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 алгоритмов консенсуса в сети blockchain»

Магистрант: Соснин К.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 Соснину Кириллу Андреевичу
(фамилия, имя, отчество)

Тема диссертации «Исследование алгоритмов консенсуса в сети blockchain»

Утверждена Ученым советом университета № 122 от 25 октября 2018

Срок сдачи законченной диссертации «25» мая 2020г.

Цель исследования состоит в исследовании существующих blockchain решений и построении приложения на основе технологии распределенных реестров.

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

1. Изучение принципов работы технологии blockchain

2.Сравнительный анализ производительности различных открытых blockchain платформ

3. Техническая реализация проекта на основе выбранной платформы

Перечень графического материала (с точным указанием обязательных чертежей)

Рисунок 1.1 – Клиент-серверная и P2P архитектура сети

Рисунок 1.5 – Хэширование блока

Рисунок 2.2 – Общая архитектура Hyperledger Fabric

Рисунок 2.5 – Пропускная способность платформ Etherium и Hyperledger Fabric

Рисунок 3.17 – Пользовательский интерфейс разработанного приложения

Рекомендуемая основная литература:

1. Bahga A., Madisetti V. Blockchain Applications: A Hands-On Approach. – 2017.- 167 стр.

2. . Hyperledger-fabricdocs Docomentation //Linux Foudation – 2015 – 141 стр.

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. Технические аспекты, образующие технологию blockchain (теоретическая часть)	12.02.2018	
3.Сравнительный анализ производительности платформ для создания blockchain приложений Hyperledger Fabric и Etherium	03.03.2018	
4. Изучение возможностей платформы Hyperledger Fabric	15.09.2019	
5. Создание приложения для записи и хранения информации об окончании студентами учебных заведений	09.11.2019	

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 _____ (Соснин К. А.)
(подпись) (Ф.И.О.)

Аңдатпа

Бұл жұмыста біз blockchain технологиясы негізінде студенттердің жоғары оқу орнын бітіргендігін растайтын куәлік алғандығы туралы ақпаратты тіркеу және сақтау үшін қолданбаны құруды қарастырамыз, бұл клиент-сервер логикасымен классикалық шешімдерден деректерді сақтау әдісімен түбегейлі ерекшеленеді. Блокчейн технологиясына және оның қазіргі шынайылықта қолданылуына жүйелік талдау жасалды. Блокчейн компоненттерін жүзеге асыру үшін математикалық модельдер мен алгоритмдері қарастырылған. Etherium және Hyperledger Fabric платформаларына қысқаша шолу жасалды, сонымен қатар Etherium және Hyperledger Fabric платформаларында құрылған сынақ қолданбаларының жұмысына салыстыру жүргізілді. Тапсырмаларды іс жүзінде орындау үшін қолданбаның клиенттік бөлігі, сондай-ақ кіріс операцияларын қабылдау және өңдеу логикасы бейімделді.

Аннотация

В настоящей работе рассматривается создание приложения для записи и хранения информации о получении студентами свидетельств, подтверждающих окончание Высшего учебного заведения на основе технологии blockchain, принципиально отличающегося от классических решений с клиент-серверной логикой методами хранения данных. Был проведен системный анализ технологии блокчейн и ее применимости в современных реалиях. Рассмотрены математические модели и алгоритмы для реализации компонентов блокчейн. Произведен краткий обзор платформ Etherium и Hyperledger Fabric, а также представлено сравнение производительности тестовых приложений, построенных на платформах Etherium и Hyperledger Fabric. Для практической реализации поставленных задач были адаптированы клиентская часть приложения, а также логика поступления и обработки входящих транзакций.

Abstract

In the present work it was considered creation of the application for recording and storage of the information on reception by students of the certificates confirming the termination of the Higher education institution on the basis of technology blockchain, essentially different from classical decisions with client-server logic methods of data storage. It was carried out the system analysis of blockchain technology and its applicability in modern realities. Mathematical models and algorithms for implementation of blockchain components are considered. It was made a brief review of Etherium and Hyperledger Fabric platforms, and a comparison of performance of test applications built on Etherium and Hyperledger Fabric platforms was presented. There were adapted client part of the application as well as the logic of receiving and processing incoming transactions for practical realization of the tasks.

Содержание

Введение

Оглядываясь назад на последние полвека компьютерных технологий и архитектур, можно наблюдать тенденцию колебания между централизацией и последующей децентрализацией вычислительной мощности, хранилища, инфраструктуры, протоколов и кода.

Мэйнфрейм был основной частью компьютерной сети. Он как правило обладает всей вычислительной мощностью, памятью, хранилищем данных и кодом. Доступ к мэйнфрейму осуществляется главным образом через «терминал», который поддерживает только ввод и вывод и не может хранить или обрабатывать данные.

С появлением персональных компьютеров и частных сетей клиенты теперь имеют одинаковую вычислительную мощность, а серверы размещаются на их собственном компьютере. Таким образом, разработчик архитектуры «клиент-сервер» поддержал разработку децентрализованной системы данных. Большие наборы данных, размещенные в мэйнфреймах, начали переходить в распределенные архитектуры. Эти данные могут реплицироваться с одного сервера на другой, а подмножества данных могут быть расположены и запущены на клиентах, а затем отправлены обратно на сервер.

Со временем архитектура Интернета и облачных вычислений стали поддерживать глобальный доступ к различным вычислительным устройствам; мэйнфреймы предназначены в первую очередь для удовлетворения потребностей крупных компаний и правительств. Несмотря на «аппаратное обеспечение» в облачной архитектуре, уровень концентрации приложений децентрализован (например, Facebook, Google и т. Д.). В чем блокчейн отличается от классической архитектуры, сделана ли корпоративная сетевая идентификация? Установление источника записи, после того как она осуществлена, может быть определено путем просмотра списка транзакций; таким образом, никакие изменения цепочки блоков уже невозможны. Каждая учетная запись участника имеет свою собственную копию текущего состояния сети.

Целью данной диссертации является изучение технологии блокчейн, ее технических аспектов, практическое применение и дальнейшее использование. Для реализации приложения были созданы записи успеваемости студентов. Для достижения этой цели необходимо решить следующие задачи:

- изучить технические основы технологии блокчейн;
- ознакомиться с существующими платформами, предлагающими решения блокчейн;
- произвести сравнительный анализ производительности различных платформ;
- создание приложения для записи и хранения информации об окончании студентами учебного процесса на основе выбранной платформы.

1 Теоретическая часть

1.1 Технология распределенных реестров

Сети P2P, как и другие распределенные системы, должны решать очень сложную проблему информатики: разрешение или согласование конфликтов. Реляционные базы данных предлагают ссылочную целостность, но в распределительной системе такой функции нет. Если два несовместимых факта поступают одновременно, система должна иметь правила для определения того, какой факт является правильнее.

Существует в основном два типа сетевых архитектур:

1. Клиент-серверная сеть;
2. Одноранговая сеть.

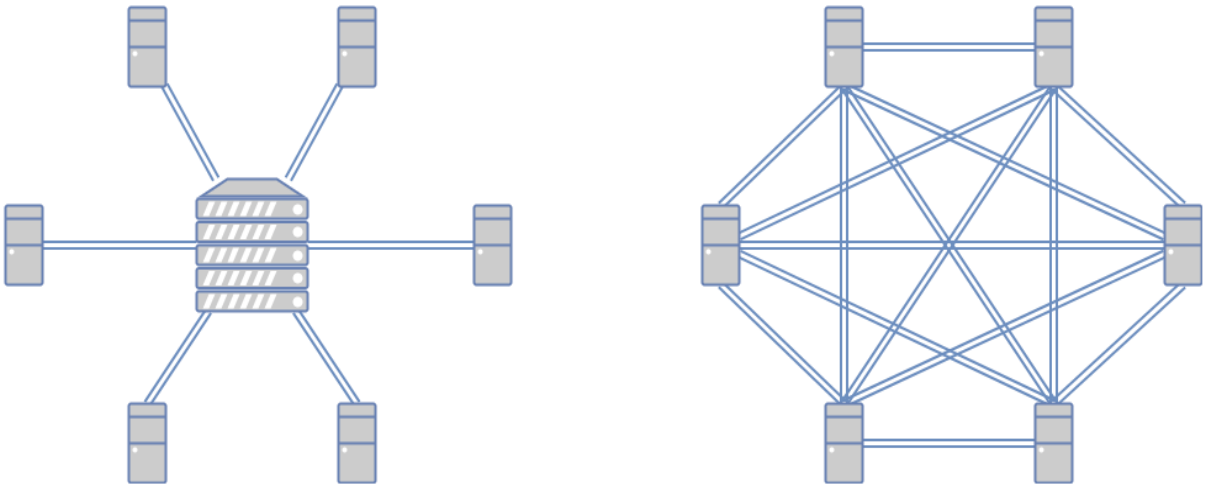

Рисунок 1.1 - Клиент-серверная и P2P архитектура сети

Сеть в первую очередь подразумевает централизованное управление всем: услугами, доступом, данными. Вся системная логика и информация скрыты внутри сервера, это позволяет снизить требования к производительности для клиентских устройств и обеспечить высокую скорость обработки. Именно этот метод стал наиболее распространенным сегодня.

Одноранговые или децентрализованные сети не имеют основного устройства, и все участники имеют равные права. В этой модели каждый пользователь является не только потребителем, но и сам становится поставщиком услуг.

Подавляющее большинство приложений и систем для нормальной работы требуют возможности оперировать набором данных. Существует множество способов организации такой работы, и один из них использует метод одноранговой сети[2]. Распределенные, или параллельные, базы данных

отличаются тем, что информация в частичном или полном составе хранится на каждом устройстве сети.

Одним из преимуществ этой системы является доступность данных: здесь нет единой точки отказа, как в случае с базой данных, расположенной на одном сервере. Такое решение также влечет за собой определенные ограничения на скорость обновления и распространения данных среди участников сети. Такая система не выдержит нагрузки миллионов пользователей, которые постоянно публикуют новую информацию.

Распределенный реестр - это тип структуры данных, которая находится на нескольких вычислительных устройствах, обычно распределенных по местоположениям или регионам.

Технология распределенного реестра включает в себя не только блокчейн и смарт-контракты. Она существовала до биткойна. Блокчейн биткойна представляет собой конвергенцию множества технологий, которая включает в себя временную метку транзакций, пиринговые (P2P) сети, хэш-функции и общую вычислительную мощность, а также совершенно новый алгоритм консенсуса.

Блокчейн фактически относится к числу видов распределенного хранения информации, использующих три ранее признанные технологии: одноранговые сети, базы данных, а также шифрование. База данных на самом деле представляет собой цепочку блоков, которая шифруется определенным образом и сохраняется практически на всех узлах сети в одном и том же типе (репликация фактически является полной копией). Весь секрет заключается в связях между блоками в результате криптографии, поэтому подделать информацию в блоках практически невозможно. Блокчейн позволяет вам легко распределять и, возможно, передавать информацию между несколькими людьми через ненадежную сеть. Появляется возможность передавать информацию, которая не требует третьей надежной стороны, подтверждающей подлинность данных. Примерами классической сети являются фактически наличные деньги (требуется участие Банка), имущественные права (требуется участие нотариуса), кредитный договор и т. д. По сути, блокчейн снимает требование о привлечении третьего доверительного управляющего.

Нововведение блокчейна состоит в использовании распределенного источника данных, формирующих блоки, которые фактически являются связанным контрольным списком (каждый следующий блок имеет идентификатор предыдущего). Каждая часть сети будет хранить копию всех операций, выполненных в течение всего времени. Это будет невозможно без особых инновационных разработок, созданных для обеспечения безопасности и эффективности сети[3].

Независимо от того, что интерес в технологии блокчейн гораздо больше связан с областью финансов, сфера разработки распределенных реестров не ограничивается ею. Вместе с банками и стартапами игроки из различных других нефинансовых рынков обратили свое внимание на технологию и ищут решения,

чтобы максимально использовать возможности, которые она предлагает. Давайте рассмотрим несколько интересных примеров практического применения технологии блокчейн, которые существуют вне сферы финансовых услуг.

Новые проекты на блокчейне будут основываться на его основных преимуществах: открытости и безопасности.

В связи с этим блокчейн станет отличным подспорьем для любых сервисов, где пользователям важна безопасность данных:

- микроплатежи;
- банковские операции;
- логистика;
- юриспруденция;
- медицина.

В телекоммуникационной индустрии блокчейн может быть использован в следующих областях: цифровая идентификация, управление данными (хранение различных документов, страхование, авиабилеты и др.), роуминг, 5G (выбор самого быстрого узла для связи), Умный город (полезен ввиду открытости и прозрачности), мобильная торговля, M2M и IoT (защищенное P2P-соединение для IoT-устройств с целью создания экономически жизнеспособной и самоуправляемой системы), Электронное здравоохранение (защищенное хранение медицинской информации в электронном виде). Роль блокчейна в обеспечении безопасности телекоммуникационных сетей сводится к тому, что технология распределенного реестра надежно хранит данные и результаты совместной деятельности устройств в системе. В случае взлома сети это никак не влияет на функционирование всей сети. Блокчейн в телекоммуникациях станет тем инструментом, который поможет операторам связи трансформироваться в настоящего поставщика цифровых услуг. Блокчейн помогает строить экосистемы, основанные на партнерском взаимодействии. Стоит отметить, что технология блокчейн находится только в начале своего развития. В ближайшие 5 лет будут проведены эксперименты, верификация и стандартизация блокчейн. Но сейчас телекоммуникационным компаниям необходимо сконцентрироваться на применении и внедрении этой технологии.

Эта информация о блокчейне помогает взглянуть на мир компьютерных технологий иначе, измерить их внутренний потенциал развития, сделать еще один шаг на пути к эволюции мышления.

Таким образом, технология распределенного реестра обычно состоит из 3 основных компонентов:

- информационная конструкция, которая фиксирует текущее состояние реестра;
- цепочка операций, которые изменяют состояние главного реестра;
- процесс, используемый для выработки консенсуса между участниками относительно того, что сделки будут подтверждены.

1.2 Технические характеристики блокчейна

1.2.1 Структура блока. Блокчейн – это подмножество технологий распределенного реестра, которые строят хронологическую цепочку блоков, отсюда и название "цепочка блоков". Блок относится к набору транзакций, которые объединяются вместе и добавляются в цепочку одновременно. В блокчейне биткоина узлы майнеров объединяют неподтвержденные и действительные транзакции в блок. Каждый блок содержит заданное количество транзакций. В сети биткоин майнеры должны решить криптографическую задачу, чтобы предложить следующий блок. Этот процесс известен как "доказательство работы" и требует значительных вычислительных мощностей[4].

Каждый блок состоит из адреса, даты и времени создания, хэша и списка транзакций:

- адрес - открытый ключ, генерируемый асимметричным алгоритмом шифрования (например, RSA), основанный на секретном ключе, сгенерированным пользователем;

- дата и время - момент создания блока (транзакция также имеет дату и время создания);

- хэш - вычисляется из адреса предыдущего блока и хэш-суммы всех транзакций текущего блока;

- информация - это любые данные, которые пользователь хочет сохранить (сумма денег (криптовалюты), документы, история болезней, программный код (смарт-контракты) и т.д).

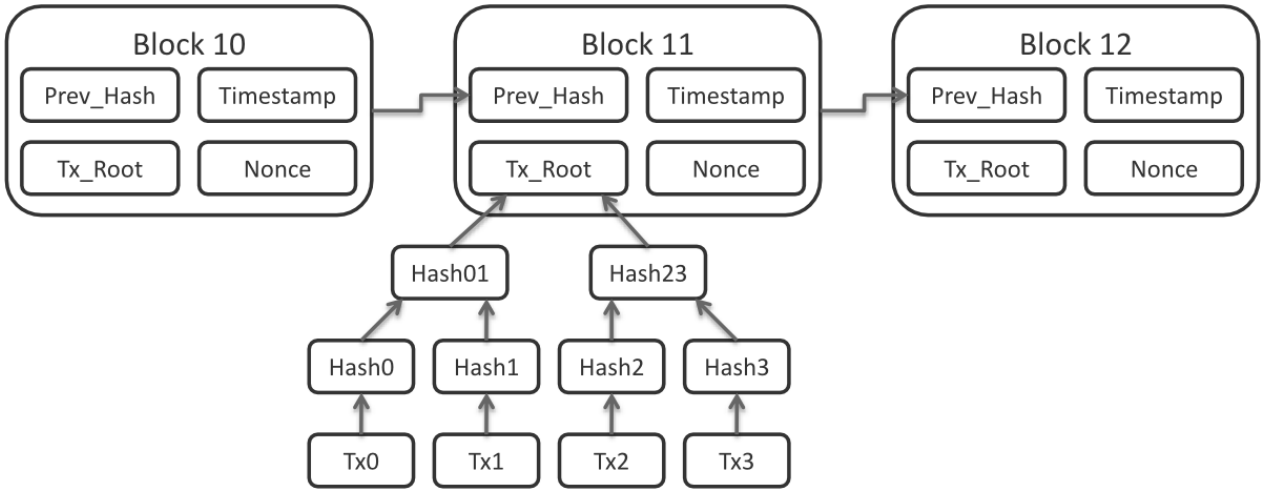

Рисунок 1.2 - Структура данных блока биткойн

Запись события, криптографически защищенная цифровой подписью, которая проверяется, упорядочивается и объединяется в блоки, образует транзакции в блокчейне. В биткойн - блокчейне транзакции включают передачу биткойнов, в то время как в других блокчейнах транзакции могут включать

передачу записи или, возможно, любого актива какой-либо оказываемой услуги [6]. Кроме того, смарт-контракт в рамках блокчейна может позволить автоматическое выполнение транзакций при соблюдении заранее определенных критериев.

1.2.3 криптография. Криптография играет решающую роль как в обеспечении безопасности, так и в неизменности транзакций, записанных в блокчейнах. Криптография включает в себя изучение методов, используемых для обеспечения безопасной связи между различными сторонами, участвующими в транзакциях, а также для обеспечения подлинности и неизменности передаваемых данных. Для блокчейн-технологий криптография фактически используется для подтверждения того, что транзакция была разработана достоверным участником сети. Она также используется для связывания транзакций в блок способом, защищенным от несанкционированного доступа, а также для создания связей между блоками, чтобы сформировать блокчейн.

Криптография является сердцем, которое обеспечивает работу системы. Архитектура блокчейна предполагает, что доверие между участниками сети основано на принципах математики и экономики, то есть оно формализовано. Криптография также гарантирует безопасность, которая основана на прозрачности всех операций[8].

Различные криптографические методы гарантируют неизменность списка транзакций блоков, также решают задачу аутентификации и контролируют доступ к сети и данным в блокчейне. Говоря о криптографии в блокчейне, речь идет о хэш-функциях, ключах и цифровых подписях. Неизменность информации и то, что хранится в блокчейне, вероятно, является одним из самых мощных, а также убедительных мотивов для развертывания основанных на блокчейне способов хранения информации для широкого спектра социально-экономических задач, которые в настоящее время фиксируются на централизованных серверах. Эта конкретная неизменность может сделать блокчейн полезным для бухгалтерского учета, денежных операций, управления идентичностью, а также владения преимуществами, передачи и управления. Когда транзакция фактически создается в блокчейне, никто не может ее изменить, или, во всяком случае, это будет невероятно трудно.

Невероятно трудно изменить транзакции внутри блокчейна, так как каждый блок на самом деле связан с предыдущим блоком с помощью хэша предыдущего блока. В случае, если одна транзакция изменялась, не только менялся хэш корня дерева Меркла, но и, следовательно, хэш, найденный в последующем блоке. Кроме того, каждый последующий блок должен быть постоянно обновлен, чтобы отразить изменения в последовательности транзакций. В ситуации доказательства выполнения, количество электроэнергии, необходимое для пересчета «nonce» из-за этого изменения, а также каждое последующее изменение будет запрещено[8]. С другой стороны, если злоумышленник действительно изменил транзакцию

внутри блока без какого-либо анализа необходимых мер для обновления последующих блоков, будет достаточно просто пересчитать хэши, примененные к блокам, и выяснить, какой узел сети пытался внести в блок недостоверные данные.

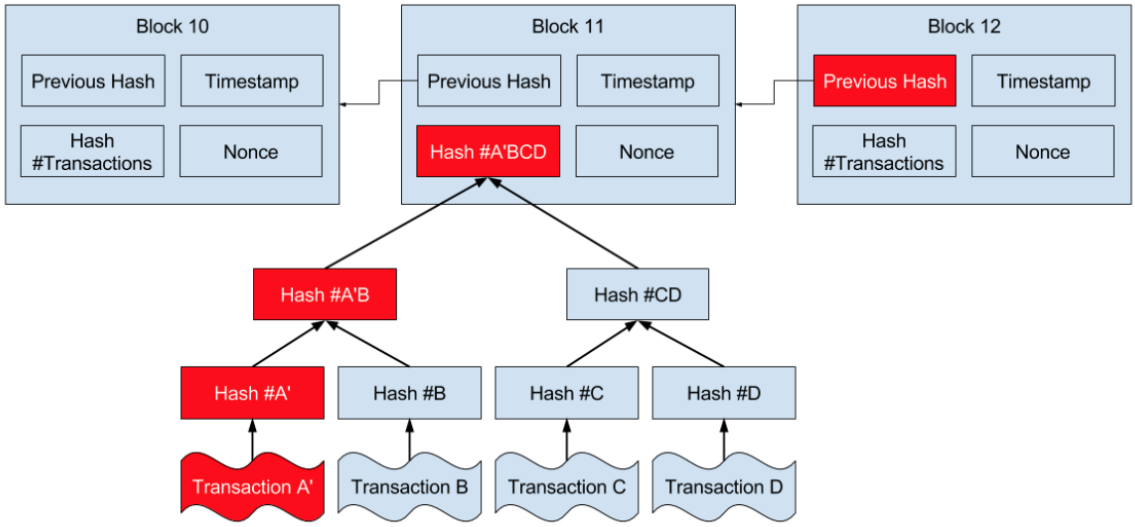

Рисунок 1.3 - Неизменность блокчейна

Основной целью использования криптографии с открытым ключом для блокчейна является создание безопасной цифровой ссылки на личность пользователя. Безопасные цифровые ссылки о том, кто есть кто и кому что принадлежит, являются основой для сделок P2P. Криптография с открытым ключом позволяет доказать свою личность с помощью набора криптографических ключей: закрытый ключ и открытый ключ. Комбинация обоих ключей создает цифровую подпись. Эта цифровая подпись доказывает право собственности на маркеры и позволяет управлять маркерами с помощью так называемого "кошелька". Цифровые подписи доказывают право собственности на токены и позволяют контролировать свои средства. Подобно тому, как мы подписываем банковскую транзакцию или чек вручную, или используем аутентификацию для интернет-банкинга, мы используем криптографию с открытым ключом для подписи транзакций блокчейна или других операций с блокчейном.

