Minter Documentation

The Authors

Dec 11, 2018

Contents

1 Minter 101 **1**

1.1 Introduction . . . 1
1.2 Install Minter . . . 1
1.3 Blockchain Specification . . . 3
1.4 Coins . . . 3
1.5 Transactions . . . 5
1.6 Minter Check . . . 11
1.7 Multisignatures . . . 12
1.8 Commissions . . . 13
1.9 Validators . . . 14
1.10 Delegator FAQ . . . 17
1.11 Minter Node API . . . 19
1.12 Minter SDKs . . . 27

2 Minter 102 **29**

2.1 Running in production . . . 29

CHAPTER 1

Minter 101

1.1 Introduction

Welcome to the Minter guide! This is the best place to start if you are new to Minter.

1.1.1 What is Minter?

Minter is a blockchain network that lets people, projects, and companies issue and manage their own coins and trade them at a fair market price with absolute and instant liquidity.

1.2 Install Minter

There are several ways you can install Minter Blockchain Testnet node on your machine:

1.2.1 Using binary

Download Minter

Get latest binary build suitable for your architecture and unpack it to desired folder.

Run Minter

```
./minter
```

Then open http://localhost:3000/ in local browser to see node's GUI.

1.2.2 Using Docker

You'll need docker and docker compose installed.

Clone Minter source code to your machine

```
git clone https://github.com/MinterTeam/minter-go-node.git
cd minter-go-node
```

Start Minter

```
docker-compose up
```

Then open http://localhost:3000/ in local browser to see node's GUI.

1.2.3 From Source

You'll need `go` installed and the required environment variables set

Clone Minter source code to your machine

```
mkdir -p $GOPATH/src/github.com/MinterTeam
cd $GOPATH/src/github.com/MinterTeam
git clone https://github.com/MinterTeam/minter-go-node.git
cd minter-go-node
```

Get Tools & Dependencies

```
make get_tools
make get_vendor_deps
```

Compile

```
make install
```

to put the binary in `$GOPATH/bin` or use:

```
make build
```

to put the binary in `./build`.

The latest `minter version` is now installed.

Run Minter

```
minter
```

Then open http://localhost:3000/ in local browser to see node's GUI.

1.3 Blockchain Specification

1.3.1 Tendermint

Minter Blockchain utilizes `Tendermint Consensus Engine`.

Tendermint is software for securely and consistently replicating an application on many machines. By securely, we mean that Tendermint works even if up to 1/3 of machines fail in arbitrary ways. By consistently, we mean that every non-faulty machine sees the same transaction log and computes the same state. Secure and consistent replication is a fundamental problem in distributed systems; it plays a critical role in the fault tolerance of a broad range of applications, from currencies, to elections, to infrastructure orchestration, and beyond.

Tendermint is designed to be easy-to-use, simple-to-understand, highly performant, and useful for a wide variety of distributed applications.

You can read more about Tendermint Consensus in official documentation.

1.3.2 Consensus

In Minter we implemented Delegated Proof of Stake (DPOS) Consensus Protocol.

DPOS is the fastest, most efficient, most decentralized, and most flexible consensus model available. DPOS leverages the power of stakeholder approval voting to resolve consensus issues in a fair and democratic way.

1.3.3 Block speed

Minter Blockchain is configured to produce `1 block per 5 sec`. Actual block speed may vary depends on validators count, their computational power, internet speed, etc.

1.3.4 Block size

We limit block size to `10 000 transactions`. Block size in terms of bytes is not limited.

1.4 Coins

Minter Blockchain is multi-coin system.

Base coin in testnet is `MNT`.
Base coin in mainnet is `BIP`.

Smallest part of *each* coin is called `pip`.
1 pip = 1^-18 of any coin. In Blockchain and API we only operating with pips.

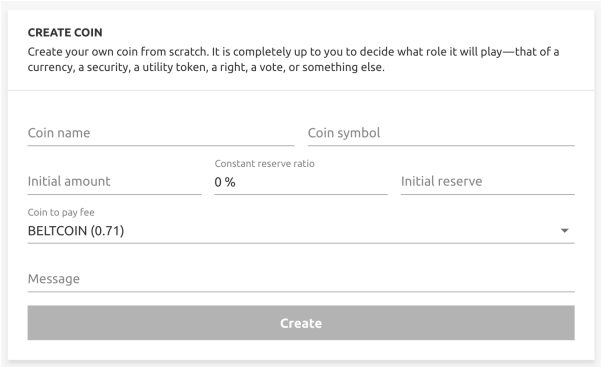
Note: Each coin has its **own** pip. You should treat pip like atomic part of a coin, not as currency.

