собственной персональной идентичности". В настоящее время некоторые сети, объединяющие несколько компаний, имеют управление идентификацией и важность для повторного использования по сравнению с восстановлением. Другие архитектуры, включают шифрование или консолидацию, некоторые из которых имеют свои собственные стандарты шифрования.

Помощь HSM (Hardware Security Module) на самом деле очень важна для защиты и управления цифровыми ключами, используемыми для аутентификации . Hyperledger Fabric обеспечивает улучшенную, а также неизменную защиту для производства, что очень важно, поддерживая другие ситуации и управление идентификацией, которые требуют повышенной защиты. Чтобы справиться с обстоятельствами управления идентификацией, HSM обеспечивает дополнительную защиту конфиденциальных данных и ключей.

Важность работы блочной сети, как правило, заключается в управлении целостностью информации и противодействии ее фальсификации. В блоке Bitcoin - все узлы соревнуются за соответствующее подтверждение транзакции, так как победитель получает награду в виде биткойнов [21].

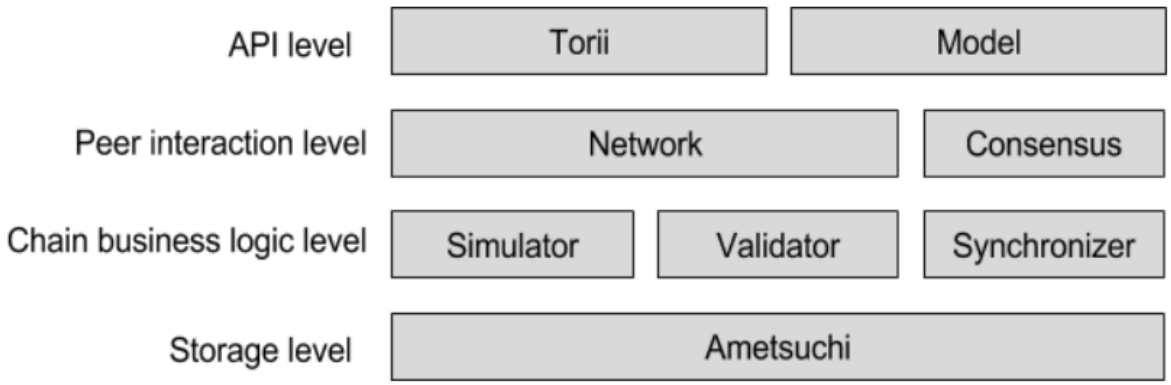

Рисунок 1.13 –Hyperledger Iroha

Эти части предлагают входные и выходные интерфейсы для клиентов Tori (Gate). Это единственный RPC сервер, который клиенты используют для общения с друзьями по сети. Если RPC вызов клиента на самом деле не блокируется, Torii на самом деле делает асинхронный сервер.

Коллаборационисты достигают консенсуса по содержимому веб-ссылки через консенсус.Механизм консенсуса, используемый Iroha, на самом деле представляет собой YAC (другой консенсус), который основан на реальном алгоритме отказоустойчивости, основанном на предубеждениях, который использует голосование по блочным хэшам.

Симулятор создает краткосрочную картину сделки, выполняя их против этого конкретного снимка и создавая законное предложение, которое состоит только из действительных сделок.

Класс верификатора проверяет правила ведения бизнеса, а также валидность (правильность формата) данной транзакции или, возможно, запроса. Hyperledger Iroha имеет 2 различных типа аутентификации:

- Аутентификация апатридов на самом деле является быстрым типом аутентификации, который определяет шаблоны транзакций, а также подписи;

- Проверка статуса является на самом деле легитимной проверкой, она определяет последователя, а также текущее глобальное представление о состоянии, и это самое последнее и наиболее практичное состояние в цепочке, чтобы выяснить, в случае, если требуемые правила и политики бизнеса действительно выполнимы; например, достаточно ли у вас денег для перевода одного счета?

Синхронизируйте новичков со структурой синхронизации или, возможно, помогите временно отключить коллег [26].

Сеть Hyperledger Iroha Network состоит из 3 основных частей: Клиенты на самом деле: "Транзакции" из " данных" относятся к статистике запросов, и они имеют право доступа/разрешения, чтобы указать на изменение поведения. Например, в транзакции пользователь может перевести деньги 3 лицам (3 независимых заказа). Тем не менее, в случае, если у него не хватает денег, вся транзакция будет отклонена.

Сотрудники на самом деле обрабатывают текущий статус вместе со своими общими счетами. Коллектив имеет единую сущность в сети и обладает доверием, идентификацией и адресом. Hyperledger была разработана для того, чтобы убедиться в том, что коллеги могут быть компьютерами или, возможно, вычислительными кластерами, что означает, что различные компьютерные системы на самом деле используются для хранения, аутентификации, индексирования и одноранговой логики.

Для того, чтобы купить информацию о транзакциях по заказу службы, на самом деле существуют альтернативы алгоритму, используемому службой подписки. Хорошим кандидатом считается Кафка.

Эти транзакции фактически защищены для криптографического хэширования и используют для подписания и верификации подписанную комбинацию публичного/частного ключа. Меркель Пелла подводит итог истории транзакций, поэтому любая попытка отредактировать и/или заменить предыдущую транзакцию может быть выполнена эффективно и результативно, поэтому все последующие транзакции, вероятно, могут быть возобновлены. Использование блок-цепочки остается на ранних стадиях, однако она основана на значительном понимании и реалистичных криптографических принципах.

Заглядывая вперед, можно сказать, что блок-цепь может быть совершенно новым приложением, которое может быть использовано для исправления новых проблем. Блокчейн, скорее всего, является одним из

крупнейших финансовых учреждений в мире. Им, возможно, придется изменить свою ценность или, возможно, полностью изменить свое поведение, чтобы иметь возможность сосредоточиться на создании достойной платформы оценки [12].

Компании, которым приходится регистрировать государственные вопросы с помощью специализированных технологий, таких как регистрация прав собственности на землю, рождений или брака, должны подумать о решении проблем, с которыми они сталкиваются. Блокчейн также предлагает возможность хранить, а также регистрировать записи о цепочке поставок. Имея такую особенность, блокчейн может в конечном итоге стать эффективным инструментом для разработки децентрализованных защищенных приложений и сетей к ним [17].

Смарт-контракты могут быть использованы в качестве инструмента для создания соглашений об обязательствах (контрактов) на блокчейне их выполнения. Однако, блокчейн - это система, которая не может быть исправлена после инициализации. Следовательно, вы не можете изменить какую-либо информацию после ее развертывания в блок-поиске.

Некоторые аспекты публичных блокчейн сетей делают их непригодными для массового принятия большим бизнесом. Это включает в себя:

- Долгосрочный процесс: каждый узел в блочной цепи Bitcoin необходим для проверки каждого движения команды;
- Большие данные: каждый узел в сети идет с историей трафика всей информации;
- Компании могут пострадать от серьезных ограничений на пространство хранения данных;
- Затраты времени: некоторые из публичных платформ блок-цепочки работают через алгоритм подтверждения работы (POW) [6] консенсуса, который требует включения процедуры "добычи". Под "добычей" мы понимаем процесс исследования хэша предыдущего блока с целью создания нового блока;
- Нет государственного контроля, который бы устанавливал некоторые стандарты или требования для общественного использования хэша.

Частные структуры блокчейн представляют собой следующие особенности:

- Частный доступ: В частные структуры цепочки могут входить только назначенные пользователи.
- Производительность: Предприятия, внедряющие частные блокировочные структуры, вряд ли могут обрабатывать большое количество транзакций в секунду (TPS). Таким образом, частные блокировки должны масштабироваться таким образом, чтобы они могли посылать общую производительность соответствующим образом.

- Безопасность: Частные блокировки могут следовать за публичными блокировочными цепями по некоторым стандартам хеширования для персонализации транзакций и повышения уровня аутентификации структуры.
- Централизация: Частные блокчейн структуры теряют функцию "децентрализованной" системы. Поскольку узлы в системе должны быть только доверенными частными блочными структурами, теряется одна из основных особенностей "децентрализованной" системы.
- Поддержка: Частная структура блокчейн должна быть поддерживаемой. Поддержка должна включать в себя администрирование, поиск и устранение неисправностей, исправление ошибок.
- Реализация: Частные структуры блокчейн должны иметь некий программный интерфейс (API) [7] для того, чтобы иметь мост между пользователями и самим бизнесом. Некоторые из платформ с таблицей сравнения приведены на рисунке 5 ниже.
- Регулирование: Каждое предприятие должно разработать методы, стандарты, процедуры и инструменты для работы с частными блокчейн структурами.

Таблица 5 – Сравнение платформ

Платформа	Консенсус	Частный / Публичный	Смарт контракты	Язык программирования
Bitcoin	PoW	Публичный	Да	Ivy, RSK, BitML
Ethereum	PoW and PoS [11]	Оба	Да	Solidity, Flint, SCILLA
Hyperledger [8]	PBFT [12]	Частный	Да	Go, Node.js, Java
Quorum [9]	Raft-based and Instanbul BFT	Частный	Да	Solidity
R3 Corda [10]	Flexible plug-in feature for consensus	Частный	Да	Kotlin

Corda - это распределенное программное обеспечение для ведения реестра, предназначенное для записи и обработки совместно используемых данных, таких как контракты, разработанное для реализации видения, содержащегося в настоящем документе. Компания Corda поддерживает "умные" контракты, соответствующие требованиям по дезактивации Clack, Bakshi, Braine; наш "умный" контракт - это соглашение, исполнение которого автоматизировано с помощью компьютерного кода, работающего с вводом и контролем, и права и обязанности которого, выраженные в юридической прозе, имеют юридическую силу.

Изначально Corda была разработана с учетом потребностей регулируемых финансовых учреждений, но оказалась гораздо более применимой. В значительной степени в ее основе лежат системы блокировки, но без выбора конструкции, которые делают традиционные блокировки непригодными для осуществления реальных деловых операций.

Нашим основным строительным элементом является «государственный объект», представляющий собой специфический экземпляр спецификационного договора, который может рассматриваться как представляющий собой реальный договор или раздел договора. В настоящем документе мы используем термины "соглашение" и "договор" на взаимозаменяемой основе. Это отличается от систем, в которых данные, по которым участники должны прийти к консенсусу, представляют собой состояние всей бухгалтерской книги или всей виртуальной машины. Corda предоставляет три основных инструмента для достижения глобального распределенного консенсуса:

- Умная логика контрактов, которая определяет ограничения, обеспечивающие действительность переходов между состояниями в соответствии с заранее согласованными правилами, описанными в коде контракта как часть CorDapps.

- Уникальные услуги, известные как нотариальные пулы для временного заказа транзакций и устранения конфликтных ситуаций.

- Уникальный компонент под названием flow framework, который упрощает процесс написания сложных многоступенчатых протоколов между многочисленными недоверяющими друг друга сторонами через Интернет.

В Corda обновления применяются с использованием транзакций, которые потребляют существующие государственные объекты и создают новые государственные объекты, тем самым создавая цепочки происхождения. Существует два аспекта консенсуса:

1. Действительность сделки: стороны могут достичь уверенности в том, что предлагаемая сделка по обновлению, аннулирующая выходные данные, является действительной, проверив, что соответствующий код договора, который должен быть детерминирован, успешно подтвержден и имеет все необходимые подписи; и что любые сделки, к которым относится данная сделка, также являются действительными.

2. Уникальность сделки: стороны могут достичь уверенности в том, что сделка, о которой идет речь, является уникальным потребителем во всех входных состояниях. Иными словами, не существует никакой другой сделки, в отношении которой мы ранее достигли консенсуса и которая потребляла бы одно и то же государство.

Стороны могут договориться о действительности сделки, самостоятельно запустив один и тот же код договора и логику валидации. Однако две действительные сделки могут существовать одновременно, и поэтому участникам необходимо определить, какая из них будет считаться

действительной. Для этого необходим заранее установленный наблюдатель, которым должна быть группа взаимно недоверчивых участников пула.
Corda имеет "подключаемые" уникальные услуги. Это позволяет повысить конфиденциальность, масштабируемость, географическую доступность, совместимость правовой системы5 и алгоритмическую подвижность. Один сервис может состоять из множества взаимно недоверчивых узлов, координируемых с помощью византийского алгоритма отказоустойчивости, или может быть очень простым, как одна машина.

Важно отметить, что эти сервисы уникальности необходимы только для того, чтобы удостовериться в том, что состояния, потребленные данной транзакцией, были ранее потреблены; от них не требуется удостоверять действительность самой транзакции, что является вопросом для сторон транзакции. Это означает, что услуги по обеспечению уникальности не требуются (и, в общем случае, не будут) для того, чтобы увидеть полное содержание любых транзакций, что значительно улучшает конфиденциальность и масштабируемость системы по сравнению с альтернативными конструкциями распределенных бухгалтерских книг и блок-сетей.

Блокчейн способен записывать каждый этап жизненного цикла продукта. На момент поставки они уже отгружены и отправлены в магазин, который был создан и куплен в конце клиента.

Могут появиться новые отрасли промышленности, такие как цифровые уведомления, которые позволяют кому-либо получить доступ к определённой информации, записывая хэш в биткойн. Существует множество потенциальных применений и возможностей для технологии блокчейн.

Технология блокчейн может нарушить работу многих отраслей. Чтобы не упустить возможности и избежать неожиданных сюрпризов, организации должны проверять на случай, если блок-цепь сможет им помочь.

Концепция инжиниринга блокировок настолько проста, насколько это возможно — это огромная общедоступная база данных, которая работает без централизованного контроля. В ситуации с биткоином, проверки транзакций на самом деле выполняются так называемыми участниками, которые проверяют подлинность выполненных шагов, а затем проводят блоки записей транзакций.

2 Практическая реализация

2.1 Цель и задачи

В рамках диссертационной работы должны быть рассмотрены следующие задачи:

1. Децентрализованное создание приложения блокчейн на базе программного фреймворка Hyperledger Sawtooth, способного хранить данные.
2. Демонстрация примера применения данной блокчейн платформы.

3. Регистрация и аутентификация основных данных в примере применения блокчейн платформы.

4. Создание многопользовательского режима передачи данных с помощью прикладного блока на базе платформы Hyperledger Sawtooth.

5. Описание архитектуры системы: компоненты, используемые для протоколирования данных, назначение компонентов прикладного блока, регистрация пользователей.

6. Расчет необходимых ресурсов для создания и технической поддержки данного блока приложений.

7. Оценка надежности и доступности этой системы по нескольким параметрам.