В криптографии с открытым ключом две стороны распространяют свои открытые ключи и позволяют любому человеку шифровать сообщения, используя свои открытые ключи. Открытый ключ математически генерируется из закрытого ключа. Хотя вычислить открытый ключ из частного ключа очень просто, обратное возможно только с помощью грубой силы; угадать ключ можно, но это потребует огромного количества вычислительных ресурсов. Следовательно, не проблема, если известен публичный ключ, но закрытый ключ всегда должен храниться в секрете. Это означает, что, даже если публичный ключ известен всем, никто не может извлечь из него закрытый ключ. Теперь сообщение может безопасно

передаваться владельцу закрытого ключа, и только владелец этого закрытого ключа может расшифровать сообщение, используя закрытый ключ, связанный с открытым ключом. Этот метод также работает наоборот. Любое сообщение, подписанное личным ключом, может быть проверено с помощью соответствующего открытого ключа. Этот метод также называется цифровой подписью.

1.2.4 хэш-функции. Хэширование - это процесс преобразования массива входных данных произвольной длины в (выходную) битовую строку фиксированной длины. Например, хэш-функция может взять строку с любым количеством символов (одна буква или целое литературное произведение) и получить на выходе строку со строго определенным количеством символов (дайджест). Хэш-функции доступны практически на любом языке программирования. Отдельной категорией хэш-функций являются криптографические хэш-функции. К ним предъявляются гораздо более жесткие требования, чем к функциям, обычно используемым в хэш-таблицах. Поэтому они используются и в более "серьезных" случаях, например, для хранения паролей. Криптографические хэш-функции разрабатываются и тщательно проверяются исследователями по всему миру [9].

Чтобы продолжать функционировать, блок должен создавать новые блоки. Поскольку блокчейны являются децентрализованными системами, новые блоки должны создаваться не единственным аутентифицирующим субъектом, а сетью в целом. Чтобы решить, каким будет новый блок, сеть должна прийти к консенсусу. Чтобы достичь консенсуса, майнеры предлагают определенные блоки, блоки проверяются, и, наконец, сеть выбирает единственный блок, который будет следующей частью реестра. Однако многие майнеры предлагают идентичные блоки, которые проходят проверку.

Хэш-функции гарантируют "необратимость" всей цепочки транзакций. Дело в том, что каждый новый блок транзакций ссылается на хэш предыдущего блока в реестре. Хэш самого блока зависит от всех транзакций в блоке, но вместо последовательной передачи транзакций хэш-функции они собираются в одно хэш-значение с помощью двоичного дерева с хэшами (Дерево Меркла). Таким образом, хэши используются в качестве замены указателей в обычных структурах данных: в связанных списках и бинарных деревьях.

Дерево Меркла, также называемое двоичным хэш-деревом, на самом деле является информационной системой, которая фактически используется для хранения хэшей информации в больших наборах данных методом, позволяющим эффективно проводить проверку набора данных. Это механизм защиты от несанкционированного доступа, чтобы убедиться, что более крупный набор данных не был изменен. Слово "дерево" фактически используется для обозначения ветвящейся информационной системы в информатике, как это показано на рисунке ниже [10].

Блокчейн состоит из различных блоков, которые связаны друг с другом (отсюда и название блок-цепочки). Хэш-дерево, или дерево Меркл, эффективно и безопасно кодирует данные в цепочке. Оно обеспечивает быструю проверку данных в цепочке, а также быстрое перемещение больших объемов данных с одного компьютерного узла на другой в одноранговой сети [9].

Каждая транзакция, происходящая в сети блокчейн, имеет связанный с ней хэш. Однако эти хэши хранятся на блоке не в последовательном порядке, а в виде древовидной структуры таким образом, что каждый хэш привязывается к своему родительскому узлу, следуя древовидной связи между родителями и детьми.

Поскольку в конкретном блоке хранится множество транзакций, все хэши транзакций в блоке также хэшируются, в результате чего получается корень Меркл.

Например, рассмотрим блок из семи транзакций. На самом низком уровне (называемом листовым) будет четыре хэша транзакций. На первом уровне, расположенном выше уровня листа, будет два хэша транзакций, каждый из которых будет соединяться с двумя хэшами, находящимися ниже них на уровне листа. На верхнем (второй уровень) будет последний хэш транзакции, называемый корнем, и он будет соединяться с двумя хэшами, находящимися под ним (на первом уровне).

Фактически, в данном случае получается перевернутое двоичное дерево, при этом каждый узел дерева соединяется только с двумя узлами, находящимися под ним (отсюда и название "двоичное дерево"). В верхней части дерева находится один корневой хэш, который соединяется с двумя хэшами первого уровня, каждый из которых снова соединяется с двумя хэшами третьего уровня (листового), и структура дерева продолжается в зависимости от количества хэшей транзакций.

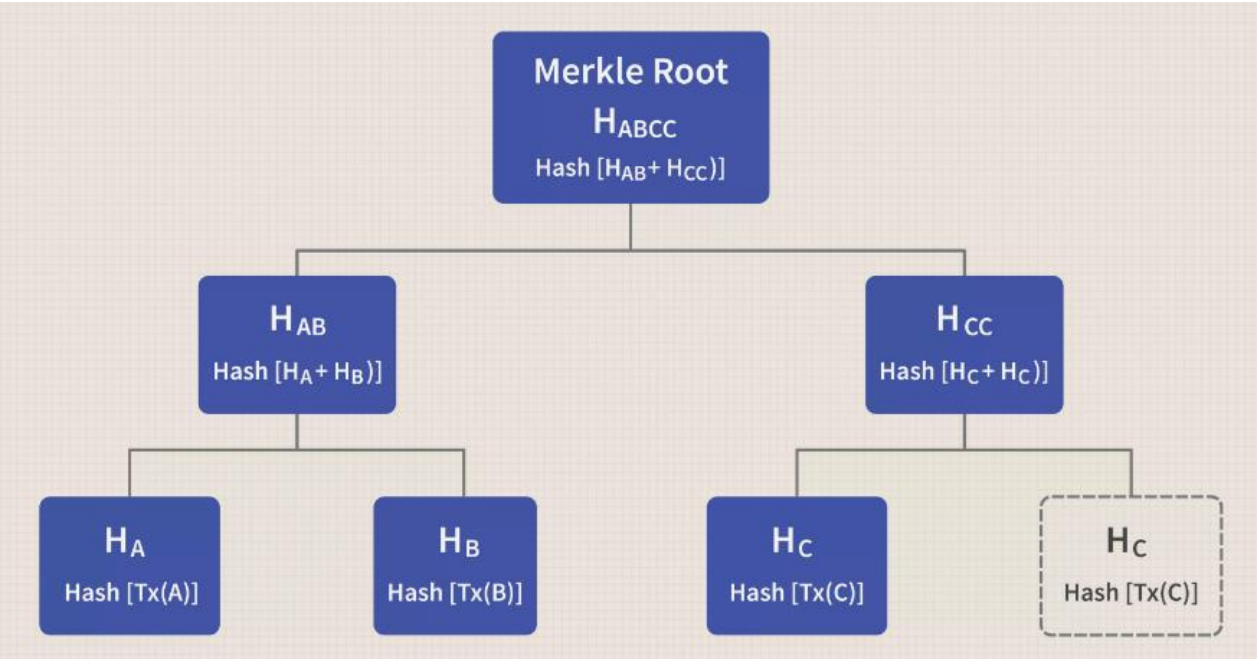

Рисунок 1.4 – Структура дерева хэша

Хэширование начинается в узлах нижнего уровня (на уровне листьев), и все четыре хэша включаются в хэш узлов, связанных с ним на первом уровне. Аналогичным образом, хэширование продолжается на первом уровне, что приводит к тому, что хэши достигают более высоких уровней, пока не достигают единственного верхнего корневого хэша.

Этот корневой хэш называется корнем Merkle, и благодаря древовидной связи хэшей, он содержит всю информацию о каждом хэше транзакции, существующем в блоке. Он предлагает одноточечное значение хэша, которое позволяет проверить все, что присутствует в этом блоке.

Чтобы понять, насколько важны деревья Меркл для технологии блокчейн, нужно представить себе блокчейн без них. В первую очередь будем ссылаться на Bitcoin, так как использование деревьев Меркла не только жизненно важно для крипто-валюты, но и легко понятно. Например, если бы в Биткойне не было Деревьев Меркл, каждый узел сети должен был бы хранить полную копию каждой транзакции, которая когда-либо происходила в Биткойне, а это невероятно огромное количество информации [10].

При подтверждении прошлой сделки, узел должен был бы связаться с сетью, чтобы получить копии распределенного реестра от других узлов сети. Узел должен был бы сравнивать каждую строку записи за строкой, чтобы удостовериться, что его собственные записи и сетевые записи точно совпадают. Если между реестрами возникнут какие-либо расхождения, это может поставить под угрозу безопасность сети.

Каждый запрос на проверку в Bitcoin требовал бы отправки по сети безумно больших пакетов информации, потому что для проверки данных нужно иметь сами данные. Компьютеру, используемому для проверки, нужно будет применить большую вычислительную мощность для сравнения копий реестров, чтобы убедиться, что в них не было изменений.

Деревья Меркл решают эту проблему. Они хэшируют записи в реестре, что эффективно отделяет доказательства данных от самих данных. Доказательство того, что сделка действительна, включает в себя только отправку небольшого количества информации по сети. Кроме того, это позволяет доказать, что обе версии блокчейна одинаковы с номинальными объемами вычислительной мощности и пропускной способностью сети.

Хэш-задачи доступны для использования практически на всех языках программирования. Например, они используются для расчета хэш-таблиц, а также наборов (HashMap / HashSet в Java, set и dict в Python, Map, Objects и Set в JavaScript и т.д.). Специфической категорией хэш-задач являются криптографические хэш-функции. К ним предъявляются гораздо более жесткие требования, чем к функциям, широко используемым в хэш-таблицах. Таким

образом, они используются в более "серьезных" случаях, например, для сохранения паролей. Криптографические хэш-задачи создаются и полностью изучаются исследователями по всему миру [10].

Все транзакции, которые были выполнены, и их последовательность будут передаваться общим значением: хэшем самого последнего блока. Таким образом, свойство инвариантности хэша 1 транзакции обещает неизменность всего блока.

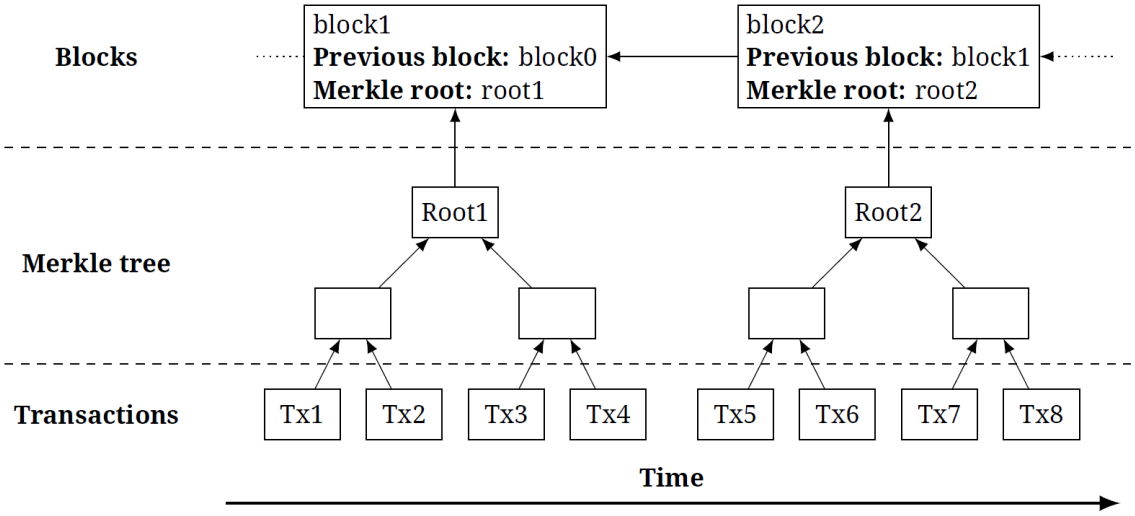

Рисунок 1.5 – Хэширование блока

Метка времени - еще одна ключевая особенность технологии блокчейн. Каждый блок имеет отметку времени, причем каждый новый блок ссылается на предыдущий блок. В сочетании с криптографическими хэшами эта временная цепочка блоков обеспечивает неизменную запись всех транзакций в сети, начиная с самого первого блока.

1.2.5 Цифровые Подписи. Чтобы информацию внутри транзакций нельзя было подделать, каждая транзакция внутри блока подписывается электронной цифровой подписью (ЭЦП).

Электронно-цифровая подпись – это последовательность байтов, формируемая путем преобразования подписываемой информации по криптографическому алгоритму и предназначенная для проверки авторства электронного документа.

ЭЦП основывается на использовании асимметричного шифрования и хэш-функциях.

Кратко о методах шифрования:

- симметричное шифрование использует один и тот же ключ и для зашифровывания, и для расшифровывания;
- асимметричное шифрование использует два разных ключа: один для зашифровывания (который также называется открытым), другой для расшифровывания (называется закрытым).

В асимметричных алгоритмах шифрования, шифрование производится с помощью открытого ключа, а расшифровка с помощью закрытого.

Но в асимметричных схемах цифровой подписи подписание производится с применением закрытого ключа, а проверка подписи — с применением открытого, то есть шифруем закрытым, а проверяем открытым

Одним из таких алгоритмов может быть RSA. Выбор ассиметричного шифрования обосновывается тем, что другие участники сети должны убедиться в том, что именно владелец блока внес изменения и подписал блок своей подписью

Цифровые подписи в блокчейне основаны на криптографии с открытым ключом. Они использует два ключа. Первый - закрытый ключ - необходим для генерации цифровых подписей и держится в секрете [11]. Второй - открытый ключ - используется для проверки электронной подписи. Открытый ключ действительно может быть вычислен на основе закрытого ключа, но обратное преобразование требует огромного количества вычислений, сравнимого с брутфорсингом.

Существует множество различных схем криптографии с открытым ключом. Двумя наиболее популярными из них являются схемы, основанные на факторинге (RSA), и схемы, основанные на эллиптических кривых. Последние более популярны в блокчейне из-за меньшего размера ключей и подписей. Например, в биткойне используется стандарт эллиптической криптографии ECDSA вместе с эллиптической кривой secp256k1. В нем закрытый ключ имеет длину 32 байта, открытый - 33 байта, а подпись - около 70 байт.

1.2.6 Закрытый и открытый ключи. Закрытый ключ генерируется самим пользователем и используется для подписания транзакций. Он держится в секрете, и тот, кто владеет секретным ключом, имеет доступ к информации, которая может быть представлена кошельком, контейнером с любыми данными (например, личной перепиской, важными документами и т. д.) [13].

Открытый ключ должен быть сгенерирован на основе закрытого ключа, то есть между ними существует математическая связь (открытый ключ не придумывается из головы). Он может быть опубликован, более того, он используется как адрес блока, а также как проверка подлинности подписи информации в других блоках, сторонними участниками сети. Знание открытого ключа не дает возможности определить закрытый ключ [14].

Чтобы создать подпись, вам понадобится:

- асимметричный алгоритм шифрования (например, RSA);
- хэш-функция (например, SHA512);
- информация, которую мы собираемся подписать.

Поскольку асимметричные алгоритмы довольно медленны по сравнению с симметричными алгоритмами, объем подписываемых данных играет большую роль, и если он велик, то они обычно берут хэш из подписываемых данных, а не сами данные. Хэш получается с помощью хэш-функций, например, SHA512,

который принимает некоторую информацию на входе и возвращает хэш определенной длины. Хэш-функция как мясорубка, вы можете прокрутить мясо и получить фарш, но обратно из фарша уже мясо не получится. Таким образом, ЭЦП размещается не на самом документе, а на его хэше. Хэш-функции не являются частью алгоритма ЭЦП, поэтому в схеме может быть использована любая надежная хэш-функция [15].

Этапы:

1) используя RSA, генерируется пара открытых и закрытых ключей;

2) подписанные данные подставляются в функцию SHA512 и мы получаем хэш;

3) помещаем полученный хэш и закрытый ключ в функцию асимметричного шифрования RSA, то есть RSA Encode (хэш информации, закрытый ключ), получаем строку - EDS на выходе.

1.3 Алгоритмы Консенсуса

Блокчейн — это распределенная система, в которой могут находиться тысячи участников. В отличие от обычных распределенных баз данных, в блокчейне почти всегда отсутствует центральный администратор, который конфигурирует узлы сети, поэтому получается, что архитектура блокчейна не просто распределена, но децентрализована. В связи с этим для блокчейна является актуальной задача распределенного консенсуса:

«Как узлам сети достичь одинаковой точки зрения на журнал транзакций блокчейна в распределенной сети при условии, что произвольные узлы могут «падать» или зависать, руководствуясь лишь общими правилами обработки сообщений в сети?»

В любой блокчейн-сети передаются два основных типа сообщений — транзакции и блоки (которые, в свою очередь, являются списками транзакций). Транзакции формируются участниками системы и их алгоритм консенсуса не касается: для того чтобы инициировать, скажем, отправку биткойнов, никакого соглашения не надо, достаточно знать правильный ключ. Блоки — совсем другое дело. Они являются основным продуктом алгоритма консенсуса и определяют, в каком порядке транзакции будут включены в журнал транзакций [15].

Можно ли обойтись без блоков, включая транзакции в журнал по отдельности? Теоретически да, но на практике блоки экономят объем трафика и вычислительные ресурсы узлов сети. Кроме того, у них есть и другие преимущества в контексте конкретных алгоритмов консенсуса — оказывается, что со слишком частыми блоками работа блокчейна становится нестабильной.

Блоки создаются особой категорией узлов сети блокчейна — так называемыми узлами консенсуса. В случае биткойна и других криптовалют эти узлы называют майнерами, поскольку они вознаграждаются за свою работу (майнинг) генерацией новых порций криптовалюты. Майнеры активно участвуют

в формировании блокчейна, постоянно группируя входящие транзакции в блоки и распространяя их по сети.

Второй тип — узлы аудита. Они не участвуют в процессе консенсуса, однако имеют у себя полную копию блокчейна [15]. «Аудиторы» регулярно проверяют работу майнеров и занимаются распределением нагрузки по сети, выполняя функцию своеобразной сети доставки контента (CDN) для данных блокчейна.

Третий тип узлов — это легкие клиенты. Легкими они называются потому, что не имеют полной версии блокчейна и содержат лишь те данные, которые важны для узла. По этой причине они являются хорошим вариантом для организации криптовалютного кошелька — всей картины сети такой клиент не даст, но позволит эффективно отслеживать баланс пользователя. Легкие клиенты требуют меньше вычислительных ресурсов и объемов памяти, поэтому могут работать на мобильных платформах.

Задача распределенного консенсуса не специфична для блокчейнов и имеет хорошо проверенные решения для многих других распределенных систем (например, баз данных NoSQL). Даже задача консенсуса, в котором узлы могут вести себя «по-плохому», — задача византийского консенсуса — впервые была сформулирована в 80-х годах прошлого века, а методы её решения появились в конце 90-х.

Но биткойн и другие блокчейны от предыдущих наработок отличаются условиями работы сети. В обычных алгоритмах византийского консенсуса у узлов сети есть «личности», выражаемые через цифровые подписи или сходные криптопримитивы, а сам список узлов известен заранее или меняется редко, но предсказуемо. В биткойн-блокчейне все наоборот.

Участники сети не только заранее неизвестны, но и могут свободно подключаться или отключаться от сети. При этом блокчейн, являясь децентрализованной системой, имеет определенные свойства: устойчивость к цензуре (никто не может запретить майнить криптовалюту) и объективность (для определения текущей версии журнала транзакций не нужно доверие неким авторитетным источникам — корень доверия находится в самом блокчейне) [16].

Из-за этого обычные алгоритмы византийского консенсуса для блокчейна не подходят. Поэтому было предложено множество различных алгоритмов, среди которых выделяются две основные категории: алгоритмы на основе доказательства работы (proof-of-work) и алгоритмы на основе подтверждения доли (proof-of-stake).

Консенсус в сети относится к процедуре получения согласия с участниками сети относительно соответствующего статуса деталей продукта. Консенсус приводит к тому, что другие узлы совместно используют те же самые детали. Таким образом, алгоритм консенсуса делает две вещи: он гарантирует, что информация в реестре фактически одинакова для всех узлов в сети, а также, в

свою очередь, останавливает злоумышленников от корректировки информации. Алгоритм консенсуса варьируется в зависимости от различных реализаций блокчейна.

Блокчейн биткойна использует доказательство работы(Proof of Work), потому что блокчейн и распределенные реестры фактически развертывают кучу алгоритмов консенсуса, подобных Proof of Stake, Proof of Burn, Proof of Capacity, Proof of Elapsed Time, а также многим другим, основанным на уникальных потребностях.

1.3.1 Доказательство работы (PoW). Алгоритм Proof of Work (PoW) состоит из решения вычислительно сложной задачи, чтобы иметь возможность производить совершенно новые блоки в блокчейне биткойна. В разговорной речи задача фактически признается "майнингом", а также узлы в сети, которые принимают участие в майнинге, на самом деле широко известны как "майнеры". Мотиватор для майнинговых транзакций заключается в том, чтобы получить экономическую выгоду, именно там, где конкурирующие майнеры фактически получают компенсацию в размере 12,5 биткойнов и крошечную плату за транзакцию [16].

Блокчейн-протоколы должны обеспечивать консенсус среди нод децентрализованной системы. Пожалуй, самым известным алгоритмом консенсуса можно считать «тормозунутый, но надежный, потому что тормознутый» алгоритм Proof-of-Work: каждая нода, имея набор новых транзакций перебирает некоторое число nonce, являющееся полем блока. Блок считается валидным, если валидны все транзакции внутри него и хэш-функция от заголовка блока имеет некоторую общепринятую особенность (например, количество нулей в начале, как в Bitcoin):

Hash(Block{transaction,nonce,…}) = 000001001...

Как известно, блокчейн — это цепочка блоков. Цепочкой он является потому, что внутри каждого блока записан id (как правило хэш от заголовка) предыдущего блока.

В процессе синхронизации ноды с другими нодами, ей необходимо осуществить валидацию всех блоков, которые ей прислали соседи – проверить хэши и транзакции всех новых блоков, а в случае первого подлючения, до самого первого блока (genesis -block). Нетрудно предположить, что это достаточно длительный и затратный процесс.

