Санкт-Петербургский государственный университет

***МАКАРОВ Михаил Алексеевич***

**Выпускная квалификационная работа**

***Система заключения пари на основе смарт-контрактов***

Уровень образования: магистратура

Направление: 02.04.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа: ВМ.5502 «Вычислительные технологии»

Научный руководитель:

доцент кафедры КММС, кандидат физико-математических наук, Корхов В.В.  
  
Рецензент: кандидат наук СПбГУ, ведущий программист   
ООО «ОСЕНСУС АРМ», Балян С.Г.

Санкт-Петербург

2020 год

Содержание

[Введение Актуальность проблемы3](#_Toc41847861)

[О проблеме4](#_Toc41847862)

[Цель работы5](#_Toc41847863)

[Постановка задачи6](#_Toc41847864)

[Обзор существующих систем и инструментов7](#_Toc41847865)

[Обзор платформ распределенных реестров9](#_Toc41847866)

[Ethereum10](#_Toc41847867)

[NEO19](#_Toc41847868)

[EOS21](#_Toc41847869)

[Глава 1. План разработки системы22](#_Toc41847870)

[1.1. Разработка системы22](#_Toc41847871)

[1.2. Тестирование работы системы24](#_Toc41847872)

[Глава 2. Разработка системы заключения пари на основе смарт-контрактов26](#_Toc41847873)

[2.1. Детальный обзор существующих методов и инструментов26](#_Toc41847874)

[2.3. Разработка собственного подхода29](#_Toc41847875)

[2.3.1. Описание системы на основе смарт-контрактов29](#_Toc41847876)

[2.3.2. Разработка собственного метода33](#_Toc41847877)

[2.4. Архитектура программного комплекса для заключения пари на основе смарт-контрактов34](#_Toc41847878)

[Глава 3. Апробация разработанной системы35](#_Toc41847879)

[3.1. Постановка эксперимента35](#_Toc41847880)

[3.2. Локальная сеть36](#_Toc41847881)

[3.3. Глобальная сеть38](#_Toc41847882)

[Результаты39](#_Toc41847883)

[Заключение41](#_Toc41847884)

[Список литературы42](#_Toc41847885)

Введение  
**Актуальность проблемы**

В современном мире довольно часто возникают спорные моменты и ситуации, для решения которых не обойтись без вмешательства третьей стороны. Так, например, существуют суды различных уровней для принятия решений по спорным ситуациям.

Несмотря на то, что суды позволяют решить множество различных вопросов, окончательное решение требует большого количества времени даже без учета процедуры апелляции и обжалования.

В контексте данной работы будет рассматриваться класс задач заключения пари, на примере спортивных событий. В дальнейшем этот класс задач можно масштабировать до заключения пари любого вида. Для того, чтобы заключить пари на спортивные события, существуют букмекерские компании, которые с одной стороны позволяют осуществить сделки удобным образом, но с другой стороны: во-первых, сделки заключаются непосредственно с букмекером, а во-вторых букмекер контролируют все денежные средства пользователя, а также имеет возможность, например, осуществлять дополнительные проверки в случае большого выигрыша и имеет право отказать в выплате положенного вознаграждения. Это часто приводит к возникновению судебных разбирательств.

В данной работе предлагается использовать технологии распределенных реестров, с целью формирования доверительной среды между участниками системы и снижения издержек, связанных с недоверием третьим лицам.

О проблеме

Если рассмотреть детальнее букмекерские конторы, то очевидно, что сделка заключается между человеком, который хочет сделать ставку и непосредственно конторой, при этом все денежные средства человека, от момента пополнения счета и до момента вывода средств контролируются букмекерской конторой. Также букмекерские конторы сотрудничают с теми, у кого есть доступ к большой аудитории, посредством реферальных программ.

Например, известный человек может рекламировать букмекерскую контору и в последующем получать проценты от проигрышей тех людей, кто зарегистрировался в системе по его приглашению.

Многочисленные случаи мошенничества в этой сфере говорят о том, что существует проблема чрезмерного контроля средств пользователей со стороны букмекерских контор, а также проблема чрезмерного вынужденного доверия пользователей.

Для того чтобы все участники заключения сделки были равноправными, предлагается использовать технологию распределенных реестров. Такой подход формирует доверительную среду между всеми участниками, позволяет заключать сделки напрямую между пользователями системы, а также освобождает пользователей от риска возможных судебных разбирательств благодаря прозрачности системы.

Цель работы

Глобальной целью работы является разработка системы с использованием технологии распределенных реестров, которая позволяет минимизировать риски всех ее участников. Также система должна удовлетворять следующему ряду требований:

Удобство и прозрачность. Система должна быть удобной для пользователя и обладать тем же функционалом, что и системы без наличия технологии распределенных реестров.

Скорость и масштабируемость. Распределенный реестр должен обладать довольно высокой скоростью совершения транзакций для того, чтобы поддерживать заключение пари как в обычном, так и в так называемом живом режиме.

Кроссплатформенное взаимодействие. Взаимодействие с реестром должно осуществляться как из браузера, так и из мобильного устройства.

Использование криптовалюты. Система должна иметь возможность интеграции с одной или несколькими наиболее популярными существующими криптовалютами. Из этого аспекта также вытекает требование анонимности пользователя.

Постановка задачи

Для достижения цели были поставлены следующие подзадачи:

* Обзор и анализ существующих систем и инструментов.
* Обзор и анализ платформ распределенных реестров.
* Разработка системы.
* Тестирование работы системы.

Обзор существующих систем и инструментов

На данный момент уже существуют некоторые букмекерские конторы с использованием технологии блокчейн, но к сожалению, многие из них уже либо перестали предоставлять свои услуги, либо обладают большим количеством недостатков. Рассмотрим некоторые из них:

* Augur [1] - одна из наиболее популярных платформ, где можно участвовать в прогнозах и делать ставки на прогнозы в BTC и Ethereum. Разработка началась в 2014, а окончательный запуск произошел в июле 2018. Augur реализован на базе Ethereum, а разработчики Ethereum являлись консультантами. По своей сути Augur представляет собой протокол для осуществления сделок на рынке предсказаний. В контексте данной работы протокол Augur представляет наибольший интерес и может быть взят за основу протокола ставок на спортивные события.
* Onehash [2] - была запущена в 2013 году. Для того, чтобы использовать систему необходим адрес BTC кошелька и электронная почта. К плюсам данной системы относят наличие мобильной версии системы и поддержку BTC. К минусам - ограниченный выбор платежных систем, а также из-за использования электронной почты и других данных - отсутствие анонимности. В последнее время система недоступна.
* Betmatch [3] - основана в 2018 году, поддерживает ставки на разные спортивные события и каждый участник может посмотреть ставки других участников. Базируется на платформе Ethereum и обладает собственными токенами XBM. К плюсам системы относят наличие мобильной версии и поддержку ETH, BTC, LTC и собственных токенов системы XBM, а к минусам - сомнение использования реальной технологии распределенных реестров и наличие дополнительных проверок при выводе средств.
* 1xBit [4] - основана в 2016 году, поддерживает более 20 криптовалют, к плюсам относят - полную анонимность и поддержку криптовалют, а к минусам отсутствие токенов и мобильной версии системы.

На основе выдвинутых требований (удобство и прозрачность, скорость и масштабируемость, кроссплатформенность, интеграция с криптовалютами) были сформулированы критерии выбора платформы распределенных реестров. В результате анализа существующих платформ распределенных реестров и статей схожей тематики было выявлено, что наиболее пригодными платформами для реализации системы являются Ethereum-подобные платформы, которые изначально были спроектированы с возможностью построения умной экономики. На основании этого были выбраны следующие кандидаты для более детального рассмотрения: Ethereum, NEO, EOS.