Для данного проекта необходимо предусмотреть, чтобы пользователь действовал исключительно от его имени. Анонимность в данном случае недопустима.

Основными задачами создания приложений на базе программного фреймворка Hyperledger Sawtooth являются:

1. Передача аутентичных данных, используемых для регистрации данных с невозможностью реализации последующих изменений в реестре.

2. Минимизация ошибок в процессе заполнения формы допинг-контроля.

3. Хранение базы данных децентрализовано с возможностью просмотра истории транзакций.

2.2 Hyperledger Sawtooth

Основываясь на данной диссертационной работе, мы заявили о создании программы блокчейн для Всемирного антидопингового агентства (WADA). Для этой цели мы выбрали платформу Hyperledger Sawtooth, основанную на операционной системе Ubuntu с открытым исходным кодом.

Hyperledger Sawtooth, предоставленная Intel, на самом деле представляет собой фреймворк блокчейн, который использует модульный клин для построения, развертывания и разработки рассылаемых реестров. Обработка реестров, созданная с помощью Hyperledger Sawtooth, способна использовать различные алгоритмы, основанные на измерениях. Он обеспечивает алгоритм доказательство истекшего времени (Proof of Elapsed Time (PoET)), который обеспечивает масштабируемость Bitcoin блокчейна без большого энергопотребления. PoET обеспечивает масштабируемое сообщество узлов валидаторов. Hyperledger Sawtooth на самом деле создается для адаптируемости, с помощью которого осуществляется каждое разрешенное и не разрешенное развертывание.

Кроме того, Sawtooth является чрезвычайно модульным. Эта специфическая модульность позволяет консорциумам и предприятиям делать выбор в пользу политики, которую они лучше всего готовы создать. Дизайн Sawtooth позволяет программам выбирать руководства по транзакциям, алгоритмы авторизации и мнения.

Принцип "первым пришел - первым обслужил". Внутри PoET каждому валидатору фактически предоставляется произвольный период ожидания.

"Валидатор с наименьшим периодом ожидания для определенного блока транзакций фактически избирается лидером". - sawtooth.hyperledger.org

Hyperledger Sawtooth на самом деле идеально подходит для данной ситуации в связи со способностью наблюдать за активами (в данном конкретном случае тунца). Соглашение о распределении статуса, новый алгоритм консенсуса, а также отделенная от консенсуса логика позволяют пользователю быть уверенной в том, что она покупает тунца, который был пойман на законных основаниях.

Компоненты Hyperledger Sawtooth:

1. Валидаторы транзакций.

2. Семейства транзакций состоят из "команды видов деятельности или, возможно, типов транзакций" (Дэн Миддлтон), которые на самом деле разрешены в общем реестре. Члены семейства транзакций состоят как из обработчиков транзакций (логика серверной стороны), так и из клиентов (для использования из мобильных приложений) или web.

3. Процессор транзакций передает бизнес-логику серверной стороны, которая работает на активах в сети.

4. Пакеты транзакций на самом деле представляют собой кластеры транзакций.

5. Сетевой уровень на самом деле отвечает за взаимодействие между валидаторами в сети Hyperledger Sawtooth, который включает в себя выполнение первого подключения, одноранговое обнаружение и обработку сообщений [10].

Глобальный экспресс имеет текущее состояние цепочки и регистр вызовов транзакций. Состояние для всех семейств транзакций фактически представлено на каждом валидаторе. Состояние фактически разбито на наименования, которые предоставляют авторам семейства транзакций свободу объяснять, делиться и повторно использовать информацию о состоянии во всем мире между процессорами транзакций [4].

Пакеты транзакций Hyperledger Sawtooth на самом деле представляют собой кластеры транзакций, которые все вместе посвящены состоянию, или, возможно, ни одна из транзакций вообще не зафиксирована. В результате пакеты транзакций часто описываются как атомарное устройство изменений, поскольку персонал транзакций на самом деле управляется как единое целое, и на самом деле они посвящены состоянию как единое целое. Каждая транзакция в Hyperledger Sawtooth на самом деле представляется в виде пакета. Пакеты могут включать лишь небольшую сумму в качестве одной транзакции.

Когда транзакция генерируется клиентом, пакет на самом деле передается валидатору (о чем мы подробнее поговорим в следующем разделе). Транзакции организованы в пакет в порядке их фиксации. Затем валидатор, в свою очередь, применяет каждую транзакцию в пределах пакета, что

приводит к сдвигу в процессе транзакции. Пакет фактически посвящен состоянию. Если одна транзакция внутри пакета действительно недействительна, то ни одна из транзакций внутри этого пакета не фиксируется.

Подводя итог, можно сказать, что пакетирование транзакций позволяет использовать команду транзакций в определенном порядке, разумеется, если какая-либо из транзакций недействительна, то ни одна из транзакций в этом пакете не применяется. Это мощный инструмент, который может использоваться многими предприятиями, так как он позволяет конечным пользователям лучше контролировать и повышать эффективность.

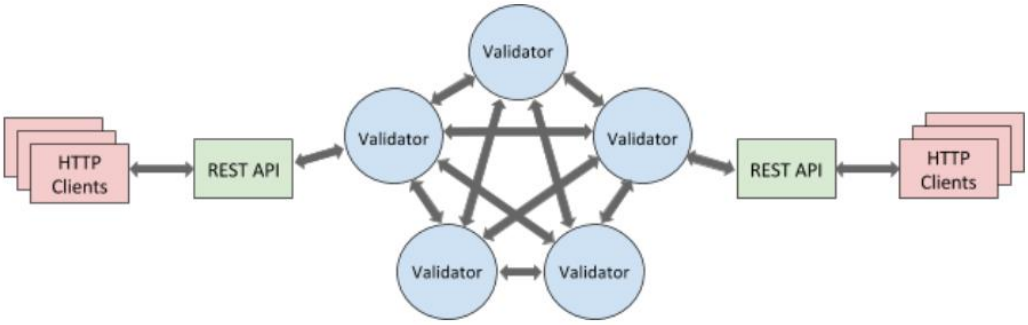

Рисунок 2.1 – Структура Hyperledger Sawtooth

В любой сети блокчейн для изменения глобального состояния необходимо создать и применить транзакцию. В Hyperledger Sawtooth валидаторы применяют блоки, которые вызывают изменение состояния. Если говорить точнее, валидаторы проверяют блоки транзакций и обеспечивают, чтобы транзакции приводили к изменениям состояния, которые последовательны для всех участников сети.

Для начала пользователь создает пакет транзакций и отправляет его валидатору через клиент и REST API. Затем валидатор проверяет пакет транзакций и применяет его, если он считается действительным, что приводит к изменению состояния. В рамках продемонстрированного нами сценария рыбак Сара создает пакет транзакций для записи информации о группе уловов тунца [11]. Затем валидатор применяет операции, и состояние обновляется, если присутствуют все соответствующие атрибуты: уникальный идентификационный номер, место и время улова, вес и кто поймал рыбу. Если какой-либо из этих элементов отсутствует, транзакции внутри партии не применяются, и состояние не обновляется.

Hyperledger - это модульная платформа для платформы Sawtooth, позволяющая создавать, назначать и продолжать распределенные реестры. Распределенные регистраторы предоставляют цифровые записи (например, владение активами) без централизованного управления или реализации. Использование технологии распределенного реестра на платформе Sawtooth

помогает найти ряд процессов и обеспечить надежность. Поэтому к любому объекту, закрепленному за человеком, могут быть добавлены датчики регистрации имущества, возможность отслеживания права собственности на объект, параметры собственности и телеметрии, такие как его местоположение, температура, влажность, трафик и т.д. Последний клиент может получить доступ к полному реестру данных и рассчитывать на их точность и полноту.

Внутри Hyperledger Sawtooth журнал ведет, а также расширяет, блокчейн для валидатора. Он отвечает за проверку потенциальных блоков, оценку легитимных блоков, чтобы выяснить, являются ли они соответствующими головками цепи, и производит совершенно новые блоки, чтобы удлинить цепь. Транзакционные партии поступают в журнал, именно там, где они оцениваются, валидируются и помещаются в блокчейн. Кроме того, журнал разрешает форки, которые возникают из-за разногласий по поводу того, кто совершает блок. Когда блоки действительно закончены, они доставляются в Chain Controller для проверки. Чтобы узнать, как именно компоненты журнала взаимодействуют друг с другом, ознакомьтесь с диаграммой на следующей странице [11].

2.3 Структура платформы

Консенсус в Hyperledger Sawtooth является модульным, что означает, что алгоритм консенсуса может быть легко изменен. Hyperledger Sawtooth предоставляет абстрактный интерфейс, который поддерживает как PBFT, так и алгоритмы в стиле Nakamoto-. Для реализации нового алгоритма консенсуса в Hyperledger Sawtooth необходимо реализовать отдельный интерфейс для: издателя блоков, верификатора блоков и разрешения форков.

- создатель блоков: создаёт новые блоки-кандидаты для расширения цепочки.

- верификатор блоков: проверяет, что блоки-кандидаты публикуются в соответствии с правилами консенсуса.

- распознаватель форков: выбирает какой форк использовать в качестве головки цепочки для алгоритмов консенсуса, в результате чего формируется форк.

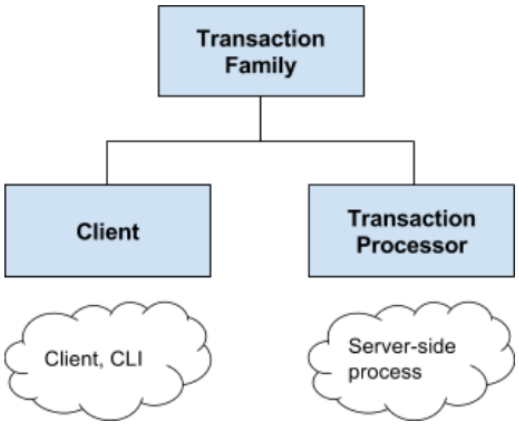

Рисунок 2.2 – Транзакции в Sawtooth

В Hyperledger Sawtooth модель данных, которая фиксирует состояние и язык транзакций, изменяющий состояние, реализуется с помощью семейств транзакций.

Семейство транзакций состоит из группы операций или типов транзакций, которые разрешены в общих записях. Это обеспечивает гибкость на уровне универсальности и риска, который существует в сети. Семейства транзакций часто называют "более безопасными" интеллектуальными контрактами, потому что в них указан предопределенный набор приемлемых шаблонов интеллектуальных контрактов, в отличие от программирования интеллектуальных контрактов с нуля.

Процессор транзакций обеспечивает бизнес-логику на стороне сервера, которая работает с активами в сети. Hyperledger Sawtooth поддерживает подключаемые процессоры транзакций, которые настраиваются в зависимости от конкретного приложения. Предприятия могут разрабатывать процессоры транзакций, которые делают именно то, что нужно их приложениям. Кроме того, операционные процессоры могут быть написаны на различных языках (Java, Python, C, C++, JavaScript и Go), что обеспечивает простоту использования и простоту при работе с активами.

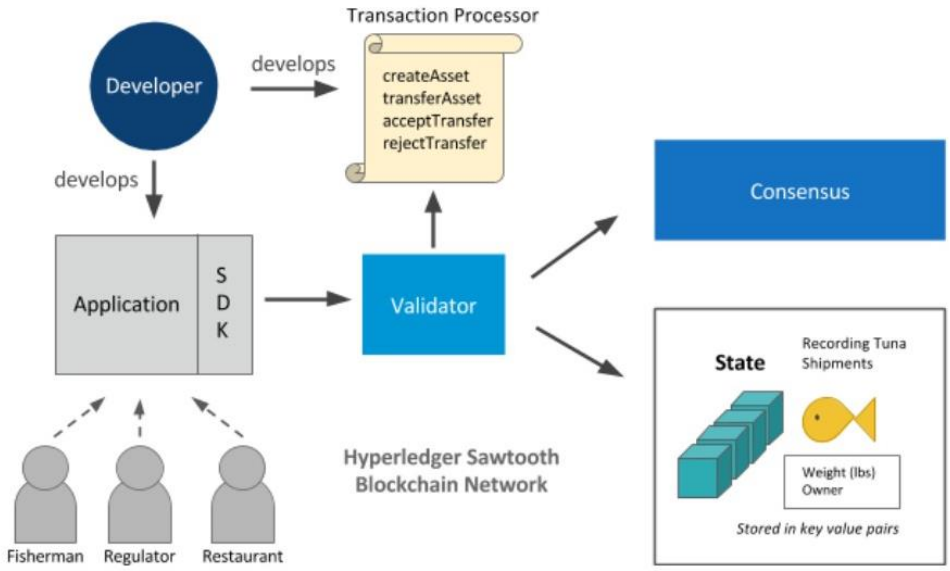

Рисунок 2.3 – Поток заявок

Hyperledger Sawtooth позволяет предприятиям безопасно обновлять и считываться реестры без участия центрального органа власти. Разработчики создают бизнес-логику приложения и процессора транзакций (интеллектуальный контракт).

Через клиентское приложение пользователи (рыбак, регулятор, ресторан) могут изменять состояние путем создания и применения транзакций. Посредством REST API клиентское приложение создает партию, содержащую одну сделку, и представляет ее валидатору. Валидатор оформляет сделку с помощью процессора транзакций, который вносит изменения в состояние (например, создает запись об улове тунца).

В Hyperledger Sawtooth пакеты представляют собой кластеры транзакций, которые стремятся к совместному состоянию. Если одна транзакция из пакета не может быть зафиксирована, ни одна из транзакций не фиксируется. В результате, пакеты транзакций часто описываются как атомарная единица изменения, так как группа транзакций рассматривается как единое целое и фиксируется в состояние как единое целое.

Каждая отдельная транзакция в Hyperledger Sawtooth отправляется в рамках пакета. Пакеты могут содержать всего одну транзакцию.

Когда транзакция создается клиентом, пакет передается валидатору (о чем мы более подробно поговорим в следующем разделе). Транзакции организуются в пакет в порядке их фиксации.

Затем валидатор, в свою очередь, применяет каждую транзакцию в рамках пакета, что приводит к изменению глобального состояния. Пакет передается государству. Если одна транзакция в рамках пакета недействительна, то ни одна из транзакций в рамках этого пакета не фиксируется.

Таким образом, пакетирование транзакций позволяет применить группу транзакций в определенном порядке, и если какая-либо транзакция окажется

недействительной, то ни одна из транзакций не будет применена. Это мощный инструмент, который может использоваться многими решениями для предприятий, поскольку он обеспечивает большую эффективность и контроль для конечных пользователей.