Хэш-функция от заголовка блока является его id. Как было сказано ранее, в сети Bitcoin, как и во многих других сетях, особенностью, по которой определяется валидность блока, является число нулей в начале записи id. Это известное и общее для всех майнеров число нулей, называют сложностью майнинга T (mining target). Валидный хэш с T нулями в начале может иметь больше нулей в начале, чем T. Конкретнее, половина блоков будет иметь только T нулей в начале; половина блоков будет иметь T+1 нуль в начале; четверть блоков T+2 нулей и.т.д. Например так может выглядеть набор валидных блоков для T = 5:

000000101… (6 нулей)
000001110… (5 нулей)
000001111… (5 нулей)
000000010… (7 нулей)
000000101… (6 нулей)
000001110… (5 нулей)
000001111… (5 нулей)

Количество нулей, превышающее T в id блока назовем уровнем μ, а блоки с уровнем μ будем называть μ — суперблоками. Если блок является μ — суперблоком, то он так же является и(μ -1)-суперблоком. Таким образом, пока ничего не изменяя, а лишь оперируя введенным параметром μ, можем представить цепочку блоков в следующем μ-уровневом виде:

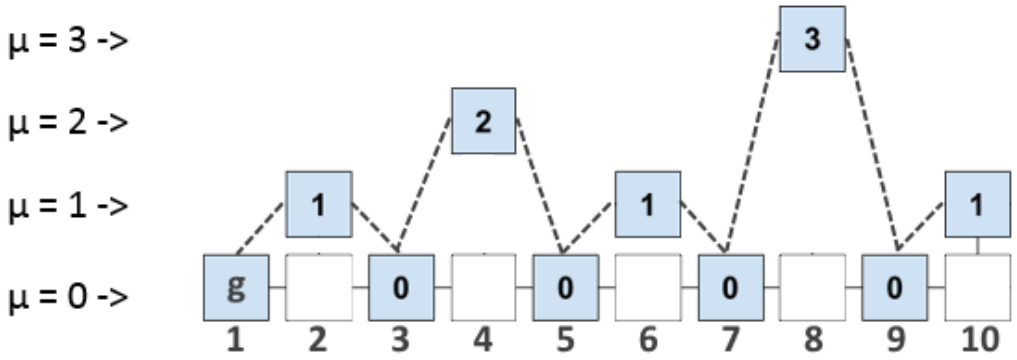

Рисунок 1.6 – Структура блоков по уровням

Блоки пронумерованы для простоты описания, нумерация не несет смысловой нагрузки.

Теперь подумаем, как мы можем это использовать. Если в заголовок каждого блока записывать не только id предыдущего блока, но и id всех последних блоков на каждом уровне, то мы позволяем каждому блоку ссылаться на более «древние» блоки, чем предыдущий. Набор всех последних на каждом уровне блоков будем называть interlink (множественная ссылка). Например, Interlink для блока 8 выглядит так:

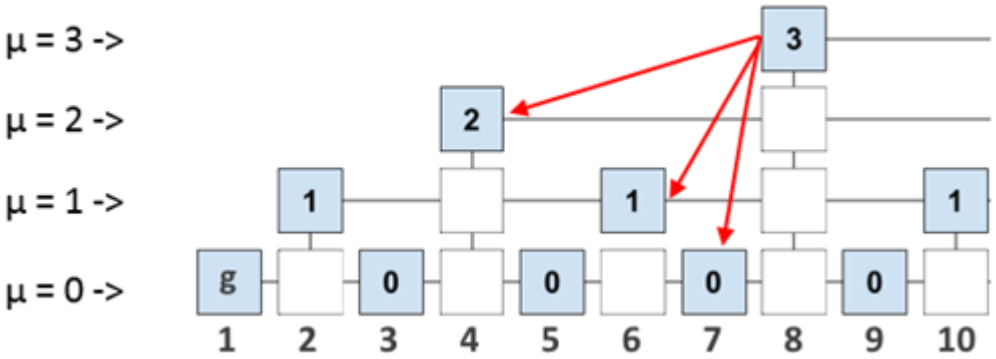

Рисунок 1.7 – Множественная ссылка для блока 8

Допустим, мы подключили новую ноду и теперь хотим безопасно синхронизировать её. Как мы уже сказали, для полноценной валидации нового блока, ноде нужно «прошагать» до genesis – блока по всему блокчейну [16]. Однако, если мы будем иметь в валидируемом блоке ссылки на некоторые «опорные» блоки, то сможем «прошагать» до genesis – блока, запросив у других нод не весь блокчейн, а лишь некоторое доказательство (proof), которое будет содержать короткий маршрут до самого первого блока. Сам маршрут будет являться валидной подцепочкой самого блокчейна, так как блоки подцепочки последовательно ссылаются друг на друга.

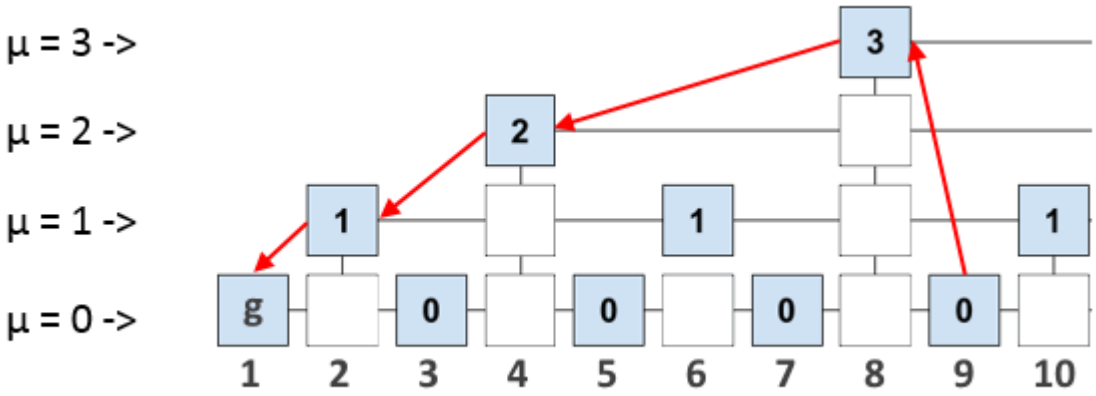

Рисунок 1.8 – Маршрут proof от 9 блока до первого блока

Доказательство – это набор заголовков нескольких предыдущих блоков. Строго говоря, доказательство содержит не только «короткий маршрут», но и ещё несколько заголовков других блоков. Это сделано для верификации доказательства (verify) по задаваемым параметрам безопасности m, k и др.

Нода, которая не хранит весь блокчейн, а лишь запрашивает proof у full-нод, хранящих всю историю, называется PoPoW – нодой. Теоретически такую ноду можно развернуть на маломощном компьютере, смартфоне.

Алгорим работы PoPoW-протокола следующий:

1. PoPoW-нода запрашивает доказательство для блока у full – ноды.

2. Full – нода (proover) формирует доказательство и отправляет его.

3. PoPoW-нода (verifier) проверяет доказательство, сопоставляет с доказательствами других нод и делает заключение о валидности блока.

Также стоит отметить, что сложность создания PoPoW доказательства не уступает сложности создания полной цепочки из валидных заголовков (хэш функция заголовка содержит Interlink, поэтому «подделывать» блоки нечестной ноде пришлось бы с учетом μ-уровневой иерархичности). Поэтому использование для валидации блока PoPoW доказательства не влечет потери безопасности [17].

Несколько критических замечаний можно найти для алгоритма Proof of Work. PoW требует участия большого количества электроэнергии при условии использования вычислительно важного алгоритма. Кроме того, алгоритм имеет чрезмерную задержку подтверждения транзакций, а также основное внимание компаний майнеров уделяется местам, где электричество фактически стоит недорого. Что касается мер сетевой безопасности, то PoW фактически уязвим для "атаки пятидесяти одного процента", которая может быть в теории осуществлена командой майнеров, контролирующих более пятидесяти процентов вычислительной энергии этой сети.

1.3.2 Доказательство доли владения (Proof-of-Stake - PoS). Доказательство доли владения - это другой вид механизма консенсуса, который можно использовать для согласования единственной верной записи истории данных. В то время как в PoW майнеры тратят энергию (электричество) на обновление существующих блоков, в PoS валидаторы берут на себя обязательства по подтверждению (или "валидации") существовавших блоков.

Валидаторы - это участники сети, которые управляют узлами (называемыми узлами валидаторов) для предложения и подтверждения блоков в цепочке блоков PoS. Они делают это с помощью криптографического стека (в случае Ethereum 2.0, ETH) в сети и делают себя доступными для случайного выбора, чтобы предложить блок. Другие валидаторы затем "подтверждают", что они получили новый блок. Когда для блока собрано достаточное количество подтверждений, блок добавляется в блокчейн. Валидаторы получают вознаграждение как за успешное предложение блоков (так же, как и в PoW), так и за подтверждение того, что они подтвердили блок [18].

Крипто-экономические стимулы для PoS предназначены для создания более убедительных вознаграждений за правильное поведение и более строгих наказаний за злонамеренное поведение. Вместо того, чтобы учитывать вторичную стоимость электроэнергии для работы узла PoW, валидаторы в цепочках PoS вынуждены напрямую вносить значительную денежную сумму в сеть.

Валидаторы начисляют вознаграждение за изготовление блоков и аттестации, когда наступает их очередь. Они наказываются за невыполнение своих

обязанностей, когда наступает их очередь - т.е. если они находятся в автономном режиме. Штрафы за то, что они находятся вне сети, относительно мягкие и равны примерно тем же, что и ожидаемые вознаграждения с течением времени. Однако, если валидатор попытается атаковать или скомпрометировать блокчейн, пытаясь предложить новый набор данных истории, срабатывает другой механизм штрафов: существенная часть их ставки будет урезана (возможно, до полной суммы ставки), и они будут выброшены из сети. В результате возникает огромный финансовый риск неудачной атаки со стороны валидатора. Более того, такая архитектура передает безопасность сети непосредственно в руки тех, кто обслуживает сеть и держит в самом протоколе ее собственный крипто-актив.Proof of Stake решает три обсуждавшиеся ранее проблемы сетей PoW - доступность, централизация и особенно масштабируемость:

Доступность: Доказательство доли владения не требует от валидаторов беспокоиться о первоначальных затратах на оборудование или обращать внимание на тарифы на электроэнергию так же, как это должны делать майнеры в сетях PoW. Следовательно, здесь значительно более низкий барьер для входа для человека, чтобы запустить узел валидатора в цепи PoS, чем для майнера в цепи PoW. Однако существует заметный барьер для входа в PoS. Валидаторы должны поставить минимальное количество криптографической суммы, чтобы запустить полный узел валидатора. Для Ethereum 2.0, например, эта сумма равна 32 ETH [18]. Для многих это значительная сумма денег и сдерживающий фактор активного участия. Точно так же цепочки PoW имеют добывающие пулы, однако, будут иметь стейк-пулы, которые объединяют средства участников, не способных или не желающих делать ставку 32 ETH. Пул будет делать ставки от их имени, и они будут получать вознаграждение в процентах от своей доли.

Централизация: С уменьшением барьеров для входа и устранением опасений по поводу минимизации затрат на электроэнергию, сети PoS значительно более децентрализованы на уровне узлов, чем сети PoW. Для участия в сети PoS требуется только ненулевое количество криптографической информации, подключение к Интернету и компьютер (или телефон/планшет). Это открывает двери участия и получения дохода для гораздо большей группы людей. Кроме того, экономия от масштаба гораздо ниже в PoS, чем PoW. В системах PoW, чем больше хэш-мощность, контролируемая майнером, тем больше процент вознаграждения, которое он сможет получить. В PoS процент возврата валидатора остается постоянным, независимо от того, управляет ли он 1 узлом или 1000.

Масштабируемость: архитектура PoS позволяет реализовать решение для масштабируемости, известное как шардинг, без снижения уровня безопасности. Шардинг - это механизм масштабирования базы данных, в котором блокчейн разбивается на несколько шардинговых цепочек, каждая из которых способна обрабатывать блоки. Это избавляет блокчейн от необходимости обрабатывать каждый блок одновременно, а вместо этого позволяет обрабатывать несколько

блоков (и, другими словами, больше наборов данных) одновременно. Например, в Ethereum 2.0 шардинг разбивает блок-цепочку на 64 отдельных шарда.

1.3.3 Доказательство истекшего времени (PoET). PoET был разработан гигантом по производству микросхем Intel еще в 2016 году как эффективный механизм консенсуса, в первую очередь для блокчейн сетей [19]. В настоящее время PoET является моделью консенсуса для модульного фреймворка Hyperledger и популярным инструментом для реализации и экспериментов с распределенными системами ведения распределенных реестров.

Важнейшим компонентом консенсуса PoET является инновационная технология, с которой она работает в сочетании - Software Guard Extensions (SGX). Представленное в 2015 году с процессорами Intel Core Processors 6-го поколения, SGX функционирует как доверенная среда выполнения (Trusted Execution Environment - TEE), которая позволяет выбирать доверенный код для выполнения независимо от приложения, в котором он запущен.

Прежде чем понять, как работает общий консенсус PoET, необходимо немного разобраться в том, как работает SGX.

SGX - это сложная технология, но по своей сути она представляет собой набор инструкций для процессора, которые используются приложениями для изоляции определенных, доверенных участков кода и данных. Код, который выполняется в TEE с использованием SGX, может генерировать подписанный аттестат внутри платформы или приложения, которое напрямую обращается к процессору и обеспечивает аутентификацию. Эта функция имеет значительные последствия для функциональности консенсуса PoET, но также создает барьер для входа и ограничения его использования [19].

Память, в которой хранится защищенный код в SGX, даже защищена от злоумышленников, которые контролируют физический доступ к платформе и имеют самую высокую степень аутентификации для доступа к ее памяти.

В контексте консенсуса PoET, SGX функционирует как механизм, позволяющий участникам присоединиться к сети и проверить, что они запустили доверенный код, необходимый для выполнения консенсуса PoET.

Консенсус PoET является эффективной формой доказательства работы, которая устраняет необходимость в трудоемком процессе решения криптографических задач и заменяет его системой рандомизированных таймеров для участников сети. В принципе, каждый участник сети получает случайный объект таймера и первый таймер, который истекает сигнализирует о том, какой именно узел сети будет формировать новый блок [20].

Что касается византийской отказоустойчивости, то такой механизм алгоритма консенсуса обеспечивает необходимое и эффективное рандомизированное решение "проблемы случайного выбора лидера".

Сеть работает следующим образом:

1. Узел загружает код PoET и генерирует подтверждение (ключ) кода с помощью SGX.
2. Узел пересылает этот ключ при запросе на подключение к сети. Узлы, которые уже являются частью сети, проверяют этот ключ.
3. Новый узел теперь имеет свой объект таймера, который инициализируется случайным значением. Эта случайность гарантируется кодовой защитой, предлагаемой SGX.
4. Все узлы инициализируются случайным временем; победителем становится тот, срок действия которого истекает первым. Это означает, что он создает новый блок, присоединяет его к текущему блоку и получает вознаграждение. Затем узлы инициализируются снова.

PoET является отличной альтернативой другим алгоритмам консенсуса. В то же время, он также обеспечивает отличное решение "проблемы случайного выбора лидера", не будучи ресурсоемким и не требуя сложной механики и структур стимулирования, необходимых с доказательством консенсуса внутри алгоритма PoW.

PoET также является отличным механизмом консенсуса для разрешенных сетей, вот почему он является механизмом консенсуса для Hyperledger. Кроме того, он эффективно масштабируется и может использоваться в качестве модели "подключи и работай" для тестирования сред с Hyperledger [20].

1.4 Смарт-Контракты

Смарт-контракты - это компьютерные программы, которые выполняют предопределенные действия при выполнении определенных условий внутри системы. Смарт-контракты предоставляют язык транзакций, позволяющий изменять состояние распределенного реестра. Они могут способствовать обмену и передаче чего-либо ценного (например, акций, денег, контента, имущества).

Чтобы иметь возможность заключить классический договор, вы должны посетить адвоката или, возможно, нотариуса, заплатить и подождать, пока вам передадут документы. Разумные контракты работают как торговые автоматы: вы просто бросаете плату в машину (которая на самом деле является реестром), а также соглашение, поддерживаемое третьей стороной, и, возможно, какую-то дополнительную услугу, которую вы приобрели, которые затем попадают на ваш банковский счет [21].

Кроме того, в отличие от обычных соглашений, смарт-контракты не только содержат информацию о штрафах сторон, но также и об ответственности за их нарушение, но и мгновенно гарантируют выполнение всех условий соглашения [21].

В конечном итоге именно эта система подтверждает выполнение условий договора и мгновенно определяет, должна ли указанная награда в виде криптовалюты либо другой вид активов перейти к одному из участников сделки

или, возможно, быстро вернуться к другому участнику (а может быть, обстоятельства несколько сложнее). Все это время документ фактически сохраняется, а также дублируется в децентрализованном реестре, что гарантирует его сохранность и не позволяет ни одной из сторон изменять условия соглашения.

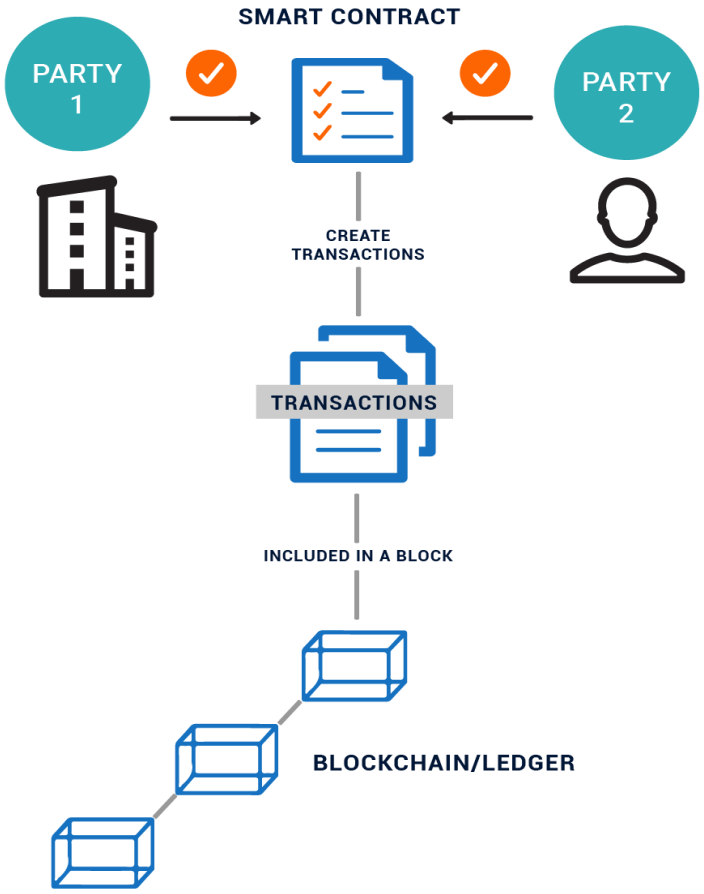

Рисунок 1.9 - Структура смарт-контрактов

Когда мы говорим о распределенных вычислениях, то обычно понимаем распределение нагрузки. Допустим, перед нами стоит задача вычислить вхождение конкретного слова в огромный текстовый файл. Мы разрезаем файл на несколько частей, распределяем эти части по разным узлам (например, с помощью Hadoop), каждый выполняет подсчет и выдает ответ. Мы суммируем ответы и получаем результат. Таким образом, мы значительно ускоряем выполнение этой задачи [21].

Если говорить о смарт-контрактах, то это совершенно другая ситуация. Здесь мы не режем файл на отдельные части, мы даем каждому узлу целый файл, и каждый узел дает нам один и тот же результат (в идеале). Давайте вернемся к нашему вопросу: попадает ли такое действие под определение распределенных

вычислений? Только в данном случае речь идет не о распределении нагрузки, а о распределении доверия.

И да, следует отметить, что такое понятие трудно масштабировать, поскольку оно изначально не содержит необходимых механизмов. К чему такая конструкция? Наиболее очевидным примером является создание общественного договора (контракта) между двумя или более сторонами. Отсюда и название понятия - "смарт-контракт". Стороны фиксируют договоренности и условия выполнения сценария развития отношений между собой, используя язык программирования, таким образом, что наступление определенных событий автоматически вызовет выполнение заранее определенного кода.

Смарт-контракт - это условие, написанное на компьютерном языке, при котором стороны, подписывающие смарт-контракт, обмениваются любыми активами: валютой, недвижимостью, акциями и т.д. Например, валюта покупателя переводится в программу и замораживается там до тех пор, пока продавец не выполнит свою часть договора. Если условие нарушается, то сумма возвращается на счет клиента, а смарт-контракт аннулируется. Если все условия выполнены, то происходит обмен активами [22]. Этот обмен фиксируется в смарт-контракте и записывается в блокчейн, после чего он не может быть отменен, заменен или уничтожен. При отслеживании выполненных условий программа включается в автоматическом режиме, Контроль или участие людей не требуется. Другими словами, смарт-контракты работают непосредственно между заинтересованными сторонами, исключая посредников.

Смарт-контракты дают возможность безопасно обмениваться деньгами, акциями, имуществом и другими активами напрямую, без посредников.

Для того чтобы заключить любую сделку, вам необходимо обратиться к нотариусу или адвокату, оплатить документы и дождаться их оформления. Зачастую многие пункты этих документов содержат ссылки на законодательные статьи, которые могут быть истолкованы сами по себе, обойдены стороной. В случае невыполнения условий сделки, в реальной жизни людям приходится обращаться в суд, снова тратить деньги на этот процесс и доказывать свою правоту. При заключении таких сделок вообще не может быть и речи о доверии сторон договора.

Смарт-контракты могут регулировать самые разные финансовые (и не только) отношения между людьми. Самый очевидный вариант-это торговля в Интернете. Электронная коммерция сегодня охватывает практически все виды товаров. Мы заказываем не только технику, но и готовые блюда, продукты.

Выше уже приводился пример с покупкой недвижимости. Давайте разберемся, как можно реализовать вариант с его арендой. Мы должны внести деньги за первый месяц аренды и депозит. Сумма фиксируется в блокчейне, после чего хозяин сдает ключи.

Чтобы полностью автоматизировать смарт-контракт, нужно немного добавить "Интернет вещей": желательно установить навороченный замок в съемном жилище, который будет автоматически заблокирован, если оплата будет задержана или по истечении оговоренного срока. Когда срок аренды подходит к концу, двери блокируются, и недвижимость автоматически возвращается к арендатору.

Кроме того, смарт-контракты могут быть использованы при распределении наследства. Пожилой миллиардер, не доверяющий исполнителям (человеческий фактор, миллиардное наследство - вы понимаете), прописывает в смарт-контракте счета бенефициаров наследства на случай своей смерти. Система периодически отслеживает информацию из государственного реестра умерших. Как только появляется запись о разыскиваемом миллиардере, деньги автоматически отправляются его счастливым наследникам.

1.5 IOTA

Впервые криптовалютное сообщество узнало об этой валюте в 2015 году, и все это время разработчики занимались доведением своих идей до ума, и выглядели они так. Криптовалюта не должна никем контролироваться. Пользователи криптовалюты должны иметь возможность проводить микроплатежи. Отсутствие комиссии за проведенные сделки, а также быстрота и проведение - это один из самых важных критериев. Все эти идеи реализованы в блокчейне iota. У этой криптовалюты есть еще одна очень интересная функция, для того чтобы вы провели транзакцию, она должна быть подтверждена другими участниками сети (3 человека), и чем раньше вы получите это подтверждение, тем быстрее совершите транзакцию. Люди, которые подтверждают вашу транзакцию, ждут их подтверждения, что позволяет вам совершать быстрые транзакции. Такую систему можно использовать не только при продаже вещей в Интернете. Если вы не знаете, то IOTA изначально задумывалась как криптовалюта для Интернета вещей. Этот принцип может быть использован при голосовании и в других сферах [22].