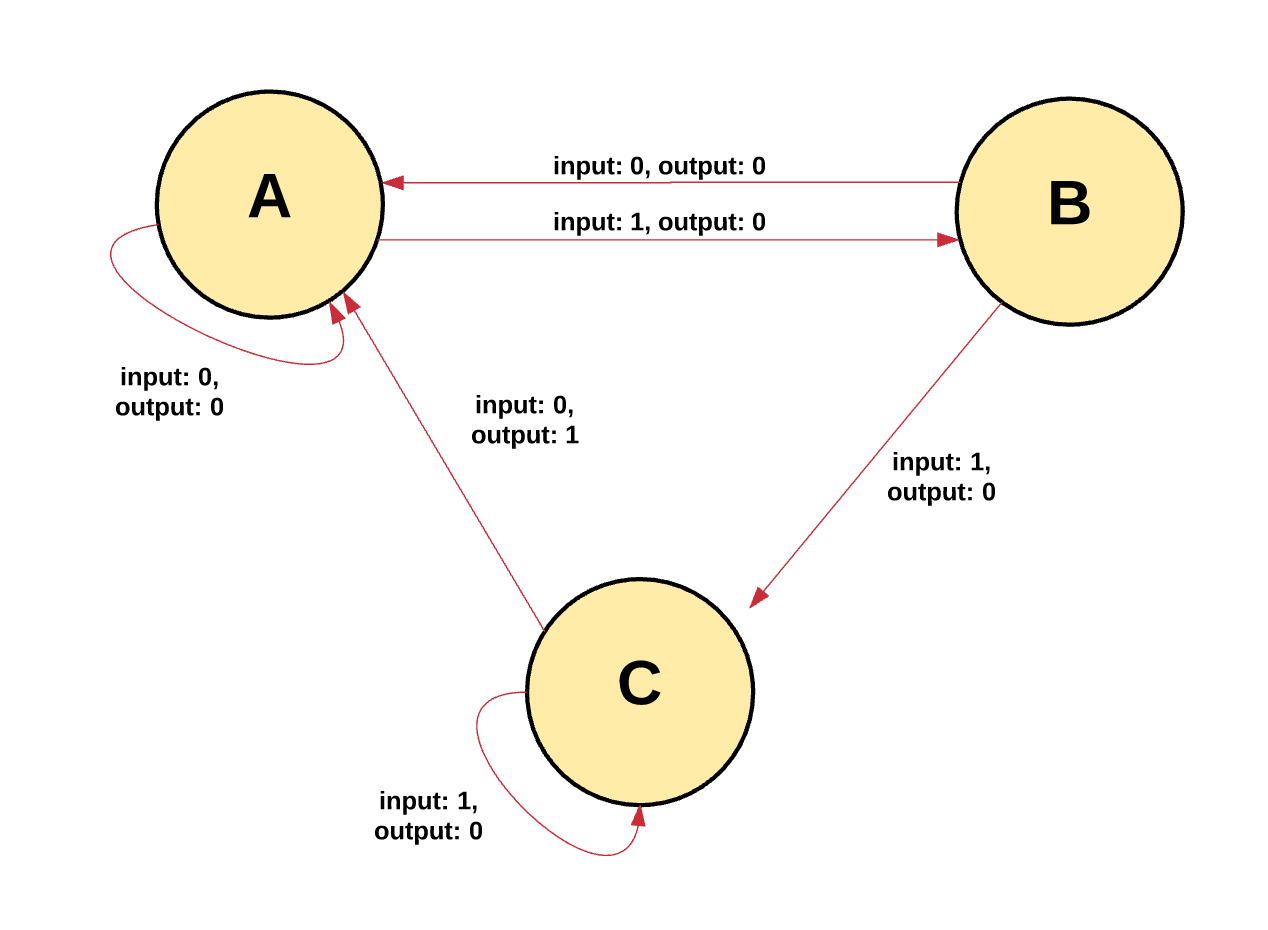
Обзор платформ распределенных реестров

На основе выдвинутых требований для системы можно сформулировать критерии выбора платформы распределенных реестров. Из требования скорости и масштабируемости можно вывести параметр tps (transactions per second - количество транзакций в секунду). Из требования кроссплатформенности следует поддержка реестром JavaScript взаимодействия. А из требования наличия поддержки криптовалют следуют поддержка анонимизации и реализованная криптовалюта на базе реестра либо механизм токенов.

Ethereum

Прежде всего следует детально рассмотреть принципы работы Ethereum. В Ethereum блокчейн представляет собой криптографически защищенную машину, которая может находиться в одном состоянии, при этом состояние доступно всем. На базе Ethereum реализована собственная криптовалюта. Операции с криптовалютой защищены математическими алгоритмами.

Машина реагирует на транзакции и переходит в определенное состояние, представляющее собой некую глобальную правду, которой все доверяют. При этом определенное состояние доступно каждому. Машина может считывать последовательности ввода, и на основе этих последовательностей совершать транзакцию, которая переводит машину в новое состояние (Рисунок 1). Существует нулевое состояние - genesis state (Рисунок 2). Перед запуском каждой транзакции, нулевое состояние переходит в финальное состояние, которое по факту отображает текущее состояние системы Ethereum.

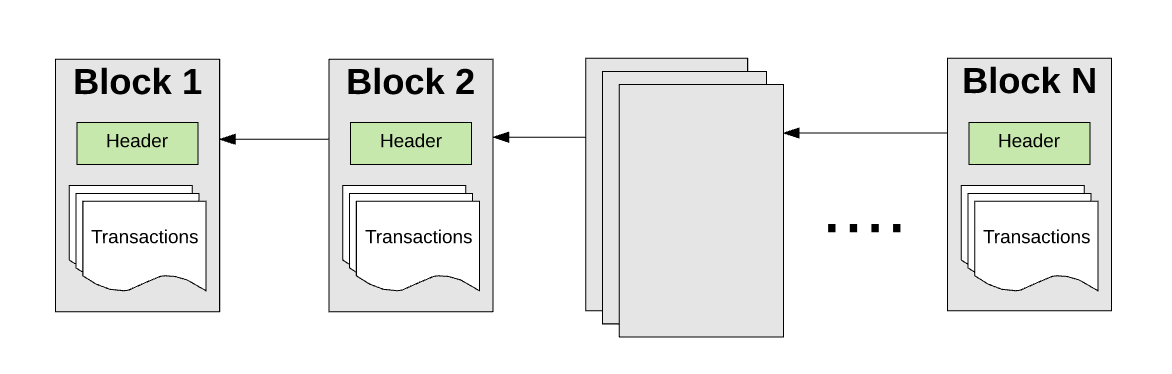


*Рисунок 1: Схематичное устройство машины Ethereum*



*Рисунок 2: Переход путем транзакций из нулевого состояние в текущее*

Текущее состояние машины Ethereum складывается из огромного количества транзакций, которые уже были совершены. Транзакции группируются в блоки (Рисунок 3).

*Рисунок 3: Схематичное представление цепочки блоков*

Перед тем как совершить изменение состояния транзакцию необходимо подтвердить, а процесс подтверждения транзакции называется майнингом. Группа узлов распределенной сети тратит свои вычислительные ресурсы, для того чтобы создать блок подтвержденных транзакций.

Любой узел, который объявил себя майнером, может принять участие в процессе подтверждения блоков транзакций. Каждый майнер предоставляет математическое доказательство решения задачи полного перебора, а в момент отправки блока в цепочку, это доказательство служит гарантией подтверждения блока.

Для того, чтобы блок был добавлен в блокчейн конкретным майнером, этому майнеру необходимо предоставить доказательство быстрее остальных. По-другому, такой алгоритм добавления новых блоков на основе соперничества в подборе математических доказательств называется алгоритмом консенсуса proof-of-work.

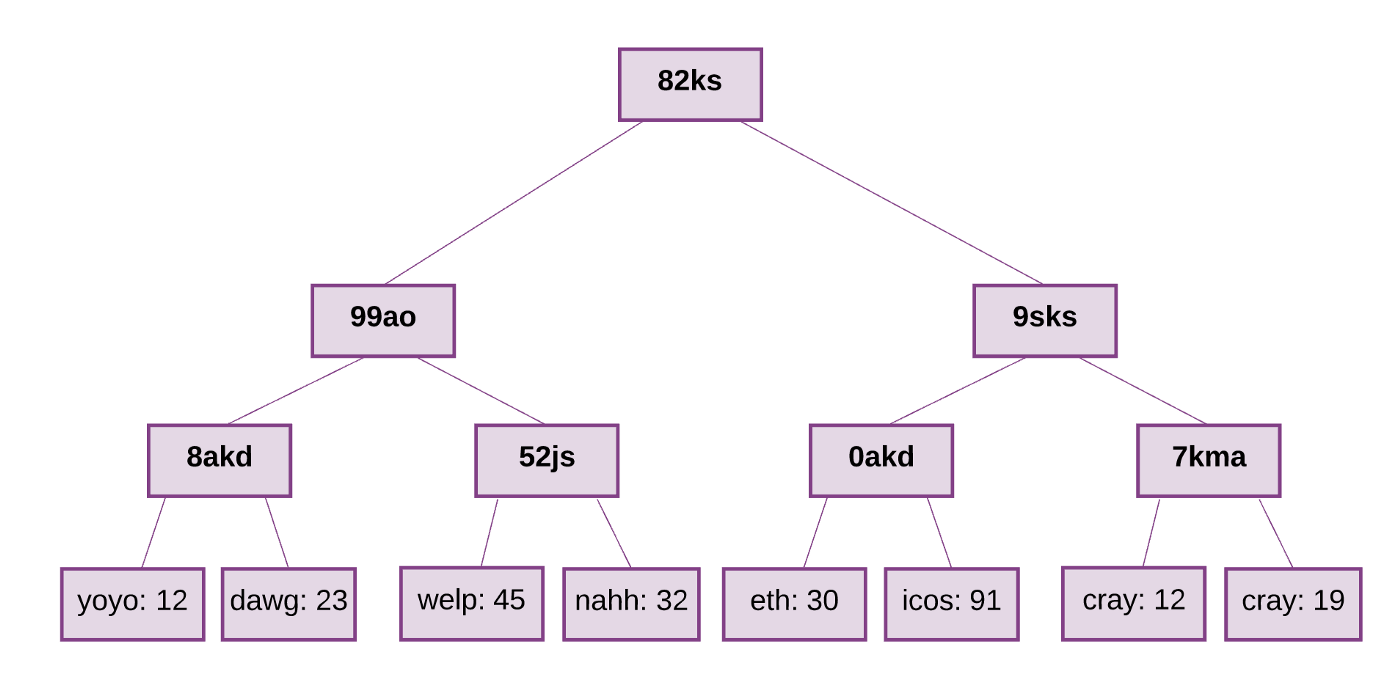
Ethereum содержит в себе следующие процессы и компоненты: аккаунты, состояния, gas, транзакции, блоки, майнинг и алгоритм консенсуса proof-of-work.