2.4 Структура сети

Sawtooth на самом деле является модульной платформой, предоставленной Intel в апреле 2016 года, с несколькими основными инновационными разработками. Основное внимание сосредоточено на адаптируемом использовании в различных частях бизнеса, с запуском других семейств, основанных на консенсусе и транзакциях.

Сделки на самом деле отделены от суммы консенсуса, используя для этой цели совершенно новую идею, известную как семейство сделок. Вместо транзакций, которые отдельно связаны с цепочкой, используются семейства транзакций, что обеспечивает повышенную гибкость, а также безграничную компоновку бизнес-логики. Операции придерживаются схем, а также структур, определенных в семействах транзакций.

Корпорация Intel представила совершенно новый алгоритм консенсуса PoET, свидетельствующий о событиях, имевших место в прошлом. Протокол произвольно выбирает победителя, хотя денежного стимула нет, как в ситуации с добычей полезных ископаемых.

Распределенные реестры представляют собой цифровую историю (например, владение активами), которая ведется без основного органа власти.

Hyperledger на самом деле является модульной платформой для производства и управления сетевыми лазерными приложениями. Разработка программных приложений на самом деле облегчается путем отделения первичного процесса от количества в программном объеме. Разработчики приложений должны знать внешний вид основного ядра процесса, работать с собственным языком программирования по своему выбору и информировать о нем бизнес-логику приложения [15].

Валидаторы Sawtooth проверяют транзакции. Валидаторы фактически несут ответственность за объединение партий транзакций в блоки, распределение их в реестре и утверждение легитимных блоков на основе алгоритма мнения сети.

Программы Sawtooth, фактически, распределяются в виде интеллектуальных протоколов, которые отделены от основного фреймворка. Программное обеспечение для транзакций описывает поведение, а также действия субъекта, который предлагает средства для действий, совершаемых в семействе. В функциональных приложениях фактически предоставляются два процессора (логика серверной стороны) и один или даже несколько клиентов (сеть, тип команды CLI и использование мобильных программ).

Большинство приложений из нескольких узлов процессора транзакций выполняют определенную ситуацию использования или, может быть, пара одинаковых сетей в каждой транзакции.

Пакеты фактически сгруппированы вместе. В пакетах, ориентированных на состояние, они могут быть одной транзакцией или даже несколькокими связанными транзакциями.

Сетевой уровень, отвечающий за взаимодействие между валидаторами в сети Hyperledger Sawtooth, который включает выполнение первого подключения, поиск и контроль записей.

Worldwide express имеет текущее состояние цепочки и книги вызовов транзакций. Состояние всех приложений, работающих в сети, отображается на каждом узле. Система аутентификации каждой транзакции гарантирует, что транзакции приводят к одинаковым переходам статуса. [18]. Состояние конкретной программы разбито на пространства имен приложений, которые описывают писателя приложения, делят между собой состояние процессора транзакций по всему миру, а также дают возможность вновь обрести свободу ношения.

Свидетельство об истекшем времени (POET) было единогласно согласовано в Hyperledger Sawtooth, что сводит на нет применение алгоритмов, основанных на времени, для больших распределенных сетей с алгоритмами в качестве доказательства работы.

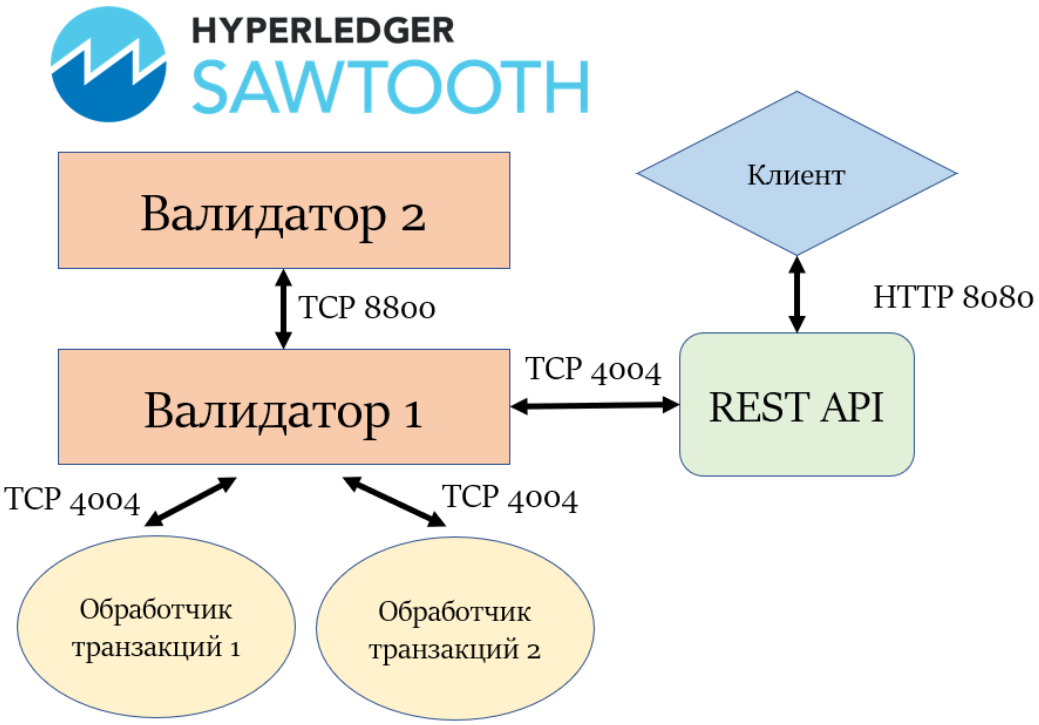

Рисунок 2.4 – Структура Hyperledger Sawtooth

Валидаторы, как было сказано выше, добавляют комбинации в блоки и сверяют их с реестром. Кроме того, в соответствии с данный консенсусом они добавляются в блок-цепочку. В этом случае показывается структура только с валидатором 1.

Транзакционные процессоры представляют собой бизнес-логику из серверной части.

REST API определяет набор функций, на которые разработчики могут делать запросы и получать ответы. Взаимодействие происходит по протоколу HTTP. Преимуществом такого подхода является широкое распространение протокола HTTP, поэтому REST API можно использовать практически с любыми языка программирования.

С помощью семейства традиционных пользователей IntegerKey пользователи могут добавлять, увеличивать и уменьшать значение архивных записей в государственном словаре.

Запросы на проведение транзакций в семействе IntegerKey определяются следующими значениями:

- описательные торговые значения;
- название записи или имя, которое необходимо изменить;
- настройки записи или измененные значения;

Глагол 'set' используется для создания новых записей. Начальное значение записи устанавливается в значение, указанное в запросе транзакции. В статусном словаре для изменения значения существующей записи используются "inc" и "dec".

Поскольку мы уже рассматривали компоненты основной архитектуры Hyperledger Sawtooth, необходимо перейти к методам подключения этих компонентов.

Валидаторы соединяются между собой по протоколу TCP 8800, а также соединяются с другими компонентами Hyperledger Sawtooth по протоколу TCP через порт 4004.

TCP - это протокол более высокого уровня, позволяющий приложениям, запущенным на различных хост-компьютерах, совместно использовать потоки данных. TCP разделяет потоки данных на цепочки, которые называются TCP сегментами, и передает их по IP. В большинстве случаев каждый TCP-сегмент передается в одной IP-датаграмме. Однако при необходимости TCP разделяет сегменты на несколько IP датаграмм, которые вписываются в физические рамки данных, используемые для передачи информации между компьютерами в сети. Поскольку IP не гарантирует, что датаграммы будут получены в той же последовательности, в которой они были отправлены, TCP собирает TCP сегменты на другом конце маршрута, чтобы сформировать непрерывный поток данных.

HTTP (HyperText Transfer Protocol - Протокол передачи гипертекста) является протоколом OSI седьмого уровня для передачи данных, который основан на клиент-серверной архитектуре. Изначально протокол HTTP разрабатывался для передачи HTML документов между сервером и клиентом с помощью HTTP сообщений. Поскольку протокол основан на клиент-серверном взаимодействии, предполагается, что есть клиент, который выполняет HTTP-запросы, и есть HTTP-сервер, который обрабатывает эти запросы и дает клиенту HTTP-ответы. Все ответы сервера содержат коды состояния, и все запросы клиента имеют HTTP-методы. Эта серия публикаций

поможет нам понять, как клиент и сервер взаимодействуют по протоколу HTTP.

В реальности существует ряд конфигураций и архитектур кэшей, а также прокси-серверов, которые в настоящее время разрабатываются или даже развертываются во всемирной паутине; эти методы включают национальные иерархии прокси-кэшей, которые защищают полосу пропускания межокеанских каналов, устройства, которые распространяют содержимое кэша, предприятия, которые распространяют подмножества кэшированной информации на CD ROM и т.д. Методы HTTP используются в интрасетах компаний с высокоскоростными линиями переписки, а также для доступа к КПК с нестабильной связью и маломощными линиями. Замысел HTTP / 1.1 фактически заключается в поддержании ряда конфигураций, уже сделанных с запуском более ранних типов процесса.

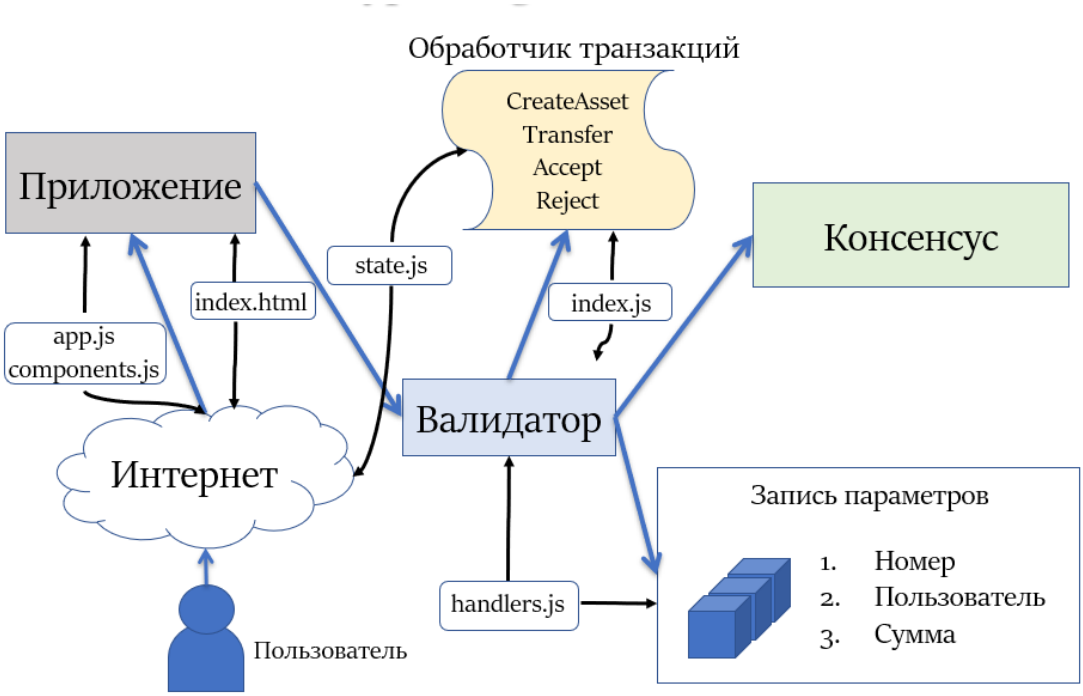

Рисунок 2.5 – Схема процессов в платформе смарт-контрактов

Давайте рассмотрим структуру Hyperledger Sawtooth, чтобы узнать, какие программы взаимосвязаны с компонентами нашего фреймворка.

В Hyperledger Sawtooth держатель учетной записи будет контролировать систему, но история транзакций не изменится. Приложение обычно состоит из двух частей.

Клиентское приложение используется для блокировки транзакций, которые обычно отправляются через API Sawtooth.

Клиентом может быть интерфейс командной строки, веб-страница, мобильное приложение, сенсор IoT или любой другой тип интерфейса, способный отправлять HTTP-запросы.

Процессор транзакций применяет код бизнес-логики, связывается с клиентом для аутентификации и отправляет полученную клиентом транзакцию в процессор транзакций.

Клиентская часть содержится из программ, которые используются для связи нашей программы с конечным пользователем. Эти программы состоят из app.js, components.js, state.js. В данной папке содержится код клиента на основе браузера на основе блокчейн, написанный на Javascript. После запуска композиционного файла docker, доступ к клиенту осуществляется по адресу 'localhost: 8000'.

Папка, содержащая код клиента на Javascript, который соединяет бэкенд процессора транзакций с клиентом (в основном в state.js). На следующем рисунке показана структура файла state.js. Этот файл создает новую пару ключей для приложения, сохраняет пары ключей в локальном хранилище.

Для валидатора он получает текущее состояние блокировки Sawtooth из валидатора и отправляет подписанную транзакцию валидатору.

```
const $ = require('jquery')
const {
  signer,
  BatchEncoder,
  TransactionEncoder
} = require('sawtooth-sdk/client')

// Config variables
const KEY_NAME = 'transfer-chain.keys'
const API_URL = 'http://localhost:8080'

const FAMILY = 'transfer-chain'
const VERSION = '0.0'
const PREFIX = '19d832'

// Fetch key-pairs from localStorage
const getKeys = () => {
  const storedKeys = localStorage.getItem(KEY_NAME)
  if (!storedKeys) return []

  return storedKeys.split(';').map((pair) => {
    const separated = pair.split(',')
    return {
      public: separated[0],
      private: separated[1]
    }
  })
}

// Create new key-pair
const makeKeyPair = () => {
  const privateKey = signer.makePrivateKey()
  return {
    public: signer.getPublicKey(privateKey),
    private: privateKey
  }
}
```

Рисунок 2.6 – Настройка переменных и создание новых пар ключей

Components.js и app.js управляют html-элементами пользовательского интерфейса.