По сути, платформа включает в себя обобщение блокчейн-протокола (известного как Tangle), который находится в бэкэнде на платформе IOTA.

Вместо того, чтобы платить майнерам за проверку транзакций, структура сети предполагает одноранговую проверку. Мы можем представить себе простую аналогию с наставником, оценивающим домашнюю работу студентов: ученики на самом деле являются клиентами/пользователями в рамках Биткойн-процесса, а учитель на самом деле является майнером/валидатором. Tangle engineering предлагает ученикам (пользователям) проводить качественные исследования каждого другого, создавая спрос на преподавателя (внешнего валидатора), а также воздерживаться от расходов, связанных с предоставлением результатов преподавателем/валидатором. Это позволяет платформе стать полностью

свободной от плат, не занимаясь проблемами масштабирования, которые на самом деле являются естественными в самом первом поколении блокчейнов [23].

Кроме того, возможно использование платформы с подключенными устройствами Интернета вещей.

Многие люди считают, что принцип работы IOTA решил все проблемы биткоина и эта система еще более перспективна. Мы не будем столь категоричны, но перечислим основные преимущества IOTA перед биткоином. В сети нет разделения и привилегий - нет обычных участников и тех, кто подтверждает транзакции. Все пользователи абсолютно равны, и никто ни от кого не зависит. IOTA - это абсолютно децентрализованная система. В том же биткойне майнеры могут собираться в пулы (что и происходит) и влиять на сеть (что также происходит). В IOTA это просто невозможно.

В сети IOTA вообще нет комиссий. Опять же, потому что здесь нет майнеров. Поэтому система идеально подходит для микроплатежей и не имеет ограничений по их объему и сумме. На самом деле, данную платформу еще лучше продвигают Ripple или Stellar, которые позиционируют себя как платежные решения.

Все сети, такие как биткоин, имеют проблемы с масштабируемостью, которые мы уже наблюдаем. Уникальный алгоритм, используемый IOTA, решает эту проблему почти навсегда. Здесь наоборот - чем больше участников, тем лучше работает система.

Используя различные алгоритмы консенсуса, такие как Proof-of-Work или Proof-of-Stake, система снижает требования к оборудованию. Для IOTA достаточно обычных устройств, которыми пользуются люди, и которых большинство. В техническом плане минимальным техническим требованием является недорогой микроконтроллер с 16 Кб оперативной памяти. IOTA использует другой тип шифрования. Не вдаваясь в подробности, это принцип квантового доказательства, который разработан и настроен на квантовые компьютеры. И хотя последние еще не появились, но это уже перспектива ближайшего будущего.

1.6 Смарт Контракты Ethereum

Блокчейн Ethereum – это криптографически защищённый одноэлементный механизм записи транзакций с совместно используемым состоянием. Блокчейн Эфириума, по сути, является машиной состояний, функционирующей посредством транзакций. В компьютерных науках определение машины состояний подразумевает, что этот механизм считывает серию входных данных и, основываясь на них, переходит в новое состояние.

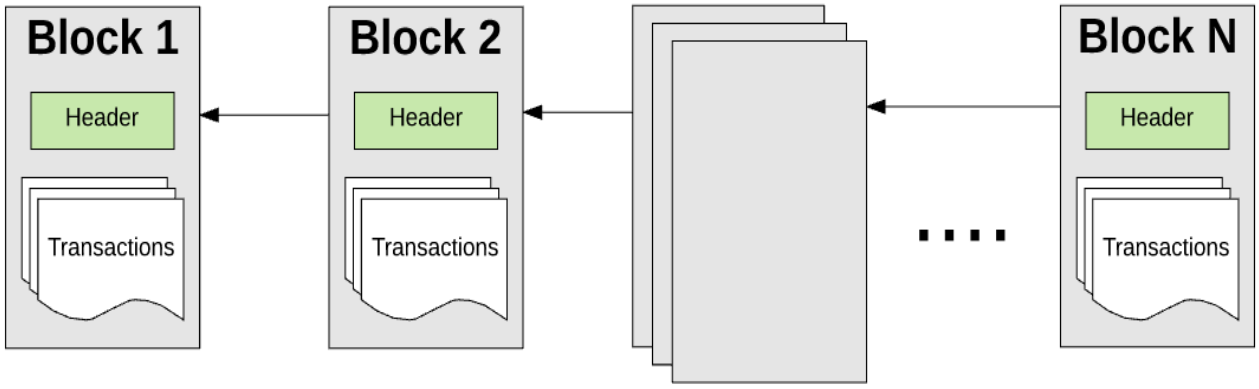

Рисунок 1.10 – Блокчейн Etherium

Любой узел в сети, объявляющий себя майнером, может попытаться создать и проверить блок транзакций. Распространенным опытом является попытки множества майнеров одновременного создания и проверки блока транзакций [23]. Каждый майнер предоставляет свое математическое «доказательство» при отправке блока в блокчейн, и это доказательство выступает в роли своеобразной гарантии: в случае если доказательство существует, транзакции в блоке считаются корректными.

Майнер, который обосновывает новый блок, получает определенное вознаграждение за выполнение этой работы. О каком вознаграждении идет речь? В блокчейне Эфириума используется встроенный цифровой токен, который носит название «эфир» (от англ. ether— «эфир»). Каждый раз, когда майнер обосновывает свой блок транзакций, создается новый токен или новый эфир, а майнер получает вознаграждение за его создание.

Для того чтобы вызвать переход сети из одного состояния в другое, транзакция должна быть действительной. Для того чтобы транзакция была признана действительной, она должна пройти через процесс валидации (проверки и утверждения), известный как майнинг. Майнингом в сети Эфириум называется процесс, в котором группа узлов сети (т.е. компьютеров) расходует свои вычислительные ресурсы на создание блока действительных транзакций [24].

Любой вычислительный узел сети (их также называют «нодами», от англ. node – узел сети), декларирующий себя в качестве майнера, может претендовать на создание и валидацию блока транзакций. Многие майнеры со всего мира одновременно пытаются создавать и валидировать блоки. Каждый майнер при записи блока в блокчейн предоставляет математическое «доказательство» (англ. proof), и это доказательство действует как гарантия: если доказательство существует, блок должен быть валидным (действительным). Для того чтобы добавить блок к основному блокчейну, майнер должен подтвердить его раньше других, конкурирующих с ним, майнеров. Процесс валидации каждого блока путём предоставления майнерами математического доказательства называется Proof-of-Work (доказательство выполнения работы). Майнер,

подтверждающий новый блок, получает за выполнение этой работы вознаграждение. Какое вознаграждение? В блокчейне Эфириума используются внутренние цифровые токены, называемые «эфирами». Каждый раз, когда майнер подтверждает блок, генерируются новые эфиры и выплачиваются майнеру [24].

Смарт-контракты обычно пишутся на языке высокого уровня, например на языке Solidity. Но для запуска они должны быть скомпилированы в байт-код низкого уровня, который работает в EVM. После компиляции они развертываются на платформе Ethereum с помощью специальной транзакции создания контракта, которая идентифицируется как таковая путем отправки на специальный адрес создания контракта блока размером 0x0. Каждый контракт идентифицируется по адресу Ethereum, который является производным от транзакции создания контракта в зависимости от исходного счета и nonce. Ethereum-адрес контракта может быть использован в транзакции в качестве получателя, отправляя средства на контракт или вызывая одну из функций контракта. Обратите внимание, что, в отличие от EOAs, здесь нет ключей, связанных с учетной записью, созданной для нового смарт-контракта. Как создатель контракта, вы не получаете никаких особых привилегий на уровне протокола (хотя вы можете явно кодировать их в смарт-контракте). Вы, конечно, не получаете закрытый ключ для контрактной учетной записи, которой на самом деле не существует—можно сказать, что смарт-контрактные учетные записи владеют сами собой.

EVM-это виртуальная машина, которая запускает специальную форму кода, называемую байт-кодом EVM, аналогичную процессору вашего компьютера, который запускает машинный код [24]. В то время как можно запрограммировать смарт-контракты непосредственно в байт-коде, байт-код EVM довольно громоздок и очень труден для чтения и понимания программистами. Вместо этого большинство разработчиков Ethereum используют высокоуровневый язык для написания программ и компилятор для преобразования их в байт-код.

В то время как любой язык высокого уровня может быть адаптирован для написания смарт-контрактов, адаптация произвольного языка для компиляции в байт-код EVM является довольно громоздким упражнением и в целом приведет к некоторой путанице. Смарт-контракты работают в сильно ограниченной и минималистичной среде исполнения (EVM). Кроме того, должен быть доступен специальный набор специфичных для EVM системных переменных и функций. Таким образом, проще построить язык смарт-контрактов с нуля, чем сделать язык общего назначения подходящим для написания смарт-контрактов. В результате появился целый ряд специальных языков для программирования смарт-контрактов. Ethereum имеет несколько таких языков, а также компиляторы, необходимые для создания EVM-исполняемого байт-кода.

В целом языки программирования можно разделить на две широкие парадигмы программирования: декларативную и императивную, также известные как функциональная и процедурная соответственно. В декларативном

программировании мы пишем функции, которые выражают логику программы, но не ее поток. Декларативное программирование используется для создания программ, в которых отсутствуют побочные эффекты, то есть нет никаких изменений состояния вне функции [25]. Декларативные языки программирования включают Haskell и SQL. Императивное Программирование, напротив, - это когда программист пишет набор процедур, которые объединяют логику и поток программы. Императивные языки программирования включают C++ и Java. Некоторые языки являются "гибридными", что означает, что они поощряют декларативное программирование, но также могут использоваться для выражения императивной парадигмы программирования. Такие гибриды включают Lisp, JavaScript и Python. В общем случае любой императивный язык может быть использован для написания декларативной парадигмы, но это часто приводит к неэлегантному коду. Для сравнения, чисто декларативные языки не могут быть использованы для написания в императивной парадигме. В чисто декларативных языках нет никаких "переменных".

1.7 Hyperledger

Hyperledger - это сообщество с открытым исходным кодом, ориентированное на разработку набора стабильных фреймворков, инструментов и библиотек для развертывания блокчейна корпоративного уровня [26].

Он служит источником различных систем распределенного реестра, включая Hyperledger Fabric, Sawtooth, Indy, а также инструменты, такие как Hyperledger Caliper и библиотеки, такие как Hyperledger Ursa.

Возможности технологии Hyperledger:

- Hyperledger позволяет создать приватный, а не публичный блокчейн.
- Отсутствие необходимости майнинга и выпуска токенов. Проверяющие узлы самостоятельно уведомляют друг друга об операциях, достигают консенсуса и создают новые блоки.
- Гибкая настройка прав доступа к сети, возможность создавать частные системы с закрытыми для чтения блоками – посторонние не смогут получить данные из них.
- Конфиденциальные транзакции доступные для просмотра только избранным пользователям, имеющим необходимые ключи шифрования.
- Большой выбор доступных языков программирования: C++, JavaScript, Python, Golang, Java.

Hyperledger может быть использован для создания блокчейна общего назначения, в то время как решения конкурентов предназначены для узкопрофильных задач:

- Ripple – организация платежных систем.
- R3 CEV – проведение традиционных транзакций и заключение соглашений.
- Ethereum – создание публичного блокчейна.

1.5.2 Hyperledger Sawtooth. Hyperledger Sawtooth - это корпоративная блокчейн-платформа для создания распределенных бухгалтерских приложений и сетей. Философия дизайна нацелена на то, чтобы распределить бухгалтерские книги и сделать смарт-контракты безопасными, особенно для корпоративного использования.

Sawtooth также обладает высокой модульностью. Эта модульность позволяет предприятиям и консорциумам принимать стратегические решения, для принятия которых они лучше всего подготовлены. Основной дизайн Sawtooth позволяет приложениям выбирать правила транзакций, разрешения и алгоритмы консенсуса, которые поддерживают их уникальные бизнес-потребности [26].

Sawtooth упрощает разработку и развертывание приложения, обеспечивая четкое разделение между уровнем приложения и уровнем базовой системы. Sawtooth предоставляет абстракцию смарт-контрактов, которая позволяет разработчикам приложений писать логику контрактов на выбранном ими языке.

Приложение может быть собственной бизнес-логикой или виртуальной машиной смарт-контракта. На самом деле оба типа приложений могут сосуществовать в одном и том же блокчейне. Sawtooth позволяет принимать эти проектные решения на уровне обработки транзакций, что позволяет нескольким типам приложений существовать в одном экземпляре блокчейн-сети.

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 [26].

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 [26].

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 [26].

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 [26]. Это мощный инструмент, который может использоваться многими предприятиями, так как он позволяет конечным пользователям лучше контролировать и повышать эффективность.

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.

Процесс 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 [26].

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 [26].

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 [26]. 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.

1.5.3 Hyperledger Fabric. Hyperledger Fabric-это корпоративная распределенная платформа, которая предлагает модульность и универсальность для широкого набора отраслевых вариантов использования. Модульная конструкция архитектуры Hyperledger Fabric вмещает в себя все многообразие предприятий, примеры использования с помощью компонентов plug and play, таких как консенсус, конфиденциальность и членские услуги.

Hyperledger Fabric это open-source проект, одна из ветвей открытого проекта Hyperledger, консорциума Linux Foundation. Hyperledger Fabric был изначально стартован Digital Assets и IBM. Основной особенностью платформы Hyperledger Fabric является направленность на корпоративное применение. Поэтому платформа разрабатывалась с учетом обеспечения высокой скорости проведения транзакций и их низкой стоимости, а также идентификации всех участников [27]. Данные преимущества достигаются за счет разделения службы проверки транзакций и формирования новых блоков распределенного реестра, а также применения центра сертификации и авторизации участников.

Hyperledger Fabric — это распределенная блокчейн сеть, состоящая из различных функциональных компонентов, которые устанавливаются на узлы сети. Компоненты Hyperledger Fabric представляют из себя Docker контейнеры, которые

можно свободно скачать из DockerHub. Hyperledger Fabric также можно запустить в Kubernetes среде.

Для написания смарт-контрактов (chaincode в контексте Hyperledger Fabric) мы использовали Golang (хотя Hyperledger Fabric позволяет использовать и другие языки). Для разработки пользовательского приложения в нашем случае использовался Node.js с соответствующим Hyperledger Fabric SDK.

На узлах выполняется бизнес логика (смарт-контракт) – chaincode, хранится состояние распределенного реестра (ledger data) и исполняются другие системные службы платформы. Узел – это только логическая единица, разные узлы могут существовать на одном физическом сервере [27]. Гораздо важнее – это как узлы сгруппированы (Trusted domain) и с какими функциями блокчейн сети они ассоциированы.

Fabric обладает модульной и конфигурируемой архитектурой, позволяющей использовать инновации, универсальность и оптимизацию для широкого круга отраслевых приложений, включая банковское дело, финансы, страхование, здравоохранение, управление персоналом, цепочку поставок.

Fabric является первой платформой распределенного реестра, поддерживающей "умные" контракты, составленные на языках программирования общего назначения, таких как Java, Go и Node.js, а не на ограниченных языках, специфичных для домена (DSL). Это означает, что большинство предприятий уже имеют набор навыков, необходимых для разработки "умных" контрактов, и нет необходимости в дополнительной подготовке для изучения нового языка или DSL.

Платформа Fabric также является разрешенной, а это означает, что, в отличие от сети без разрешений, участники известны друг другу, а не анонимны и, следовательно, полностью доверяют друг другу. Это означает, что, хотя участники могут не полностью доверять друг другу (они могут, например, быть конкурентами в одной отрасли), сеть может функционировать в рамках модели управления, построенной на том, какое доверие существует между участниками, например, юридическое соглашение или рамки для разрешения споров [27].

Одним из наиболее важных отличительных признаков платформы является ее поддержка подключаемых протоколов консенсуса, которые позволяют более эффективно адаптировать платформу к конкретным случаям использования и моделям доверия. Например, при развертывании в рамках одного предприятия или под управлением доверенного органа полностью византийский консенсус, основанный на отказоустойчивости, может считаться ненужным и чрезмерно затягивающим работу и пропускную способность. В таких ситуациях консенсусный протокол crash fault-tolerant (CFT) может быть более чем уместным, в то время как в случае децентрализованного использования может потребоваться более традиционный byzantine fault tolerant (BFT).

Fabric может использовать консенсусные протоколы, не требующие наличия собственной криптографической валюты для осуществления дорогостоящей

добычи или интеллектуального исполнения контрактов. Избегание криптовалюты снижает некоторые значительные риски/векторы атаки, а отсутствие криптографических операций по добыче означает, что платформа может быть развернута с приблизительно такими же операционными затратами, как и любая другая распределенная система [25].

Сочетание этих отличительных конструктивных особенностей делает Fabric одной из наиболее эффективных платформ, доступных сегодня как с точки зрения обработки транзакций, так и с точки зрения задержки подтверждения транзакций, а также обеспечивает конфиденциальность и конфиденциальность транзакций и "умных" контрактов, которые их реализуют.

2 Исследование существующих blockchain решений

2.1 Постановка задачи

Одна из главных причин, почему блокчейн так привлекателен, заключается в том, что цепочки почти всегда имеют открытый исходный код. Это означает, что другие пользователи или разработчики имеют возможность изменять его по своему усмотрению, но в то же время это невероятно затрудняет незаметное изменение ранее зарегистрированных данных. Последнее обстоятельство делает блокчейн особенно надежной технологией.

Преимущества технологии блокчейн:

- блокчейн база данных безопасна, в отличие от классической базы данных, где информация хранится на сервере, и все люди, имеющие доступ, имеют возможность уничтожать, красть или использовать данные.

- еще одна особенность блокчейна - это открытость. Каждый узел самостоятельно проверяет информацию и имеет возможность выполнить транзакцию. Блокчейн характеризуется прозрачностью передаваемых данных.

Еще одна серьезная причина привлекательности блокчейна заключается в том, что технология не предполагает центрального пункта сбора данных. Вместо того чтобы управлять большим дата-центром и проводить через него все транзакции, блокчейн фактически позволяет отдельным транзакциям иметь свою собственную систему аутентификации и авторизации, чтобы обеспечить их связь друг с другом. Информация о конкретных блоках цепочки разбросана по разным серверам по всему миру, и это гарантирует, что даже если эта информация попадет к посторонним, например, хакерам, то будет скомпрометирована лишь небольшая часть данных, а не вся сеть [24].

В качестве платформы для проекта был выбран Hyperledger Fabric. Преимущества этой платформы будут раскрыты ниже. Формулировка задач заключается в следующем:

- изучение архитектуры выбранной платформы;
- изучение процесса приема и обработки данных;
- внесение изменений в программный код для реализации заявки на запись успеваемости студента;
- внесение изменений в клиентскую часть приложения;
- расчет времени создания блока;
- расчет количества транзакций внутри блока;
- расчет отказоустойчивости;
- планирование дальнейшей разработки приложения.

2.2 Обоснование выбора платформы Hyperledger Fabric

Hyperledger - это продукт Linux foundation, созданный в декабре 2015 года для продвижения технологии блокчейн. Проект представляет собой совместную работу по созданию открытого, распределенного реестра, который может быть использован для открытой разработки и внедрения блокчейн-приложений и систем. Основной упор делается на создание и запуск платформ, поддерживающих глобальные бизнес-транзакции. Проект также направлен на повышение надежности и производительности работы компаний.

Проекты в Hyperledger проходят различные стадии развития: от идеи и ее развития до активной работы и даже застоя. Для того чтобы проект перешел от идеи к стадии инкубации, он должен иметь полностью функционирующую кодовую базу и активное сообщество разработчиков.

Fabric - это блокчейн - фреймворк, первоначально предложенный IBM и DAH (Digital Asset Holdings). Он призван создать основу для разработки решений для блокчейна и базируется на модульной архитектуре, где при необходимости могут быть подключены различные компоненты, например, консенсусный алгоритм. Смарт-контракты в Fabric называются chaincode [27]. Сеть Hyperledger Fabric включает в себя "одноранговые узлы", которые выполняют код блока, доступ к данным регистра, поддерживают транзакции и интерфейс приложения. Fabric предназначен для проектов, требующих технологии распределенных реестров (DLT). Поддерживает chaincode в Go (Golang), Java и JavaScript (через Hyperledger Composer или с версии 1.1) и поэтому потенциально более гибок, чем язык смарт-контрактов. Hyperledger Fabric - это платформа для распределенных реестров, основанная на модульной архитектуре, обеспечивающей высокую степень конфиденциальности, отказоустойчивости, гибкости и масштабируемости. Он предназначен для поддержки подключаемых реализаций различных компонентов и учитывает сложность и запутанность, существующие во всей экономической экосистеме.

Hyperledger Fabric обеспечивает уникальную эластичную и расширяемую архитектуру, отличающую ее от альтернативных блокчейн-решений.

2.2.1 Функциональные Возможности Hyperledger Fabric. Hyperledger Fabric - это фактически реализация технологии распределенного реестра (DLT), которая обеспечивает корпоративную готовую сетевую безопасность, масштабируемость, производительность и конфиденциальность в модульной архитектуре блокчейна. Hyperledger Fabric предоставляет следующие функциональные возможности блокчейн-сети: управление идентификацией, конфиденциальность, эффективная обработка, производительность цепного кода, модульность.

Hyperledger Fabric имеет службу идентификации членства, которая управляет идентификаторами пользователей, а также аутентифицирует всех участников сети. Списки управления доступом могут использоваться для предоставления дополнительных уровней разрешений с помощью авторизации определенных частей сообщества. Например, определенный идентификатор пользователя может быть разрешен для вызова программы chaincode, но заблокирован от развертывания нового chaincode [27].

Hyperledger Fabric позволяет конкурирующим интересам компаний и любым группам, которые нуждаются в частных, конфиденциальных транзакциях, сосуществовать в одной разрешенной сети. Частные каналы - это фактически ограниченные пути обмена сообщениями, которые могут быть использованы для обеспечения конфиденциальности, а также конфиденциальности транзакций для определенных подмножеств пользователей сети. Вся информация, которая включает информацию о канале, члене и транзакции, на самом деле недоступна и невидима для любого вида участников сети, которым явно не предоставлен доступ к этому каналу.

Выполнение транзакций непосредственно перед их активацией позволяет каждому одноранговому узлу обрабатывать несколько транзакций одновременно. Это параллельное выполнение повышает эффективность обработки на каждом узле, а также ускоряет доставку транзакций в службу покупки [27]. Эта конкретная бифуркация ролей также ограничивает обработку, необходимую для аутентификации и авторизации; всем одноранговым узлам не нужно верить остальным упорядочивающим узлам, и наоборот, поэтому задачи на одной стороне могут работать независимо от проверки другой стороны [26].

