

Основные принципы функционирования Ethereum

Основы разработки
на Solidity



Оглавление

Введение	3
Словарь терминов	3
Транзакции, блоки, блокчейн	3
Узлы, сети, форки	7
Монеты, аккаунты, адреса	8
Кошельки	9
EVM, состояния, смарт-контракты	10
Газ, комиссии, оплата транзакций	12
Измерение вычислительной сложности кода	12
Вычисление размера комиссий	13
Основы синтаксиса solidity	15
Размещение (deploy) контракта с помощью IDE Remix	17
Вопросы и ответы	17
Вопросы	17
Ответы	17
Дополнительные материалы	18
Использованная литература	18

Введение

Добрый день, Друзья!

В процессе изучения курса вы познакомитесь с базовыми сведениями о блокчейне Ethereum, стандартами ERC, языком Solidity и современными средствами разработки смарт-контрактов.

Задача первой лекции - изучить основные понятия, связанные с блокчейном, и освоить элементарные пользовательские операции. Весь наш курс будет опираться на Ethereum – второй (после bitcoin) по значимости блокчейн. Большинство объяснений про блокчейн будут подразумевать именно Ethereum, хотя сейчас существует порядка тридцати [протоколов](#), которые существенно отличаются деталями и принципами реализации.

Темы лекции:

- Как устроен блокчейн Ethereum, транзакции и блоки (на примере Etherscan).
- Что такое native coin, аккаунт, адрес и крипто-кошелёк.
- Как работает EVM, что такое state, что такое смарт-контракты.
- Какую роль играет газ и как происходит оплата транзакций.

Словарь терминов

Solidity – высокоуровневый язык программирования, используемый для создания умных контрактов на платформах, основанных на блокчейне, включая Ethereum. Язык напоминает JavaScript и предназначен для разработки приложений, работающих в Ethereum Virtual Machine (EVM).

Умный контракт (Smart Contract) – программа, которая автоматически выполняет, контролирует или документирует юридически значимые события и действия согласно условиям контракта. Умные контракты работают на блокчейне и могут быть использованы для автоматизации выполнения контракта, делая его более безопасным и менее зависимым от посредников.

Ethereum Virtual Machine (EVM) – виртуальная машина в сети Ethereum, которая обеспечивает выполнение умных контрактов. EVM изолирует умные

контракты от остальной части сети, предоставляя им безопасную среду для выполнения.

Блокчейн (Blockchain) – распределенная база данных, состоящая из цепочки блоков, содержащих информацию. Каждый новый блок в цепочке содержит хеш предыдущего блока, создавая тем самым неизменяемую и целостную структуру данных.

Хеш (Hash) – функция, преобразующая входные данные любого размера в строку фиксированной длины, используемая для обеспечения целостности данных в блокчейне. Хеш блока включает в себя уникальный отпечаток всех транзакций в блоке, а также хеш предыдущего блока.

Транзакция (Transaction) – запись, отправленная в сеть блокчейна, которая переводит криптовалюту, выполняет умный контракт или изменяет данные в блокчейне.

Генезис-блок – первый блок в цепочке блокчейна. В Ethereum генезис-блок устанавливается с хешлинком, равным нулю, и является отправной точкой для всей последующей цепочки блоков.

Софт-форк (Soft fork) – изменение в протоколе блокчейна, которое совместимо с предыдущими версиями. Примером софт-форка является создание тестовых сетей с отличающимся генезис-блоком от основной сети, например, сети Sepolia или Polygon с тестовой сетью Mumbai. Также софт-форки могут вносить изменения в сам протокол, как, например, в случае с Tron.

Транзакции, блоки, блокчейн

В рамках этого курса блокчейн проще всего будет воспринимать как организованную особым образом базу данных. Эта база содержит [транзакции](#) (transaction) – записи о фактах перемещения средств между пользователями. Например, закодированное соответствующим образом сообщение "Алиса отправила Бобу 5 монет" и будет простейшей транзакцией.

Транзакции в блокчейне хранятся в составе [блоков](#), блоком называется массив транзакций. Блоки, в свою очередь, объединяются в односвязный список, т.е.

собственно blockchain – цепочку блоков. Ключевой особенностью является способ, которым блоки ссылаются друг на друга - *хешлинки* (hashlink). Каждый блок содержит значение *хеш-функции* от предыдущего блока (см. рис. 1). Первый блок в цепочке называется *генезисом* (genesis) и его хешлинк обычно устанавливается равным нулю.