В частности, components.js добавляет выбранную опцию. После этого добавляется новая строка таблицы с количеством ячеек, которые необходимо реализовать в приложении out blockchain. Также добавляются кнопки accept/reject. С их помощью информация из блокчейн может быть передана между пользователями.

```
const $ = require('jquery')

// Add select option which may be set to selected
const addOption = (parent, value, selected = false) => {
  const selectTag = selected ? ' selected' : ''
  $(parent).append(`<option value="${value}"${selectTag}>${value}</
option>`)
}

// Add a new table row with any number of cells
const addRow = (parent, ...cells) => {
  const tds = cells.map(cell => `<td>${cell}</td>`).join('')
  $(parent).append(`<tr>${tds}</tr>`)
}

// Add div with accept/reject buttons
const addAction = (parent, label, action) => {
  $(parent).append(`<div>
  <span>${label}</span>
  <input class="accept btn btn-primary" type="button" value="Accept">
  <input class="reject btn btn-caution" type="button" value="Reject">
</div>`)
}

module.exports = {
  addOption,
  addRow,
  addAction
}
```

Рисунок 2.7 – Добавление строк в таблицы

Программа, которая будет продемонстрирована ниже app.js, содержит код для приложения с блокчейн, написанный на Javascript. Эта программа выбирает пользователей, которые были добавлены в приложение, а затем создает активы, а также она может передавать и принимать их.

```
// Select User
$('[name="keySelect"]').on('change', function () {
  if (this.value === 'new') {
    app.user = makeKeyPair()
    app.keys.push(app.user)
    saveKeys(app.keys)
    addOption(this, app.user.public, true)
    addOption('[name="transferSelect"]', app.user.public)
  } else if (this.value === 'none') {
    app.user = null
  } else {
    app.user = app.keys.find(key => key.public === this.value)
    app.refresh()
  }
})

// Create Asset
$('#createSubmit').on('click', function () {
  var asset = $('#createName').val()
  asset = asset.concat(" ",$('#boxNo').val()" ",$('#mission0rder').val()" ",
$('#timeSealed').val()" ");
console.log(asset);
  const boxNo = $('#boxNo').val()
  const missionOrder = $('#mission0rder').val()
  const timeSealed = $('#timeSealed').val()
  if (asset) app.update('create', asset)
})

// Transfer Asset
$('#transferSubmit').on('click', function () {
  const asset = $('[name="assetSelect"]').val()
  const owner = $('[name="transferSelect"]').val()
  if (asset && owner) app.update('transfer', asset, owner)
})

// Accept Asset
$('#transferList').on('click', '.accept', function () {
  const asset = $(this).prev().text()
  if (asset) app.update('accept', asset)
})
```

Рисунок 2.8 – Создание, передача и прием активов

Sawtooth - одна из девяти технологий бизнес-блокировки и распространяемый журнал под покровительством Linux Foundation. То же

самое касается языка моделирования Hyperledger Composer с поддержкой REST API, основанного на JavaScript.

Платформа Hyperledger Sawtooth, над которой работали более 50 инженеров-программистов, позволяет управлять цепочкой, используемой в самоисполняющихся "умных" контрактах при настройке конфигурации блокировки и определении прав доступа к электронному журналу.

Платформа поддерживает усовершенствованный механизм транзакций, позволяющий обрабатывать их параллельно, что ускоряет создание и валидацию блоков.

```
const { TransactionProcessor } = require('sawtooth-sdk/processor')
const { JSONHandler } = require('./handlers')

const VALIDATOR_URL = 'tcp://localhost:4004'

// Initialize Transaction Processor
const tp = new TransactionProcessor(VALIDATOR_URL)
tp.addHandler(new JSONHandler())
tp.start()
```

Рисунок 2.9 – Начало работы процессора транзакций

В index.js мы инициализируем процессор транзакций, handlers.js содержит все обработчики, где мы определяем всю необходимую бизнес-логику, включая запросы и обновления до состояния блочной цепи. Package.json обработчики зависимости процессора: sawtooth-sdk.

```
placeholder="Enter athlete name...">
        <input id="boxNo" class="form-control" type="text" placeholder="Enter
box number...">
        <input id="missionOrder" class="form-control" type="text"
placeholder="Enter mission order...">
        <input id="timeSealed" class="form-control" type="text"
placeholder="Enter sealed time...">
        <input id="createSubmit" type="button" value="Create" class="btn btn-
primary">
      </div>

      <div class="form-group">
<<!--       <label>Transfer Doping Control Form</label>
        <select class="form-control" name="assetSelect">
          <option value="none" selected>Select Doping Control Form...</option>
        </select>
        <select class="form-control" name="transferSelect">
          <option value="none" selected>Select ADAMS/DCO recipient...</option>
        </select>
        <input id="transferSubmit" type="button" value="Transfer" class="btn
btn-primary">
      </div>

      <div class="form-group">
        <label>Accept DCF</label>
        <div id="transferList"></div>
      </div>
    </div>
-->
    <div id="data">
      <label>ADAMS List</label>
      <table class="table table-hover">
        <tr>
          <th>Athlete name</th>
          <th>Box number</th>
          <th>Mission order</th>
          <th>Time sealed</th>
          <th>Doping Control Officer</th>
        </tr>
        <tbody id="assetList"></tbody>
      </table>
    </div>
  </div>

<script src="dist/bundle.js"></script>
</body>
</html>
```

Рисунок 2.10 – Конфигурация параметров html страницы

HTML - это язык гипертекстовой разметки, который получил широкое распространение в Интернете. Язык HTML определяет структуру страниц,

которые отображаются в браузере. Каждый сайт в Интернете использует HTML для отображения информации.

HTML определяет структуру страниц, которые отображаются в браузере благодаря тегам HTML, браузер считывает, обрабатывает их, а затем отображает теги для вас на экране, но как элементы HTML, с некоторыми элементами HTML вы можете даже взаимодействовать с помощью мыши или клавиатуры.

Если быть точным с формальной точки зрения, то правильно говорить не о HTML-странице, а о HTML-документе, Ваш браузер общается с веб-сервером по протоколу HTTP, посылает HTTP-запросы и получает ответы сервера, в теле которого содержится HTML.

Как и протокол HTTP, язык HTML был разработан в ЦЕРНе Тимом Бернерсом-Ли (Tim Berners-Lee) в 1991 году и изначально использовался учеными для обмена научными документами. Язык HTML четко определил структуру документа и позволил выделить некоторые особенности текста документа, благодаря этому и тому, что синтаксис языка HTML был простым, он получил огромное распространение не только в научной среде, но и пошел в массы.

Index.html с пользовательским интерфейсом для применения. Откройте этот файл в браузере, чтобы начать взаимодействие с Hyperledger Sawtooth. После этого параметры можно легко записать в блок-цепочку приложения на основе фреймворка Hypeledger Sawtooth.

3 Процесс установки необходимых пакетов и анализ технических характеристик системы

3.1 Процедура инсталляции с помощью интерфейса командной строки

Sawtooth SDK предлагают пользователям сделок помощь разработчика для Javascript, Go., Java, C, C, и Python Для создания простого приложения мы будем использовать Javascript SDK, который предлагает клиенту производительность и поставляет процедуру внесения изменений в блок-схему.

Создание приложения требует нескольких важных шагов:

Во-первых, определение активов, которые будут находиться в распределенной книге, а также сделки, которые будут действовать на данные активы, чтобы изменить их состояние. Во-вторых, разработка логики транзакций, которые будут действовать на данных активах.

Для создания приложения требуется несколько важных шагов:

Во-первых, определение активов, которые будут находиться в распределенной книге, а также сделки, которые будут действовать на эти активы, чтобы изменить их состояние. Во-вторых, разработка логики транзакций, которые будут действовать на этих активах.

Затем, когда транзакции поступают на узел Sawtooth, они перенаправляются на другие узлы. Узел выбирается по консенсус-модели для публикации блока. Блок будет содержать любые транзакции, которые были получены и успешно выполнены. Затем блок транслируется на узлы публикации. Каждый узел пилы получает опубликованный блок и проверяет его действительность. Затем он уведомляет наше приложение о любых изменениях состояния.

Hyperledger Sawtooth - это набор, позволяющий создавать и использовать распределенную книгу. Установка Hyperledger Sawtooth будет включать в себя добавление ключей подписания для создателя программного обеспечения в нашу среду, включая репозиторий, который содержит код в нашей системе, и выполнение обычного обновления/установки. Узел Hyperledger Sawtooth validator может быть запущен как из предварительно созданных образов Docker, так и с помощью Ubuntu 16.04.

Прежде всего, нам необходимо загрузить файл Docker'a в виде файла sawtooth-default.yaml. и перенести его в текстовый файл.

```
version: "2.1"

services:

  settings-tp:
    image: hyperledger/sawtooth-tp_settings:0.8
    container_name: sawtooth-settings-tp-default
    expose:
      - 4004
    depends_on:
      - validator
    entrypoint: settings-tp -vv tcp://validator:4004

  intkey-tp-python:
    image: hyperledger/sawtooth-tp_intkey_python:0.8
    container_name: sawtooth-intkey-tp-python-default
    expose:
      - 4004
    depends_on:
      - validator
    entrypoint: intkey-tp-python -vv tcp://validator:4004

  xo-tp-python:
    image: hyperledger/sawtooth-tp_xo_python:0.8
    container_name: sawtooth-xo-tp-python-default
    expose:
      - 4004
    depends_on:
      - validator
    entrypoint: xo-tp-python -vv tcp://validator:4004

  validator:
    image: hyperledger/sawtooth-validator:0.8
    container_name: sawtooth-validator-default
    expose:
      - 4004
    ports:
      - "4004:4004"
    # start the validator with an empty genesis batch
    entrypoint: "bash -c \"\
        sawtooth admin keygen && \
        sawtooth keygen my_key && \
        sawtooth config genesis -k /root/.sawtooth/keys/my_key.priv && \
        sawtooth admin genesis config-genesis.batch && \
        sawtooth-validator -vv \
          --endpoint tcp://validator:8800 \
          --bind component:tcp://eth0:4004 \
          --bind network:tcp://eth0:8800 \
        \""
```

Рисунок 2.11 – Файл Docker-compose.yaml

После этого нам необходимо запустить docker-compose файл на операционной системе Ubuntu, выполнив следующую команду.

В приложении Sawtooth в реестре будет храниться состояние системы, в дополнение к неизменяемой записи о транзакциях, которые создали это состояние. Приложение обычно состоит из двух частей:

Клиентское приложение отправляет транзакции в блокировку, обычно через API Sawtooth REST. Клиентом может быть интерфейс командной строки, веб-страница, мобильное приложение, сенсор IoT или большинство других видов интерфейсов, способных отправлять HTTP-запросы.

Процессор транзакций кодирует бизнес-логику приложения. Общается с валидатором, который отправляет транзакции, полученные от клиента, в обработчик транзакций для проверки.

Перед выполнением этой команды необходимо перейти в рабочую директорию.

cd образование/LFS171x/sawtooth-material

После переноса необходимо выполнить следующую команду, чтобы запустить файл компиляции докера.

$ docker-compose -f sawtooth-default.yaml up

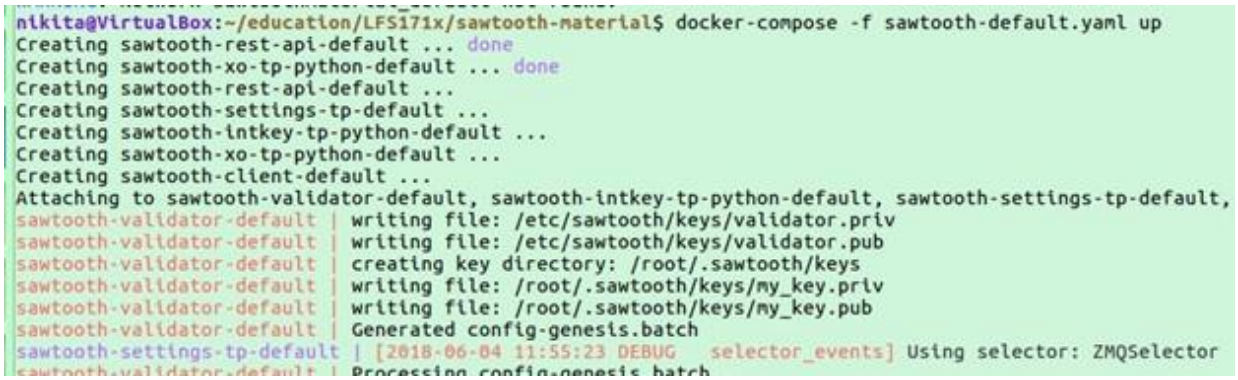

Рисунок 2.12 – Файл Running docker-compose

После выполнения команды нам нужно войти в запущенный контейнер, выполнив следующую команду:

$ docker exec -it sawtooth-shell-default bash

Мы увидим следующую корневую папку:

root@75b380886502:/#

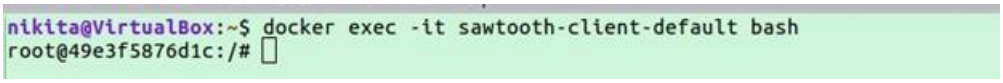

Рисунок 2.13 – Текущая папка

Теперь наше приложение настроено, и мы готовы начать экспериментировать с сетью. Но сначала нам нужно проверить, что валидатор работает правильно и доступен из клиентского контейнера. Для этого нам нужно выполнить следующую команду:

$ curl http://rest-api:8008/blocks

Рисунок 2.14 – Блоки rest-api

Как видно на экране, есть объекты json-ответа с заголовками, подписями заголовков, выходами и т.д.

Для корректной работы программы нам необходимо проверить наш хост-компьютер с помощью контейнера Docker. Для этого необходимо выполнить команду в терминальных окнах. Причины запускать ее из того же каталога, что и раньше, нет. Мы запустим ее из нового терминала.

$ curl http://localhost:8008/blocks

Рисунок 2.15 – Блоки Localhost

После выполнения данной команды должен появиться json-ответ объекта с "данными", массивом партий, заголовком и т.д.

После выполнения этих команд необходимо открыть новый терминал и инициализировать json обработчики для Hyperledger Sawtooth.

Для этого нам необходимо перейти в следующий каталог:

cd образование/LFS171x/пила-материал/пила-вада/процессор

После того, как нам нужно запустить npm (менеджер пакетов узлов). Мы используем его для распаковки контейнеров.

```
nikita@VirtualBox:~$ cd education/LFS171x/sawtooth-material/sawtooth-tuna/processor
nikita@VirtualBox:~/education/LFS171x/sawtooth-material/sawtooth-tuna/processor$ npm start

> Sawtooth-tuna-chain-processor@0.0.0 start /home/nikita/education/LFS171x/sawtooth-materi
al/sawtooth-tuna/processor
> node index.js

Initializing JSON handler for Sawtooth Tuna Chain
Connected to tcp://localhost:4004
Registration of [transfer-chain 0.0 application/json] succeeded
```

Рисунок 2.16 – Запуск менеджера пакетов узлов

После выполнения всех этих команд наша система готова к работе. Нам необходимо открыть файл index.html, который содержит интерфейс нашего приложения для работы с блок-цепочками.