Модульность архитектуры Hyperledger Fabric позволяет разработчикам сети использовать различные решения для компонентов, что является значительным преимуществом. Одной из наиболее востребованных областей модульности является идентификация узлов. В некоторых сетях с несколькими компаниями уже есть управление идентификацией и они хотят повторно использовать то что имеют, а не перестраивать. Другие компоненты архитектуры, которые могут быть легко подключены, включают в себя консенсус или шифрование.

Прозрачный процесс: транзакции могут быть непрозрачными, но процесс разработки наоборот. «На этом этапе основные команды Hyperledger были чрезвычайно готовы сбалансировать потребности, чтобы получить возможности с открытым и прозрачным процессом развития», - заметил основатель Skuchain Заки Маниан.

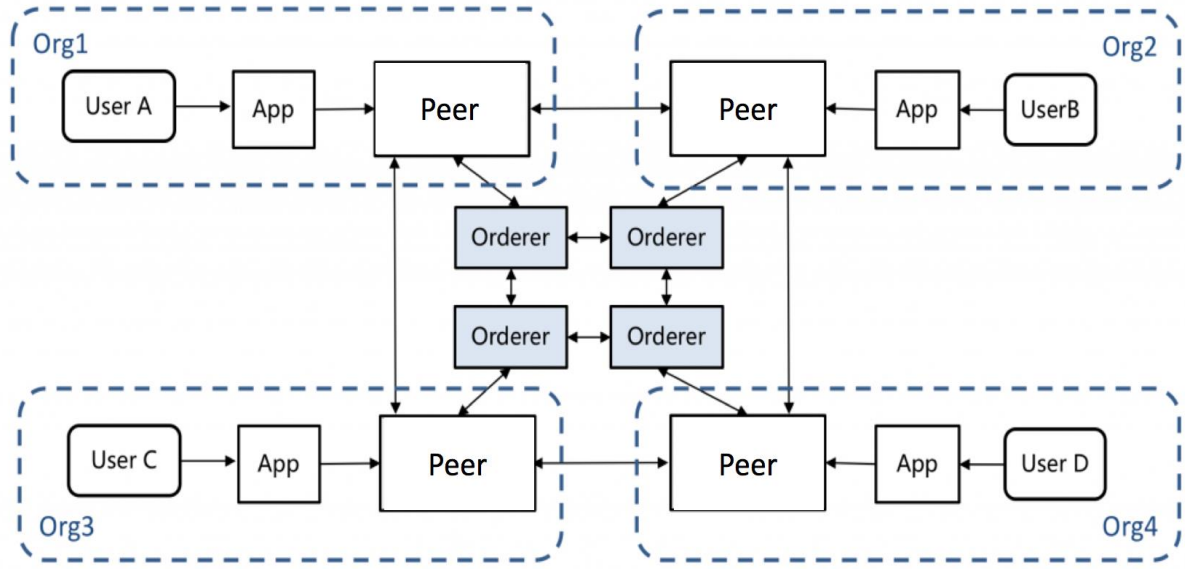

Рисунок 2.1-Роль участников в Сети Hyperledger Fabric

Как видно из диаграммы-каждый участник имеет:

- свой собственный узел (peer), который обеспечивает работоспособность блокчейной сети и является частью HLF;

- приложение, которое обычно состоит, по крайней мере, из веб-сервера и пользовательского интерфейса;

- служба заказа, которая обеспечивает взаимодействие между узлами [23].

В приложении пользователи авторизуются с помощью собственных ключей или связующего логина и пароля и имеют доступ к смарт-контракту.

Смарт-контракт - это запрограммированная бизнес-логика и правила взаимодействия с данными, размещенными в распределенной книге учета. Реестр содержит цепочку блоков транзакций, которые были зафиксированы участниками сети. Каждый последующий блок хранит хэш предыдущего блока, что гарантирует, что некоторая информация не может быть изменена. Процесс согласования корректности блока и добавления его в распределенный реестр называется консенсусом. Различные блочные платформы имеют различные механизмы консенсуса: доказательство работы (PoW), доказательство доли (PoS), византийская отказоустойчивость (BFT) и другие [28].

Поддержка HSM (Hardware Security Module) жизненно важна для защиты и управления цифровыми ключами для надежной аутентификации. Hyperledger Fabric предоставляет модифицированный и немодифицированный PKCS11 для

генерации ключей, который поддерживает такую функцию, как управление идентификацией, которая нуждается в большей защите. Для сценариев управления идентификацией HSM повышает защиту ключей от взлома и конфиденциальных данных.

Каналы Hyperledger могут быть одной из самых недооцененных функций. Каналы дают возможность осуществлять разделение данных. Это позволяет нам защищать данные, которые нам необходимо защитить.

Эта возможность очень полезна, когда финансовые компании, которые намереваются внедрить блок-цепь, выражают глубокую озабоченность в защите данных. Мы говорим о компаниях и банках, где даже очень хорошей криптографии недостаточно чтобы обезопасить данные [28].

С каналами в Hyperledger Fabric вы можете предоставлять данные, которые необходимы только в разделённом виде, или хранить данные, чувствительные к разделам данных.

2.3 Сравнительный анализ производительности платформ Hyperledger Fabric и Etherium

В настоящее время вопросы надежности систем приобретают все большее значение, это формирует новые требованиям к технологиям обработки и хранения данных. Остановимся подробнее на анализе технологии блокчейн (blockchain или block chain – цепочка блоков транзакций) – выстроенной по определённым правилам цепочки из формируемых блоков транзакций, ориентированной на обеспечение взаимодействия большого количества пользователей между собой без использования «доверенных посредников» [21].

Впервые термин появился как название распределённой базы данных, реализованной в криптовалюте биткоин. Следующим этапом стоит считать Ethereum, который, добавив умные контракты в систему блокчейн, сделал платформу для разработки децентрализованных приложений. Также стоит отметить Hyperledger, реализовавший модульную структуру, которая упрощает интеграцию с существующими системами [22]. В основе протоколов более высокого уровня лежит понятие «умный контракт» – электронной алгоритм, описывающий набор условий, необходимых для выполнения транзакции, который хранится в узлах сети. Под транзакцией понимаются действия участников системы, связанные с взаимодействием с данными. При совершении транзакции ее следует зафиксировать в блокчейне, для этого транзакции помещаются в блоки. Но для добавления нового блока в блокчейн необходимо разрешение от других участников сети. Алгоритм получения этого разрешения получил название «консенсус».

Одной из главной проблем блокчейн является достоверность данных, это определяет необходимость применения эффективных алгоритмов шифрования. Они должны гарантировать достаточную криптографическую стойкость для

информации в сети, а также позволять реализовать цифровую подпись . Рассмотрим алгоритм работы ассиметричного шифрования RSA [22]. Сначала выбираются два простых числа *p* и *q*. Далее находятся модуль для открытого и закрытого ключа (1) и функция Эйлера от модуля (2):

$$n = p * q \quad (2.1)$$

$$\varphi(n) = \varphi(pq) = (p-1)(q-1) \quad (2.2)$$

После этого выбирается целое число *e* (открытая экспонента) от 1 до $\varphi(n)$, взаимно простое с $\varphi(n)$. Обычно в качестве *e* берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, но не слишком малые, для быстрого возведения в степень. [8]

Далее находится число d, отвечающее формуле (3):

$$d * e \bmod \varphi(n) = 1 \quad (2.3)$$

Таким образом формируется приватный ключ {d, n} и публичный ключ {e, n}, при помощи которых производится шифрование (4) данных.

$$c = m^e \bmod(n) \quad (2.4)$$

При попытке же взломать (подобрать) закрытый ключ придется перебрать 2^N комбинаций, где N – длина ключа. Например, при длине ключа в 256 бит и скорости подбора паролей 1024 в секунду потребуется 1,23e + 67 лет, что очень и очень много, да и информация к тому времени будет уже не актуальна.

Также используются более прогрессивные алгоритмы, такие как Elliptic Curve Digital Signature Algorithm (ECDSA), которые работают схожим образом, но имеют свои тонкости. Не менее важен вопрос специальных алгоритмов конкурентного доступа и разрешения коллизий в сети.

Ниже рассмотрены некоторые из них:

-PBFT – запрос на добавление блока рассылается всем участникам, и все производят вычисления хэша следующего блока, после чего рассылают свое решение остальным участникам, в итоге каждый участник получает массив ответов и принимает ответ с общим числом более 50 % за достоверный, недостаток – время выполнения тран закции увеличивается в зависимости от размера сети;

-PoW – узлы сети (майнеры) решают задачу по вычислению хэша следующего блока с определенным условием, и кто быстрее посчитает хэш, того блок и будет следующим. Минусы - энергоемкость из-за сложности вычислений и присутствие некоторой централизации - майнеров;

-PoS – альтернатива PoW, не требует больших вычислительных мощностей: участники сети имеют внутрисистемную валюту, и кто богаче, тот имеет приоритет для формирования блока, недостатки – отсутствие случайности[21].

Ethereum - это платформа и одноименная криптовалюта для создания смарт-контрактов и децентрализованных приложений. Платформа позволяет создавать полноценные децентрализованные приложения с решением вопросов безопасности и масштабируемости и использовать глобальную сеть для

транзакций любого уровня сложности. [22] При этом не нужно создавать инфраструктуру бизнеса с нуля. Прозрачная и надежная технология смарт-контракта дает возможность регистрации любых сделок с активами на основе распределенной базы контрактов, реализованных с использованием хэшей блокчейн цепочек, не прибегая к традиционным юридическим процедурам. Многочисленные пользователи сети Ethereum сами обеспечат необходимые вычисления хэш-сумм для блокчейна, используя мощности своих компьютеров, что позволит отказаться от вложений в аппаратное и программное обеспечение. Ethereum блокчейн обладает встроенным Тьюринг-полным языком программирования Solidity, в котором можно писать смарт-контракты с произвольными правилами владения, форматами транзакций и произвольными функциями изменения состояния.

Hyperledger Fabric (HLF) — платформа с открытым исходным кодом, использующая технологию распределенного реестра (DLT — distributed ledger technology), предназначенная для разработки приложений, работающих в среде бизнес-сетей, созданных и контролируемых консорциумом организаций с применением правил доступа (permissioned). [21]

Платформа поддерживает смарт-контракты, в терминах HLF — чейнкоды (chaincode), создаваемые на языках общего назначения, таких как Golang, JavaScript, Java, в отличие, от, например, Ethereum, в котором используется контрактно-ориентированный, ограниченный по функциональности язык Solidity (LLL, Viper и др). [21]

Общая архитектура выглядит следующим образом :

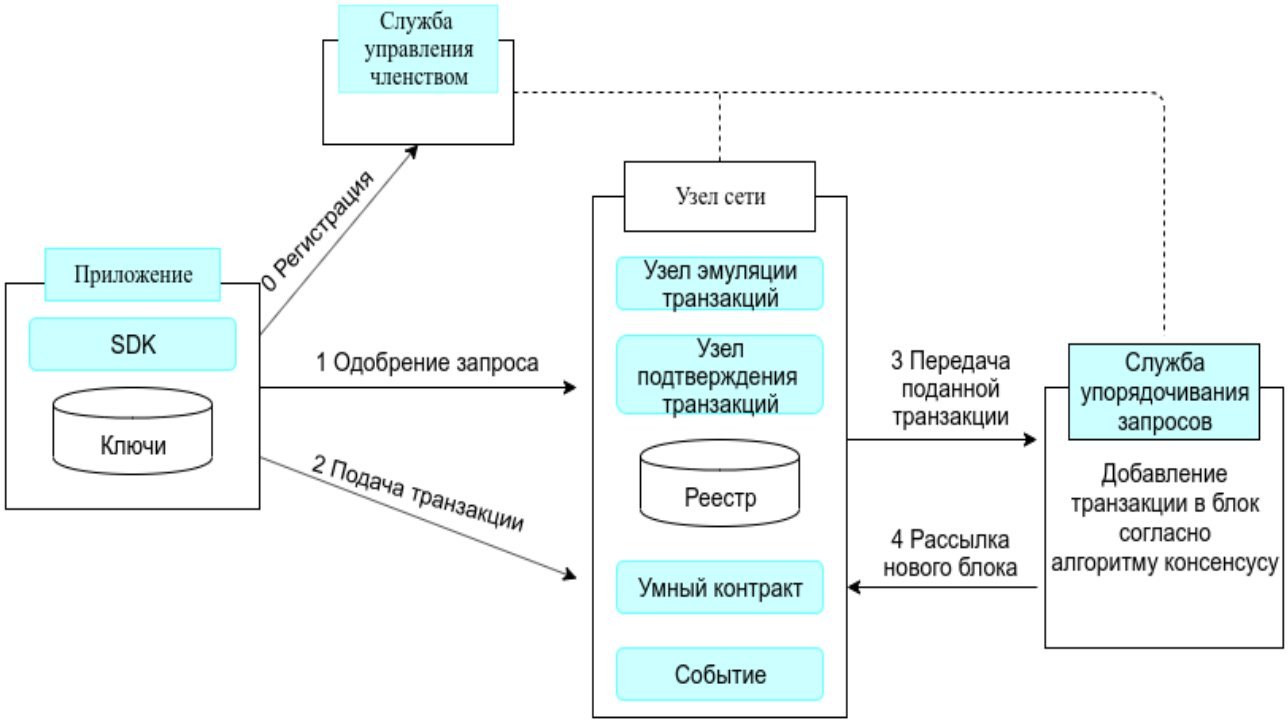

Рисунок 2.2 – Общая архитектура Hyperledger Fabric

Резюмируя вышесказанное, можно объединить в сводную таблицу информацию об основных достоинствах и недостатках блокчейн платформ Etherium и Hyperledger Fabric (Таблица 2.1).

Таблица 2.1. Достоинства и недостатки Etherium и Hyperledger Fabric.

Платформа	Достоинства	Недостатки	ЯП
Etherium	широкое распространение, гибкость, низкий порог вхождения.	Оплата транзакций в сети (gas), Высокая нагруженность.	Solidity, Serpent, Mutan.
Hyperledger Fabric	Ориентированность на корпоративный сегмент, Растущая экосистема, Крупное сообщество пользователей.	Высокий порог вхождения, Платная официальная поддержка.	Go, Java, JavaScript.

В недавнем исследовании была измерена эффективность двух реализаций блокчейн: Ethereum и Hyperledger Fabric, варьируя количество транзакций (от 1 до 10000). Исследование проводилось на примере простого приложения, осуществляющего 3 основные функции:

- Создание учетной записи,
- Получение денег,
- Перевод денег.

Функция создания учетной записи используется для создания новых пользователей, в то время как две другие функции получения и трансфера денег используются, соответственно, для выдачи денег на счет и перевода денег от одного пользователя другому. Результаты этого эксперимента показали, что Hyperledger превосходит Ethereum во всех метриках оценки производительности системы, таких как время выполнения транзакции, пропускная способность и задержка, что подробно показано на рисунках 2.3-2.5.

Рисунок 2.3 – Время исполнения различного количества транзакций

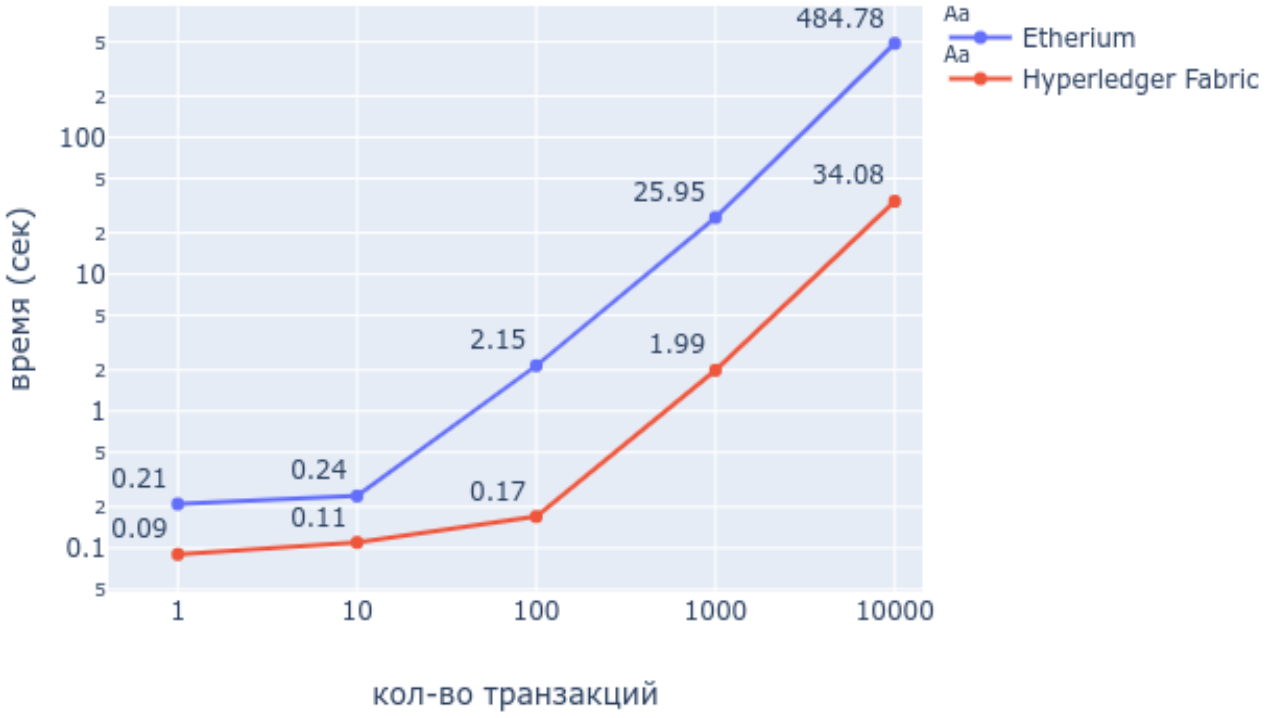

Рисунок 2.4 –. Задержка при исполнении различного количества транзакций

Рисунок 2.5 – Пропускная способность платформ Etherium и Hyperledger Fabric

Переход от экстенсивного роста к интенсивному обуславливает сокращениею числа свободных ниш и, как следствие, кооперации многих ранее не работавших сообща людей и организаций. У каждого своя БД, своя инфраструктура данных, что порождает неточности и недостоверность информации. В мире, где цена ошибки выше, чем цена системы, требуется технология, умеющая управлять издержками, снижать издержки, обеспечивать эффективность обмена данными и минимизировать ошибки, что во многом позволяет делать технология блокчейн [29].

Приведенное выше сравнение отчетливо показывает преимущества использования платформы Hyperledger Fabric над своим прямым конкурентом в корпоративном сегменте Etherium. Скорость исполнения транзакций отличается значительным образом, показывая разницу практически в 25 раз при 1000 транзакций, что является неоспоримым аргументом в пользу использования платформы Hyperledger.

3. Практическая реализация проекта

3.1 Поток операций

В сети Hyperledger Fabric транзакции начинаются с клиентских приложений, управляющих предложениями по сделкам, или даже, другими словами, предлагающих сделку для одобрения остальными узлами.

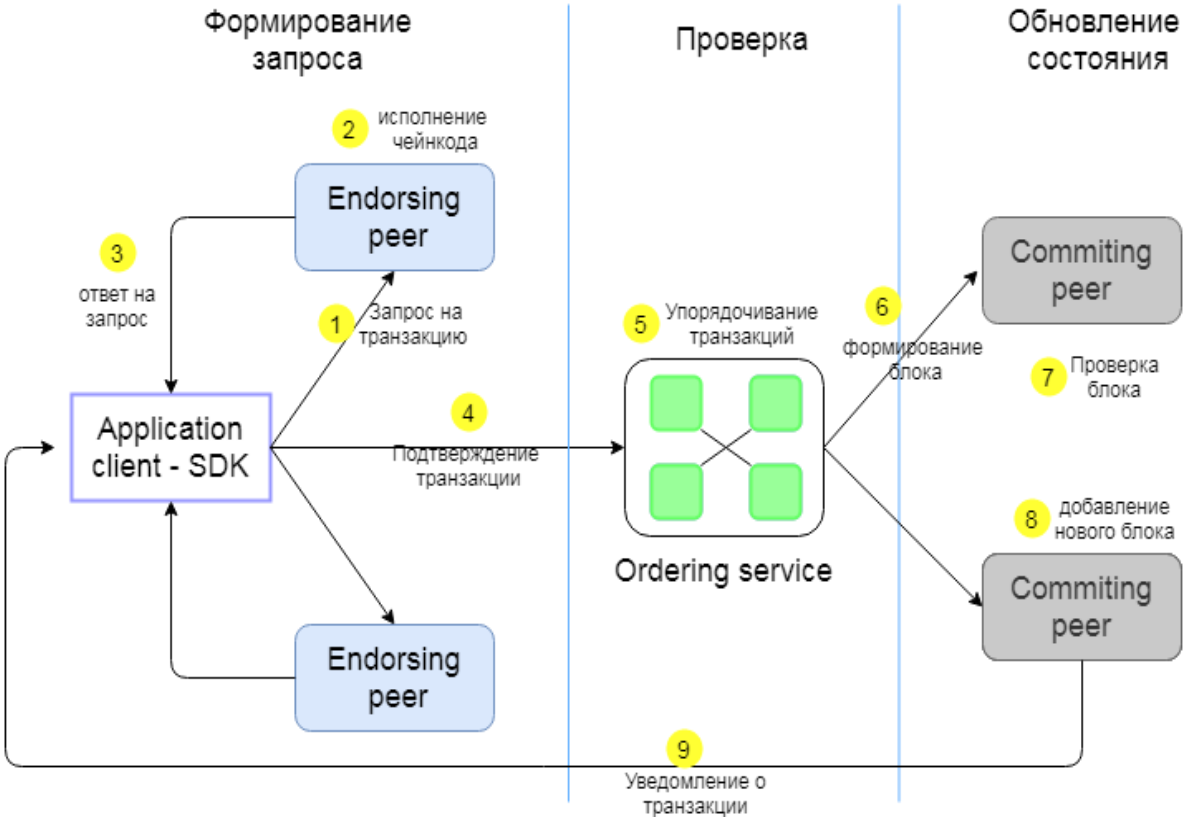

Рисунок 3.1 – Роль участников сети Hyperledger Fabric

Члены сети должны иметь "умные" контракты, имитирующие предложения по сделкам.

Стратегия одобрения транзакции зависит от политики одобрения, которая фактически указывается после того, как чейнкод действительно развернут. Хорошим примером политики одобрения будет то, что "основная масса коллег, участвующих в сделке, должна одобрить ее". Поскольку политика одобрения на самом деле указана для определенного чейнкода, различные узлы могут иметь различную политику одобрения [29].

После этого программное обеспечение отправляет в службу подтверждения согласованную сделку, а также RW наборы. Добавление транзакции происходит по всей сети, параллельно с обеспеченными чейнкодами.