Аккаунты могут взаимодействовать между собой. Каждый аккаунт содержит состояние и 20-ти байтовый адрес. Адрес позволяет идентифицировать аккаунт в сети. Существует внешние и контрактовые аккаунты.

Внешние аккаунты могут отправлять сообщения другим внешним аккаунтам, или другим контрактным аккаунтам, подписывая транзакции своим приватным ключом. Передача сообщения между внешними аккаунтами представляет собой простую передачу значения, а сообщение между внешним аккаунтом и контрактным аккаунтом - запуск кода смарт-контракта. Контрактные аккаунты не могут инициализировать и запускать транзакции сами по себе, но могут выполнять транзакции в ответ на полученные транзакции. В связи с этим, любое изменение в сети Ethereum вызывается транзакциями внешних аккаунтов.

Глобальное состояние сети Ethereum складывается из соответствия адресов аккаунтов и состояний аккаунтов. В качестве структуры данных для хранения этого соответствия используется Merkle Patricia Tree [5] - вид бинарного дерева (Рисунок 4), в котором:

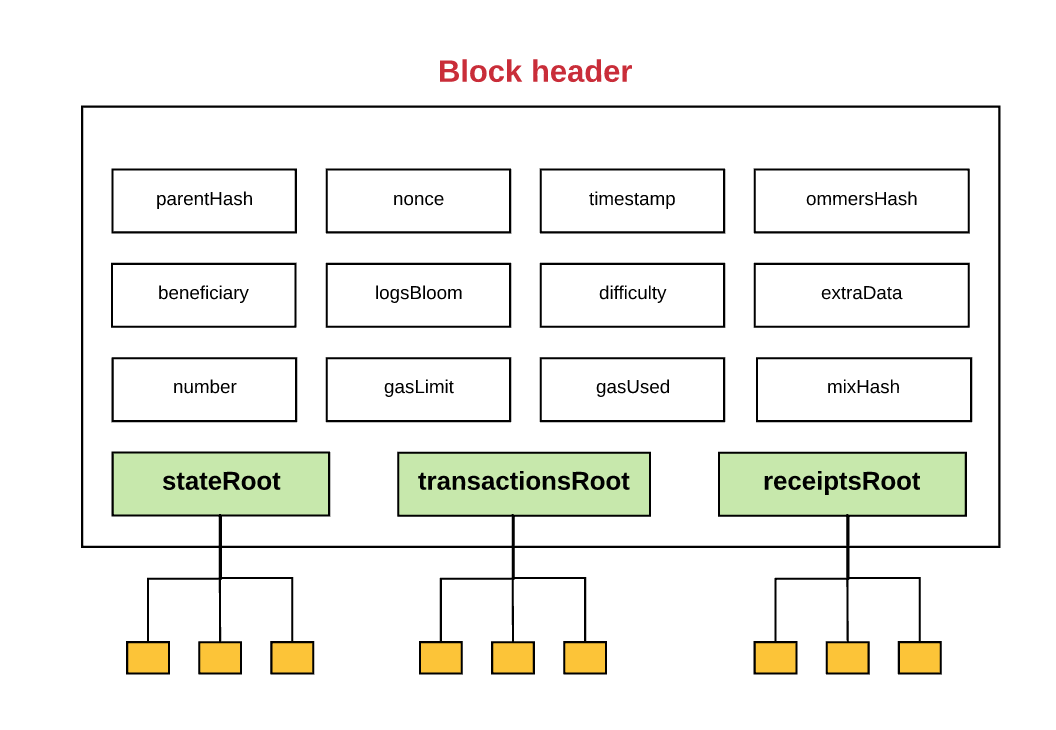
* Целевые данные хранятся в листьях дерева.
* Промежуточные узлы являются хэшем их двух потомков.
* Корень дерева, как и промежуточные узлы формируется из двух хэшей его потомков.



*Рисунок 4: Схематичное представление дерева Меркла*

Это дерево должно иметь ключ значения для каждого узла, хранящегося в нем. Начиная с каждого узла, ключ должен указывать, в какую ветку направиться, чтобы получить конечное значение, хранящееся в листьях.

Такая же структура используется для хранения транзакций. Каждый блок имеет заголовок, который содержит хэш корня трех деревьев Меркла: State tree, Transactions tree, Receipts tree (Рисунок 5).



*Рисунок 5: Внутреннее представление каждого блока*

В Ethereum существует два типа узлов: полновесный и легковесный. Полновесные синхронизируются с сетью путем выкачивания полной цепочки блоков и выполнением этой цепочки от нулевого состояния. Такие узлы используются майнерами, а выкачивание всей цепочки нужно для процесса майнинга. Легковесные выкачивают только цепочку заголовков, на основе этой цепочки заголовков можно осуществлять проверку транзакций. Так как у легковесных есть доступ к заголовкам блоков, содержащих хэши деревьев, они могут получить актуальную и подтвержденную информацию о транзакциях, балансах и аккаунтах.

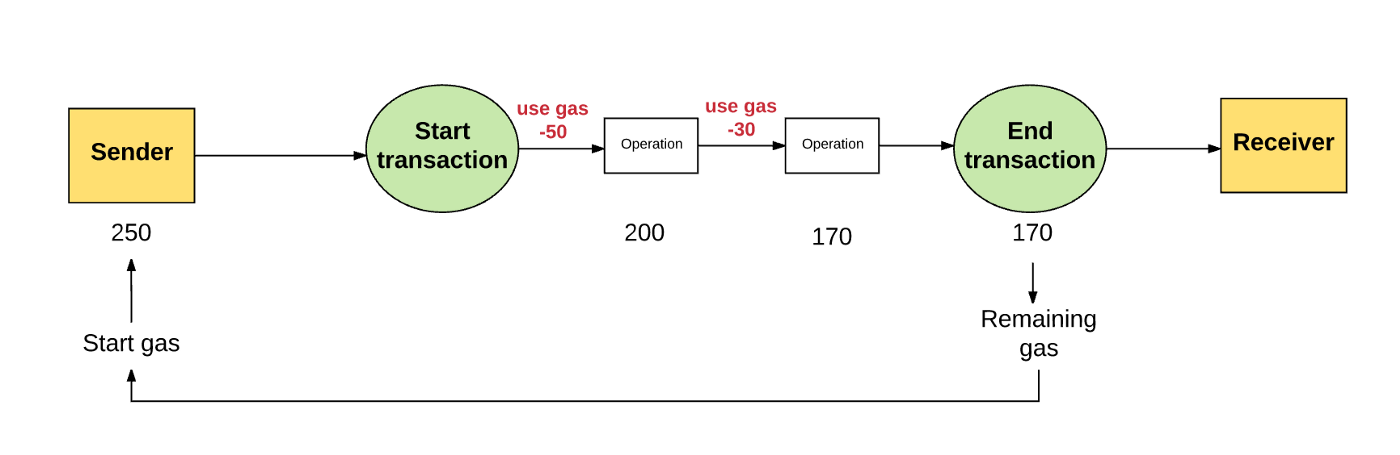
Это работает по причине того, что хэши в дереве Меркла генерируются снизу вверх, и если злоумышленник попытается подставить ложную транзакцию вниз дерева, то это изменение вызовет изменения хэша узла дерева, который находится выше и так далее до корня.

Каждый, кто читает доказательство может проверить формирование хеширования ветки и тем самым убедиться, что проверяемые данные находятся на нужном месте.

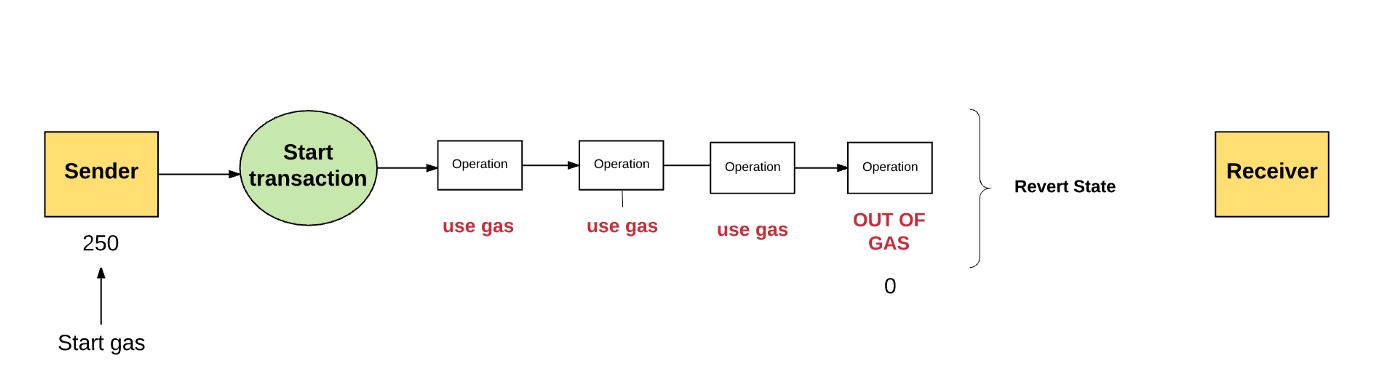
Преимущество использования дерева Меркла заключается в том, что корень криптографически зависит от данных, которые хранятся в дереве, тем самым для подтверждения данных не нужно сверять все данные дерева целиком, а достаточно сверить только хеши.