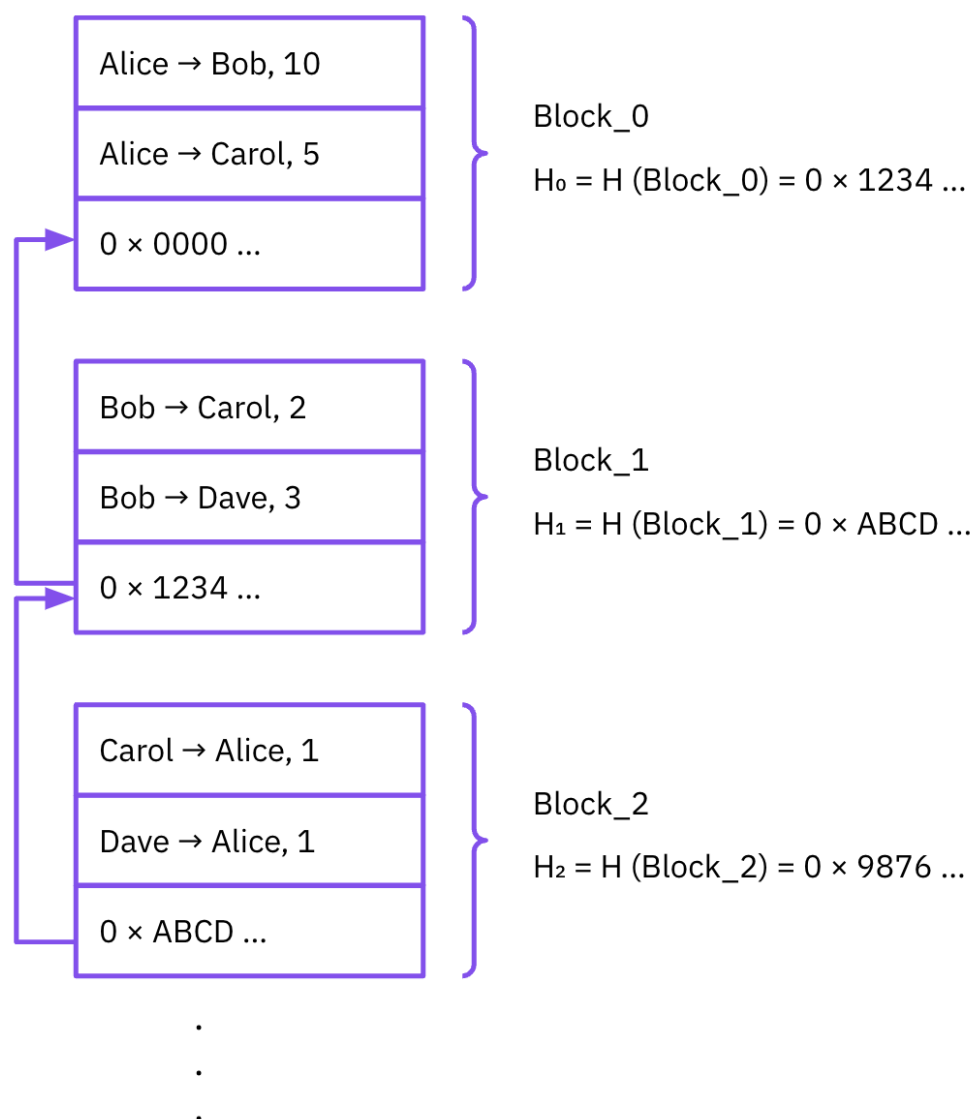


Рис 1. Упрощённая структура блокчейна

Это крайне упрощенное представление о содержимом блокчейна, тем не менее на него можно опираться при решении практических задач. На рис. 2-4 приведены фрагменты страниц сайта etherscan.io, на котором можно просмотреть содержимое блокчейна Ethereum. На заглавной странице отображается список последних выпущенных блоков. При нажатии на

соответствующую ссылку, можно увидеть содержимое блока, список транзакций в блоке и содержимое каждой транзакции.

Block Height:

17908697

<>

Status:

Finalized

Timestamp:

1 day 9 hrs ago (Aug-13-2023 09:42:59 PM +UTC)

Proposed On:

Block proposed on slot 7094913, epoch 221716

Transactions:

123 transactions and 61 contract internal transactions in this block

Withdrawals:

16 withdrawals in this block

Fee Recipient:

builder0x69 in 12 secs

Block Reward:

0.024740592574331574 ETH (0 + 0.244880946391170376 - 0.220140353816838802)

Total Difficulty:

58,750,003,716,598,352,816,469

Size:

127,150 bytes

Gas Used:

16,277,738(54.31%)

+9% Gas Target

Gas Limit:

29,970,705

Base Fee Per Gas:

0.000000013524013829 ETH (13.524013829 Gwei)

Burnt Fees:

0.220140353816838802 ETH

Extra Data:

by builder0x69 (Hex:0x6279206275696c64657230783639)

Ether Price:

\$1,839.10 / ETH

Hash:

0x31d5740d2c356d85097e1693d549f4f91b1321a28b9dcbe9c338b06ff8659f86

Parent Hash:

0xe7c0d78792d4548e26225c1a4baf70d7ccee8a2b3c740238cdc44ac13bea4077

StateRoot:

0xac1a0d2a7b89fc592572a21f6cd63bd130199ebc581b192c77bb026c61f41c8b

WithdrawalsRoot:

0xc815fd7cea21e616e0130ae56d12545a1862f15db0877edbb4dfc32d92de1510

Nonce:

0x0000000000000000

Рис. 2 Содержимое блока

A total of 123 transactions found

First<Page 1 of 3>Last

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x579665831f1da805a...	Transfer	17908697	1 day 9 hrs ago	builder0x69.eth	Lido: Execution Layer R...	0.024060599 ETH	0.00029902
0xe3f57c0648c84efdf...	0x66b210ac	17908697	1 day 9 hrs ago	0xF9c2F2...582eaeEE	0xB51785...fAd510c4	0.01 ETH	0.00209537
0xc0e29db5d2743876...	Commit	17908697	1 day 9 hrs ago	sermoon.eth	ENS: ETH Registrar Co...	0 ETH	0.00059868
0xf4df9a881215d9d0f...	Execute	17908697	1 day 9 hrs ago	0x882EEd...05F6e040	Uniswap: Universal Ro...	0 ETH	0.00253861
0x56b397ad0b555d54...	Transfer*	17908697	1 day 9 hrs ago	0x0F5070...E7ab014b	0x000000...F6573532	0 ETH	0.00029718
0x2df5695d7f8e471d2...	Transfer	17908697	1 day 9 hrs ago	0x26ec47...06731Ccd	0x5f91b1...E790B3fb	0.11 ETH	0.0002844
0x6701bdde4cef15c3f...	Batch Edit Qu...	17908697	1 day 9 hrs ago	gutterskelly.eth	The Captainz: Captainz...	0 ETH	0.00390556
0xa6bedecdbc49c314...	Transfer	17908697	1 day 9 hrs ago	0xDe1A0...d842066a	0x8Aa2b5...6B150976	0.008806314 ETH	0.00028449

Рис. 3 Фрагмент списка транзакций

Transaction Hash:	0x2df5695d7f8e471d27271c8643577f3fe0dbf8fd0f9b44cad6f3fe6b8636d3ce
Status:	Success
Block:	17908697 10051 Block Confirmations
Timestamp:	1 day 9 hrs ago (Aug-13-2023 09:42:59 PM +UTC) Confirmed within 10 secs
Transaction Action:	Transfer 0.11 Ether To 0x5f91b1...E790B3fB
Sponsored:	
From:	0x26ec47Fc631963329934EF660AE127E706731Ccd
To:	0x5f91b125EbdEd24FFa8a1D0abEc6A5B5E790B3fB
Value:	0.11 ETH \$202.28
Transaction Fee:	0.000284403885759 ETH \$0.52
Gas Price:	13.543042179 Gwei (0.000000013543042179 ETH)
Ether Price:	\$1,839.10 / ETH
Gas Limit & Usage by Txn:	21,000 21,000 (100%)
Gas Fees:	Base: 13.524013829 Gwei Max: 16.608086761 Gwei Max Priority: 0.01902835 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.000284004290409 ETH (\$0.52) Txn Savings: 0.000064365936222 ETH (\$0.12)
Other Attributes:	Txn Type: 2 (EIP-1559) Nonce: 3 Position In Block: 117
Input Data:	0x

Рис. 4 Содержимое транзакции

Так, можно увидеть, что блок **17908697** выпущен **13 августа 2023 года 9:42:59 UTC**, содержит **123** транзакции и имеет хеш **0x31d5...9f86**. В числе прочих этот блок содержит транзакцию с хешем **0x2df5...d3ce**, в которой перечисляются **0.11 ETH** от пользователя с адресом **0x26ec...1Ccd** пользователю **0x5f91..B3fB**.