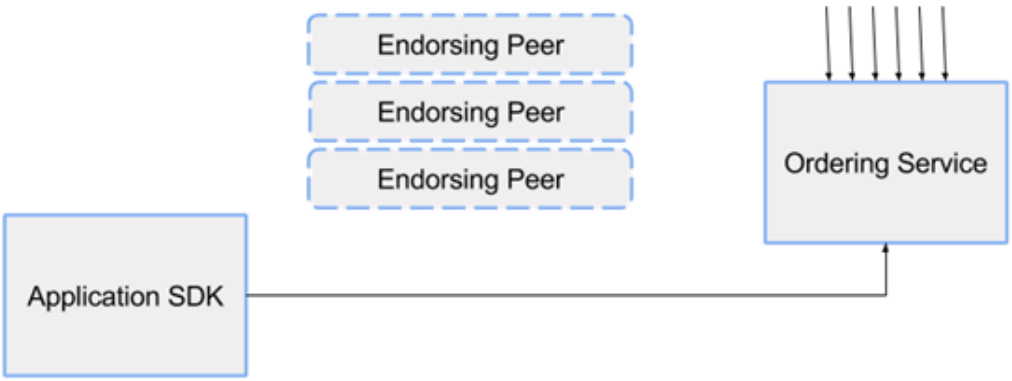

Рисунок 3.2 – Заявка клиента подается в службу заявок

Сервис заявок принимает одобренные транзакции и наборы RW, добавляет эту информацию в блок и доставляет блок всем участникам сети блокчейн.

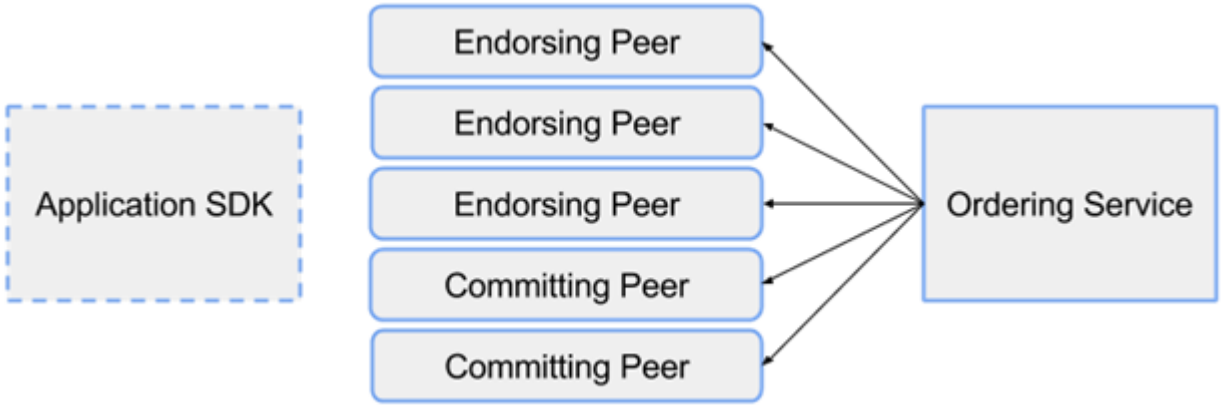

Рисунок 3.3 – Endorsing peer отправляет блок последовательных транзакций всем узлам.

Сервис заявок, состоящий из группы заказчиков, не обрабатывает транзакции, не выполняет смарт-контракты и не принимает участия в формировании блоков. Сервис заявок принимает одобренные транзакции и указывает порядок, в котором эти транзакции будут переданы в распределенный реестр. Архитектура Fabric v1.0 была разработана таким образом, что конкретная реализация «упорядочивания» (Solo, Kafka, BFT) становится подключаемым компонентом. Сервисом заявок по умолчанию для Hyperledger Fabric является Kafka. Таким образом, сервис заявок является модульным компонентом Hyperledger Fabric.

В блокчейн-сети транзакции должны записываться в распределенный реестр в согласованном порядке. Порядок транзакций должен быть установлен, чтобы гарантировать, что обновления состояния блокчейна действительны, когда они зафиксированы в сети. В отличие от биткойн-блокчейна, где упорядочение происходит путем решения криптографической задачи или майнинга, Hyperledger

Fabric позволяет организациям, управляющим сетью, выбрать механизм упорядочения, который лучше всего подходит для этой сети. Эта модульность и гибкость делает Hyperledger Fabric невероятно выгодным для корпоративных приложений.

Архитектура Hyperledger предоставляет три механизма упорядочения: SOLO, Kafka и упрощенная Византийская отказоустойчивость (SBFT), которые еще не были реализованы в версии v1.0.

SOLO-это утилита для создания очередей, которая обычно используется разработчиками, экспериментирующими с сетью Hyperledger Fabric. SOLO состоит из одного последовательного узла.

Kafka-это механизм упорядочивания блоков Hyperledger, который рекомендуется для использования в производстве. Командная строка использует Apache Kafka - платформу обработки потоков с открытым исходным кодом, которая обеспечивает унифицированную, высокочастотную платформу для обработки потока данных в реальном времени. В этом случае данные содержат подтвержденные наборы транзакций. Кафка предлагает отказоустойчивое решение для классификации систем [29].

SBFT - это алгоритм упрощенной задачи Византийской отказоустойчивости. Этот последовательный механизм является коллизионно - толерантным, поэтому соглашение может быть достигнуто при наличии вредоносного узла или дефектного узла. Сообщество Hyperledger Structure еще не внедрило эту систему, но оно планирует ее использовать.

Важно отметить, что состояние сети поддерживается партнерами, а не службой заказа или клиентом. Обычно вы проектируете свою систему так, чтобы разные машины в сети играли разные роли. Таким образом, машины, которые являются частью сервиса заявок, не должны быть настроены также для поддержки или подтверждения транзакций, и наоборот. Тем не менее, существует совпадение между одобрением и фиксацией пиров в системе. Подтверждающие одноранговые узлы должны иметь доступ и иметь смарт-контракты, в дополнение к выполнению роли обязывающего однорангового узла.

Подтверждающие одноранговые узлы проверяют подпись клиента и выполняют функцию чейнкода для имитации транзакции. Ответ на предложение отправляется обратно клиенту вместе с подписью подтверждения. Эти ответы на предложения отправляются заказчику для дальнейших действий. Затем заказчик упорядочивает транзакции в блок, который он направляет endorsing и committing пирам. Наборы RW используются для проверки того, что транзакции по-прежнему действительны до обновления содержимого распределенного реестра. Наконец, одноранговые узлы асинхронно уведомляют клиентское приложение об успехе или неудаче транзакции.

Каналы позволяют организациям использовать одну и ту же сеть, сохраняя при этом разделение между несколькими цепочками блоков. Только члены канала,

на котором была выполнена транзакция, могут видеть подробности транзакции. Другими словами, каналы разделяют сеть, чтобы обеспечить видимость транзакции только для заинтересованных сторон [30]. Этот механизм работает путем делегирования транзакций различным регистрам. Только участники канала участвуют в консенсусе, в то время как другие члены сети не видят транзакции на канале.

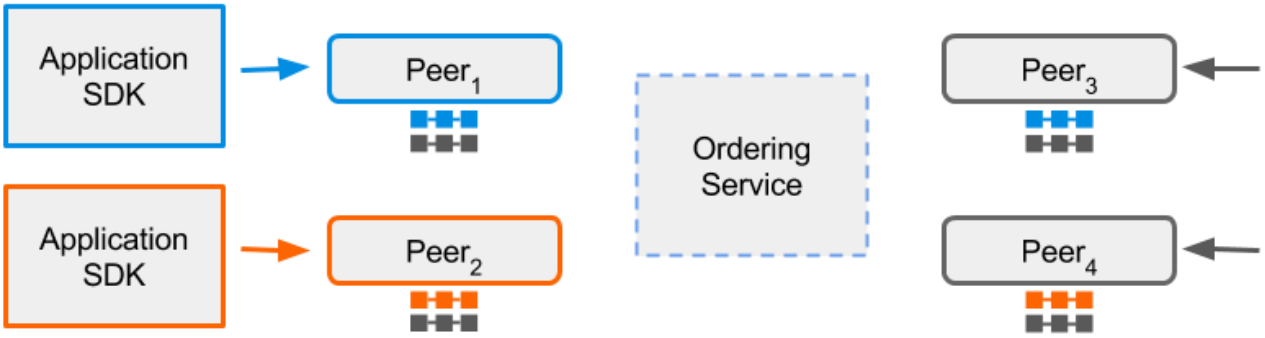

Рисунок 3.4 – Разделение общей сети с помощью каналов

Пиры могут принадлежать нескольким сетям или каналам. Пиры, которые участвуют в нескольких каналах, моделируют и передают транзакции в разные реестры. Сервис заявок одинаков для любой сети или канала.

3.2 Расчет основных параметров сети

Предположим, что у нас есть блокчейн-сеть, состоящая из 6 узлов, где узловой граф выглядит так, как показано на рисунке 2.

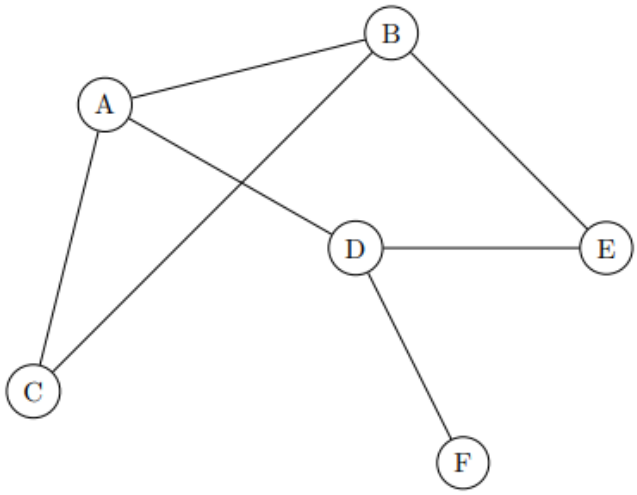

Рисунок 3.5 – Сеть узлов

Допустим, заданы следующие переменные, относящиеся к таб. 3.2, и разработчик блокчейна никак не может их изменить. Многие из них диктуются пользователями / узлами блокчейна, заставляя их регулярно меняться:

Размер заголовка-72 байта. [12]
Размер транзакции-102 400 байт. [12]
количество узлов в системе – 6.
Размер блока – 1 Мб (1 048 576 байт).
Сложность криптографической задачи = 500 [12]

Таблица 3.1 – Мощность майнинга

Мощность майнинга по узлам(hashes/second)	A	B	C	D	E	F
	1000	1500	500	750	900	1100

Таблица 3.2-пропускная способность узлов

		A	B	C	D	E	F
Пропускная способность	A	x	9	10	7	-	-
	B	-	x	6	-	10	-
	C	-	-	x	-	-	-
	D	-	-	-	x	7	10
	E	-	-	-	-	x	-
	F	-	-	-	-	-	x

Общая мощность майнинга $= \sum_{i \in nodes} m(i) = 1000 + 1500 + 500 + 750 + 900 + 1100 = 5750$ hashes/second.

Количество транзакций в блоке(3.1):

$$N = \frac{\text{размер блока} - \text{размер заголовка}}{\text{размер транзакции}} \quad (3.1)$$

$$N = \frac{1\,048\,576 - 72}{102\,400} = 10.24$$

$$\text{Частота образования блоков} = \frac{\text{общая мощность майнинга}}{\text{сложность криптозадачи}} \quad (3.2)$$

$$\text{Частота образования блоков} = \frac{5750}{500} = 115 \text{ блоков/мин}$$

Количество транзакций n в секунду:

$$n = \frac{\text{Частота образования блоков}}{60} * N \quad (3.3)$$

$$n = \frac{115}{60} * 10.24 = 19.63$$

Время установления связи между узлами немного сложно рассчитать. Это происходит потому, что нужна пропускная способность между всеми узлами, даже теми, которые не имеют прямой связи. Для того чтобы получить их, сначала нужен "самый быстрый" маршрут между не связанными напрямую узлами.

$$\text{Inter nodes times} = \frac{\text{размер блока}}{b(i,j)} \tag{3.4}$$

где

b(i,j) – пропускная способность между узлами i и j, i ≠ j

$$AE = ABE = \frac{1}{9} + \frac{1}{10} = \frac{19}{90}, \text{сек}$$

$$BD = BED = \frac{1}{10} + \frac{1}{7} = \frac{17}{70}, \text{сек}$$

$$AF = ADF = \frac{1}{9} + \frac{1}{10} = \frac{19}{90}, \text{сек}$$

$$BF = BEDF = \frac{1}{10} + \frac{1}{7} + \frac{1}{10} = \frac{12}{35}, \text{сек}$$

$$CE = CAEB = \frac{1}{10} + \frac{1}{9} + \frac{1}{10} = \frac{14}{45}, \text{сек}$$

$$CF = CADF = \frac{1}{10} + \frac{1}{7} + \frac{1}{10} = \frac{12}{35}, \text{сек}$$

$$CD = CAD = \frac{1}{10} + \frac{1}{7} = \frac{17}{70}, \text{сек}$$

$$EF = EDF = \frac{1}{7} + \frac{1}{10} = \frac{17}{70}, \text{сек}$$

$$\text{Общая пропускная способность} = \max\{\text{inter nodes times}\} = \frac{12}{35}, \text{сек}$$

Для начала предполагается, что количество блоков, найденных в системе за заданный промежуток времени, следует пуассоновскому процессу. Давайте посмотрим на это так: существует множество узлов (большое n) и очень малая вероятность нахождения блока (малое p), создающего более или менее "стабильное" λ. В этом случае вероятность создания злоумышленниками нового блока представляет собой число блоков, найденных в единицу времени λ [17].

Добавим следующие обозначения:

Nt = число блоков, созданных в единицу времени (0,t) ~ Poisson(0.1t)
$P(fork) = \mathrm{P}(\{> 1 \text{ блока в единицу времени } (0, \Delta t)\}) = \mathrm{P}(\mathrm{N}(\Delta t) > 1)$
Распишем вероятность:

$$P(fork) = 1 - P(N(\Delta t) \leq 1$$

$$= 1 - P(N(\Delta t) = 0 - P(N(\Delta t) = 1)$$

$$= 1 - \frac{e^{-0.1 \cdot \frac{12}{35}} \left(0.1 \cdot \frac{12}{35}\right)^0}{0!} - \frac{e^{-0.1 \cdot \frac{12}{35}} \left(0.1 \cdot \frac{12}{35}\right)^1}{1!}$$

$$= 1 - e^{-0.1 \cdot \frac{12}{35}} - 0.1 \cdot \frac{12}{35} \cdot e^{-0.1 \cdot \frac{12}{35}}$$

$$\approx 0.966295 - 0.03313013$$

$$\approx 0.00057449$$

Другими словами вероятность добавления в сеть блока злоумышленником с этими заданными параметрами была бы около 0,057%

2.6.1 Расчет параметров отказоустойчивости сети на основе технологии блокчейн, состоящей из 2 узлов.

Готовность отражает способность системы непрерывно выполнять свои функции.

Фактор доступности - это вероятность того, что система будет работать в любой момент времени.

MTTR (Mean Time to Repair) - это среднее время восстановления [30].

MTBF (Mean Time Between Failure) - это технический параметр, характеризующий надежность восстановленного устройства или технической системы. Измеряется статистически, путем тестирования различных приборов.

Вероятность отказа компонента во время MTBF равна 1, и если MTBF измеряется в годах, то вероятность отказа компонента в течение одного года равна:

$$\mathrm{P} = \frac{1}{\mathrm{MTBF}} \tag{3.5}$$

Вероятность выхода из строя всего узла в течение года рассчитывается как сумма вероятностей выхода из строя отдельных узлов:

$$\mathrm{P_t} = \sum \mathrm{P} \tag{3.6}$$

Сбой дублированного компонента приведет к сбою сети только при условии, что резервный компонент также выйдет из строя в течение времени, необходимого для "горячей " замены компонента, который вышел из строя первым. Если гарантированное время замены компонента составляет 24 часа (1/365 года), то вероятность такого события в течение года:

$$P_d = \frac{P*P}{365} * 2 \qquad (3.7)$$

Вероятность одновременного наступления этих событий равна произведению их вероятностей. Для случая, когда сначала отказывает компонент № 2, а затем компонент № 1, вероятность будет одинаковой. Теперь, зная вероятность Pi отказа каждого из N компонентов (дублированных и не дублированных) сервера, можно рассчитать вероятность отказа сервера в течение одного года. Давайте проведем расчет следующим образом: как уже было сказано выход их системы из любого компонента будет означать выход из строя сервера в целом.

Таблица 3.3-параметры надежности сетевых компонентов

Компоненты узла	Заявленная надежность			Число компонентов в сети	Вероятность отказа с учетом дублирования
	MTBF (часов)	MTBF (лет)	Вероятность отказа в течение года		
Блок питания	135 000	15.41	0,06489	2	0,0000231
HDD	100 000	11.42	0,08757	2	0,0000420
Кулер	80 000	9.13	0,10952	2	0,0000657
Материнская плата	110 000	12.56	0,07962	2	0,0000347
Видеокарта	90 000	10.27	0,09737	2	0,0000519
CPU	200 000	22.83	0,04380	2	0,0000105
Для сети в целом:			0,4828		0,0002279

Исходя из приведенных выше значений, можно рассчитать вероятность отказа узла в течение года:

$$P_{t(1)} = 0.4828$$

Вероятность выхода из строя всей сети в течение года:

$$P_t = 0.0002279$$

Вероятность одновременного отказа узлов вычисляется по формуле:

$$P_{t(2)} = \frac{P_{t(1)} * P_{t(1)}}{365} * 2 = \frac{0.4828 * 0.4828}{365} * 2 = 0.001277$$

Поскольку отказы компонентов равномерно распределены по времени, зная вероятность отказа узла в течение года, можно определить время, необходимое для отказа (время, через которое узел выйдет из строя со 100% вероятностью):

$$\text{MTBFc} = \frac{1}{P_{t(2)}} = \frac{1}{0.001277} = 4388 \text{ лет}$$

Фактор доступности - это вероятность того, что компьютерная система будет находиться в рабочем состоянии в любое время.

Этот коэффициент определяется по формуле:

$$K = \frac{\text{MTBFc}}{\text{MTBFc} + \text{MTTR}} = \frac{4388}{4388 + 24} = 0.99727$$

MTTR (Mean Time to Repair) - среднее время восстановления.

3.3 Структура сети и архитектура приложения

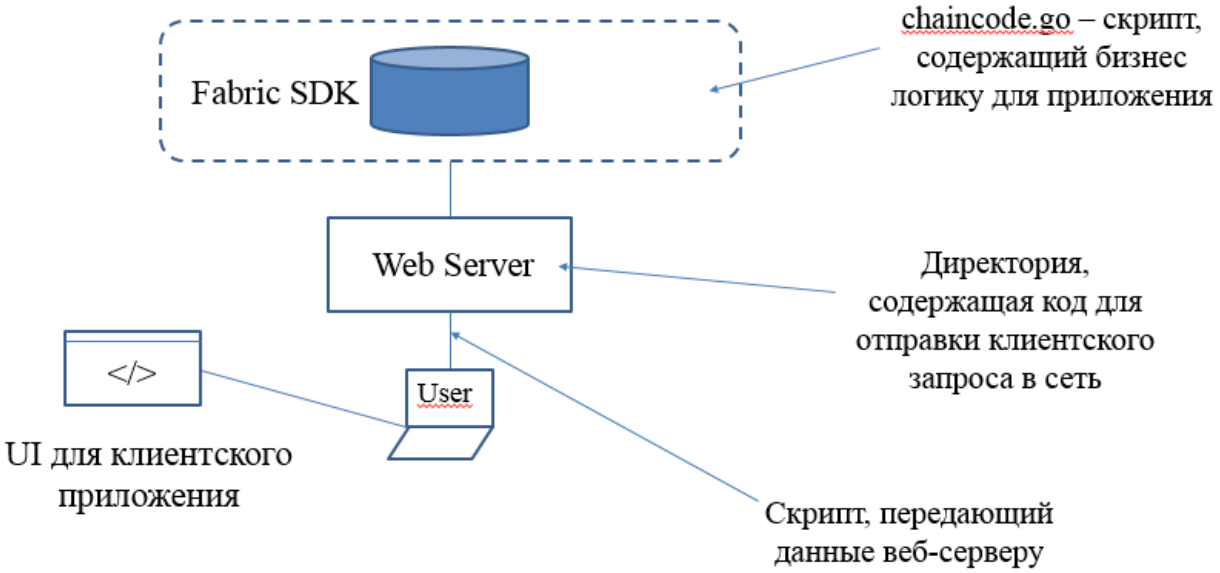

Рисунок 3.6 – Архитектура разрабатываемого приложения

Давайте рассмотрим поближе каждый элемент системы.

Основная часть разработанного приложения - это SDK, который включают в себя Node SDK, Java SDK и CLI, поэтому он объединил все понятия в единый Fabric SDK. Напомню, что смарт-контракты на самом деле являются компьютерными программами, которые обладают логикой для выполнения

транзакций и изменения состояния распределенного реестра. Student-chaincode.go - это файл кода blockchain, который содержит всю бизнес-логику приложения. Он развернут на узлах сети, и из бэкенда приложения он используется SDK для вызова функций в смарт-контракте.

```
func (s *SmartContract) recordTuna(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

        if len(args) != 5 {
                return shim.Error("Incorrect number of arguments. Expecting 5")
        }

        var tuna = Tuna{ Vessel: args[1], Location: args[2], Timestamp: args[3], Holder: args[4] }

        tunaAsBytes, _ := json.Marshal(tuna)
        err := APIstub.PutState(args[0], tunaAsBytes)
        if err != nil {
                return shim.Error(fmt.Sprintf("Failed to record tuna catch: %s", args[0]))
        }

        return shim.Success(nil)
}
```

Рисунок 3.7 – Функция recordTuna() из файла tuna-chaincode.go

Chaincode в Fabric является ничем иным, как обычными смарт-контрактом. Chaincode может быть написан на Go, Java и, в ближайшем будущем, JavaScript. В диссертации рассматривается реализация Chaincode на Go [30].

Небольшое предисловие о том, как Chaincode используется на стороне узла сети:

1. Узел получает запрос на добавление записи и ищет docker-контейнер. Если он не находит его, создает новый docker-контейнер с Chaincode внутри, если он находит его, то использует найденный.

2. После создания контейнера он запускает исполняемый файл chaincode.go, который подключается к gRPC узлу и начинает прослушивать его инструкции.

3. Когда запрос Invoke / Query поступает на узел, он отправляет команды в исполняемые файлы в исполняемые файлы обрабатывают их и отправляют результат обратно.

В языке GO нет таких понятий, как класс, конструктор и деструктор (вместе с соответствующими зарезервированными словами). Однако существуют структуры, заимствованные из C++, к которым можно привязать функции, поэтому в GO можно создавать код в стиле ООП. Наличие "сборщика мусора" облегчает работу с памятью, по сравнению с C или C++. Есть также указатели, но арифметика для них не предусмотрена. Поэтому, даже зная адрес переменной, невозможно итерироваться относительно нее. Это делается по соображениям безопасности. Также отсутствуют заголовочные файлы и неявные преобразования типов. Многопоточность поддерживается на уровне языка, каналы используются для связи потоков [29].