В Ethereum используется концепция вознаграждений и газа. Любые вычисления, происходящие внутри сети Ethereum, являются результатом выполнения транзакции. Газ - единица измерения вознаграждения.

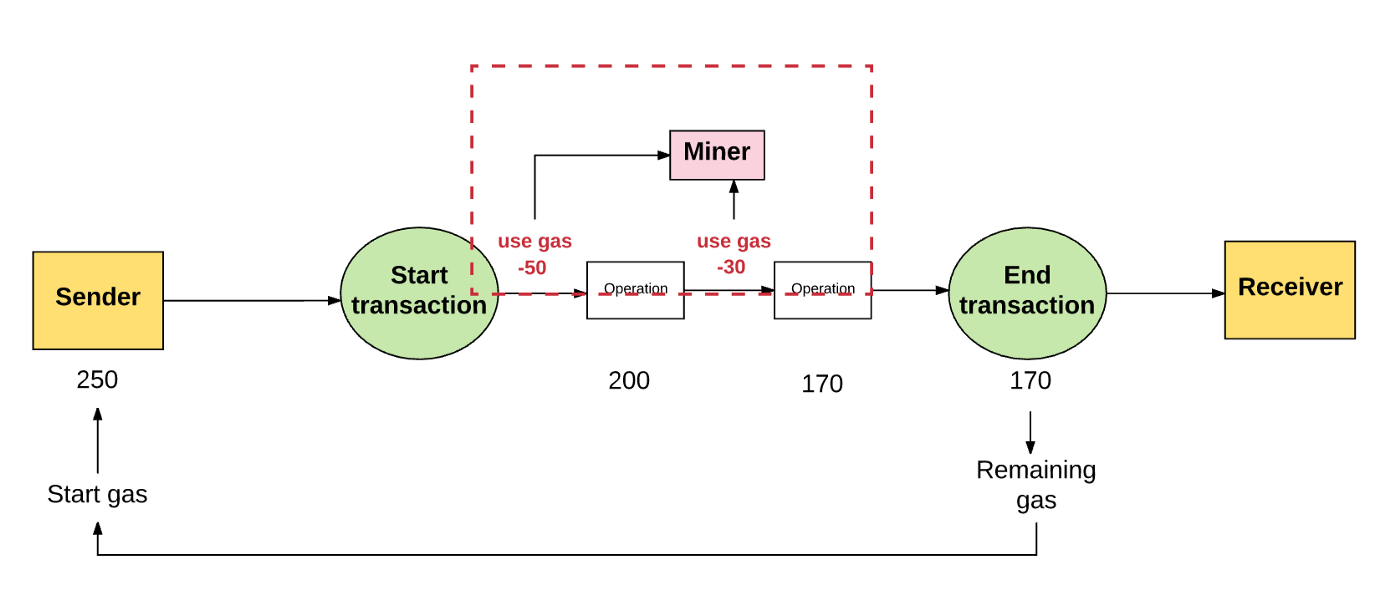
При каждой транзакции, инициатор указывает лимит газа, и цену газа. Их произведение означает максимум, который инициатор готов потратить на транзакцию. Если баланс инициатора транзакции больше лимита, то транзакция запускается, а остатки газа будут возвращены на баланс пользователя по исходному курсу (Рисунок 6).

  
*Рисунок 6: Описание механизма транзакции в случае остатка газа*

В случае если пользователь не может покрыть заявленный лимит, транзакция помечается не валидной, машина Ethereum откатится к состоянию до запуска транзакции, а остаток не появится в принципе (Рисунок 7).

  
*Рисунок 7: Описание механизма транзакции в случае нехватки газа*

Все средства, потраченные на лимит газа, поступают на счет майнера в качестве благодарности за произведенные вычисления (Рисунок 8).

  
*Рисунок 8: Принцип начисления газа майнеру*

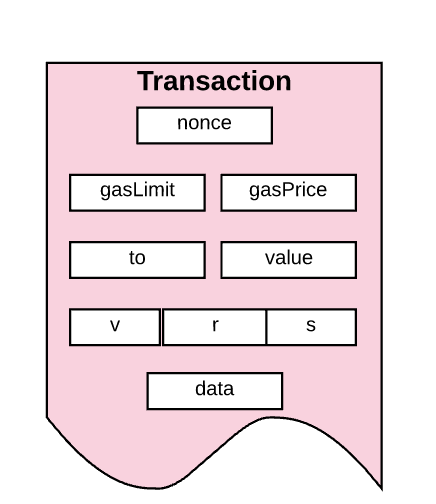
Чем больше заявленный лимит газа, тем большее вознаграждение получит майнер, и с тем большей вероятностью майнер включит транзакцию в новый блок, так как майнер может задать минимальный уровень лимита для включения транзакции в блок.

Газ используется не только для платы за вычисления, но и для платы за хранение данных, при этом если транзакция освобождает место, то плата за выполнение этой операций отменяется, а инициатору такой транзакции возвращается компенсация за освобожденное место.

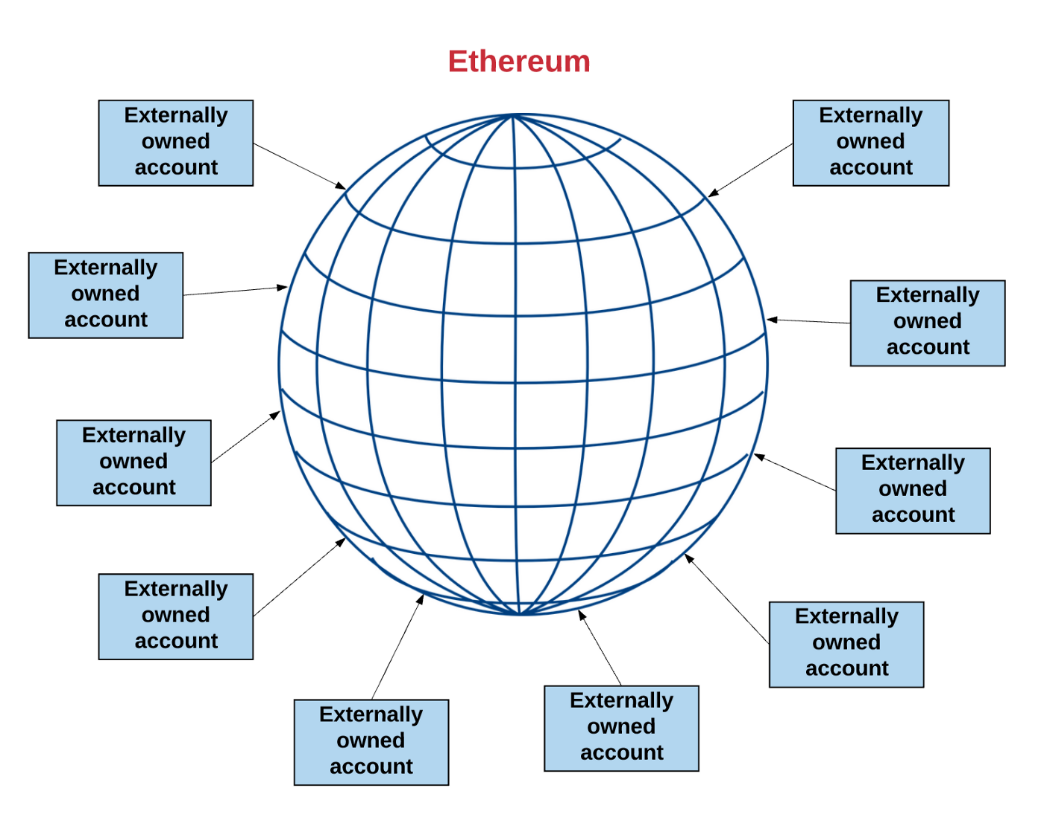
Каждая операция, выполняемая сетью, одновременно выполняется каждым полновесным узлом. Именно поэтому вычисления в Ethereum обходятся так дорого, вследствие чего смарт-контракты платформы Ethereum целесообразно использовать для реализации относительно простой бизнес логики и задач. Взимание платы предотвращает перегрузку сети лишней работой.

Ethereum - основанная на транзакциях машина состояний. Другими словами - транзакции, возникающие между аккаунтами, переводят глобальное состояние машины Ethereum от одного к другому.

Транзакция - криптографически подписанный фрагмент инструкции, который инициализируется внешним аккаунтом, сериализуется, а затем отправляется в глобальную сеть (Рисунок 9).

  
*Рисунок 9: Сущность транзакции*

Также транзакции можно считать связующим звеном между внешним миром и внутренним состоянием машины Ethereum (Рисунок 10).

  
*Рисунок 10: Схематичное взаимодействие внешних аккаунтов в глобальной сети*

Контракты могут взаимодействовать между собой с помощью вызовов функций или внутренних транзакций другого контракта. Такое взаимодействие отличается от обычных транзакций тем, что инициаторами этих транзакций являются именно контракты, а не внешние аккаунты.

Как уже было отмечено ранее, транзакции группируются в блоки. Блокчейн состоит из последовательности блоков, связанных между собой. Ожидаемое время поиска решения прямо пропорционально сложности, чем сложность выше, тем сложнее подобрать nonce. Тем самым изменяя сложность блока, можно регулировать время на валидацию блока.