💡 Такой формат транзакций называется *account based* и применяется в большинстве блокчейнов, но не является единственно возможным. Например, в bitcoin используется формат *UTXO* - unspent transaction output.

Узлы, сети, форки

Копии блокчейна хранятся на компьютерах, называемых *узлами* (nodes). По специальному протоколу отдельные узлы синхронизируют между собой состояние блокчейна, образуя *сеть*. Консистентность сети обеспечивается тем, что на всех узлах запущены совместимые экземпляры программного обеспечения и используется одинаковый генезис-блок.

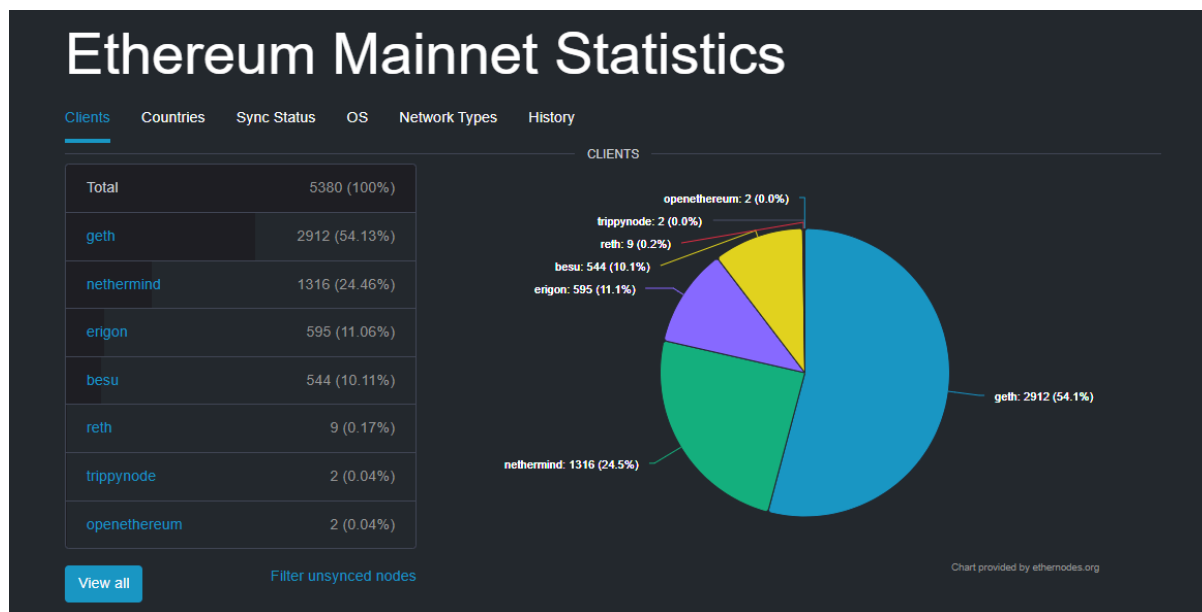


Рис 5. Состояние основной сети Ethereum (<https://www.ethernodes.org>)

Например по данным сайта <https://www.ethernodes.org>, в основной сети Ethereum запущено 5280 узлов, и основная их часть (2912) использует клиент geth.

Также широко распространена практика запуска стандартного клиента Ethereum, но использующего другой генезис-блок. В этом случае мы говорим, что был запущен софт-форк (soft fork). Софтфорками могут быть тестовые сети как, например, [Sepolia](#), так и отдельные проекты, например, [Polygon](#), со своей собственной тестовой сетью [Mumbai](#).

Кроме того, софт-форки могут реализовывать некоторые изменения в протоколе, как, например, [Tron](#).



Использование тестовых сетей для отладки приложений или обучения – это хорошая практика, позволяющая избежать трат на комиссии.

Монеты, аккаунты, адреса

Итак, мы выяснили, что блокчейн представляет собой упорядоченный список транзакций. В простейшем случае транзакции содержат данные о пересылках нативной монеты (native coin). В отличие от других активов, нативная монета обладает особыми свойствами, например, именно её списывают в качестве комиссии за транзакции, ею награждают майнеров PoW или её стейкают майнеры PoS. В основной сети Ethereum основной монетой является Ether, или, сокращённо ETH.



Предыдущий абзац справедлив для Ethereum, в других сетях функции нативной монеты могут отличаться. В сети Tomochain, например, возможно выплачивать комиссию в иных активах. А, например, для отправки токенов в сети Tron помимо основной монеты понадобится "энергия".

Активы пересылаются между аккаунтами. *Аккаунтом* в Ethereum называется сущность, которая имеет баланс ETH и может отправлять транзакции. В Ethereum есть два типа аккаунтов - внешние и контракты. Во всех примерах выше мы говорили о внешних аккаунтах, которые генерируются на основе пары криптографических ключей. Транзакции подписываются секретным ключом аккаунта, что позволяет подтвердить их подлинность. Открытый ключ используется для получения адреса аккаунта.

Адресом в Ethereum называется 42 символьная шестнадцатеричная строка, полученная из 20 байтового значения с префиксом 0x, например, 0x0000000073c77bE04773D65aa3c5459019613052. Адрес внешнего аккаунта вычисляется на основе первых 20 байт значения хеш-функции от открытого ключа аккаунта.