Для этого нам необходимо перейти в следующий каталог:

cd education/LFS171x/sawtooth-material/sawtooth-wada/client/index.html.

Захватить html файл и вставить его в браузер. На экране появится интерфейс приложения.

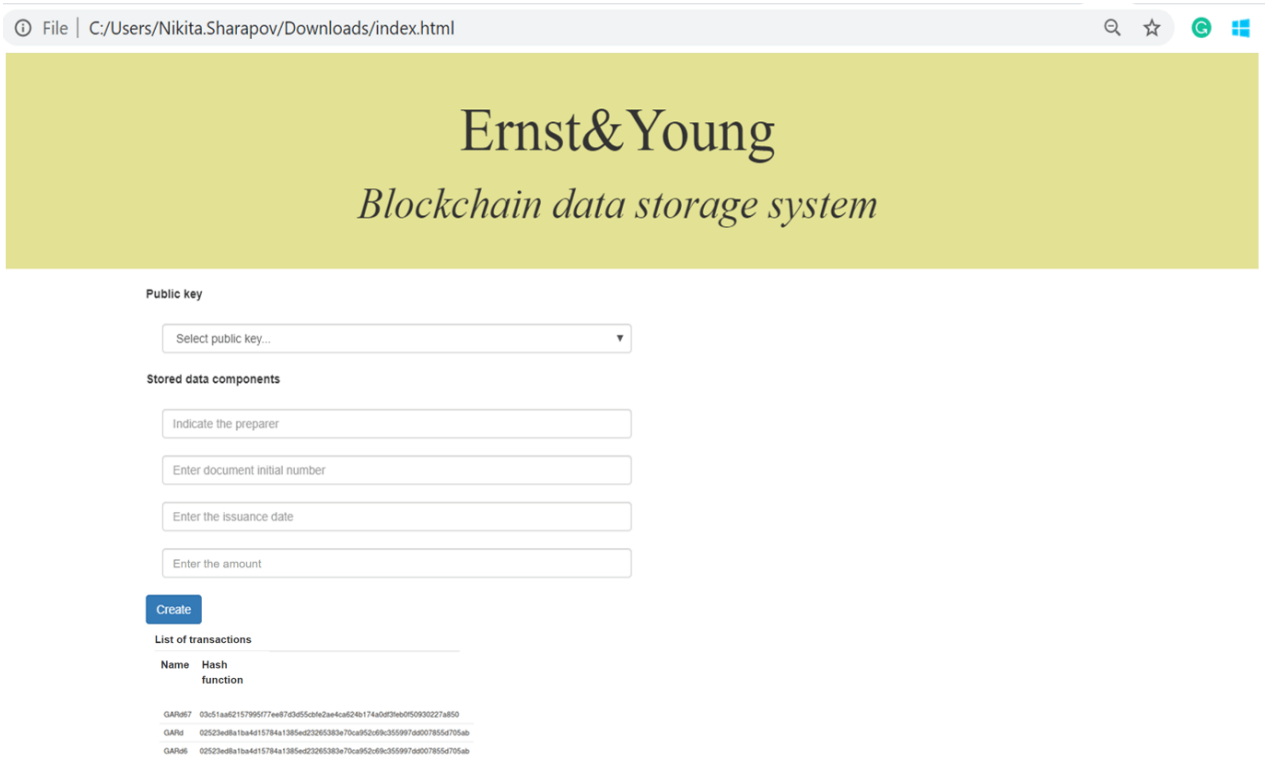

Рисунок 2.17 – Интерфейс приложения

С помощью данного приложения мы можем передавать формы допинг-контроля во Всемирное антидопинговое агентство с помощью блок-цепочки.

Сотрудник допинг-контроля создает публичный ключ и затем вводит его:

- имя спортсменов;
- номер коробки с пробой;
- порядок выполнения миссии;
- время запечатывания пробы;
- .

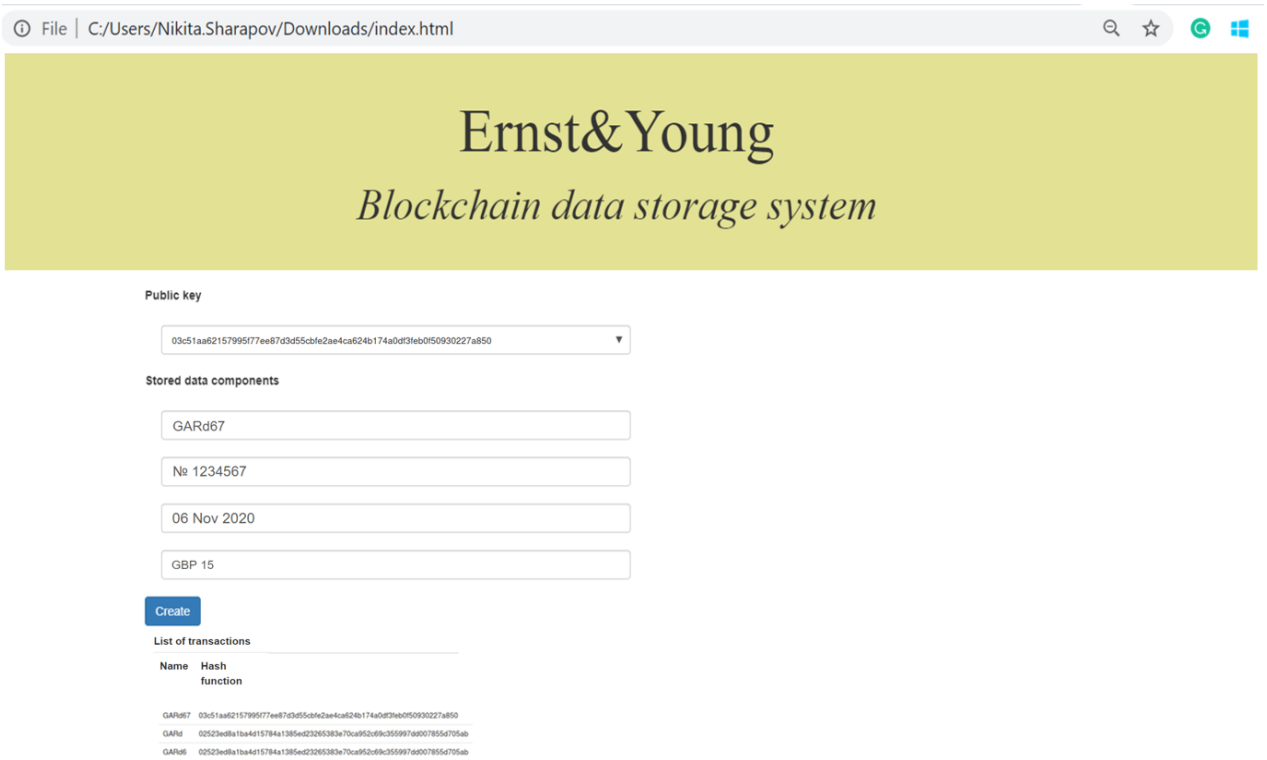

Рисунок 2.18 – Интерфейс

Мы выбрали следующие параметры для записи в приложение out blockchain:

- имя: GARd67;
- номер документа: №1234567;
- Дата транзакции: 6 ноября 2020 года;
- Сумма: GBP 15;

Процесс добавления параметров в блок-цепь проиллюстрирован ниже.

Рисунок 2.19 – Добавление параметров в блокчейн

Наконец, мы разработали блокчейн приложение с помощью платформы Hyperledger Sawtooth. Были выполнены следующие задачи, и в будущем это приложение может быть реализовано.

Проведена полная передача аутентичных данных, используемых для регистрации документов с невозможностью последующего внесения изменений в реестр. Исправлены ошибки в процессе заполнения формы допинг-контроля. Хранилище нашей базы данных имеет возможность просматривать истории транзакций.

3.2 Расчеты компонентов сети

3.2.1 Кластер надежности. Готовность отражает способность системы непрерывно выполнять свои функции.

Фактором готовности является вероятность того, что компьютерная система будет работать в любой момент времени.

Этот коэффициент определяется по формуле:

$$K = \frac{MTBF}{MTBF+MTTR} \quad (3.1)$$

Где:

MTBF означает среднее время между ошибками

MTTR – Среднее время на исправление

В отличие от надежности, значение которой определяется только значением MTBF, доступность также зависит от времени, необходимого для возвращения системы в рабочее состояние.

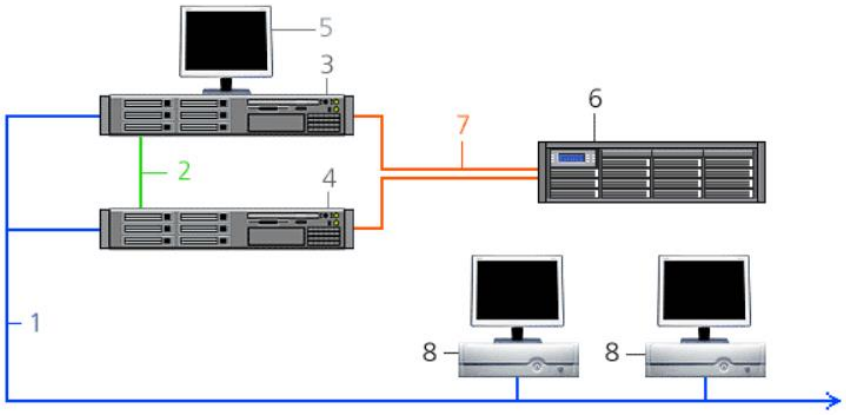

Рисунок 3.1 – Структура кластера

Где 1 - Общая сеть; 2 - Частная сеть; 3 - Узел 1; 4 - Узел 2; 5 - Консоль управления; 6 - Общий дисковый массив; 7 - Интерфейсы SCSI и волоконно-оптических каналов; 8 - Клиенты

Кластер состоит из двух узлов (серверов), соединенных общим дисковым массивом. Все основные компоненты этого дискового массива - блок питания, дисковые накопители, контроллер ввода-вывода - имеют резервирование с возможностью горячей замены. Внутренняя сеть для обмена информацией об их текущем состоянии соединяет узлы кластера. Питание кластера осуществляется от двух независимых источников. Подключение каждого узла к внешней локальной сети также дублируется.

Таким образом, все подсистемы кластера имеют резервирование, поэтому в случае отказа какого-либо элемента, кластер в целом останется работоспособным. Более того, замена вышедшего из строя элемента возможна без остановки кластера.

Поэтому мы делаем вид, что в нашей системе также есть два узла (сервера), подключенных к общему дисковому массиву. На самом деле, это число может отличаться, так как мы строим приложение с дюжиной узлов.

Приложение вместе со своей ресурсной группой не привязано к конкретному узлу, а, наоборот, может быть запущено на любом из этих узлов (и на каждом из них могут одновременно запускаться несколько приложений). В свою очередь, клиенты этого приложения (сервиса) "увидят" не узлы кластера в сети, а виртуальный сервер (сетевое имя и IP-адрес), на котором запущено приложение.

Сначала приложение работает на одном из узлов. Если этот узел по какой-то причине перестает функционировать, то другой узел перестает получать от него сигнал активности ("сердцебиение") и автоматически запускает все приложения вышедшего из строя узла, т.е. приложения вместе со своими группами ресурсов "мигрируют" на рабочий узел. Миграция приложения может длиться от нескольких секунд до нескольких десятков секунд, и в течение этого времени приложение недоступно клиентам.

В зависимости от типа приложения после перезапуска сессия возобновляется автоматически или может потребоваться повторная

авторизация клиента. Нет необходимости изменять настройки на клиентской стороне. После ремонта неисправного узла, его приложения могут мигрировать обратно.

По мере того, как мы строим кластер высокой доступности, нам нужно учитывать тот факт, что эта система может выйти из строя. В мире нет системы, полностью защищенной от сбоев на 100%.

Если нам нужно сделать несколько остановок, мы можем сделать это легко без необходимости останавливать всю систему.

На кластере эти операции можно выполнять последовательно на разных узлах, не прерывая работу кластера в целом.

Незапланированные остановки происходят из-за программных или аппаратных сбоев. В случае программного сбоя на обычном сервере необходимо перезагрузить операционную систему или приложение, в случае кластера приложение перейдет на другой узел и продолжит работу.

В любом случае, нам нужно иметь несколько цифр, на которые можно положиться при построении этой системы.

Как мы видели выше, формула MTBF означает среднее время между ошибками. Как мы можем теперь получить это значение? Для этого есть специальная формула:

$$\text{MTBF} = \frac{\text{Время тестирования} * \text{Количество тестируемых продуктов}}{\text{Количество ошибочных продуктов}}$$

(3.2)

Давайте притворимся, что это для нашей системы блокчейн:

Время тестирования = 5 лет;

Количество протестированных продуктов = 5000 штук;

Количество ошибочных продуктов = 500 единиц

$$\text{MTBF} = \frac{5 * 5000}{500} = 50 \text{ (лет)}$$

Это означает, что только через 50 лет все объекты будут неисправны.

Концепция MTBF совсем не отражает того, что очевидно следует из его названия. "Среднее время между отказами" буквально означает время, которое составляет только половину ССПБ. Так, в нашем примере это "среднее время" будет не 50 лет, а всего 25 лет, потому что в среднем все экземпляры изделия будут работать не 10 лет, а наполовину меньше. Те. MTBF , заявленный производителем - это время, в течение которого продукт выйдет из строя с вероятностью 100% [31].

Поэтому можно сделать вид, что вероятность выхода из строя MTBF равна 1 и если измерять MTBF в годах, то вероятность выхода из строя компонента в течение одного года будет такой:

$$\text{P} = \frac{1}{\text{MTBF}} = \frac{1}{50} = 0{,}02$$

Например, нам нужно заменить сломанный компонент системы в течение 24 часов (1/365 года), после чего вероятность этого события приравнивается:

$$P_d = \frac{P * P}{365} * 2$$

Есть две разные возможности:

1. Отказ компонента № 1 в любой момент времени в течение года (вероятность P).
2. Отказ компонента № 2 в течение 24 часов после выхода из строя компонента № 1 (вероятность P / 365).

Вероятность безотказной работы любого компонента в течение года равна:

$$P'_i = 1 - P_i \quad (3.3)$$

где P_i - вероятность выхода из строя компонента в течение одного года.

Обычно срок службы материнской платы - год, в некоторых других случаях - 10 лет. Однако средний срок службы материнской платы обычно составляет 5-6 лет. Многое зависит от интенсивности работы этой платы, и если вы будете работать с компьютером экономно, то материнская плата сможет проработать даже 15 лет.