Часть протокола, отвечающая за обработку транзакций, называется EVM (Ethereum Virtual Machine). EVM полная по Тьюрингу виртуальная машина и имеет архитектуру, базирующуюся на стеке, оперативную память и постоянную память.

EVM имеет свой низкоуровневый язык - EVM bytecode. Контракты разрабатываются на высокоуровневом языке Solidity, которые затем компилируются в EVM bytecode. Перед выполнением каких-либо вычислений, память и стек машины пусты. Далее EVM рекурсивно выполняет транзакции, вычисляя на каждой итерации глобальное состояние сети и внутреннее состояние машины. В конце каждой итерации существует 3 возможности:

* Столкнуться с исключением (недостаточное количество газа, неверная инструкция и тд.).
* Переход к следующей итерации.
* Контролируемая остановка по завершении всех итераций.

В случае контролируемой остановки блок будет добавлен в цепочку транзакций.

Ethereum обладает большим количеством преимуществ. Платформа реализует собственную криптовалюту, которая является довольно популярной, что обеспечивает надежность использования платформы. С помощью механизма токенов, можно реализовать интеграцию приложения и криптовалюты.

Взаимодействие с распределенным реестром можно осуществлять с помощью языка JavaScript.

К минусам платформы относят довольно низкое значение параметра tps, однако, в протоколе Ethereum 2.0 разработчиками планируется существенно увеличить значение этого параметра до 3000 путем использования другого алгоритма консенсуса.

NEO

NEO является конкурентом Ethereum и представляет собой прежде всего платформу для создания распределенных приложений. Так же, как и у Ethereum основная цель платформы NEO удобство для построения умной экономики. Платформа обладает собственными токенами NEO.

NEO содержит следующие компоненты:

* Цифровые активы - активы, которые существуют в электронном виде. В NEO можно регистрировать и распространять цифровые активы.
* Цифровые идентификаторы - механизм идентификации пользователя на основе сертификатов X509 [6].
* smartNEO - система смарт-контрактов, которые могут быть реализованы на различных языках программирования.
* NEO VM - виртуальная машина NEO для выполнения смарт-контрактов, напоминающая по архитектуре JVM (Java Virtual Machine) [7].
* Экосистема распределенных приложений

В NEO применяется алгоритм консенсуса dBFT (Delegated Byzantine Fault Tolerant). Держатель токена может выбрать узел, которому доверяет, путем голосования. Далее из выбранных узлов формируется группа, которая достигает консенсуса посредством алгоритма BFT [8] и генерирует блок. dBFT обеспечивает хорошую окончательность блока, что означает отсутствие ветвления цепочек блоков. На проведение голосования и формирование нового блока тратится около 15-20 секунд, а пропускная способность транзакций может достигать 1000 tps.

Также в официальной документации утверждается, что NEO поддерживает взаимодействие с реестром через JavaScript.

Несмотря на большое количество преимуществ, а именно: поддержку различных языков для написания смарт-контрактов, алгоритм консенсуса, позволяющий достичь 1000 tps и наличие собственных токенов, NEO обладает и недостатками. К ним относится слабый механизм анонимизации, так как в NEO используются цифровые активы и нужно связывать реального пользователя с цифровыми данными. Также минусом является недостаток алгоритма консенсуса dBTF - склонность к централизации.

EOS

EOS - относительно новая платформа, была разработана в 2018 году. Обладает собственной криптовалютой EOS. Ее основной целью являются бесплатные транзакции и увеличение количества транзакций в секунду.

При наличии плюсов EOS: большое количество транзакций в секунду, поддержка взаимодействия через JavaScript и наличие собственной криптовалюты по заявлению разработчиков, EOS на текущий момент довольно новая и нестабильная платформа, а также некоторые исследователи считают [9], что платформа по своей структуре больше похожа на облачный сервис по предоставлению вычислений без криптографии.

На основе обзора, была составлена Таблица 1, отражающая наличие требуемых характеристик у исследуемых платформ распределенных реестров. Несмотря на то, что Ethereum уступает в скорости NEO и EOS, эта платформа вызывает доверие и обладает хорошим механизмом анонимизации. Также Ethereum используется в протоколе Augur, что является весомым аргументом в пользу выбора Ethereum.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Платформа | Скорость | JavaScript | Анонимизация | Криптовалюта / Токены |
| Ethereum | 20 tps | + | + | + |
| NEO | 1000 tps | + | ? | + |
| EOS | 300000 tps | + | ? | + |

Таблица 1. Преимущества и недостатки рассматриваемых платформ распределенных реестров.

Глава 1. План разработки системы

1.1. Разработка системы

В результате анализа плюсов и минусов платформ распределенных реестров с учетом выдвинутых требований было принято решение использовать платформу Ethereum. Рассмотрим подробнее каждый из аспектов системы.

Удобство и прозрачность: на базе этой платформы имеется возможность разработки приложений с уже готовой инфраструктурой. Ethereum предоставляет глобальную сеть с большим количеством узлов, а также большое количество тестовых сетей [10] с различными алгоритмами консенсуса.

Скорость и масштабируемость: не смотря на количество узлов глобальной сети, Etherium позволяет обеспечить в среднем 20 транзакций в секунду, что в целом неплохо для не высоко нагруженных приложений.

Кроссплатформенность: работать с Ethereum можно напрямую из браузера и мобильных устройств с помощью расширения Metamask [11].  
В общем и целом существует 4 вида электронных кошельков:

* Hardware
* Desktop
* Web
* Paper

Metamask поддерживает первые три вида кошельков. В случае web-кошелька, Metamask представляет собой расширение для браузеров Chrome и FireFox. Для создания кошелька необходимо запомнить контрольную фразу из 12 слов (Mnemonic), на основе которой будет сгенерированы публичный и приватный ключи. Metamask хранит ключи в браузере, при этом промежуточные серверы отсутствуют, а кошелек можно восстановить по контрольной фразе.

С точки зрения разработчика, чтобы работать с Metemask используется библиотека web3js, которая осуществляет доступ к ключам, хранящимся внутри Metamask, а затем взаимодействует с помощью этих ключей со смарт-контрактами посредством запросов, реализованных библиотекой web3js [12].

Использование криптовалюты: в приложениях на базе Ethereum имеется возможность использования механизма токенов. Токен представляет собой смарт-контракт, реализующий один из интерфейсов, например, ERC20 [13] или ERC721 [14]. Следует отметить, что это наиболее распространенные стандарты, но существуют и другие. Смарт-контракт такого вида обладает методами для передачи единиц токенов и методами для проверки баланса токенов пользователя. Метод передачи принимает адрес отправителя, адрес получателя и идентификатор токена и реализует процесс передачи единицы токена от одного пользователя к другому. Метод проверки баланса должен быть доступен только владельцу токена и возвращать баланс единиц токенов. Внутри смарт-контракта такого вида реализуется защищенный механизм хранения адресов и балансов пользователей.

Преимущество использования токенов в разработке системы заключается в следующем: если децентрализованное приложение реализует интерфейс токена, например, ERC20, то токен этого приложения можно обменять на токен другого приложения, который реализует тот же интерфейс.

Токены можно купить с помощью криптовалюты Ethereum, тем самым использовать криптовалюту напрямую внутри системы.

1.2. Тестирование работы системы

Тестирование работы системы можно разделить на несколько составляющих:

* Тестирование непосредственно смарт-контрактов и их методов.
* Тестирование фронтенда клиентской части приложения.
* Тестирование работы приложения в локальной сети.
* Тестирование смарт-контракта после развертывания в глобальной сети.

Для каждого аспекта применяются различные инструменты, помогающие облегчить процесс тестирования.

Для тестирования смарт-контрактов используется инструменты Mocha [15] и Chai [16]. Эти инструменты также применяются и в тестировании JavaScript и Node JS кода. А благодаря использованию фреймворка Truffle [17], процесс написания тестов для смарт-контрактов, реализованных на языке Solidity, становится очень похожим на написание тестов для JavaScript кода.

В тестировании клиентской части системы удобно использовать фреймворк Ganache [18]. Этот инструмент позволяет довольно просто развернуть локальную сеть Ethereum, при этом в локальной сети будет заведено 10 аккаунтов с балансами 100 ETH внутри сети. Это позволяет совершать большое количество транзакций, при этом используя разные аккаунты для инициации транзакций.