Для начала взаимодействия с Hyperledger Fabric клиенты используют HTML-файл под названием index.html, который содержит необходимый пользовательский интерфейс для клиентского приложения. Соответственно, существует набор основных функций, которые пользователь может использовать для создания и хранения записи. После того, как пользователь заполнит все предложенные поля и нажмет кнопку «Создать» или «запрос», запись отправляется на проверку и подтверждение другим узлам, после чего она успешно сохраняется в блоке, а состояние сети для всех узлов обновляется.

```
      <div class="form-group">
        <label>Create Mark Record</label>
        <h5 style="color:green;margin-bottom:2%" id="success_create">Success! Tx ID: {{create_student}}</
h5>
        <br>
        Enter mark id: <input class="form-control" type="text" placeholder="Ex: 11" ng-model="student.id">
        Enter Student name: <input class="form-control" type="text" placeholder="Ex: Kirill Sosnin" ng-
model="tuna.vessel">
        Enter discipline: <input class="form-control" type="text" placeholder="Ex: Packet Switching and
Softswitching Networks" ng-model="student.holder">
        Enter mark: <input id="createName" class="form-control" type="text" placeholder="Ex: 97" ng-
model="student.longitude">
        Enter timestamp: <input class="form-control" type="text" placeholder="Ex: 13.09.2018" ng-
model="student.timestamp">

        <input id="createSubmit" type="submit" value="Create" class="btn btn-primary" ng-
click="recordStudent()">
      </div>
```

Рисунок 3.8 – Создание записи на стороне клиента

Но простого заполнения полей недостаточно, должна существовать программа, которая будет обрабатывать введенные транзакции и отправлять их на проверку другим узлам. В нашем случае за это действие отвечает приложение. Файл содержит всю необходимую логику для того, чтобы входящие транзакции были правильно обработаны одноранговыми узлами. JavaScript не предназначен для создания автономных приложений. Язык программирования JavaScript обрабатывается непосредственно исходным кодом HTML-документа и интерпретируется браузером по мере загрузки документа. Используя JavaScript можно динамически изменять текст загруженного HTML-документа и реагировать на события, связанные с действиями посетителя или изменениями состояния документа или окна. Важной особенностью JavaScript является объектная ориентация. Программист может получить доступ к многочисленным объектам, таким как документы, гиперссылки, формы, фреймы и т. д. Объекты характеризуются описательной информацией (свойствами) и возможными действиями (методами). Поскольку JavaScript обеспечивает взаимодействие

пользователя с веб-страницей после ее загрузки, разработчики обычно используют его для решения следующих задач:

- динамическое добавление, редактирование и удаление HTML-элементов и их значений;
- проверка содержимое веб-форм перед отправкой на сервер;
- создания файлов cookie на машине клиента для сохранения и извлечения данных при последующих обращениях к сервису.

HTML был разработан как язык для обмена технической и научной документацией, пригодный для использования теми, кто не является специалистами форматирования документов. HTML эффективно справился с поставленной проблемой, определив небольшой диапазон семантических и структурных компонентов, используемых для создания простых и легких, но великолепно созданных файлов. Помимо упрощения структуры документа, HTML поддерживает гипертекст. Мультимедийные функции были добавлены позже. Сначала язык HTML был задуман и развит как средство структурирования и форматирования документов без привязки их к способам воспроизведения. Предпочтительно, чтобы текст с HTML-разметкой воспроизводился на аппаратном обеспечении с различным специализированным оборудованием (цветной дисплей современного компьютера, монохромный дисплей органайзера, небольшой дисплей сотового телефона или, возможно, приложения и оборудование для речевого воспроизведения копии) без структурных и стилистических искажений [29].

```
$scope.queryStudent = function(){

        var id = $scope.student_id;

        appFactory.queryStudent(id, function(data){
                $scope.query_student = data;

                if ($scope.query_student == "Could not locate the mark"){
                        console.log()
                        $("#error_query").show();
                } else{
                        $("#error_query").hide();
                }
        });
}

$scope.recordStudent = function(){

        appFactory.recordStudent($scope.student, function(data){
                $scope.create_student = data;
                $("#success_create").show();
        });
}
```

Рисунок 3.9 – Функции queryStudent() и recordStudent() в app.js

Смысл работы блокчейн-сети заключается в контроле целостности данных и противодействии их фальсификации. В самом известном блокчейне - в блокчейне Биткоина - все узлы соревнуются за право верифицировать транзакцию, потому что победитель получает вознаграждение в виде биткоинов. В HLF, как уже говорилось, члены сети имеют разные права и выполняют разные задачи. Поэтому за формирование блоков отвечает отдельная служба. Сервисные работы - это одна из самых интересных технических деталей в HLF. В отличие от других существующих платформ, здесь возможны различные механизмы достижения консенсуса, которые позволяют нам сформировать следующий блок. Целью проекта Hyperledger является создание платформы с открытым исходным кодом для создания и выполнения надежных приложений и отраслевых платформ для проведения бизнес-транзакций. Техническое сообщество, построенное вокруг проекта Hyperledger, является отличной платформой для обмена знаниями и предложениями между пользователями и разработчиками, что положительно сказывается на развитии проекта. В одной из реализаций технологии блокировки в рамках Hyperledger - Fabric мы используем различные алгоритмы подтверждения транзакций proof-of-work, которые демонстрируют хорошую пропускную способность. Все это делает Hyperledger отличным претендентом на роль блокчейн системы для бизнеса.

Последний элемент системы - это часть, отвечающая за связь клиентского приложения с распределенной базой данных. Для этого у Hyperledger есть целый каталог, в котором хранится набор программ, хранящих данные, введенные пользователем в блок, и предоставляющих информацию для запросов.

```
var mark id = array[0]
var timestamp = array[2]
var student name = array[1]
var mark = array[4]
var student name = array[3]


var fabric_client = new Fabric_Client();

// setup the fabric network
var channel = fabric_client.newChannel('mychannel');
var peer = fabric_client.newPeer('grpc://localhost:7051');
channel.addPeer(peer);
var order = fabric_client.newOrderer('grpc://localhost:7050')
channel.addOrderer(order);
```

Рисунок 3.10 – Настройка сети Fabric в функции recordStudent()

Новые блокчейн сервисы в IBM Cloud: данный вид сервисов позволяет создавать и управлять блокчейн сетями для обеспечения работы нового класса

распределенных приложений. Теперь разработчики смогут создавать цифровые активы и соответствующую бизнес-логику, что откроет возможность более безопасной и конфиденциальной передачи активов между участниками тестовой сети блокчейн с ограниченными и защищенными правами доступа. Используя новые технологии блокчейн от IBM, доступные на GitHub, разработчики смогут использовать полностью интегрированные инструменты DevOps для создания, развертывания, запуска и мониторинга приложений блокчейн на IBM Cloud. Приложения-блокчейн могут получать доступ к существующим транзакциям на распределенных серверах через API, поддерживая тем самым новые виды платежей, транзакций, поставок и бизнес-процедур. В результате реализации проекта по открытию исходного кода блокчейн технологии, бизнес получил усовершенствованный процесс управления идентификацией, основанный на последних достижениях в криптографии, мониторинге и контроле конфиденциальности, а также "умные" контракты на основе Java и Go с возможностью определения прав пользователей, имеющих доступ к этим контрактам.

3.4 Процесс создания приложения для регистрации записей о получении студентами дипломов

Для успешной установки Hyperledger Fabric на вашем компьютере должны быть установлены следующие утилиты: cURL, Node.js, менеджер пакетов npm, язык Go, Docker и Docker Compose.

Далее загружаются последние выпущенные образы Docker для Hyperledger Fabric. Чтобы подтвердить и посмотреть список только что загруженных образов Docker, запустите следующую команду:

$ docker images

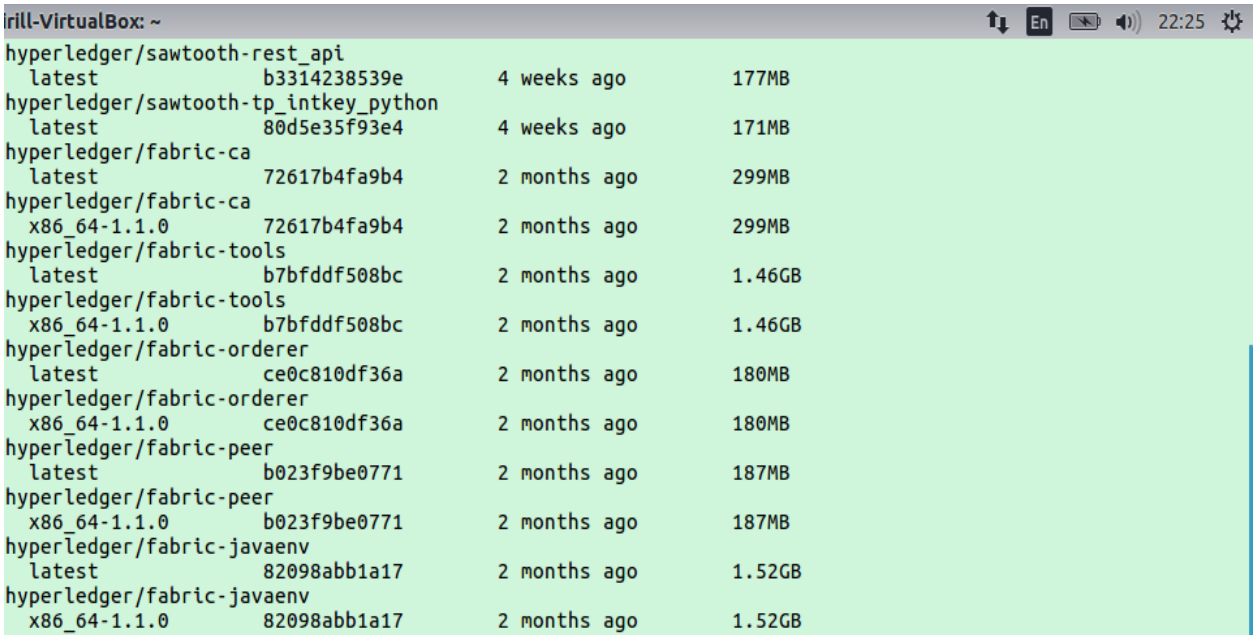

Рисунок 3.11 – Образы докера Hyperledger Fabric

Чтобы установить пример кода Hyperledger Fabric, который будет использоваться в проекте, выполняется команда:

$ git clone https://github.com/hyperledger/fabric-samples.git

Теперь, когда мы успешно установили Hyperledger Fabric, мы можем перейти к настройке простой сети, состоящей из двух узлов. Чтобы вернуться к нашему продемонстрированному сценарию, сеть включает в себя управление активами каждой проверенной, переданной транзакции, каждая из которых обслуживает двух равноправных членов и службу заказов.

Мы будем использовать образы Docker для загрузки нашей первой сети Hyperledger Fabric. Она также запустит контейнер для выполнения сценария, который присоединит узлы к каналу, развернет и проинсталлирует chaincode, а также выполнит транзакции.

Далее выполняется команда:

$ ./byfn.sh -m up

Докер на самом деле является открытой платформой для улучшения, эксплуатации, а также исполнения транзакций. Докер создан для быстрого развертывания и использования. Докер позволяет быстрее распространять ваш код, быстрее тестировать, быстрее составлять электронные таблицы, а также сократить время между написанием кода и введением кода в эксплуатацию. Докер осуществляет это с помощью легкой контейнерной платформы виртуализации, используя процедуры, а также утилиты, которые помогают контролировать, а также разворачивать вашу программу.

Из своего ядра докер позволяет управлять практически любой программой, которая надлежащим образом изолирована в контейнере. Защищенная изоляция позволяет одновременно запускать множество контейнеров на одном и том же хосте. Небольшая динамика контейнера, работающего без дополнительной нагрузки на гипервизор, позволяет получить гораздо больше от вашего аппаратного обеспечения.

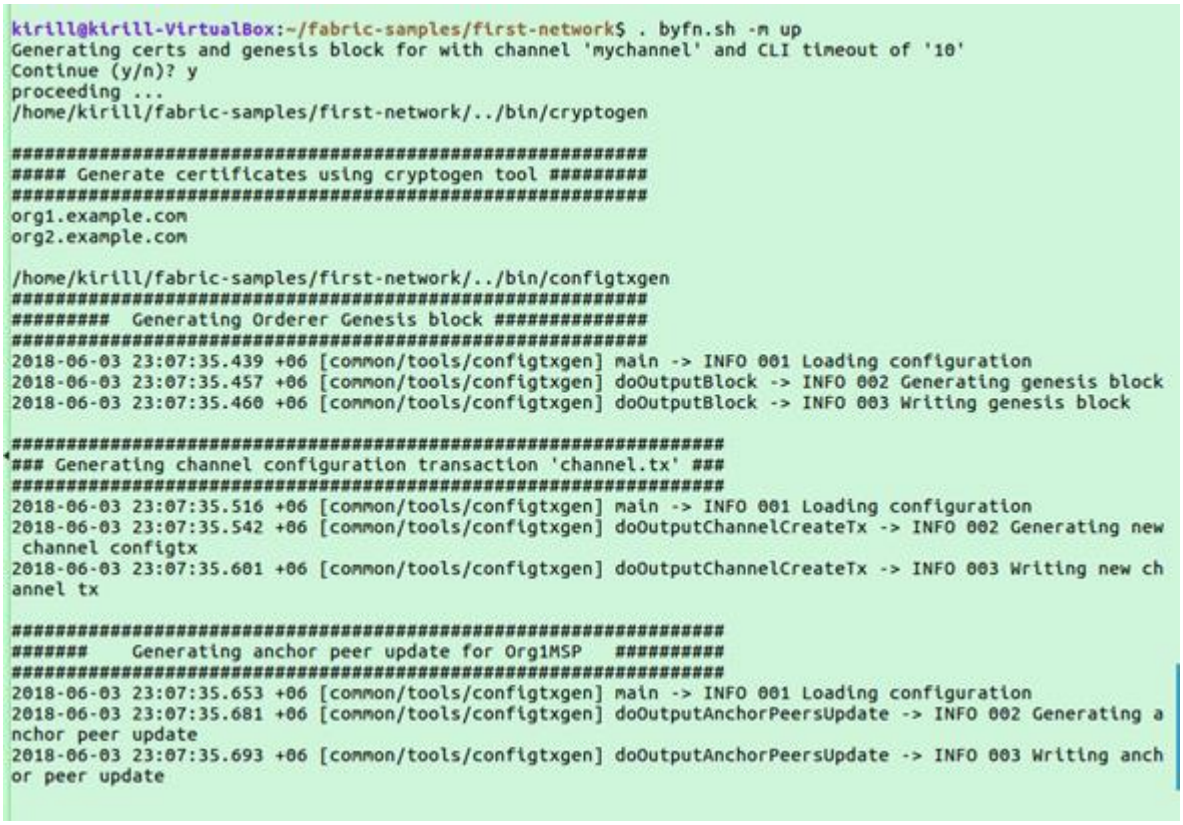

Рисунок 3.12 – Тестовая сеть Hyperledger Fabric

В блокчейн-программе блокчейн хранит состояние базы данных, в дополнение к неизменяемой истории транзакций, произведенных этим состоянием. Клиентское приложение используется для отправки транзакций в блокчейн. Приложения используют API для запуска смарт-контрактов. Внутри Hyperledger Fabric эти смарт-контракты помечаются как chaincode. Эти контракты фактически размещаются в сети, а также идентифицируются по времени создания и названию. API на самом деле доступны с помощью пакета разработки программного обеспечения или SDK. В настоящее время Hyperledger Fabric имеет 3 варианта для разработчиков: Node.JS SDK, Java SDK и CLI [31].

После того как мы убедились, что тестовая сеть функциональна и все ее компоненты фактически добавлены, можно приступать к созданию приложения для учета успеваемости студентов. Переходим в каталог, содержащий основу нашего приложения, и запускаем его с помощью следующих команд:

$ cd education/LFS171x/fabric-material/tuna-app

$. /startFabric.sh

Node или Node.js -это фактически программная платформа, зависящая от движка V8 (перевод JavaScript в машинный код), который превращает JavaScript из очень специализированного языка в язык общего назначения.

Node.js дает возможность JavaScript работать вместе с продуктами ввода-вывода через API (составленный на языке C), подключать различные внешние

библиотеки, созданные на разных языках, снабжая их вызовами из кода JavaScript. Node.js используется в основном на сервере. Node.js основан на асинхронном и событийно-ориентированном (или реактивном) программировании с неблокирующим вводом-выводом.

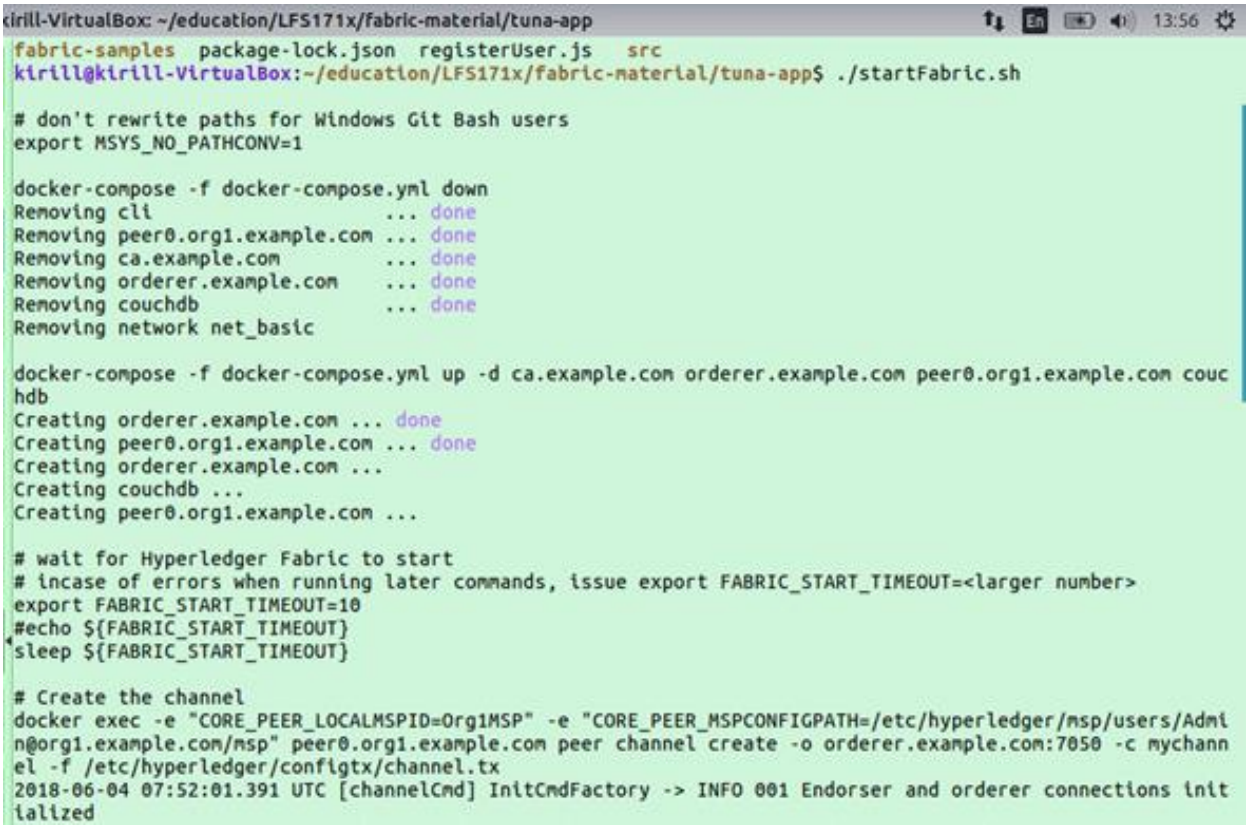

Рисунок 3.13 – Тестовая сеть Hyperledger Fabric

После этого необходимо зарегистрировать админа и пользовательские компоненты нашего приложения и активировать серверную часть. Для этого выполняются следующие команды:

$ node registerAdmin.js

Рисунок 3.14 – Окно терминала для регистрации администратора

Затем, необходимо зарегистрировать пользователя с правами администратора. Это можно сделать, набрав команду в терминале:

$ node registerUser.js

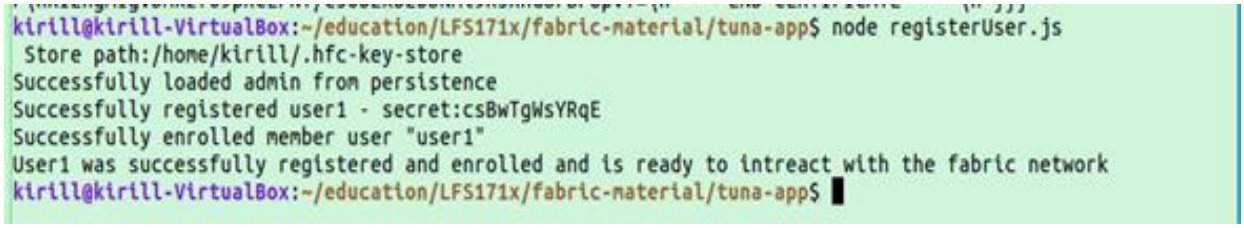

Рисунок 3.15 – Окно терминала для регистрации Пользователя

И наконец, необходимо активировать серверную часть децентрализованного приложения, чтобы взаимодействовать с ним по команде:

$ node server.js

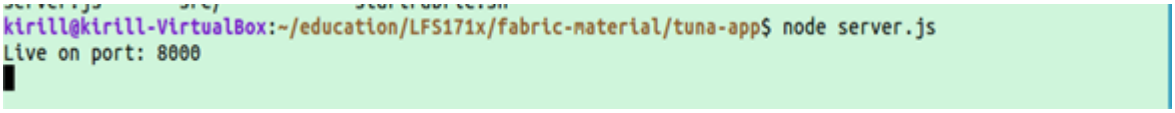

Рисунок 3.16 – Окно терминала для серверной части

Эта команда запустит сеть. Как показано на рисунке выше, мое приложение сейчас находится в рабочем процессе, поэтому я могу легко взаимодействовать с ним через любой Web-браузер, просто набрав в адресной строке "localhost:8000".