На сайте etherscan.io можно посмотреть состояние любого аккаунта

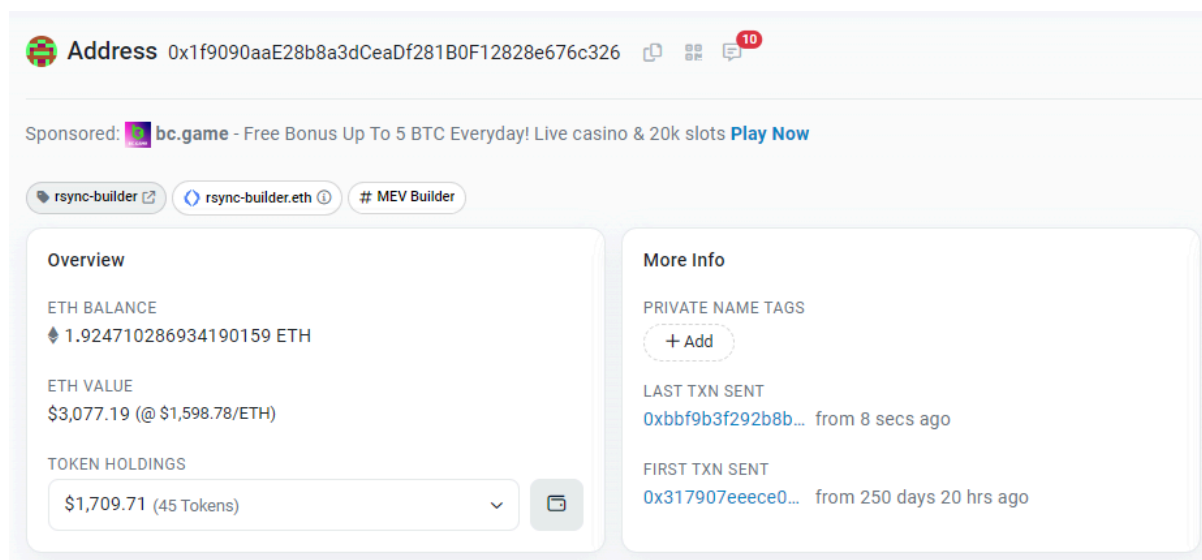


Рис. 5 Фрагмент страницы аккаунта в etherscan

💡 Для удобства отображения владелец аккаунта может добавить на etherscan своё отображаемое имя, например, rsync-builder.eth как на примере выше. Это имя существует только на сайте etherscan, влияет только на внешний вид страниц, служит только для удобства пользователей и не хранится в блокчейне.

Кошельки

Кошельком называется приложение, которое позволяет управлять аккаунтом - проверять баланс, отправлять транзакции. Для начала работы с кошельком необходимо скачать и установить приложение, причём способ установки может отличаться в зависимости от платформы. Например, популярный кошелек metamask является расширением для браузера, а trust wallet - поставляется и в виде браузерного расширения и в виде мобильного приложения.

Помимо выполнения основных функций – отправки транзакций, кошельки также часто имеют дополнительную функциональность, например, возможность покупки криптовалюты у доверенных продавцов, продвинутое управление активами с помощью финансовых инструментов (стейкинг, фарминг), аутентификация на сторонних сайтах, возможность подключения к нескольким сетям одновременно.

Как правило, кошельки генерируют аккаунты (секретные ключи) из так называемой сид-фразы (*seed phrase*). Сид-фраза – это способ представления случайной числовой последовательности с помощью заранее заданного словаря. Например, шестнадцатеричное значение

**3bd0bda567d4ea90f01e92d1921aacc5046128fd0e9bee96d070e1d606cb7922
5ee3e488bf6c898a857b5f980070d4d4ce9adf07d73458a271846ef3a8415320**


после кодирования по стандарту [BIP 39](#) соответствует фразе

indoor dish desk flag debris potato excuse depart ticket judge file exit

Фраза удобнее числовой последовательности тем, что её легче переносить вручную между разными устройствами или записывать на нецифровом носителе (бумаге).

По стандарту [BIP 44](#) с помощью алгоритма [PBKDF2](#) (Password-Based Key Derivation Function 2 - функция генерации ключа на основе пароля) из одной сид-фразы можно сгенерировать множество секретных ключей. Важной особенностью алгоритма является невозможность установить сгенерированы ли два ключа из одной сид-фразы.

При первом запуске кошелька пользователю предлагается либо сгенерировать новую сид-фразу либо экспортировать уже существующую.

 Важно сохранить сид-фразу в надёжном месте, желательно не на цифровом носителе (записать на бумаге, выгравировать на жетоне и т.д.). Компрометация сид-фразы приведёт к компрометации **ВСЕХ** сгенерированных из неё кошельков!

EVM, состояния, смарт-контракты

Часто можно увидеть описание блокчейна как структуры, с помощью которой вычисляется леджер (ledger или бухгалтерская книга, гроссбух) - перечень всех аккаунтов с их балансами. На самом деле, поскольку блокчейн содержит упорядоченную последовательность транзакций, то каждая транзакция изменяет балансы каких-то аккаунтов, обновляя гроссбух. По сути, речь идёт о

некой виртуальной машине, которая изменяет список аккаунтов в соответствии с записанными в блокчейне инструкциями.

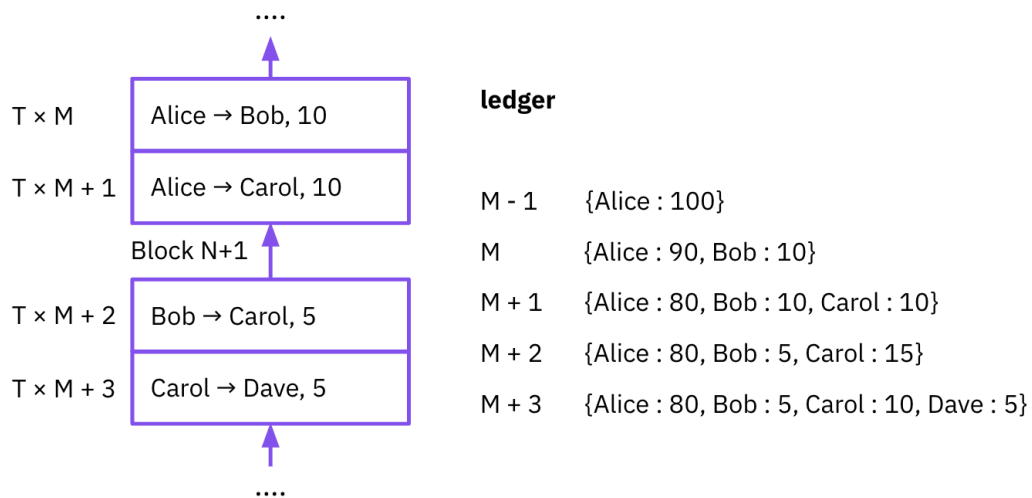


Рис. 6 Пример вычисления леджера

Например, на рисунке 6 видно, что после вычисления транзакции с номером $M+2$ состояние леджера меняется с {Alice:80, Bob: 10, Carol: 10} на {Alice:80, Bob: 5, Carol: 15}