Развернуть децентрализованное приложение, перед загрузкой в основную сеть Etherium можно в одной из следующих тестовых сетей:

* Ropsten - тестовая сеть, в которой используется алгоритм консенсуса proof-of-work. Была запущена в ноябре 2016 года. В феврале 2017 на эту сеть была произведена DdoS-атака, что привело к снижению скорости и расходованию большого количества дискового пространства. Работа сети была оперативно восстановлена уже в марте 2017. К преимуществам относится поддержка как протокола Geth, так и протокола Parity. Также эта сеть наиболее точно имитирует Ethereum, благодаря использованию того же алгоритма консенсуса. Внутри Ropsten можно майнить ETH, либо получить с помощью faucet-сервисов. К минусам относится уязвимости к спам-атакам. Время записи блока занимает в среднем 30 секунд.
* Kovan - тестовая сеть, которая использует алгоритм proof-of-authority. Была запущена в марте 2017 разработчиками из команды Parity. К плюсам сети относится отсутствие спам-атак. К минусам - отсутствие поддержки Geth протокола, а также получением ETH только через faucet-сервисы. Среднее время записи блока - 4 секунды.
* Rinkeby - тестовая сеть, которая была запущена в апреле 2017 командой разработчиков Ethereum. К плюсам сети относится устойчивость к спам-атакам. К минусам - поддержка только протокола Geth, неточное отражение поведения сети Ethereum из-за использования другого алгоритма консенсуса. В Rinkeby также получить ETH можно только с помощью faucet-сервисов. Время записи блока занимает в среднем 15 секунд.
* Goerli - тестовая сеть, которая использует алгоритм консенсуса proof-of-authority. Была запущена командой Parity в ноябре 2018. К плюсам сети относится поддержка всех протоколов Ethereum, к минусам - относительная новизна в силу того, что запуск сети произошел относительно недавно. Время записи блока занимает в среднем 15 секунд.
* Локальная сеть - c помощью фреймворка Ganache можно развернуть собственную локальную сеть. Локальная сеть использует Ethereum JS, что позволяет довольно точно симулировать клиентское взаимодействие и все доступные возможности реальной сети.

Глава 2. Разработка системы заключения пари на основе смарт-контрактов

2.1. Детальный обзор существующих методов и инструментов

Наибольший интерес представляет система Augur, так как в процессе ее создания разработчики консультировались непосредственно с основателем платформы Ethereum и на данный момент Augur является проектом с открытым исходным кодом. Согласно протоколу Augur, участники имеют право создать рынок, купить акцию рынка или продать акцию рынка в случае обладания. Под рынком подразумевается какое-то конкретное событие, которое произойдет в будущем. Жизненный цикл рынков состоит из следующих этапов:

* Создание - процесс инициализация рынка.
* Торговля - процесс покупки и продажи рынков, который начинается сразу после создания.
* Репортинг - процесс сообщений об исходах событий рынка пользователями. А пользователь, отправляющий сообщение называется репортером.
* Расчет - процесс заключения консенсуса исхода рынка и начисления средств пользователям на основе процесса репортинга.

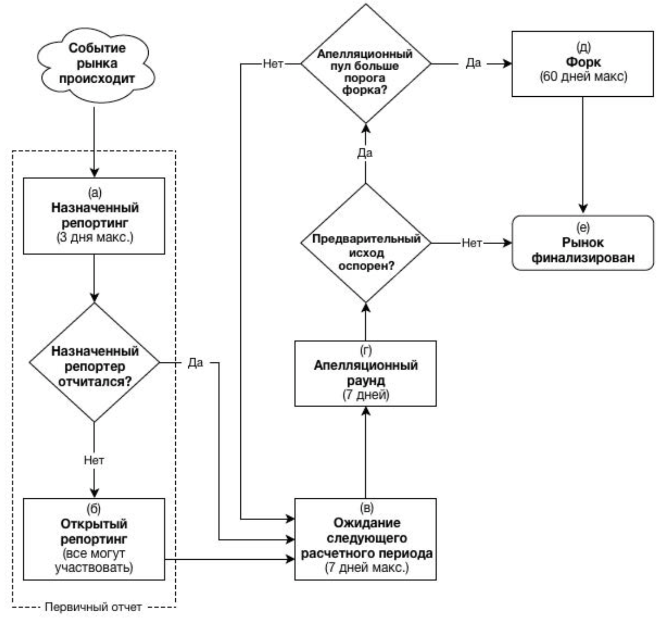
Платформа обладает собственным токеном REP (Reputation), который активно используется на этапе создания рынков и репортинга, в момент, когда пользователи сообщают о результатах события им необходимо поставить определенное количество REP на один из возможных исходов рынка, который, по их мнению, произошел. В случае если один или несколько пользователей поставили REP на исход, который не был признан большинством, то REP полученные от меньшинства пользователей распределяются между теми, кто поставил на исход, признанный большинством.

Держатели REP также получают комиссию от рынка в соответствие с суммой REP поставленной на исход на этапе репортинга. Если пользователь не планирует принимать участие в процессе репортинга, то токены REP такому пользователю не нужны.

  Создать рынок может любой пользователь, а на этапе создания рынка, создателю необходимо указать описание события, время окончания события, репортера события (репорт пользователя будет общедоступен, а у участников имеется возможность оспорить репорт), источник решения (ресурс, на котором можно наиболее точно получить информацию об исходе описываемого события), размер комиссии для трейдеров. С создателя взимается залог на валидность рынка в ETH и залог на бездействие выбранного репортера в REP и ETH. Все это упаковывается в Ethereum транзакцию и как только транзакция попадет в блок, рынок начнет действовать, а пользователям предоставляется возможность торговли.

Участники прогнозируют исходы событий путем покупки акций рынка события за ETH. Под акцией понимается доля единицы определенного исхода события.

Процесс репортинга состоит из нескольких раундов (Рисунок 11).



*Рисунок 11: Процесс репортинга*

По завершении процесса репортинга, наступает этап расчетов по рынку.  Трейдер может закрыть свою позицию либо продав свои акции другому трейдеру, либо дождаться этапа расчета.

2.3. Разработка собственного подхода

На основании выбора платформы распределенного реестра Ethereum, в системе были реализованы смарт-контракты на языке Solidity.

2.3.1. Описание системы на основе смарт-контрактов

В системе имеется несколько типов сущностей:

* Пользователь.
* Событие.
* Исход.
* Токен.

Сущность Токен – смарт-контракт, реализующий интерфейс ERC20. Токен можно использовать в качестве валюты внутри системы и обменивать на реальную криптовалюту Ethereum.

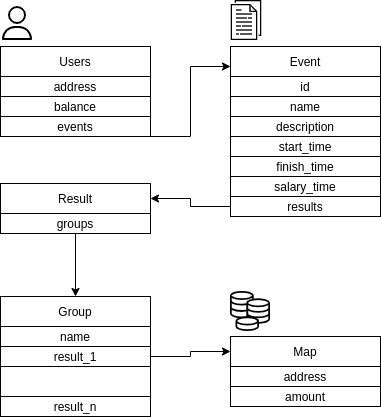
Сущность Пользователь - структура, содержащая Ethereum адрес пользователя системы и текущий баланс в токенах системы.

Сущность Исход - структура, содержащая описание исхода какого-либо события в реальной жизни. Включает в себя тип, для того, чтобы понять среди каких исходов происходило пари, связь адресов пользователей и их токенов на этот конкретный исход.

Сущность Событие - структура, содержащая имя, описание, время начала и конца события, которое произойдет в реальной жизни, время, после которого произойдет перечисление токенов тем пользователям, которые угадали исход события, набор исходов и текущую фазу события, которая может принимать одно из следующих значений:

* Заключение пари.
* Ожидание исхода.
* Проверка исхода.
* Событие завершено.

Внутреннее устройство организации хранения данных внутри смарт-контракта представлено на Рисунок 12.

  
*Рисунок 12: Внутренняя организация хранения данных смарт-контракта*

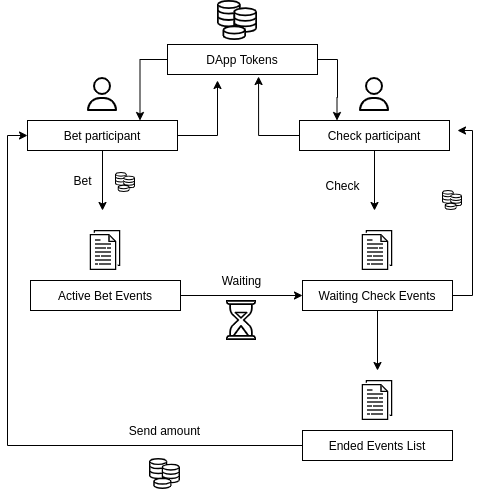
Помимо того, что в любой момент пользователю доступна покупка токенов системы, в системе существует 2 способа участия: заключение пари и проверка исходов событий. При входе в систему пользователю предоставляется выбор типа участия.

Если пользователь выбрал в качестве типа участия проверку исходов, то далее будет предоставлен список всех событий в фазе проверки исходов. Для проверки исхода, пользователю необходимо выбрать, конкретный исход в группе исходов события. За каждый проверенный исход пользователь получает небольшую вознаграждение.

Иначе пользователю будет предоставлен список всех событий, находящихся в фазе заключения пари и возможность создания своего события, на случай, если требуемого события в списке не оказалось. На этом этапе пользователь имеет возможность заключить пари, выбрав исход и указав желаемое количество токенов до наступления фазы события ожидания исхода. Заключив пари, пользователь автоматически соглашается с корректностью описания события и фактом того, что это событие действительно произойдет.

Заключившим пари пользователям доступны события в фазе ожидания исхода, проверки исхода и завершения.

Возможные действия пользователя представлены на Рисунке 13.

 *Рисунок 13: Возможные действия пользователя в системе*

С точки зрения жизненного цикла события, в момент инициализации происходит проверка текущего времени системы и времени начала события, времени конца события и времени перечисления токенов.