Затем, в окне браузера вы должны получить пользовательский интерфейс, похожий на тот, что изображен на картинке ниже:

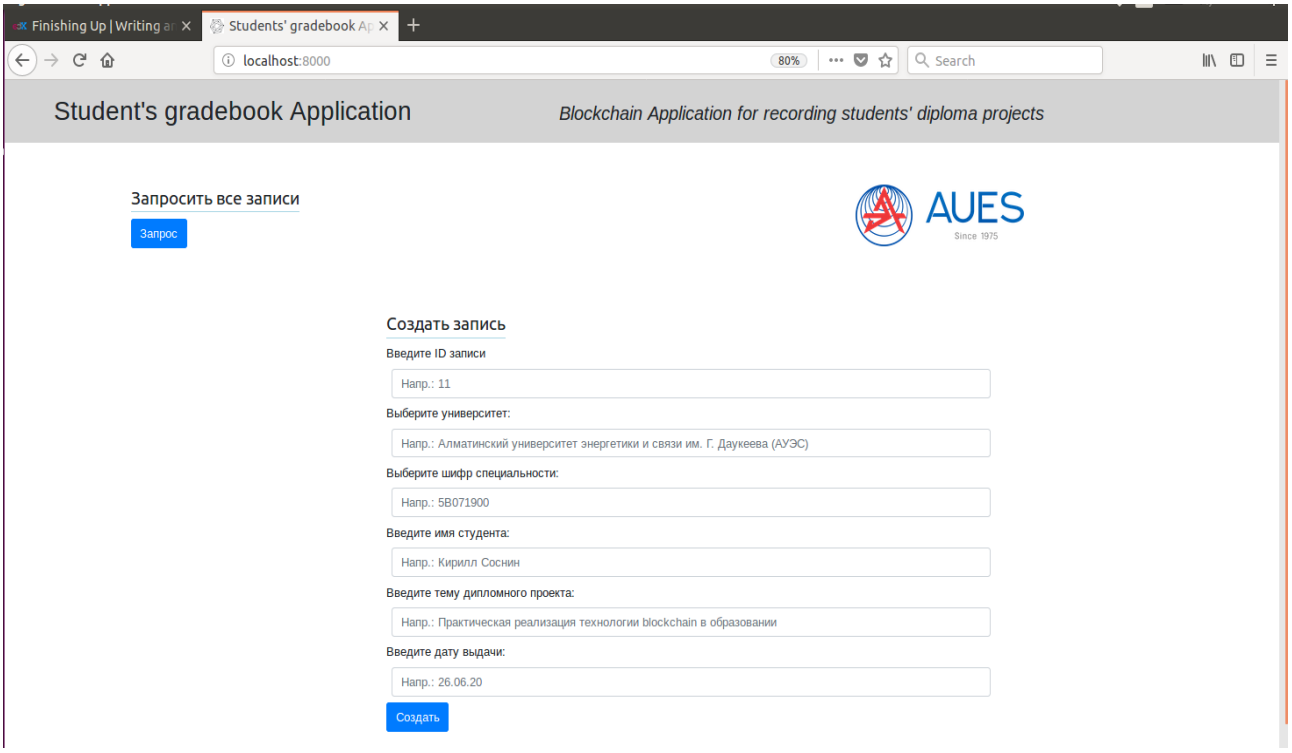

Рисунок 3.17 – Пользовательский интерфейс разработанного приложения

Как вы можете видеть приложение состоит из 2 основных частей:

1) пользователь может узнать все отметки, уже хранящиеся в распределенном реестре, нажав на кнопку "Query". Эта опция в настоящее время недоступна, сейчас она находится в процессе разработки.

2) Создание новой записи на специальных полях и подтверждение этой записи нажатием кнопки “Create”.

Созданное в рамках данной диссертации блокчейн приложение дает пользователю возможность заполнить такие поля как:

- название ВУЗа;
- шифр специальности;
- имя студента;
- тема дипломного проекта;
- отметка времени.

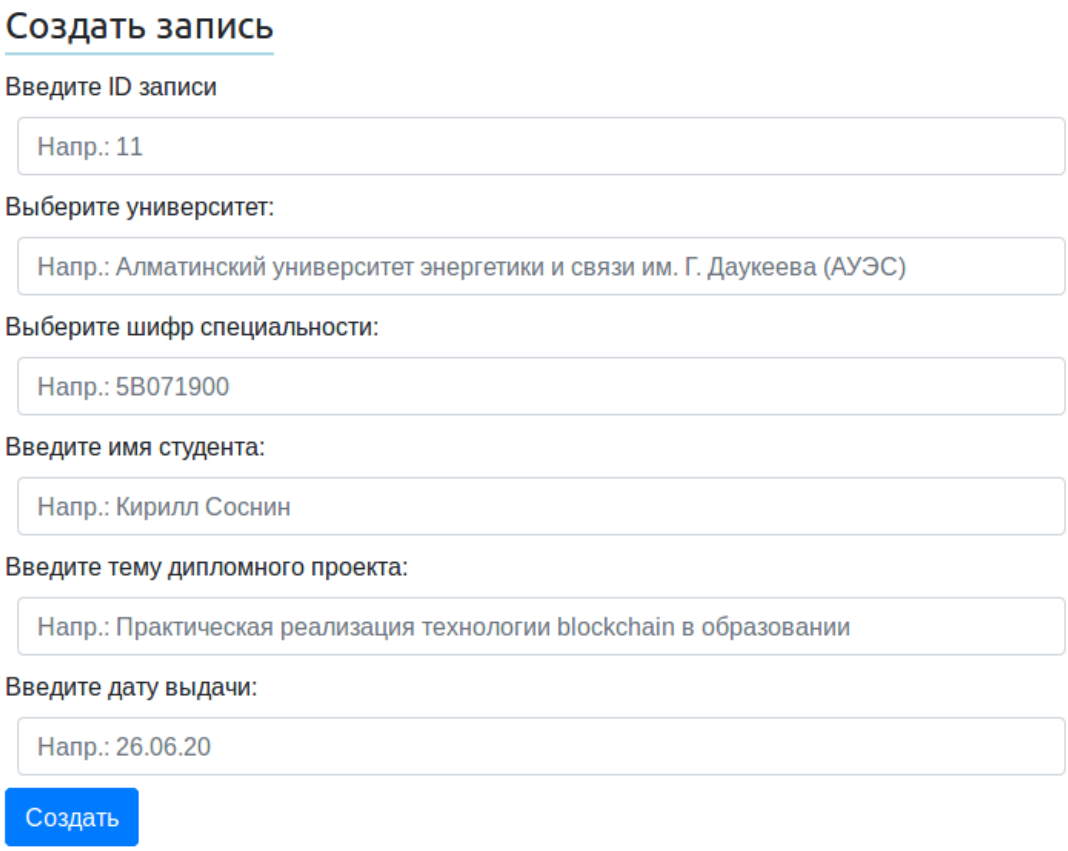

Рисунок 3.18 – Поля для создания записи

После заполнения всех полей Пользователь может создать необходимую запись, нажав кнопку "Создать". После этого запись будет принята сетью и сохранена в распределенной базе данных. Таким образом, текущее состояние блокчейна изменится. Эти изменения можно увидеть в окне терминала.

Свойство неизменности на самом деле достигается за счет использования криптографии, и никогда за счет доверия к бизнесу или даже физическим лицам. 2 самых простых криптографических алгоритма, используемых в блокчейн структуре, на самом деле являются электронные подписи и хэш-функции, которые обеспечивают целостность транзакций и, следовательно, несут ответственность за авторизацию. Важнейшей характеристикой журнала транзакций в блочной структуре является его неизменность. Именно это свойство означает, что вы не можете произвольно удалить транзакцию в журнале и даже включить в его середину совершенно новую информацию.

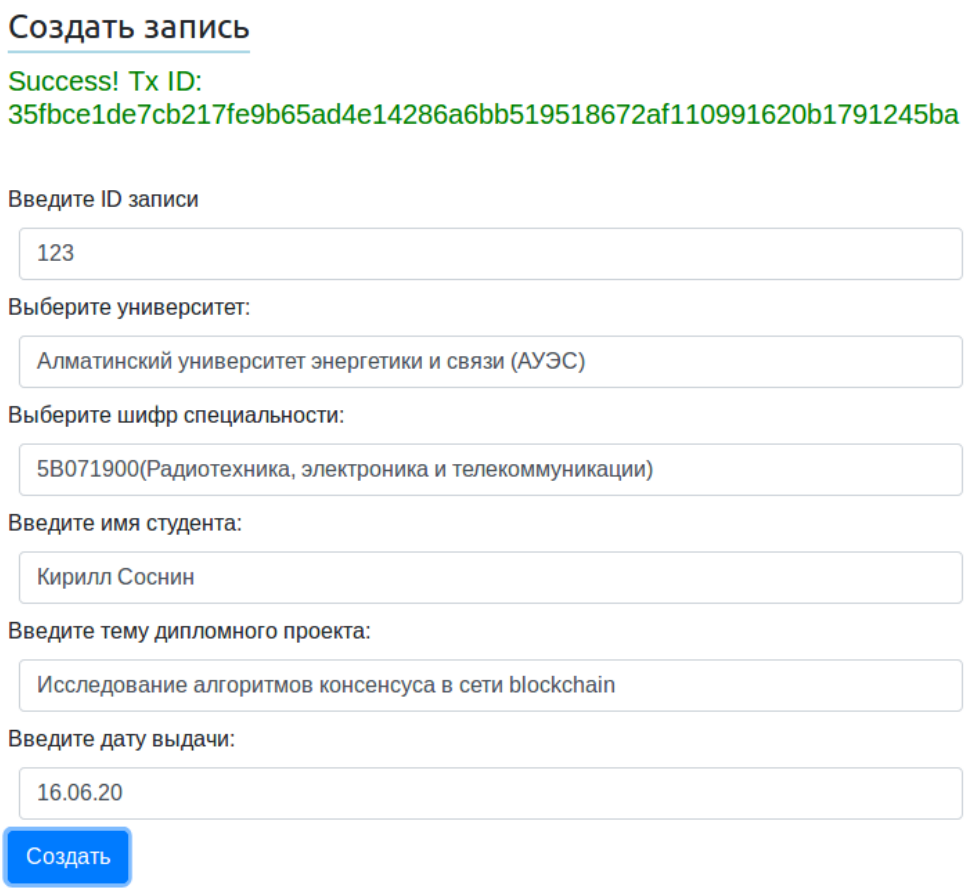

Рисунок 3.19 – Подтверждение новой записи

```
submit recording of a tuna catch:
[ '11', '97', '13.09.2018', 'PS&SN', 'Kirill Sosnin' ]
Store path:/home/kirill/.hfc-key-store
Successfully loaded user1 from persistence
Assigning transaction_id:  f4270152dcf16d28283318bdc9da97db41f5aee97a044c1cc40dd9f794836dae
Transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status - 200, message - "OK"
info: [EventHub.js]: _connect - options {}
The transaction has been committed on peer localhost:7053
Send transaction promise and event listener promise have completed
Successfully sent transaction to the orderer.
Successfully committed the change to the ledger by the peer
```

Рисунок 3.20 – Успешное изменение состояния реестра

Кроме того, в разработанном приложении существует возможность просмотреть все записи, уже внесенные в реестр ранее. За это действие отвечает опция «Запросить все данные»:

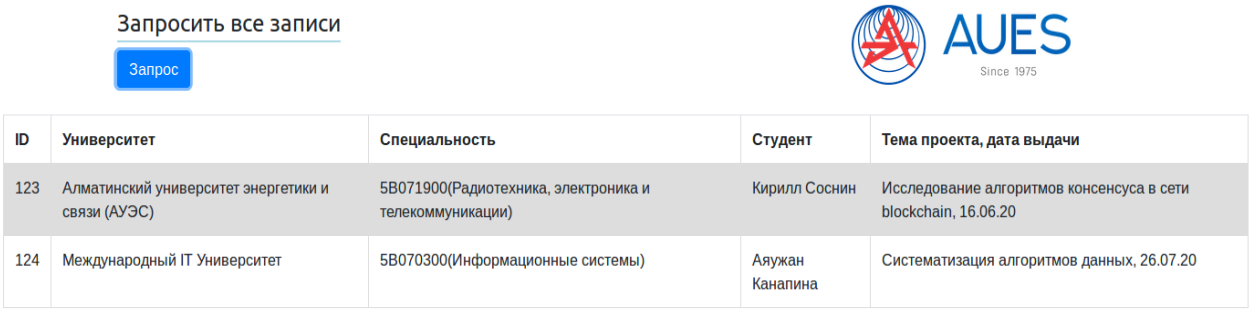

Рисунок 3.21 – Запрос данных, хранящихся в реестре

На данный момент были рассмотрены компоненты инфраструктуры Hyperledger Fabric, включая различные типы узлов в сети, частные каналы и функции баз данных. Было подробно разъяснено, как пользователь может взаимодействовать с сетью через приложение, которое позволяет пользователям запрашивать и обновлять состояние распределенного реестра. Построенное приложение обрабатывает пользовательский запросы и отправляет транзакции в сеть, которая затем вызывает chaincode. В настоящее время Fabric имеет три варианта для разработчиков: Node SDK, Java SDK и интерфейс командной строки или CLI [31].

Заключение

Данная магистерская диссертация является первой в своем роде, посвященной технологии распределенного реестра, или блокчейн. Блокчейн находит все более широкое применение в самых различных отраслях науки и промышленности, и эта диссертация является отправной точкой в дальнейшем изучении данной технологии. В ходе работы были изучены основные принципы технологии, такие как криптография, алгоритмы консенсуса и популярные платформы блокчейн для создания приложений на основе технологии распределенных реестров.

В качестве демонстрации возможностей технологии блокчейн было решено создать приложение, позволяющее записывать и хранить информацию о получении студентами высших учебных заведений свидетельств, подтверждающих успешное окончание учебного процесса и выдачу диплома. Для реализации данной задачи была выбрана платформа Hyperledger Fabric и изучены ее преимущества, проведено сравнение основных параметров производительности Hyperledger Fabric со своими прямыми конкурентами. Внедренное приложение полностью отражает основные принципы технологии блокчейн, такие как децентрализация и невозможность изменения уже сохраненных данных. Функциональность приложения позволяет вводить данные о студенте, наименовании учебного заведения, которое он окончил, тему дипломного проекта, а также дату получения свидетельства об окончании. Все данные, введенные после подтверждения остальными участниками сети, записываются в распределенную базу данных, после чего их нельзя изменить. Кроме того, вычисляются основные параметры построенной сети, такие как время одной транзакции, количество транзакций в одном блоке и отказоустойчивость такой сети.

Список литературы

1. Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59.

2. Ali M. Trust-to-trust design of a new Internet. – 2017. – p.58

3. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. – 2008. – p.47

4. Bahga A., Madisetti V. Blockchain Applications: A Hands-On Approach. – 2017.-pg. 14-18

5. Iansiti M., Lakhani K. R. The Truth About Blockchain //Harvard Business Review. – 2017. – T. 95. – №. 1. pg. 118-127.

6. 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. pg. 75-78.

7. Benchoufi M., Ravaud P. Blockchain technology for improving clinical research quality //Trials. – 2017. – T. 18. – №. 1. pg. 335.

8. Hackius N., Petersen M. Blockchain in logistics and supply chain: trick or treat?. – epubli, 2017.

9. Cretarola A, Figà-Talamanca G, Patacca M. A sentiment-based model for the BitCoin: theory, estimation and option pricing. (2014) p 6.

10. Garcia D, Tessone CJ, Mavrodiev P, Perony N. The crypto traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin economy. Journal of the Royal Society Interface. (2013) p.7.

11. Geweke, J. Measurement of linear dependence and feedback between multiple time series. Journal of the American statistical association,(1982). p.304-313.

12. Hafner, C. Cryptocurrencies with time-varying volatility(2018) p.5

13. Hayes AS. Cryptocurrency Value Formation: An empirical study leading to a cost of production model for valuing Bitcoin. Telematics and Informatics. (2016). p11.

14. Kristoufek, L. What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis. (2015, October) p.12.

15. Kroll, J. A., Davey, I. C., & Felten, E. W. (2013, June). The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In Proceedings of WEIS (Vol. 2013) p.13.

16. Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In International Conference on Financial Cryptography and Data Security pages 72–86. Springer, 2014.

17. Virtual Box Cloud Software. // virtualbox.org //Сервер компании Virtual Box. 2017. URL: https://www.virtualbox.org/ (date of the request: 15.02.2017).

18. K. Hooda S., Kapadia Sh., Krishnan P. Using TRILL, FabricPath, and VXLAN: Designing Massively Scalable Data Centers (MSDC) with Overlays. – N.:-Cisco Press, 2014. – 127c

19. W. Del Signore K. Measuring and Simulating Cellular Switching System IP Traffic // Bell Labs Tech Journal. – 2014. -№4. – С 159-180

21. Т.М.Попова, Т.В.Ходанова. Дипломное проектирование. Методические указания к выполнению экономической части.- Алматы: АИЭС. 2000.-27 с.

22. Hyperledger-fabricdocs Docomentation //Linux Foudation – 2015.p. 58-141

23. Explanation of Hyperledger Fabric block structure // URL: https://blockchain-fabric.blogspot.com/2017/04/hyperledger-fabric-v10-block-structure.html (date of request: 5.05.2019)

24. What is the Maximum transaction size in Hyperledger Fabric // URL: https://stackoverflow.com/questions/50225696/what-is-the-maximum-transaction-size-in-hyperledger-fabric(date of request: 8.05.2019)

25. What is Size of block in Hyperledger Fabric framework // URL:https://stackoverflow.com/questions/40954717/what-is-the-size-of-a-block-in-hyperledger-v0-6(date of request: 12.05.2019)

26. Vaquero L. M., Rodero-Merino L., Caceres J., and Lindner M. A Break in the Clouds: Towards a Cloud Definition. ACM Computer Communications Review, 39(1):50–55, Dec. 2008. Cited on 30

27. Zhang Q., Cheng L., and Boutaba R. Cloud Computing: State of the Art and Research Challenges. Journal of Internet Services and Applications, 1(1):7–18, May 2010. Cited on 30

28. Blockchain. Research paper. A. Shanti Bruyn, 2017. p. 78

29. Е.Хакимжанов. Расчет аспирационных систем. Дипломное проектирование. Для студентов всех форм обучения всех специальностей. – Алматы: АИЭС, 2002 - 30 стр

30. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Ph.d., University of California, Irvine, 2000.

31. Pautasso C., Zimmermann O., and Leymann F. Restful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In 17th International World Wide Web Conference, pages 805–814, New York, NY, Aug. 2008. ACM Press

Приложение А.1

Код скрипта startFabric.sh, запускающего сеть Hyperledger Fabric

```
#!/bin/bash
#

set -e

# don't rewrite paths for Windows Git Bash users
export MSYS_NO_PATHCONV=1

starttime=$(date +%s)

if [ ! -d ~/.hfc-key-store/ ]; then
        mkdir ~/.hfc-key-store/

# launch network; create channel and join peer to channel
cd ../basic-network
./start.sh

# Now launch the CLI container in order to install, instantiate chaincode
# and prime the ledger with our 10 tuna catches
docker-compose -f ./docker-compose.yml up -d cli

docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp" cli
peer chaincode install -n tuna-app -v 1.0 -p github.com/tuna-app
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp" cli
peer chaincode instantiate -o orderer.example.com:7050 -C mychannel -n tuna-app -v 1.0 -c '{"Args":[""]}'
-P "OR ('Org1MSP.member','Org2MSP.member')"
sleep 10
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp" cli
peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n tuna-app -c
'{"function":"initLedger","Args":[""]}'

printf "\nTotal execution time : $(($(date +%s) - starttime)) secs ...\n\n"
printf "\nStart with the registerAdmin.js, then registerUser.js, then server.js\n\n"
```

Приложение А.2

Код скрипта registerUser.js, регистрирующего в сети пользователя

```
'use strict';

var Fabric_Client = require('fabric-client');
var Fabric_CA_Client = require('fabric-ca-client');

var path = require('path');
var util = require('util');
var os = require('os');

//
var fabric_client = new Fabric_Client();
var fabric_ca_client = null;
var admin_user = null;
var member_user = null;
var store_path = path.join(os.homedir(), '.hfc-key-store');
console.log(' Store path:'+store_path);

// create the key value store as defined in the fabric-client/config/default.json 'key-value-store'
setting
Fabric_Client.newDefaultKeyValueStore({ path: store_path
}).then((state_store) => {
    // assign the store to the fabric client
    fabric_client.setStateStore(state_store);
    var crypto_suite = Fabric_Client.newCryptoSuite();
    // use the same location for the state store (where the users' certificate are kept)
    // and the crypto store (where the users' keys are kept)
    var crypto_store = Fabric_Client.newCryptoKeyStore({path: store_path});
    crypto_suite.setCryptoKeyStore(crypto_store);
    fabric_client.setCryptoSuite(crypto_suite);
    var tlsOptions = {
        trustedRoots: [],
        verify: false
    };
    // be sure to change the http to https when the CA is running TLS enabled
    fabric_ca_client = new Fabric_CA_Client('http://localhost:7054', null , '', crypto_suite);

    // first check to see if the admin is already enrolled
    return fabric_client.getUserContext('admin', true);
}).then((user_from_store) => {
    if (user_from_store && user_from_store.isEnrolled()) {
        console.log('Successfully loaded admin from persistence');
        admin_user = user_from_store;
    } else {
        throw new Error('Failed to get admin.... run registerAdmin.js');
    }
    return fabric_ca_client.register({enrollmentID: 'user1', affiliation: 'org1.department1'},
admin_user);
}).then((secret) => {
    // next we need to enroll the user with CA server
    console.log('Successfully registered user1 - secret:'+ secret);
    return fabric_ca_client.enroll({enrollmentID: 'user1', enrollmentSecret: secret});
}).then((enrollment) => {
  console.log('Successfully enrolled member user "user1" ');
  return fabric_client.createUser(
     {username: 'user1',
     mspid: 'Org1MSP',
     cryptoContent: { privateKeyPEM: enrollment.key.toBytes(), signedCertPEM: enrollment.certificate }
     });
}).then((user) => {
     member_user = user;
     return fabric_client.setUserContext(member_user);
}).then(()=>{
     console.log('User1 was successfully registered and enrolled and is ready to intreact with the fabric
network');

}).catch((err) => {
    console.error('Failed to register: ' + err);
        if(err.toString().indexOf('Authorization') > -1) {
                console.error('Authorization failures may be caused by having admin credentials from a
previous CA instance.\n' +
                'Try again after deleting the contents of the store directory '+store_path);
        }
});
```

Приложение А.3

Код скрипта server.js, запускающего сервер

```
//SPDX-License-Identifier: Apache-2.0

// nodejs server setup
// call the packages we need
var express       = require('express');        // call express
var app           = express();                 // define our app using express
var bodyParser    = require('body-parser');
var http          = require('http')
var fs            = require('fs');
var Fabric_Client = require('fabric-client');
var path          = require('path');
var util          = require('util');
var os            = require('os');

// Load all of our middleware
// configure app to use bodyParser()
// this will let us get the data from a POST
// app.use(express.static(__dirname + '/client'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

// instantiate the app
var app = express();

// this line requires and runs the code from our routes.js file and passes it app
require('./routes.js')(app);

// set up a static file server that points to the "client" directory
app.use(express.sttic(path.join(__dirname, './client')));

// Save our port
var port = process.env.PORT || 8000;

// Start the server and listen on port
app.listen(port,function(){
  console.log("Live on port: " + port);
});
```