Однако такое упрощение справедливо для протоколов, допускающих только переводы средств, в то время как Ethereum позволяет хранить на аккаунтах данные. Совокупность всех данных аккаунтов называется *состоянием* (state).

💡 Изучение формата хранения состояния выходит за рамки этого курса. В этом курсе можно ограничиться краткими сведениями. Состояние хранится в структуре merkle patricia tree. Текущий размер состояния сети Ethereum составляет около 130Гб.

EVM (Ethereum Virtual Machine, виртуальная машина Ethereum) это сущность, вычисляющая новое состояние из старого и транзакции. Математически её можно описать как некую функцию $Y(S, T) = S'$, где S - предыдущее состояние, T - транзакция, а S' - новое состояние.

Изменение данных аккаунта выполняется с помощью программ, которые называются *смарт-контрактами*. Выполнение этих программ возможно благодаря тому, что в сети Ethereum транзакции могут содержать не только информацию о переводе нативной монеты, но и машинный код для EVM.

Разработчики [Ethereum](#) предпочитают утверждать, что весь протокол Ethereum и вся сеть служит только для того, чтобы поддерживать консенсус относительно содержимого цепочки блоков, которая в свою очередь, нужна для вычисления актуального состояния виртуальной машиной Ethereum.

Газ, комиссии, оплата транзакций

Измерение вычислительной сложности кода

Чтобы поддерживать работу сети, с отправителей транзакций взимаются комиссии за каждую отправленную транзакцию. Комиссия взимается в нативной монете, ETH.

Поскольку каждая операция, будь то перевод нативной монеты или выполнение сложного смарт-контракта вызывает изменение состояния, то разумно взимать с отправителя комиссию, пропорциональную сложности транзакции. Размер комиссии вычисляется специальным образом через промежуточную сущность, называемую *газом* (gas).

Каждая операция EVM имеет [стоимость](#), выраженную в газе. Таким образом, можно вычислить количество газа, требуемое для выполнения транзакции. Важно понимать, что, как [доказал](#) Алан Тьюринг, не существует способа определить количество шагов, которое потребуется для выполнения программы по её программному коду. Или, применительно к Ethereum, количество газа, которое нужно для выполнения кода смарт-контракта, можно оценить только по ходу его выполнения. Например, цикл, вычисляющий N-е число Фибоначчи, скомпилируется в небольшой фрагмент машинного кода, но количество газа, которое он потребует для выполнения будет прямо пропорционально величине N, и узнать это количество можно будет только в ходе выполнения кода.

```
1 for (i = 0; i < N; i++){
2     t = p + q;
3     p = q;
4     q = t;
5 }
6
```

Размер одного блока сети Ethereum в настоящее время ограничен 30 000 000 единиц газа. При отправке транзакции пользователь может установить

ограничение по газу (по умолчанию оно равно 21 000 единиц).

Вычисление размера комиссий

Пользователь оплачивает требуемый для её выполнения газ. Для этого в транзакции содержится два значения – gas price и gas limit (цена газа и ограничение по газу). Gas limit это максимальное количество газа, которое пользователь готов оплатить, а gas price - цена за одну единицу газа. Рассмотрим подробнее **рисунок 4**, содержащий данные с сайта etherscan.

Мы видим, что пользователь установил gas limit 21000 и выполнение транзакции израсходовало именно столько газа. В совпадении нет ничего удивительного - 21000 это значение по умолчанию для gas limit, и это количество газа, требуемое для простого перевода нативной монеты ETH. Цену газа пользователь установил в 13.543042179 Gwei (или 0.000000013543042179 ETH) за единицу. Таким образом, комиссия за транзакцию составила $0.000000013543042179 \text{ ETH/unit} * 21000 \text{ units} = 0.000284403885759 \text{ ETH}$.

Если пользователь установит слишком низкий gas limit, то в процессе обработки транзакции может выясниться, что на её выполнение не хватает газа. Тогда транзакция будет считаться отклонённой, ей будет присвоен статус Failed, а с пользователя взимается комиссия в полном объёме.

В чём мотивация пользователя устанавливать высокий gas price? Как уже было сказано, размер блока ограничен (в настоящее время значением 30 000 000 газа), поэтому между транзакциями происходит конкуренция за попадание в блок - майнеры выбирают транзакции с наибольшим gas price, чтобы получить большие комиссии. Поэтому пользователи, которые хотят увеличить вероятность попадания транзакции в блокчейн, выставляют высокую цену газа.

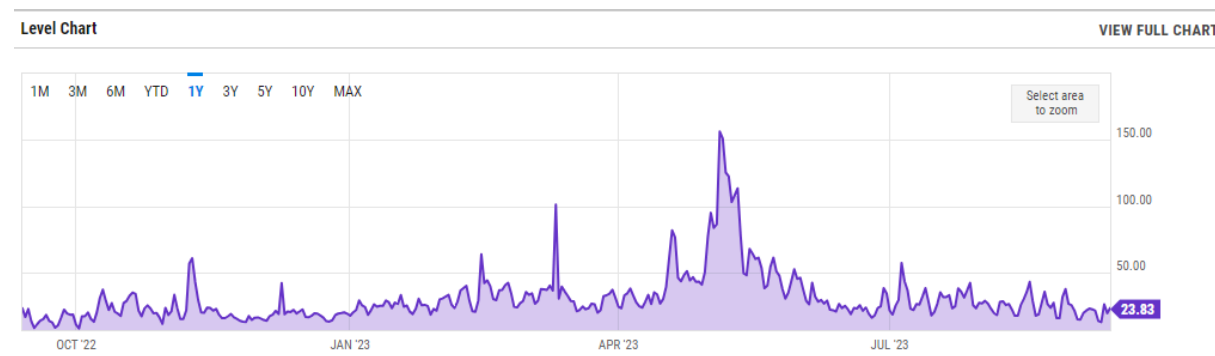


Рис. 7 Колебания цены газа за год
(https://ycharts.com/indicators/ethereum_average_gas_price)

Как видно из рисунка 7, средняя цена газа подвержена значительным колебаниям, и в периоды пиковой нагрузки она может увеличиваться в 5-6 раз. Поэтому распространённой практикой является ожидание удачного момента для отправки транзакции, если она не является срочной.