1 MNT = 10^18 pip (MNT's pip)

1 ABC = 10^18 pip (ABC's pip)
1 MNT != 1 ABC

1.4.1 Coin Issuance

Every user of Minter can issue own coin. Each coin is backed by base coin in some proportion. Issue own coin is as simple as filling a form with given fields:



- **Coin name** - Name of a coin. Arbitrary string up to 64 letters length.
- **Coin symbol** - Symbol of a coin. Must be unique, alphabetic, uppercase, 3 to 10 letters length.
- **Initial supply** - Amount of coins to issue. Issued coins will be available to sender account.
- **Initial reserve** - Initial reserve in base coin.
- **Constant Reserve Ratio (CRR)** - uint, should be from 10 to 100.

After coin issued you can send is as ordinary coin using standard wallets.

1.4.2 Issuance Fees

To issue a coin Coiner should pay fee. Fee is depends on length of Coin Symbol.

3 letters – 1 000 000 bips + standard transaction fee
4 letters – 100 000 bips + standard transaction fee
5 letters – 10 000 bips + standard transaction fee
6 letters – 1000 bips + standard transaction fee
7 letters – 100 bips + standard transaction fee
8 letters – 10 bips + standard transaction fee
9-10 letters - just standard transaction fee

1.4.3 Coin Exchange

Each coin in system can be instantly exchanged to another coin. This is possible because each coin has "reserve" in base coin.

Here are some formulas we are using for coin conversion:

CalculatePurchaseReturn Given a coin supply (s), reserve balance (r), CRR (c) and a deposit amount (d), calculates the return for a given conversion (in the base coin):

```
return s * ((1 + d / r) ^ c - 1);
```

CalculateSaleReturn Given a coin supply (s), reserve balance (r), CRR (c) and a sell amount (a), calculates the return for a given conversion

```
return r * (1 - (1 - a / s) ^ (1 / c));
```

1.5 Transactions

1.5.1 Semantic

Transactions in Minter are RLP-encoded structures.

Example of a signed transaction:

```
f873230101aae98a4d4e540000000000000094a93163fdf10724dc4785ff5cbfb9
ac0b5949409f880de0b6b3a764000080801ba06838db4a2197cfd70ede8d8d184d
bf332932ca051a243eb7886791250e545dd3a04b63fb1d1b5ef5f2cbd2ea12530c
da520b3280dcb75bfd45a873629109f24b29
```

Each transaction has:

- **Nonce** - int, used for prevent transaction reply.
- **Gas Price** - big int, used for managing transaction fees.
- **Gas Coin** - 10 bytes, symbol of a coin to pay fee
- **Type** - type of transaction (see below).
- **Data** - data of transaction (depends on transaction type).
- **Payload** (arbitrary bytes) - arbitrary user-defined bytes.
- **Service Data** - reserved field.
- **Signature Type** - single or multisig transaction.
- **Signature Data** - digital signature of transaction.

```
type Transaction struct {
    Nonce         uint64
    GasPrice      *big.Int
    GasCoin       [10]byte
    Type          byte
    Data          []byte
    Payload       []byte
    ServiceData   []byte
    SignatureType byte
    SignatureData Signature
}

type Signature struct {
    V           *big.Int
    R           *big.Int
```

(continues on next page)

(continued from previous page)

```
    S           *big.Int
}

type MultiSignature struct {
    MultisigAddress [20]byte
    Signatures      []Signature
}
```

1.5.2 Signature Types

Type Name	Byte
TypeSingle	0x01
TypeMulti	0x02

1.5.3 Types

Type of transaction is determined by a single byte.

Type Name	Byte
TypeSend	0x01
TypeSellCoin	0x02
TypeSellAllCoin	0x03
TypeBuyCoin	0x04
TypeCreateCoin	0x05
TypeDeclareCandidacy	0x06
TypeDelegate	0x07
TypeUnbond	0x08
TypeRedeemCheck	0x09
TypeSetCandidateOnline	0x0A
TypeSetCandidateOffline	0x0B
TypeCreateMultisig	0x0C
TypeMultisend	0x0D

1.5.4 Send transaction

Type: **0x01**

Transaction for sending arbitrary coin.

Data field contents:

```
type SendData struct {
    Coin  [10]byte
    To    [20]byte
    Value *big.Int
}
```

Coin - Symbol of a coin.

To - Recipient address in Minter Network.
Value - Amount of **Coin** to send.

1.5.5 Sell coin transaction

Type: **0x02**

Transaction for selling one coin (owned by sender) in favour of another coin in a system.

Data field contents:

```
type SellCoinData struct {
    CoinToSell          [10]byte
    ValueToSell         *big.Int
    CoinToBuy           [10]byte
    MinimumValueToBuy   *big.Int
}
```

CoinToSell - Symbol of a coin to give.
ValueToSell - Amount of **CoinToSell** to give.
CoinToBuy - Symbol of a coin to get.
MinimumValueToBuy - Minimum value of coins to get.

1.5.6 Sell all coin transaction

Type: **0x03**

Transaction for selling all existing coins of one type (owned by sender) in favour of another coin in a system.

Data field contents:

```
type SellAllCoinData struct {
    CoinToSell          [10]byte
    CoinToBuy           [10]byte
    MinimumValueToBuy   *big.Int
}
```

CoinToSell - Symbol of a coin to give.
CoinToBuy - Symbol of a coin to get.
MinimumValueToBuy - Minimum value of coins to get.