Предполагается, что коэффициенты на исходы события рассчитываются динамически в фазе ожидания исхода. Это означает, что пользователь не знает, какое вознаграждение он получит в момент заключения пари. Для решения этой проблемы, можно предоставить пользователю выбор вида коэффициентов. Динамические коэффициенты предполагают расчет в фазе ожидания исхода, а статические можно запрашивать, например, через API спортивного портала. Но даже в случае статических коэффициентов, итоговая выплата может отличаться, так как пользователи заключают сделки друг с другом, для расчета со всеми пользователями по статическим коэффициентам может не хватить суммы ставок на противоположные исходы. В этом случае придется производить расчет не по всей сумме ставки, а только по части суммы.

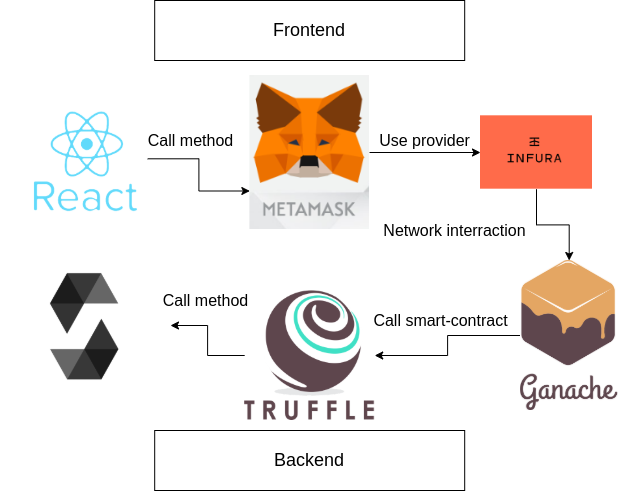
2.3.2. Разработка собственного метода

Одной из основных проблем предлагаемого решения является валидация исходов пари, заключенных между пользователями системы. Так как событие, по которому происходит начисление средств на счет пользователей, еще не произошло. Следовательно, требуется подтверждение информации о наступлении определенного исхода пари. В случае валидации исходов пользователями системы требуется платить комиссию, которая будет списана с участников пари. Предположим, что пользователь не хочет платить комиссию. Для таких пользователей можно внести альтернативный источник подтверждения исхода, а на этапе инициализации заявки пользователь может выбрать, каким способом будет подтверждена информация о наступлении определенного исхода:

* С помощью API новостного портала - по наступлению события из заявки, происходит обращение к API новостного портала, а после ответа, выполнится код, отвечающий за распределение средств между пользователями заявки.
* С помощью валидации исхода другими участниками системы (частично подход был описан выше) - по наступлению события проверяющим пользователям приходит список событий с исходами, чтобы получить комиссию за валидацию события им необходимо выбрать исход в течении фазы проверки исхода. Далее, исход набравший абсолютную сумму произведений числа голосов на значение авторитета пользователя признается произошедшим, а соответствующим пользователям, угадавшим исход будет начислено определенное количество токенов, при этом значения авторитетов пользователей валидаторов будут пересчитаны.
* Смешанный вариант - комбинация предыдущих двух вариантов, когда API новостного портала обладает большим весом. Это позволяет избежать атаки 51% [19] на децентрализованное приложение от недоброжелательных участников системы.

2.4. Архитектура программного комплекса для заключения пари на основе смарт-контрактов

Для того, чтобы облегчить процесс разработки смарт-контрактов на языке Solidity, был использован фреймворк Truffle для структурирования приложения и написания тестов, фреймворк Ganache, чтобы развернуть локальную сеть с готовыми аккаунтами внутри. Также в системе был использован инструмент Metamask для взаимодействия с локальной сетью посредством библиотеки web3js. В свою очередь библиотека web3js использует провайдер для того, чтобы непосредственно направлять запросы в сеть, было опробовано 2 вида провайдера: Infura [20] и HttpProvider. Клиентская часть приложения была реализована с помощью фреймворка React. Также, был реализован интерфейс токенов ERC20. Взаимодействие компонентов и фреймворков представлено на Рисунке 14.

  
*Рисунок 14: Схематическое представление взаимодействия компонентов и фреймворков внутри приложения*

Глава 3. Апробация разработанной системы

3.1. Постановка эксперимента

В качестве эксперимента можно развернуть систему в локальной сети и в одной из глобальных тестовых сетей. Локальную сеть поднять с помощью инструмента Ganache и инициировать транзакции от разных пользователей, предполагается, что в системе есть 10 аккаунтов с балансом в 100 ETH каждый. Также необходимо измерить трафик сети и ее пропускную способность.

В качестве тестовой глобальной сети были выбраны сети Rinkeby, Kovan и Ropsten с использованием инструмента Infura, который позволяет получить подробную статистику по децентрализованному приложению. После чего можно осуществить взаимодействие с системой используя реальные аккаунты Ethereum. Получить ETH на баланс аккаунта в сети для оплаты транзакций можно с помощью faucet-сервиса [21]. А убедиться в совершенных транзакциях можно с помощью сервиса проверки транзакций в сети [22].

В контексте эксперимента можно убедиться в достоверности преимуществ платформы Ethereum и функционала, на основе которого был осуществлен выбор платформы распределенного реестра. Проверить корректность взаимодействия работы фреймворков и технологий, используемых в реализации системы.

3.2. Локальная сеть

В общей сложности локальная сеть была развернута с помощью двух различных инструментов.

Первая - с помощью инструмента Ganache на одном компьютере, для приближения локальной сети к реальным условиям можно задать время генерации блоков, лимит газа и другие параметры. Инструмент Ganache предоставляет 10 аккаунтов с балансом в 100 ETH каждый. В локальной сети были совершены транзакции от каждого предоставляемого аккаунта, при этом ошибок не наблюдалось. В качестве параметра времени генерации блока, измеряемого в секундах, были подставлены значения из интервала от 0 до 60 с шагом в 5. При совершении транзакций в 13 запусках сети время генерации блока соблюдалось точно, а ошибки отсутствовали.

Были реализованы тесты и найдено критическое значение пользователей, совершающих транзакции для заключения пари (первая для того, чтобы посмотреть текущее значение средств, поставленных на определенный исход, вторая — для совершения ставки, третья — для того, чтобы убедиться, что ставка была совершена). В результате был построен график зависимости параметра tps от количества пользователей, представленный на Рисунке 15. Подсчет времени выполнения транзакций для нахождения параметра tps был выполнен на машине с ОС Ubuntu, процессором Intel Core i5-8250U CPU 1.60GHz × 8 и 8 Гб оперативной памяти.

Вторая – с помощью, предоставляемой Ethereum, утилиты geth. Несмотря на то, что утилита приближает локальную сеть более точно к реальной глобальной сети, настройка и конфигурация сети осуществляется сложнее и медленнее.

*Рисунок 15: Зависимость параметра tps от количества пользователей*

3.3. Глобальная сеть

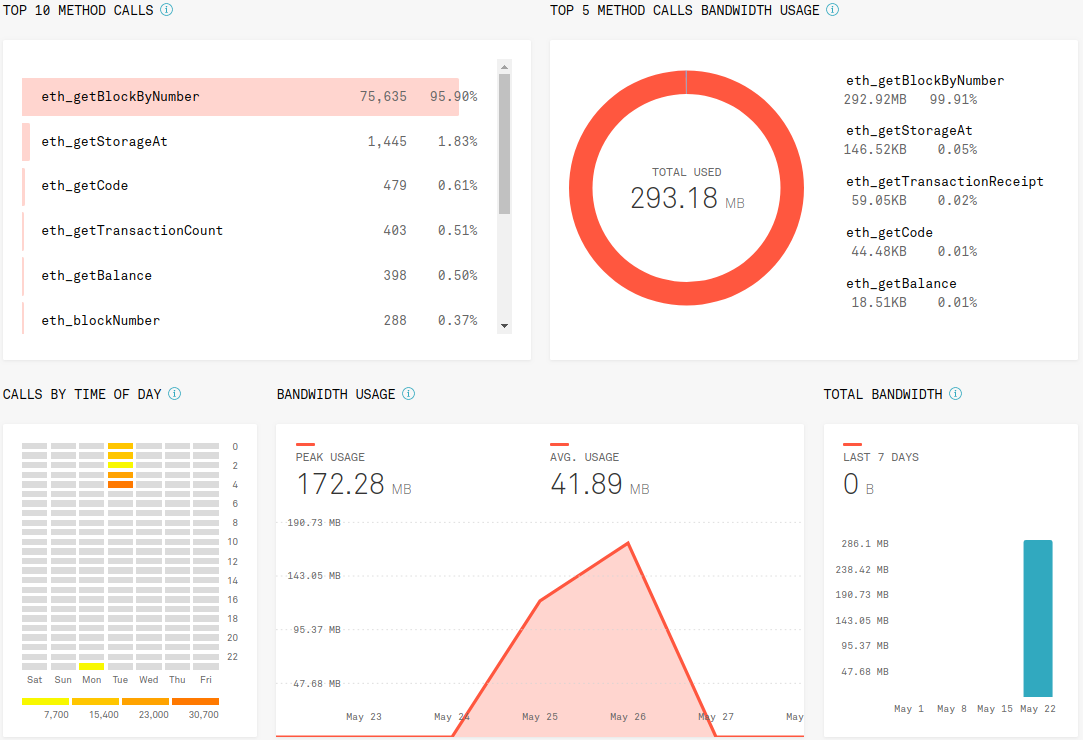
Децентрализованное приложение было развернуто в трех тестовых сетях Ethereum: Ropsten, Kovan и Rinkeby. В результате поведение системы не изменилось от используемой сети, что дает основания полагать, что и в глобальной сети система будет вести себя стабильно. Развертывание приложения в глобальной тестовой сети произведено не было, в связи с необходимостью платы реальной криптовалюты, но оно мало чем отличается от развертывания в главной глобальной. Отличие заключается в конфигурационном файле фреймворка Truffle. Требуется указать идентификатор сети и передать адрес провайдеру для связи с приложением. В качестве провайдера был использован сервис Infura, который обеспечивает доступность адреса приложения, а также предоставляет статистику по обращениям к приложению и трафику сети.