Жесткий диск (жесткий диск) является одним из самых слабых с точки зрения выживаемости. Современные модели рассчитаны на работу от трех лет, хотя на практике они служат дольше. Замечено, что старые модели были жестче, а современные диски выходят из строя даже через 5 лет.

Большая часть энергии, потребляемой источником питания, идет на питание компонентов, а не на нагрев самого устройства. Поэтому у дорогих надежных моделей может не быть даже вентилятора, так как он там не нужен. Срок службы компьютерного блока питания такого типа может составлять 10-20 лет. Поэтому источник питания с вентилятором работает примерно 10-20 лет, поэтому можно сделать вывод, что и вентилятор работает 10-20 лет.

Срок службы процессора составляет 10 лет, однако, если он еще работает и способен регулировать напряжения и частоты, то, внедрив эффективную систему отвода тепла от системного блока, можно продлить срок службы процессора до 50 лет. В любом случае, давайте возьмем 10 лет.

В среднем срок службы видеокарты при нормальной работе составляет от 3 до 8 лет. Но все зависит от степени загруженности устройства. Если вы постоянно играете в игры или даже зарабатываете на этом валюту криптографии, то вы не можете рассчитывать на длительный период работы. Такие карты будут работать 2-3 года [13].

ОЗУ может выйти из строя в случае высокой температуры внутри системного блока или подачи на него некорректного напряжения (это

происходит при выходе из строя материнской платы). Кроме того, планка может физически сломаться при прикосновении. Поэтому давайте возьмем 100 лет оперативной памяти.

Вероятность безотказной работы любого компонента в течение года:

$$P'_m = 1 - P_m = 1 - \frac{10}{365} = 1 - 0{,}0273 = 0{,}9726$$

$$P'_{HDD} = 1 - P_{HDD} = 1 - \frac{5}{365} = 1 - 0{,}0136 = 0{,}9863$$

$$P'_{ps} = 1 - P_{ps} = 1 - \frac{15}{365} = 1 - 0{,}0411 = 0{,}9589$$

$$P'_v = 1 - P_v = 1 - \frac{20}{365} = 1 - 0{,}0547 = 0{,}9452$$

$$P'_{CPU} = 1 - P_{CPU} = 1 - \frac{10}{365} = 1 - 0{,}0273 = 0{,}9726$$

$$P'_{vc} = 1 - P_{vc} = 1 - \frac{7}{365} = 1 - 0{,}0191 = 0{,}9801$$

$$P'_{RAM} = 1 - P_{RAM} = 1 - \frac{0{,}1}{365} = 1 - 0{,}00002 = 0{,}9999$$

Вероятность безотказной работы всех компонентов в течение года равна произведению вероятностей этих независимых событий:

$$P_{s\prime} = P'_m * P'_{HDD} * P'_{ps} * P'_v * P'_{CPU} * P'_{vc} * P'_{RAM} \quad (3.4)$$

$$P_{s\prime} = 0{,}9726 * 0{,}9863 * 0{,}9589 * 0{,}9452 * 0{,}9726 * 0{,}9801 * 0{,}9999 = 0{,}8287$$

Вероятность отказа сервера в течение года равна:

$$P_s = 1 - P_{s\prime} = 1 - 0{,}8287 = 0{,}1713$$

Зная вероятность отказа сервера в течение года, вы можете определить время, необходимое для отказа (время, через которое сервер выйдет из строя со 100% вероятностью):

$$MTBFs = \frac{1}{P_s} = \frac{1}{0{,}1713} = 5{,}8377 \text{ (лет)}$$

Теперь мы можем определить коэффициент доступности:

$$K_s = \frac{MTBF_s}{MTBF_s + MTTR_s} = \frac{5{,}8377}{5{,}8377 + \frac{1}{365}} = \frac{5{,}8377}{5{,}8404} = 0{,}9995$$

Обобщим данные производителей о надежности отдельных компонентов в следующей таблице.

Таблица 3.1 – Надежность отдельных компонентов

Компоненты сервера	MTBF (часы)	MTBF (часы)	Вероятность отказ в течение года	Количество компонентов на сервере	Вероятность отказа
Материнская плата	87,600	10	0.0273	1	0.0273
HDD	43,800	5	0.0136	1	0.0136
Источник питания	131,400	15	0.0411	2	0.0016
Вентилятор	175,200	20	0.0547	2	0.0029
Центральный процессор (CPU)	87,600	10	0.0273	1	0.0273
Видеокарта	61,320	7	0.0191	1	0.0191
Оперативная память (RAM)	876,000	100	0.00002	1	0.00002
Итого			0.18312		0.09182

Таким образом, мы получаем следующие результаты:

- вероятность отказа сервера в течение года - 0,09182
- MTBF сервера - 58377 (лет)
- среднее время устранения неисправности - 24 (часа)
- доступность серверов - 99,9 %
- среднее время простоя в году: 3,65 (часы)

3.2.2 Расчет размера блока. В настоящее время существует жёсткое ограничение на размер блока 1 МБ; все основные клиенты Bitcoin, независимо от блокировки 1, считаются недействительными [3]. Это ограничение оценивается в 4000 транзакций на блок (предполагая, что средний размер транзакции является наиболее релевантным, приблизительно 200-250 байт). Так как блоки добываются раз в 10 минут, то количество транзакций в секунду (TPS) составляет около 7. Улучшенное поведение дает максимальное время обработки актуальных идей в очереди, и некоторые исследования подсчитали, что они никогда не были ограничены 3,5 TPS [4].

Математическое моделирование показывает, что возросло давление как можно больше, например, 2.8 транзакций в секунду (до 80%). Негативное

влияние подтвержденного времени транзакции на сеть является отрицательным. Время подтверждения первой транзакции составляет 18.5 минут; подтверждена половина скорости транзакции. Для сравнения, если вход 1 TPS, то первое подтвержденное центральное время подтверждения составляет 7 минут.

Построение математической модели фактически является основным этапом проектирования или исследования любого метода. Весь последующий анализ объекта зависит от качества модели. Создание модели не является правильным процессом. Высоко определяется исследователем, его вкус и опыт, как правило, зависит от конкретного экспериментального материала. Прибор должен быть полностью точным, достаточным и удобным для использования.

Таблица 3.2 Время обработки транзакций в зависимости от размера блока и нагрузки на сеть

Пропускная способность, tps	Размер блока, MB	Загруженность сети, %	Время обработки, мин	Время на обработку 90% tx, мин
1,75 (нормальная величина)	1 2 3 4 8 20	50.0 25.0 16.7 12.5 6.3 2.5	8.5 7.0 7.0 7.0 7.0 7.0	29.0 23.5 23.0 23.0 23.0 23.0
2,5 (пиковая величина)	1 2 3 4 8 20	70.0 35.0 23.3 17.5 8.8 3.5	13.2 7.4 7.0 7.0 7.0 7.0	42.8 24.7 23.0 23.0 23.0 23.0
3,5 (максимальная величина)	1 2 3 4 8 20	100.0 50.0 33.3 25.0 12.5 5.0	129.1 8.5 7.4 7.0 7.0 7.0	380.0 29.0 24.7 23.5 23.0 23.0

Пропускная способность является еще одним аспектом проблемы: если параметр является подтверждением того, что вся пыль в транзакции может быть замедлена большую часть времени - чем в транзакции, то выходная стоимость средней стоимости транзакции еще ниже. [7]

Для обслуживания возражений (DOS) обычно используется пыль, которая была самой большой в июле 2015 г. [2]. Некоторые виды операций с

полюсами не транслируются по очень низкой цене и не включаются в блок. Больше ограничений на торговлю: как альтернатива остановке роста.

Как правило, высшей целью злоумышленника является полное предотвращение интернет-ресурса - "отказ в обслуживании". Нападающий также может потребовать наличные деньги, чтобы остановить нападение. В некоторых случаях DDoS-атака может представлять собой попытку дискредитации и/или уничтожения компании-конкурента [16].

Для отправки чрезвычайно большого количества запросов на ресурс жертвы киберпреступник обычно создает сеть зараженных "зомби-компьютеров". Поскольку злоумышленник регулирует деятельность каждого зараженного персонального компьютера в зомби-сети, забастовка может оказаться слишком эффективной для полезного сетевого ресурса жертвы.

Таблица 3.3 Потребление ресурсов на узлах по мере увеличения размера блока

Характеристики	Коэффициент маштабируемости	Размер блока, MB (=N/2)						
		0,5	1	2	4	8	16	32
Пропускная способность транзакций, tps	N	1.75	3.5	7	14	28	56	112
Количество txs в блоке	N	1050	2100	4200	8400	16800	33600	67200
Хранение блокчейн в день, МБ	N	72	144	288	576	1152	2304	4608
Хранение блокчейн в год, ГБ	N	26	51	103	205	411	821	1643
Время обработки транзакции, ms	1	0.33	0.33	0.33	0.33	0.33	0.33	0.33
Время проверки блока, s	N+ 0:09N log2 N	0.07	0.15	0.33	0.71	1.51	3.23	6.86
Средняя пропускная способность, kB/s	N	74	148	296	592	1184	2368	4736
Дневной траффик, ГБ	N	6.2	12.4	24.8	49.6	99.2	198	397
Годовой трафик, ТБ	N	2.2	4.4	8.8	17.7	35.4	70.7	141

Использование оперативной памяти, ГБ	N	2	4	8	16	32	64	128
Немедленно исключаемые узлы, %	n/a	0	20	40	75	90	95	95
Исключенные узлы через 6 месяцев, %	n/a	5	25	50	80	95	95	95

Увеличение размера блока может привести к дополнительной нагрузке на узлы сети Bitcoin. Наиболее дорогостоящей операцией во время ретрансляции транзакций является проверка ECDSA подписей входов транзакций (каждая транзакция проверяется перед ретрансляцией на другие узлы с целью защиты сети от DoS-атак). Современные процессоры способны проверять несколько тысяч транзакций в секунду [11]. Таким образом, текущая пропускная способность транзакций в несколько tps не является большой нагрузкой для сети.

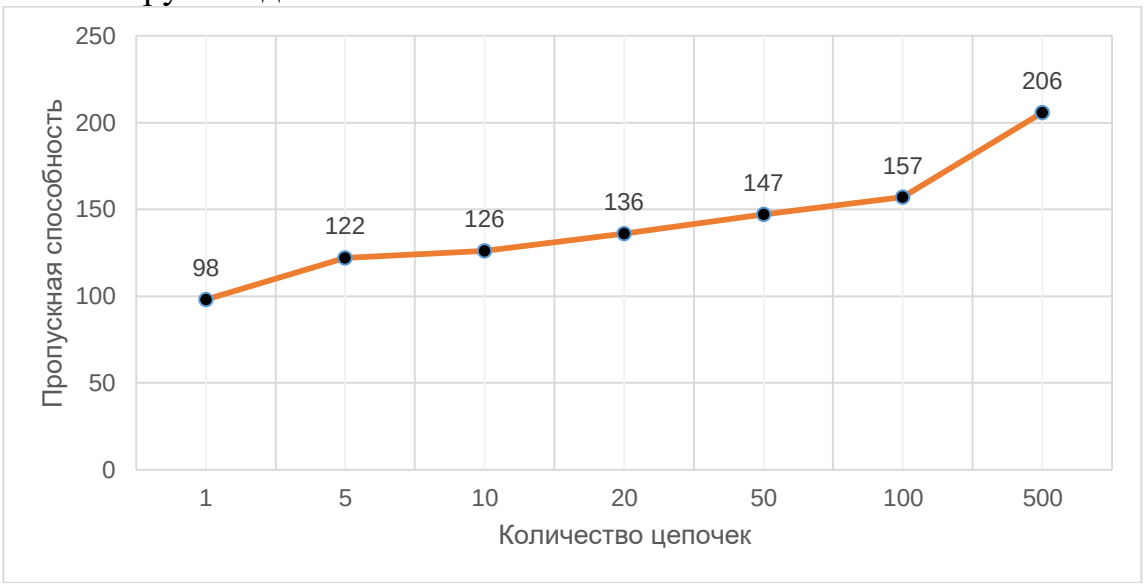

Рисунок 3.2 – Пропусная способность vs Количество цепочек

На рисунке 3.2 показано, что увеличение количества цепочек увеличивает общую пропускную способность, основной причиной, по-видимому, являются параллельные процессы валидации и блокировки коммитов. В данной работе мы используем десять (10) цепочек, четыре (4) смоделированные параллельные транзакции в каждой цепочке, 20 000 блоков во всех цепочках , равномерно распределенными по цепочкам, четыре (4) ключа, случайным образом смоделированные и модифицированные для каждой транзакцией, размер блока 200 байт и 50 транзакций в каждом блоке.

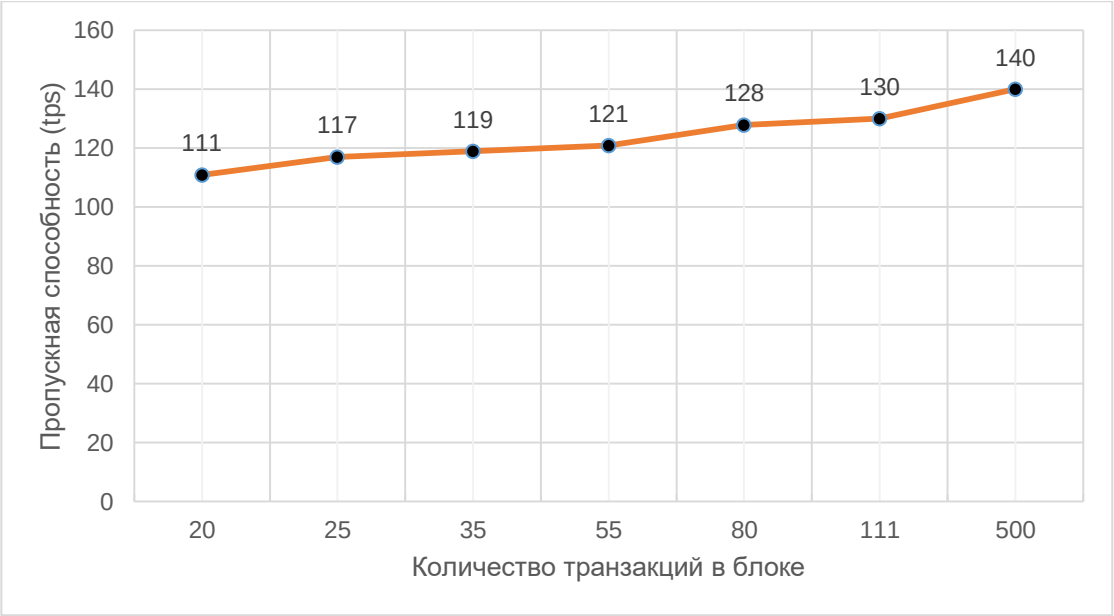

Рисунок 3.3 – Пропускная способность vs Количество транзакций в блоке

В нашей работе мы рассчитали параметры потребления ресурсов и взял среднее значение размера блока, которое в настоящее время составляет 0.5 Мб.