Для получения актуальных цен на газ можно воспользоваться страницей <https://etherscan.io/gastracker>

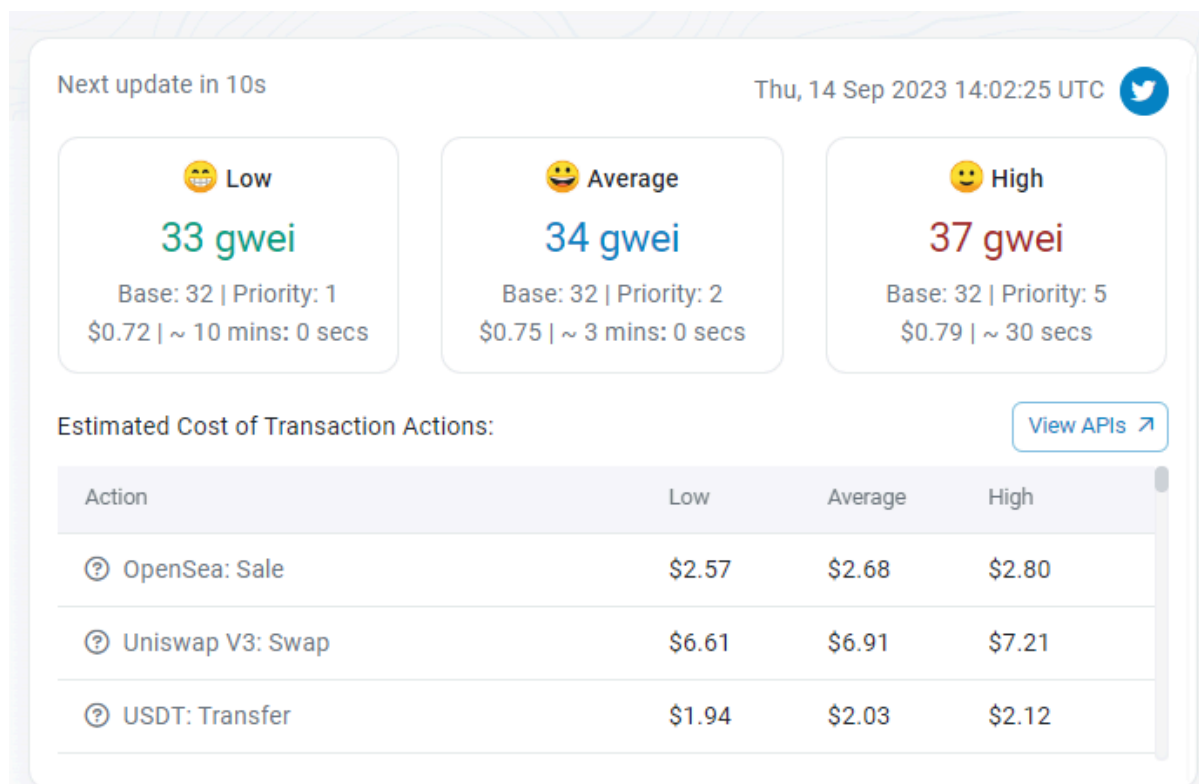


Рис 8. Фрагмент содержимого страницы <https://etherscan.io/gastracker>

На примере мы видим три рекомендованные цены - 33 gwei, 34 gwei и 37 gwei. Они рекомендованы для отправки с низким, средним и высоким приоритетами соответственно. Если в текущий момент времени отправить транзакцию с gas price меньшим 33 gwei велика вероятность, что она вообще не попадёт в блокчейн.

Основы синтаксиса solidity

В настоящий момент основным языком для разработки Ethereum смарт-контрактов является Solidity.

Solidity – это компилируемый объектно-ориентированный язык со статической типизацией. Solidity во многом основан на C++, и в меньшей степени на JavaScript и Python. От C++ были заимствованы регистрозависимость, синтаксис и механизмы ООП, от JS - синтаксис определения функций и экспорта. Программа на Solidity компилируется компилятором solc в байт-код, выполняемый на EVM.

Из языка C++ были заимствованы форматы идентификаторов, основные ключевые слова и синтаксис условного оператора *if*, операторов циклов *for*, *while* и *do..while*, определения переменных. Так, идентификаторами в solidity являются последовательности латинских букв, цифр и знака подчёркивания, причём идентификаторы не могут начинаться с цифр.

Рассмотрим код примера смарт-контракта, реализующего хранение и доступ к скалярной переменной (рис. 9).