Более подробно была рассмотрена сеть Rinkeby. На момент написания работы глобальная тестовая сеть Rinkeby содержит 42 активных полновесных узла, среднее время записи блока - 15 секунд, а цепочка состоит из 6 569 502 блоков. Присоединиться в качестве полновесного узла может любой желающий, что обеспечивает очень хорошую масштабируемость сети. По факту отличия других тестовых сетей заключаются только в использовании прочих алгоритмов консенсуса и во времени генерации блока.

После развертывания приложения были совершены транзакции от 5 разных реальных аккаунтов Ethereum. Все транзакции были выполнены и проверены с помощью сервиса, предоставляющего список всех транзакций сети.

Результаты

В процессе разработки системы были использованы следующие фреймворки и технологии. Локальная сеть была поднята с помощью инструмента Ganache и с помощью утилиты geth, предоставляемой самой платформой, для развертывания локальной сети с нужными параметрами. В результате эксперимента все транзакции были выполнены без ошибок. С помощью инструмента Infura после развертывания приложения в сети Rinkeby были полученные графики, представленные на Рисунке 16, которые показывают частоту вызовов методов приложения и объем трафика в сети, порожденный этими вызовами.

  
*Рисунок 16: Статистика провайдера Infura по децентрализованному приложению*

Из графиков следует, что чаще всего был вызван метод получения номера блока, что обусловлено обращениями майнеров для валидации транзакций и генерации последующих блоков. При этом, чем большее количество полновесных узлов присутствует в сети, тем выше вероятность того, что приложение и сеть справятся с нагрузками.

Таким образом, можно сделать вывод, что приложение, развернутое в глобальной сети, получает все преимущества готовой инфраструктуры сети и довольно хорошо масштабируется, что подкреплено работоспособностью сети Ethereum. Использование библиотеки web3js и инструмента Metamask обеспечивает взаимодействие с реестром из браузера и мобильных устройств.

Заключение

Системы заключения пари на спортивные события с использованием технологии распределенных реестров обладают существенными недостатками. Разработанная система была реализована на основе модификации протокола Augur, заключающейся в адаптации к кейсу спортивных событий. Система удовлетворяет почти всем выдвинутым требованиям. Удобство использования, прозрачность и кроссплатформенность достигается средствами инструмента Metamask. Скорость и масштабируемость обеспечивается за счет глобальной сети Ethereum. Анонимность - за счет использования аккаунта Ethereum и реализацией собственных токенов системе на основе стандарта ERC20. К минусам разработанной системы можно отнести отсутствие поддержки криптовалют, которые базируются не на основе Ethereum.  
 Было установлено, что использование платформы Ethereum оправдано в контексте данной работы, а используемый стек технологий обеспечивает удобство разработки и обладает только одним существенным недостатком - количество инструментов.

Использование различных способов расчета коэффициентов исходов позволяет закрыть потребности любого пользователя системы, а относительно короткая фаза подтверждения исходов события экономит время от момента заключения пари до итоговой выплаты вознаграждения.

В качестве направления для дальнейшего улучшения системы можно предложить разработку собственного механизма расчета коэффициентов на основе информации из открытых источников. В перспективе систему можно расширить на использование не только для спортивных событий, но и для любых событий, происходящих в реальной жизни. Однако, следует учитывать проблемы, которые возникли в системе Augur, а также предстоит найти способ их решения.

Список литературы

1. Augur whitepaper  [Электронный ресурс]: URL: <https://github.com/AugurProject/whitepaper/blob/master/v1/russian/whitepaper_ru.pdf> (дата обращения: 15.03.2020)
2. OneShash [Электронный ресурс]: URL: h[ttps://bukmeker-expert.com/bukmakers/onehash/](https://bukmeker-expert.com/bukmakers/onehash/) (дата обращения 15.03.2020)
3. BetMatch [Электронный ресурс]: URL: <https://betmatch.io/> (дата обращения 15.03.2020)
4. 1xBit [Электронный ресурс]: URL: <https://1xbit1.com/ru/> (дата обращения 15.03.2020)
5. Saurabh Dhumwad, Mandar Sukhadeve, Chetan Naik, Manjunath K.N., Srikanth Prabhu  A Peer to Peer Money Transfer Using SHA256 and Merkle Tree//[2017 23RD Annual International Conference in Advanced Computing and Communications (ADCOM)](https://proxy.library.spbu.ru:2281/xpl/conhome/8682006/proceeding) 18.04.2019
6. Сертификаты X509 [Электронный ресурс]: URL: <https://habr.com/ru/post/194664/> (дата обращения: 30.05.2020)
7. Java Virtual Machine [Электронный ресурс]: URL: h[ttps://docs.oracle.com/javase/specs/jvms/se8/html/jvms-1.html#jvms-1.2](https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-1.html#jvms-1.2) (дата обращения: 30.05.2020)
8. Practical Byzantine Fault Tolerance [Электронный ресурс]: URL: <http://pmg.csail.mit.edu/papers/osdi99.pdf> (дата обращения: 30.05.2020)
9. EOS is not blockchain [Электронный ресурс]: URL:  <https://thenextweb.com/hardfork/2018/11/01/eos-blockchain-benchmark/> (дата обращения: 15.03.2020)
10. Тестовые сети Ethereum [Электронный ресурс]: URL: <https://ethereum.stackexchange.com/questions/27048/comparison-of-the-different-testnets> (дата обращения 15.03.2020)
11. Metamask [Электронный ресурс]: URL: <https://docs.metamask.io/> (дата обращения 15.03.2020)
12. Web3js  [Электронный ресурс]: URL: <https://web3js.readthedocs.io/en/v1.2.0/web3-eth.html> (дата обращения: 15.03.2020)
13. ERC-20 [Электронный ресурс]: URL: <https://eips.ethereum.org/EIPS/eip-20> (дата обращения: 15.03.2020)
14. ERC-271 [Электронный ресурс]: URL: http://erc721.org/ (дата обращения 15.03.2020)
15. Mocha [Электронный ресурс]: URL: <https://www.npmjs.com/package/mocha> (дата обращения: 15.03.2020)
16. Chai [Электронный ресурс]: URL: <https://www.chaijs.com/> (дата обращения 15.03.2020)
17. Truffle [Электронный ресурс]: URL: <https://www.trufflesuite.com/docs/truffle/getting-started/compiling-contracts> (дата обращения 15.03.2020)
18. Ganache [Электронный ресурс]: URL:  <https://github.com/trufflesuite/ganache-cli> (дата обращения: 15.03.2020)
19. Атака 51% процента [Электронный ресурс]: URL: h[ttps://en.bitcoinwiki.org/wiki/51%25\_attack](https://en.bitcoinwiki.org/wiki/51%25_attack) (дата обращения: 23.05.2020)
20. Infura provider [Электронный ресурс]: URL: [https://infura.io](https://infura.io/dashboard/ethereum/15cbe4df6aab4504b0b4cccf0493bea3/settings) (дата обращения: 15.03.2020)
21. Rinkeby faucet-сервис [Электронный ресурс]: URL: <https://faucet.rinkeby.io/> (дата обращения: 15.03.2020)
22. Etherscan [Электронный ресурс]: URL: <https://rinkeby.etherscan.io/> (дата обращения: 15.03.2020)
23. Loom test network  [Электронный ресурс]: URL: <https://loomx.io/developers/en/deploy-loom-testnet.html#run-against-loom-testnet> (дата обращения 15.03.2020)
24. Metamask tutorial [Электронный ресурс]: URL: [https://www.bitdegree.org/tutorials/metamask-wallet-review](https://www.bitdegree.org/tutorials/metamask-wallet-review/#Cryptocurrency_Wallet_Bottom_Line)/ (дата обращения: 15.03.2020)
25. Truffle samples [Электронный ресурс]: URL: h[ttps://www.trufflesuite.com/boxes](https://www.trufflesuite.com/boxes) (дата обращения: 15.03.2020)
26. Ganache [Электронный ресурс]: URL: <https://github.com/trufflesuite/ganache-cli> (дата обращения: 15.03.2020)
27. EOS whitepaper [Электронный ресурс]: URL: [https://github.com/EOSIO/Documentation/blob/](https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md) (дата обращения: 15.03.2020)
28. NEO blockchain [Электронный ресурс]: URL: <https://docs.neo.org/docs/en-us/basic/whitepaper.html> (дата обращения: 15.03.2020)
29. Детальный разбор Ethereum  [Электронный ресурс]: URL: <https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369> (дата обращения: 15.03.2020)