Пропускная способность транзакции - это размер блока, поделенный на ожидаемый интервал времени между блоками (600 секунд) и на текущий средний размер транзакции (чуть менее 0,5 килобайт).

Этот метод дает результат приблизительно 1:75 tps.

Количество транзакций в блоке - это размер блока, поделенный на средний размер транзакции; в качестве альтернативы, он равен пропускной способности транзакции, умноженной на 600.

$$\text{Транзакций в блоке} = \text{Пропускная способность транзакции} * 600 = 1050$$

Хранилище блокчейн - это средний размер блока, умноженный на ожидаемое количество блоков. 144 блока в день; 144 * 365 = 52560 блоков в год.

$$\frac{\text{Хранилище блокчейн}}{\text{день}} = \text{размер блока} * \frac{\text{блоки}}{\text{день}} \quad (3.3)$$

$$\frac{\text{Хранилище блокчейн}}{\text{день}} = 0.5 * 144 = 72 \text{ MB}$$

$$\frac{\text{Хранилище блокчейн}}{\text{год}} = \text{размер блока} * \frac{\text{блоки}}{\text{дни}} * \text{дней в году} \quad (3.4)$$

$$\frac{\text{Хранилище блокчейн}}{\text{год}} = 0.5 * 144 * 365 = 26280 \text{ MB}$$

Время обработки транзакций основано на предположении, что узел может обрабатывать 3000 транзакций в секунду [11]. Обратите внимание, что каждая транзакция должна быть обработана, затем узел должен просмотреть неиспользованные выходы транзакций, соответствующие входам транзакций, и проверить каждую из подписей, встроенных во входы.

Время проверки блока - это время обработки транзакции, умноженное на среднее количество транзакций в блоке, умноженное на 0:2. Последний фактор соответствует понятию, что большинство транзакций в блоке уже верифицировано при поступлении блока; узлу достаточно вычислить хэши транзакций и просмотреть их в кэше транзакций.

$$\text{Время проверки блока} = 0.2 * 1050 * 0.0003333 = 0.07 \text{ сек.}$$

По данным statoshi.info средняя пропускная способность узла составляет 74 Кб. (13)

В настоящее время узел обрабатывает более 6 гигабайт (ГБ) трафика в сутки (13).

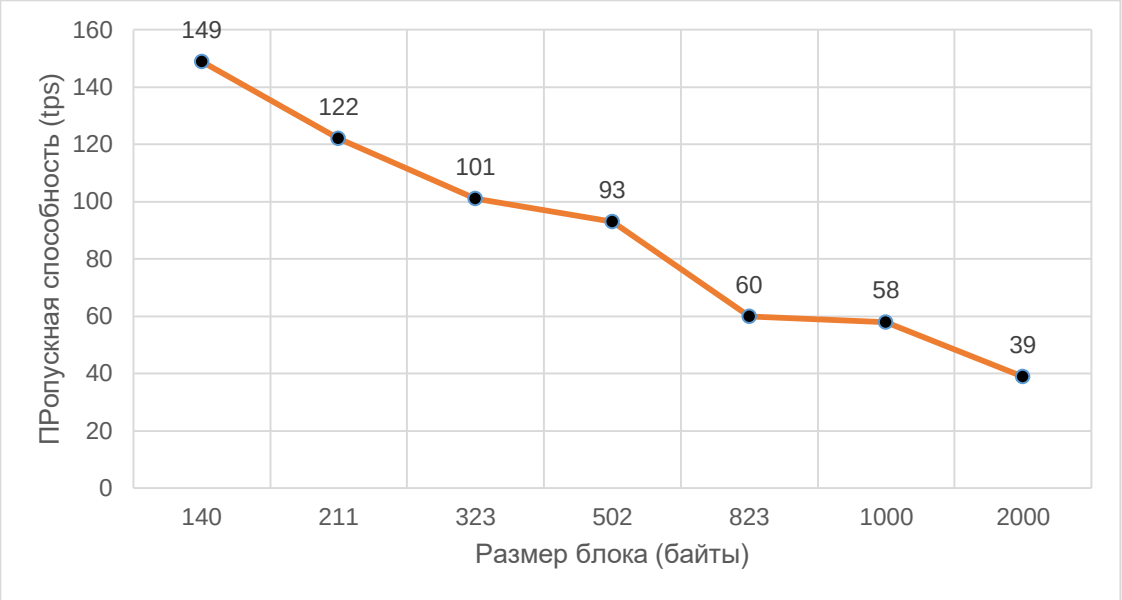

Рисунок 3.4 – Пропускная способность vs Размер блока

На рисунке 3.4 показано, что увеличение размера блока снижает общую пропускную способность. В данной работе мы используем десять цепочек, четыре моделированные параллельные транзакции в каждой цепочке, 20 000 Блоков во всех цепочках с ключами, равномерно распределенными по цепочкам, четыре (4) ключа, случайным образом считанные и модифицированные каждой транзакцией, размер Блока 200 байт и 50 транзакций в каждом Блоке. С увеличением размера блока увеличивается задержка по мере увеличения скорости поступления с увеличением размера блока. Таким образом, с увеличением размера блока увеличивается количество транзакций, обрабатываемых в секунду, что увеличивает пропускную способность всех транзакций.

Использование оперативной памяти - наиболее сложная для оценки переменная, так как она зависит от многих факторов. Кроме того, использование оперативной памяти различается для различных типов узлов. Одним из основных компонентов, способствующих потреблению оперативной памяти, является неизрасходованный выходной набор транзакций, который в настоящее время занимает более 4 Гб [14]. Множество не обязательно должно полностью располагаться в оперативной памяти, хотя другие методы хранения приводят к увеличению времени проверки транзакций. Используем эмпирическое значение 2 ГБ, исходя из рекомендаций [15], которое ниже, чем показывают другие измерения [16].

Очевидно, что для специализированных узлов, таких как узлы добычи, требования к оперативной памяти выше, так как на этих узлах необходимо хранить весь набор выходных данных о неизрасходованных транзакциях в оперативной памяти для быстрой верификации входящих транзакций и блоков.

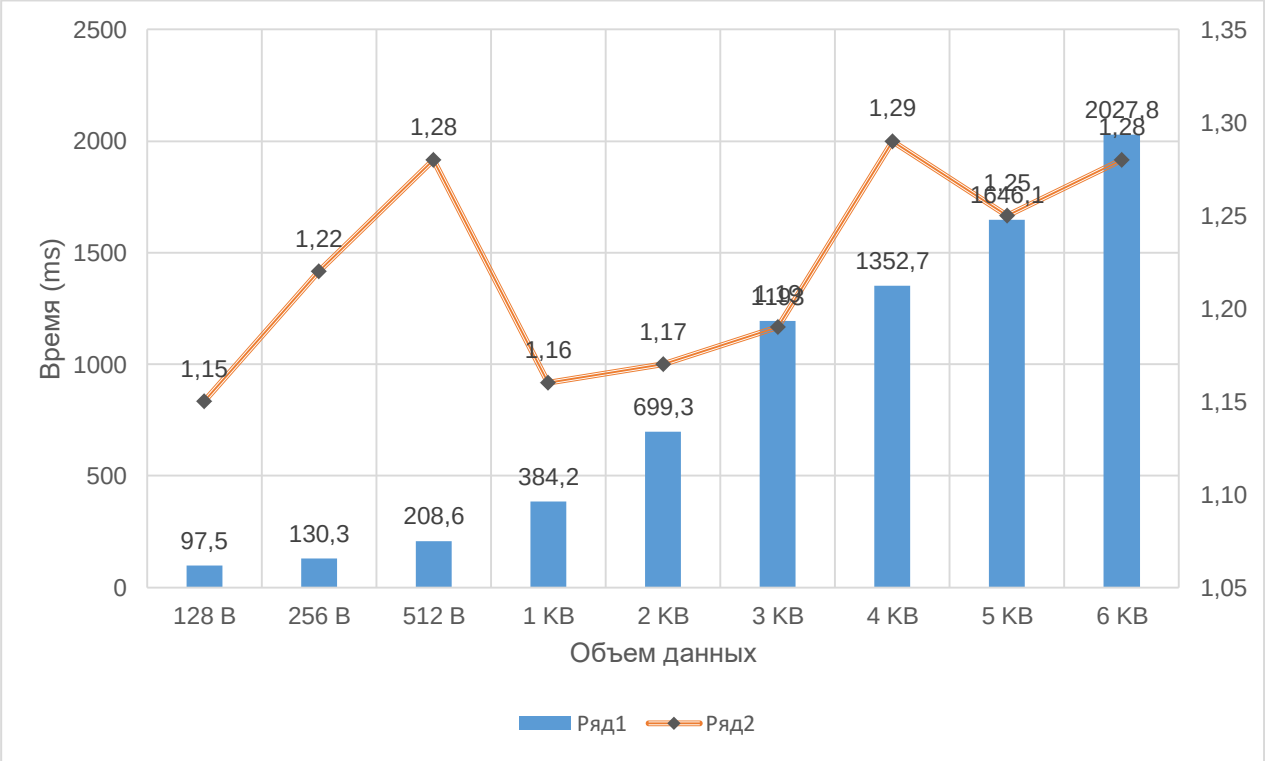

Рисунок 3.5 – Время обработки транзакции

В этом разделе будет проведено сравнение HyperLedger и реляционных баз данных, таких как MySQL, с двумя способами чтения и записи данных. Это сравнение даст некоторые практические представления о сочетании технологии блок-цепочки и централизованных баз данных. В этой работе используется HyperLedger Sawtooth для блок-цепочки и база данных MySQL для реляционных баз данных. Что касается таких механизмов, то у нас есть два стандарта. Первый заключается в том, является ли тестируемый объект функционально полным или нет.

Во-вторых, Hyperledger Sawtooth для реализации различных функций и исполнений [16]. Также MySQL можно использовать для хранения различных типов данных. На рисунке 10 показано время выполнения каждой транзакции, Blockchain тратит больше времени на больший объем данных, достигая 2027 мс с данными по 6 КБ, среднее время чтения и записи составляет 1.22 мс. Однако мы обнаружили, что время, используемое для чтения и записи данных, которое тратится Blockchain'ом, в 1660 раз больше, чем MySQL.

На рисунке 3.5 показана зависимость между пропускной способностью и объемом данных на транзакцию между Blockchain и базой данных MySQL. По мере увеличения объемов данных и превышения порога в 2KB, пропускная способность Hyperledger увеличивается. Сравнивая результат, Hyperledger показывает более высокую производительность при гораздо большем объеме данных и линейные отношения с данными. Установлено, что Hyperledger работает на 80-200 быстрее, чем MySQL. С увеличением объема данных за одну транзакцию увеличивается необходимое время обработки. Время и объем используемых данных приводит к экспоненциальным отношениям. Обратите внимание, что в блок-цепочке восемь узлов, линейная зависимость может быть результатом количества узлов.

Рассчитаем оценки параметров ключевых узлов для увеличенных размеров блоков, умножив наши базовые значения на масштабные коэффициенты во втором столбце таблицы, при этом N обозначает множитель для размера блока. Эти масштабные коэффициенты устанавливаются следующим образом. Мы предполагаем, что средний размер транзакции остается прежним, поэтому пропускная способность транзакции и количество транзакций на блок масштабируется линейно с размером блока.

Точно так же очевидно, что и размер блока линейно зависит от размера блока. Обработка транзакций требует поиска в неизрасходованном выходном наборе транзакций. При оптимальной реализации среднее время поиска логарифмически зависит от размера множества, последнее должно масштабироваться линейно (как это подразумевается под историческими данными [17]). Поскольку в настоящее время количество нерастраченных выходов транзакций значительно превышает 107, время обработки транзакций останется практически неизменным с увеличением размера блока.

Время проверки блока растет наиболее быстрыми темпами по мере увеличения размера блока; в то время как большинство транзакций в блоке уже должно быть проверено при его поступлении, узел все равно должен вычислять хэши для всех транзакций и искать их в кэш-памяти. Как и в предыдущем случае, время поиска масштабируется логарифмически в зависимости от размера кэша.

Пусть S обозначает текущий размер кэша транзакций, а t - среднее время поиска одной транзакции. Если размер блока увеличится в N раз, то время поиска каждой транзакции увеличится до $t * \log_2(NS)$; таким образом, время проверки блока увеличится в 1 раз.

$$N\frac{t*\log_2(NS)}{t*\log_2 S} = N\left(1+\frac{\log_2 N}{\log_2 S}\right)$$

(3.6)

Согласно statoshi.info, S = 2000 и предположим, что размер блока увеличивается N = 0.1 раз за счет двукратного увеличения размера блока, что подразумевает масштабный коэффициент:

$$\text{Время обработки блока} = 0.1\left(1+\frac{\log_2 0.1}{\log_2 2000}\right) = 0.07 \text{ сек}$$

Транспортный узел, а также размер блока зависят от пропускной способности транзакции. Оба эти значения одинаковы.

Другие компоненты, помогающие снизить затраты (такие как размер кэша), демонстрируют линейный или подлинейный рост и, следовательно, не могут быть уменьшены от поведения при длительной настройке. Как и другие компоненты, использование оперативной памяти в основном осуществляется через пользовательские настройки; узел может работать с относительно небольшим объемом оперативной памяти, но это может привести к задержке транзакции.

Так как общий размер блока увеличивается, количество узлов, не имеющих аппаратного обновления, находится в пределах таблицы. Эти предположения предполагают, что многие клиенты запускают полные узлы на аппаратном уровне клиента, а затем запускают их на отдельном ПК или, возможно, в облаке. Аппаратное обеспечение узла распознается по измерению количества [19]; мы думаем, что игроки компьютерных игр, а также энтузиасты Bitcoin имеют точно такое же количество информации, сфокусированной на аппаратном обеспечении своих узлов. ОЗУ на самом деле является исключением: мы считаем, что узел, поддерживающий персональный компьютер, на самом деле меньше 3 ГБ, и что для работы с отступом в качестве узла требуется не менее 2 ГБ ОЗУ [15].

Например, в случае увеличения размера блока до 2 Мб узел должен выделить 8 Гб оперативной памяти на перспективу Bitcoin, а что касается опроса, то гораздо больше половины на самом деле находятся в оперативной памяти.