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ≥0.8.2 <0.9.0;
4
5 /**
6  * @title Storage
7  * @dev Store & retrieve value in a variable
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract Storage {
11
12     uint256 number;
13
14     /**
15      * @dev Store value in variable
16      * @param num value to store
17      */
18     function store(uint256 num) public {
19         number = num;
20     }
21
22     /**
23      * @dev Return value
24      * @return value of 'number'
25      */
26     function retrieve() public view returns (uint256){
27         return number;
28     }
29 }
30
```

Рис. 9 Пример смарт-контракта на Solidity (источник [Remix IDE](#))

- На строке 1 стоит комментарий, указывающий тип используемой лицензии. Эти данные не являются обязательными, однако компилятор будет выдавать предупреждение об их отсутствии.
- На строке 3 стоит директива компилятора, задающая диапазон возможных версий компилятора. Язык solidity по большей части является обратно совместимым, тем не менее в контрактах со сложным кодом часто указывается наименьшая доступная версия компилятора.
- На строке 10 определяется контракт с именем *Storage*. В одном исходном файле, может находиться несколько контрактов, однако, согласно сложившимся практикам, рекомендуется каждый контракт хранить в отдельном файле, т.к. большинство систем (hardhat, remix) исходят из этого.
- На строке 12 определяется переменная-член *number* типа *uint256* (встроенные типы данных будут рассмотрены подробно в следующем уроке).
- На строках 18 и 26 определены функции *store* и *retrieve*. Функция определяется ключевым словом *function* с последующим заданием её имени, аргументов, типа возвращаемого значения и модификаторами. Так, функция *store* принимает один аргумент типа *uint256* и не возвращает значение, а функция *retrieve* не принимает аргументов и возвращает одно значение типа *uint256*. Функции в solidity могут возвращать множество значений, в этом случае их типы перечисляются в скобках после ключевого слова *returns*.
- В примере используется модификатор доступа *public*, означающий, что функция будет публичной, т.е. доступной для внешнего вызова. Ключевое слово *view* указывает, что функция не модифицирует состояние EVM - не изменяет переменные состояния, не создаёт событий, не создаёт и не уничтожает контракты и не пересылает ETH.

ABI

В контексте Ethereum, ABI относится к формату, который определяет, как взаимодействовать с контрактами на базе блокчейна Ethereum. ABI описывает структуру и типы данных контракта, а также список функций и событий, доступных для вызова или просмотра внешне.

ABI включает следующие элементы:

- Функции контракта: ABI определяет список функций, которые контракт может предоставлять. Каждая функция имеет свое имя, типы аргументов и тип возвращаемого значения.
- События контракта: ABI также описывает список событий, которые могут возникнуть в контракте. Каждое событие определяет свое имя и список параметров, которые будут передаваться при возникновении события.
- Типы данных: ABI определяет различные типы данных, которые могут использоваться в контракте, такие как целочисленные и строковые значения, адреса, массивы и т.д.

ABI используется для взаимодействия с контрактами Ethereum. Используя ABI, разработчики могут вызывать функции контрактов, передавать аргументы и получать результаты. Также ABI облегчает чтение и фильтрацию событий, которые контракты могут генерировать.

В целом, ABI является способом стандартизации и описания интерфейса контрактов Ethereum, что позволяет различным компонентам взаимодействовать и обмениваться данными на блокчейне.

Размещение (deploy) контракта с помощью IDE Remix

Remix – это онлайн IDE для разработки смарт-контрактов на solidity. Remix включает в себя редактор, компилятор solc и встроенную виртуальную машину Ethereum для запуска контрактов, а также обеспечивает интеграцию с github для управления версиями кода.

1. Откройте Remix IDE (интегрированная среда разработки) в веб-браузере, перейдя по ссылке <https://remix.ethereum.org/>.
2. Создайте новый файл контракта или скопируйте и вставьте существующий код контракта в текстовое поле "Solidity Compiler" в верхней панели.
3. Убедитесь, что выбрана правильная версия компилятора Solidity.
4. Нажмите на кнопку "Compile" для компиляции вашего контракта. Если компиляция прошла успешно, вы увидите информацию о контракте и его интерфейсе.

5. Перейдите на вкладку "Deploy & run transactions", расположенную в верхней панели над кодом контракта.
6. Выберите аккаунт, с которого вы хотите выполнить деплой контракта. Вы можете использовать локальный аккаунт, загрузить свой собственный аккаунт или подключиться к MetaMask для работы с аккаунтом из внешнего кошелька.
7. При необходимости установите значение для конструктора контракта во вкладке "Deploy & run transactions". Значение должно соответствовать типу переменной конструктора.
8. Нажмите на кнопку "Deploy" для выполнения деплоя контракта. Вы увидите информацию о транзакции деплоя и адресе только что созданного контракта.
9. После успешного деплоя контракта вы сможете вызвать его функции, отправлять транзакции и взаимодействовать с контрактом через интерфейс Remix.

Валидация контракта в Etherscan

[Etherscan](#) — обозреватель блоков и аналитическая платформа для блокчейна Ethereum. Проект запустила в 2015 году команда разработчиков из Малайзии. Это независимая организация, которая не финансируется Ethereum Foundation. Сервис предназначен для навигации по общедоступным данным сети: отслеживает транзакции и отображает результаты, как поисковая система. Платформа содержит детальные сведения об адресах, смарт-контрактах, блоках, состоянии сети и другие ончейн-показатели. Основные функции Etherscan — отслеживание состояния сети, транзакций и поиск информации об активах, хранящихся на общедоступных адресах Ethereum.

Валидация контрактов на Etherscan важна по нескольким причинам:

- Установление доверия: валидация контракта подтверждает, что код, опубликованный на Etherscan, соответствует фактическому контракту, запущенному на Ethereum блокчейне. Это помогает установить доверие

в проект и предотвратить возможные попытки мошенничества или подделки контракта.

- Проверка безопасности: валидация контракта может помочь в обнаружении потенциальных уязвимостей или ошибок в коде контракта. Это позволяет разработчикам получить результаты статического анализа контракта и предотвратить возможные проблемы безопасности, которые могут быть использованы злоумышленниками.
- Публичность: валидация контракта на Etherscan позволяет другим участникам сообщества Ethereum просматривать и изучать код контракта, а также анализировать его функциональность. Это способствует открытости и прозрачности в экосистеме Ethereum.

В целом, валидация контрактов на Etherscan является важным шагом для разработчиков Ethereum, который помогает поддерживать доверие, обеспечивать безопасность, способствовать прозрачности и облегчить анализ и интеграцию контрактов в экосистеме Ethereum.

Infura

Infura – сервис, предоставляющий инфраструктуру для доступа к блокчейну Ethereum через удаленные узлы. Она позволяет разработчикам легко взаимодействовать с блокчейном Ethereum, необходимым для выполнения транзакций, чтения данных и взаимодействия с контрактами.

Infura обеспечивает высокую доступность и масштабируемость, предоставляя удаленные, готовые к использованию узлы Ethereum. Это особенно полезно для разработчиков, которые не хотят разворачивать и поддерживать собственную инфраструктуру узлов Ethereum или желают избежать сложностей настройки сетевых параметров.

Чтобы использовать Infura, вы должны создать проект на их платформе и получить уникальный идентификатор проекта (Project ID), который будет использоваться для доступа к их узлам через их API. Infura предоставляет различные узлы для различных сетей Ethereum, таких как Mainnet, Ropsten, Rinkeby, Kovan и другие, чтобы вы могли выбрать нужную сеть для взаимодействия.

Разработчики могут использовать Infura API для отправки запросов к блокчейну Ethereum и получения информации о блоках, транзакциях, адресах

и выполнениях функций контрактов без необходимости развертывания и настройки собственной инфраструктуры узлов.

Infura является популярным и широко применяемым инструментом в разработке на Ethereum, облегчая разработчикам доступ к блокчейну и упрощая процесс разработки децентрализованных приложений (DApps) и интеграции сети Ethereum в различные проекты.

Использование hardhat

Hardhat – это среда для разработки контрактов, позволяющая легко и эффективно размещать и отлаживать код в тестовой среде и в рабочей сети.

Hardhat устанавливается в отдельную директорию с помощью менеджера пакетов npm.