Мы также предоставляем оценку того, как именно будут удалены текущие сети узлов в ближайшие 6 месяцев. Первичное снижение отбора узлов связано, прежде всего, с использованием процессора и оперативной памяти, а также, в конце концов, есть дополнительные ограничения, включая дисковое пространство, а также Интернет-трафик. Например, при обычном размере блока 8 МБ узел должен иметь емкость более 34 ГБ дискового пространства для обработки примерно 3 ТБ посетителей в месяц.

Заключение

Данная диссертационная работа была посвящена анализу принципов построения, алгоритмов и структур данных технологии блокчейн. Однако, основное направление данной диссертационной работа являлось исследование технологии для построения платформы смарт-контрактов.

В частности, мы представляем краткий обзор технологий смарт контрактов и блокчейн. Затем мы указываем на проблемы в интеллектуальных контрактах в различных аспектах создания, развертывания, исполнения, завершения интеллектуальных контрактов. В то же время, были обсуждены последние достижения в решении этих проблем.

Далее было произведено сравнение нескольких основных платформ смарт-контрактов. Кроме того, была произведена классификация приложения для смарт-контрактов с примерами применения различных типов смарт-контрактов. В целом, я надеюсь, что данный документ послужит руководством для разработки безопасных и масштабируемых приложений смарт-контрактов и будет способствовать развитию технологий блокчейн.

С использованием технологии распределенных реестров исполнение смарт-контрактов происходит автоматически, что дает дополнительные возможности для сокращения затрат участников отношений, возникающих при заключении сделки и исполнении ее условий. Реализуемые через смарт-контракты многосторонние взаимодействия позволяют уменьшить затраты на проведение операций и контроль за ними, увеличить скорость выполнения операций и уменьшить риски, связанные с недобросовестными действиями сторон, максимально сократить или полностью исключить посредников из сделки.

Многие участники финансового рынка, а также представители других отраслей экономики проводят различные эксперименты по применению смарт-контрактов с целью оптимизации бизнес-процессов и сокращения издержек на проведение финансовых операций. Применение смарт-контрактов в традиционных процессах потенциально может создать более удобную среду для взаимодействия между государством, организациями и гражданами.

Во многих случаях проведение экспериментов и разработка решений в области применения смарт-контрактов является неотъемлемой составляющей использования технологии распределенных реестров.

Смарт-контракты идут в ногу с развитием технологии блокчейн, хотя все еще существует ряд проблем, которые необходимо решить. Большинство текущих исследовательских тем, касающихся смарт-контрактов, сосредоточены на вопросах языка программирования, безопасности и конфиденциальности, в то время как популяризация приложений, основанных на технологии для построения платформ смарт-контрактов, также создает определенный ряд новых проблем.

Как и другие инструменты компьютерного программного обеспечения, смарт-контракты также содержат ряд ошибок, которые возникают непреднамеренно. Однако, выявление этих ошибок потребует значительных усилий в аспектах разработки программного обеспечения и анализа данных. Интеграция технологий программного обеспечения, обработки языка и искусственного интеллекта может поспособствовать решению данных проблем в будущем.

Стоит отметить, что смарт-контракты также имеют обширную область для применения не только в финансовом секторе, но и в иных отраслях экономики, и мировой тренд на цифровизацию является одним из основополагающих драйверов развития данного инструмента. Тем не менее не стоит ожидать быстрого и повсеместного внедрения смарт-контрактов, так как любые инновации, прежде чем получить широкое применение, должны пройти определенный путь развития.

Также в работе была рассмотрена структура блока со всеми его компонентами: хэш-функциями, блоками, транзакциями, ключами шифрования, жесткими вилками и софт-вилами. Был продемонстрирован процесс установки фреймворка Hyperledger Sawtooth, с помощью которого было разработано данное приложение со всеми компонентами, позволяющими взаимодействовать без помех. Помимо создания приложения были произведены расчеты надежности системы и вычислены размеры блоков в зависимости от их размера.

Список аббревиатур

1. HS – Hyperledger Sawtooth
2. HF – Hyperledger Fabric
3. HB – Hyperledger Burrow
4. HI – Hyperledger Iroha
5. P2P – Peer-to-peer
6. PoET – Proof of Elapsed Time
7. PoW – Proof of Work
8. PoS – Proof of State
9. SBFT – Simplified Byzantine Fault Tolerant
10. YAC – Yet another Consensus
11. REST – Representational State Transfer
12. CLI – Command Line Interface
13. FTP – File Transport Protocol
14. HTTP – HyperText Transfer Protocol
15. JS – JavaScript
16. IoT – Internet of Things
17. SDK – Software Development Kit
18. HTML – HyperText Markup Language
19. JSON – JavaScript Object Notation
20. MTBF - Mean Time Between Failure
21. MTTR – Mean Time To Repair
22. API – Application Programming Interface
23. IP – Internet Protocol
24. RAM – Random Access Memory
25. CPU – Central Processing Unit
26. TB – Terabyte
27. GB – Gigabyte
28. UI – User Interface
29. OSI – Open System Interconnection
30. URL – Uniform Resource Locator
31. DLT – Distributed Ledger Technology
32. DB – DataBase
33. KYC – Know Your Customer
34. DAO – Decentralized Autonomous Organization

Список литературы

1. Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfede, S., Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton University Press, 2016.
2. Nakamoto, S., “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.
3. https://bitcoin.org/bitcoin.pdf
4. Polyzos G. C., Fotiou N. Blockchain-assisted Information Distribution for the Internet of Things //2017 IEEE International Conference on Information Reuse and Integration (IRI). – IEEE, 2017. – p. 75-78.
5. https://www.hyperledger.org/category/hyperledger-sawtooth (date of request 25.03.2018)
6. Bakre A., Patil N., Gupta S. Implementing Decentralized Digital Identity using Blockchain. – 2017.
7. Zheng Z. et al. An overview of blockchain technology: Architecture, consensus, and future trends //Big Data (BigData Congress), 2017 IEEE International Congress on. – IEEE, 2017. – p. 557-564.
8. Wong, J. and Kar, I., “Everything you need to know about the Ethereum ‘hardfork,’” Quartz Media, July 18, 2016. https://qz.com/730004/everything-you-needto-know-about-the-ethereum-hard-fork/
9. Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., and Smith-Tone, D., National Institute of Standards and Technology (NIST), NIST InternalReport (NISTIR) 8105, Report on Post-Quantum Cryptography, April 2016. https://doi.org/10.6028/NIST.IR.8105 (date of request 23.04.2018)
10. https://github.com/hyperledger/sawtooth-core (date of request 05.05.2018)
11. Mell, P., Kelsey, J., and Shook, J., “Cryptocurrency Smart Contracts for Distributed Consensus of Public Randomness.” October 7, 2017. https://doi.org/10.1007/978-3-319-69084-1_31
12. National Institute of Standards and Technology (NIST), report on blockchain technology review – January 2018
13. Т. Е. Хакимжанов. Расчет аспирационных систем. Дипломное проектирование. Для студентов всех форм обучения всех специальностей. - Алматы: АИЭС, 2002.
14. Голубицкая Е.А. Экономика связи: М.: -Ирмас, 2006.
15. Фурсов В.Г. Финансовый менеджмент: конспект базовых лекций. - М.: МАИ, 2010.-125с.
16. Alekseeva N. A. “Deployment and improvement of reproductive chains as a result of the influence of the intellectual capital of small business // Multi-level social reproduction: questions of theory and practice” (2010) P. 7-11.
17. Tumanov D V “Development of the Information Society, Role in the Reproductive Process. // Multilevel Public Reproduction: Questions of Theory and Practice”(2013) P. 291 -300.

18. https://github.com/hyperledger/sawtooth-supply-chain (date of request 01.04.2018)
19. Vakhrushev D. S. “Self-organization and dynamic stability of economic systems: theoretical and methodological aspects” (2009) P. 345-407
20. Badev, A and M Chen : “Bitcoin: technical background and data analysis”, Finance and Economics Discussion Series, (2014) p.104
21. Chiu, J and T-N Wong: “E-money: efficiency, stability and optimal policy”, (2014) p. 12-14
22. Fung, Band H Halaburda : “Understanding platform-based crypto currencies”, (2014) p. 12–20
23. Gandal, N and H Halaburda : “Competition in the cryptocurrency market” (2014), pp. 33
24. Ciaian, P., Rajcaniova, M., & Kancs, D. A. The economics of Bitcoin price formation. Applied Economics. (2016): p.1799-1815.
25. Corbet S, Lucey B, Yarovya L. Datestamping the Bitcoin and Ethereum bubbles. Finance Research Letters. (2017) p.433-437.
26. Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In Proceedings of the 2016 ACM Conference on Economics and Computation,pages 365–382. ACM, 2016.
27. Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In International Conference on Financial Cryptography and Data Security, pages 264–279. Spring, 2017.
28. D. Monderer and L.S. Shapley. Potential games.Games and Economic Behavior, 14:p.124–143,1996.
29. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, p 6-8, 2008.
30. Steve Huckle, Rituparna Bhattacharya, Martin White, and Natalia Beloff. Internet of things, blockchain and shared economy applications. Procedia Computer Science, 98:461–466, 2016.
31. Lei Xu, Nolan Shah, Lin Chen, Nour Diallo, Zhimin Gao, Yang Lu, and Weidong Shi. Enabling the sharing economy: Privacy respecting contract based on public blockchain. In Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, pages 15–21. ACM, 2017.
32. Araz Taeihagh. Crowdsourcing, sharing economies and development. Journal of Developing Societies, 33(2):191–222, 2017.
33. Nick Szabo. Formalizing and securing relationships on public networks. First Monday, 2(9), 1997.
34. Jonathan Warren. Bitmessage: A peer-to-peer message authentication and delivery system. White Paper, 2012.
35. Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang. Consortium blockchain for secure energy trading in industrial internet of things. IEEE Transactions on Industrial Informatics, PP(99):1–1, 2017.

Приложение А

Листинг компонента app.js

```
// Select User
$('[name="keySelect"]').on('change', function () {
  if (this.value === 'new') {
    app.user = makeKeyPair()
    app.keys.push(app.user)
    saveKeys(app.keys)
    addOption(this, app.user.public, true)
    addOption('[name="transferSelect"]', app.user.public)
  } else if (this.value === 'none') {
    app.user = null
  } else {
    app.user = app.keys.find(key => key.public === this.value)
    app.refresh()
  }
})

// Create Asset
$('#createSubmit').on('click', function () {
  var asset = $('#createName').val()
  asset = asset.concat(" ",$('#boxNo').val()" ",$('#mission0rder').val()" ", $('#timeSealed').val()" ");
console.log(asset);
  const boxNo = $('#boxNo').val()
  const missionOrder = $('#mission0rder').val()
  const timeSealed = $('#timeSealed').val()
  if (asset) app.update('create', asset)
})

// Transfer Asset
$('#transferSubmit').on('click', function () {
  const asset = $('[name="assetSelect"]').val()
  const owner = $('[name="transferSelect"]').val()
  if (asset && owner) app.update('transfer', asset, owner)
})

// Accept Asset
$('#transferList').on('click', '.accept', function () {
  const asset = $(this).prev().text()
  if (asset) app.update('accept', asset)
})

$('#transferList').on('click', '.reject', function () {
  const asset = $(this).prev().prev().text()
  if (asset) app.update('reject', asset)
})
```

Приложение Б

Листинг компонента handlers.js

```
const { createHash } = require('crypto')
const { TransactionHandler } = require('sawtooth-sdk/processor')
const { InvalidTransaction } = require('sawtooth-sdk/processor/exceptions')
const { TransactionHeader } = require('sawtooth-sdk/protobuf')

// Encoding helpers and constants
const getAddress = (key, length = 64) => {
  return createHash('sha512').update(key).digest('hex').slice(0, length)
}

const FAMILY = 'transfer-chain'
const PREFIX = getAddress(FAMILY, 6)

const getAssetAddress = name => PREFIX + '00' + getAddress(name, 62)
const getTransferAddress = asset => PREFIX + '01' + getAddress(asset, 62)

const encode = obj => Buffer.from(JSON.stringify(obj, Object.keys(obj).sort()))
const decode = buf => JSON.parse(buf.toString())

// Add a new asset to state
const createAsset = (asset, owner, state) => {
  const address = getAssetAddress(asset)

  return state.get([address])
    .then(entries => {
      const entry = entries[address]
      if (entry && entry.length > 0) {
        throw new InvalidTransaction('Asset name in use')
      }

      return state.set({
        [address]: encode({name: asset, owner})
      })
    })
}

// Add a new transfer to state
const transferAsset = (asset, owner, signer, state) => {
  const address = getTransferAddress(asset)
  const assetAddress = getAssetAddress(asset)

  return state.get([assetAddress])
    .then(entries => {
      const entry = entries[assetAddress]
      if (!entry || entry.length === 0) {
        throw new InvalidTransaction('Asset does not exist')
      }
```

Приложение В

Листинг компонента index.html

```
<header>
  <div id="left_header">ADAMS</div>
  <i id="right_header">World Anti-Doping Agency</i>
</header>

<div id="body">
  <div id="left-column">
    <div class="form-group">
      <label>Doping Control Officer's public key</label>
      <select class="form-control" name="keySelect">
        <option value="none" selected>Select public key...</option>
        <option value="new">Create new DCF</option>
      </select>
    </div>

    <div class="form-group">
      <label>Doping Control Form's components</label>
      <input id="createName" class="form-control" type="text" placeholder="Enter athlete name...">
      <input id="boxNo" class="form-control" type="text" placeholder="Enter box number...">
      <input id="missionOrder" class="form-control" type="text" placeholder="Enter mission order...">
      <input id="timeSealed" class="form-control" type="text" placeholder="Enter sealed time...">
      <input id="createSubmit" type="button" value="Create" class="btn btn-primary">
    </div>

    <div class="form-group">
<!--      <label>Transfer Doping Control Form</label>
      <select class="form-control" name="assetSelect">
        <option value="none" selected>Select Doping Control Form...</option>
      </select>
      <select class="form-control" name="transferSelect">
        <option value="none" selected>Select ADAMS/DCO recipient...</option>
      </select>
      <input id="transferSubmit" type="button" value="Transfer" class="btn btn-primary">
    </div>

    <div class="form-group">
      <label>Accept DCF</label>
      <div id="transferList"></div>
    </div>
  </div>
->
  <div id="data">
    <label>ADAMS List</label>
    <table class="table table-hover">
      <tr>
        <th>Athlete name</th>
        <th>Box number</th>
        <th>Mission order</th>
```