```
npm install --save-dev hardhat
```

Главный компонент является hardhat runner, через который происходит управление процессами. Рассмотрим использование hardhat на примере.

Создадим проект с помощью команды

```
npx hadhat
```

Выберем "проект JavaScript" и ответим на все вопросы мастера вариантами по умолчанию. Мастер конфигурирования создаст структуру проекта, состоящую из каталогов contracts, scripts, tests и файлов Lock.sol, deploy.js и Lock.js, содержащих тестовый контракт, скрипт деплоя и скрипт теста.

Скомпилируем контракт командой **npx hardhat compile**, и запустим тесты командой **npx hardhat test**. Тесты по умолчанию будут выполняться на локальной копии EVM. Тестовый скрипт написан на фреймворке mocha и использует функции web3js для взаимодействия с тестовой средой.



Практическое задание: деплой контракта из hardhat.

1. В конфигурационном файле `hardhat.config.js` добавьте блок `networks`, чтобы настроить сеть Ethereum, на которую вы хотите развернуть контракт. Например, для разворачивания на сети Rinkeby, добавьте следующее:

```
networks: {  
  sepolia: {  
    url: "https://rinkeby.infura.io/v3/your-infura-project-id",  
    accounts: [your-private-key],  
  },  
},
```

2. В поле url впишите ваш endpoint для подключения к сети Ethereum. Вместо sepolia укажите сеть, которую хотите использовать. В поле accounts укажите секретный ключ аккаунта, с которого будет происходить размещение контракта.
3. В папке проекта создайте скрипт деплоя контракта (например, `deploy.js`). В этом скрипте используйте Hardhat API для деплоя контракта.

```
async function main() {  
  const [deployer] = await ethers.getSigners();  
  
  console.log("Deploying contracts with the account:",  
    deployer.address);  
  
  const Contract = await ethers.getContractFactory("MyContract");  
  const contract = await Contract.deploy();  
  
  console.log("Contract address:", contract.address);  
}  
  
main()  
  .then(() => process.exit(0))  
  .catch((error) => {  
    console.error(error);  
    process.exit(1);  
  });
```

4. Из командной строки запустите скрипт деплоя:

```
npx hardhat run deploy.js --network sepolia
```

Замените `sepolia` на название выбранной вами сети из конфигурационного файла.

Hardhat выполнит деплой контракта на указанной сети и выведет адрес, только что развернутого контракта. Убедитесь, что у вас есть достаточный баланс и подключение к выбранной сети, чтобы выполнить деплой контракта.

Валидация контрактов в etherscan из hardhat

1. Создайте аккаунт на Etherscan, если у вас его еще нет.
2. Войдите в свою учетную запись на Etherscan.
3. Перейдите в раздел "My Account", затем выберите "API-Keys" в выпадающем меню.
4. Нажмите кнопку "Добавить ключ API" и получите вашу уникальную ключевую фразу API.
5. В вашем проекте Hardhat установите пакет `@nomiclabs/hardhat-etherscan` с помощью следующей команды:

```
npm install --save-dev @nomiclabs/hardhat-etherscan
```

6. В конфигурационном файле `hardhat.config.js` добавьте следующий блок кода с настройками `etherscan` в объект `etherscan` и используйте вашу ключевую фразу API из Etherscan:

```
module.exports = {  
  // ... оставшаяся конфигурация Hardhat  
  etherscan: {  
    apiKey: "ваша_ключевая_фраза_API"  
  }  
};
```

7. В командной строке выполните следующую команду, чтобы проверить ваш проект Hardhat:

```
1 npx hardhat verify --network sepolia <адрес_контракта>  
  "аргументы_конструктора"
```

8. Замените `sepolia` на имя сети в вашей конфигурации Hardhat, `<адрес_контракта>` на адрес вашего развернутого контракта и `"аргументы_конструктора"` на значения аргументов конструктора вашего контракта (если они есть).



Вопросы для самопроверки

Вопросы

1. Спишется ли комиссия за транзакцию, если на выполнение смарт-контракта не хватит газа?
2. Смарт-контракты и аккаунты пользователей принадлежат одинаковому пространству адресов?
3. Все ли отправленные в сеть транзакции попадают в блокчейн? Как увеличить вероятность попадания транзакции в блокчейн?

Ответы

1. Да, спишется. В ходе выполнения за каждую команду будет сжигаться газ, если смарт-контракт израсходует весь газ, указанный в поле gaslimit, то выполнение транзакции прервётся, ей будет присвоен статус failed, а с отправителя будет списана комиссия за весь сожжённый газ.
2. Да. В Ethereum есть два типа аккаунтов - внешние и контракты, и их интерфейсы неотличимы.
3. При отправке транзакции попадают в mempool, из которого они извлекаются майнерами для последующей упаковки в блоки. Майнеры в первую очередь выбирают из пула наиболее выгодные для себя транзакции. Чтобы повысить привлекательность транзакции надо увеличить её gasprice - стоимость газа, выраженную в ETH.

Дополнительные материалы

1. https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf
2. <https://iancoleman.io/bip39/>

Использованная литература

1. <https://ethereum.org/en/developers/docs/>
2. <https://docs.soliditylang.org/>

3. <https://hardhat.org>