1.5.7 Buy coin transaction

Type: **0x04**

Transaction for buy a coin paying another coin (owned by sender).

Data field contents:

```
type BuyCoinData struct {
    CoinToBuy           [10]byte
    ValueToBuy          *big.Int
    CoinToSell          [10]byte
    MaximumValueToSell  *big.Int
}
```

CoinToBuy - Symbol of a coin to get.
ValueToBuy - Amount of **CoinToBuy** to get.
CoinToSell - Symbol of a coin to give.
MaximumValueToSell - Maximum value of coins to sell.

1.5.8 Create coin transaction

Type: **0x05**

Transaction for creating new coin in a system.

Data field contents:

```
type CreateCoinData struct {
    Name                 string
    Symbol               [10]byte
    InitialAmount        *big.Int
    InitialReserve       *big.Int
    ConstantReserveRatio uint
}
```

Name - Name of a coin. Arbitrary string up to 64 letters length.
Symbol - Symbol of a coin. Must be unique, alphabetic, uppercase, 3 to 10 symbols length.
InitialAmount - Amount of coins to issue. Issued coins will be available to sender account.
InitialReserve - Initial reserve in BIP's.
ConstantReserveRatio - CRR, uint, should be from 10 to 100.

1.5.9 Declare candidacy transaction

Type: **0x06**

Transaction for declaring new validator candidacy.

Data field contents:

```
type DeclareCandidacyData struct {
    Address    [20]byte
    PubKey     []byte
    Commission uint
    Coin       [10]byte
    Stake      *big.Int
}
```

Address - Address of candidate in Minter Network. This address would be able to control candidate. Also all rewards will be sent to this address.
PubKey - Public key of a validator.
Commission - Commission (from 0 to 100) from rewards which delegators will pay to validator.
Coin - Symbol of coin to stake.
Stake - Amount of coins to stake.

1.5.10 Delegate transaction

Type: **0x07**

Transaction for delegating funds to validator.

Data field contents:

```
type DelegateData struct {
    PubKey []byte
    Coin   [10]byte
    Stake  *big.Int
}
```

PubKey - Public key of a validator.
Coin - Symbol of coin to stake.
Stake - Amount of coins to stake.

1.5.11 Unbond transaction

Type: **0x08**

Transaction for unbonding funds from validator's stake.

Data field contents:

```
type UnbondData struct {
    PubKey []byte
    Coin   [10]byte
    Value  *big.Int
}
```

PubKey - Public key of a validator.
Coin - Symbol of coin to unbond.
Value - Amount of coins to unbond.

1.5.12 Redeem check transaction

Type: **0x09**

Transaction for redeeming a check.

Data field contents:

```
type RedeemCheckData struct {
    RawCheck []byte
    Proof    [65]byte
}
```

RawCheck - Raw check received from sender.
Proof - Proof of owning a check.

1.5.13 Set candidate online transaction

Type: **0x0A**

Transaction for turning candidate on. This transaction should be sent from address which is set in the "Declare candidacy transaction".

Data field contents:

```
type SetCandidateOnData struct {
    PubKey []byte
}
```

PubKey - Public key of a validator.

1.5.14 Set candidate offline transaction

Type: **0x0B**

Transaction for turning candidate off. This transaction should be sent from address which is set in the "Declare candidacy transaction".

Data field contents:

```
type SetCandidateOffData struct {
    PubKey []byte
}
```

PubKey - Public key of a validator.

1.5.15 Create multisig address

Type: **0x0C**

Transaction for creating multisignature address.

Data field contents:

```
type CreateMultisigData struct {
    Threshold uint
    Weights   []uint
```

(continues on next page)

(continued from previous page)

```
    Addresses [][20]byte
}
```

1.5.16 Multisend transaction

Type: **0x0D**

Transaction for sending coins to multiple addresses.

Data field contents:

```
type MultisendData struct {
    List []MultisendDataItem
}

type MultisendDataItem struct {
    Coin  [10]byte
    To    [20]byte
    Value *big.Int
}
```

1.6 Minter Check

Minter Check is like an ordinary bank check. Each user of network can issue check with any amount of coins and pass it to another person. Receiver will be able to cash a check from arbitrary account.

1.6.1 Introduction

Checks are prefixed with “Mc”. Here is example of a Minter Check:

```
Mcf89b01830f423f8a4d4e5400000000000000843b9aca00b8419b3beac2c6ad88a8bd54d2
4912754bb820e58345731cb1b9bc0885ee74f9e50a58a80aa990a29c98b05541b266af99d3
825bb1e5ed4e540c6e2f7c9b40af9ecc011ca0387fd67ec41be0f1cf92c7d0181368b4c67a
b07df2d2384192520d74ff77ace6a04ba0e7ad7b34c64223fe59584bc464d53fcdc7091faa
ee5df0451254062cfb37
```

Each Minter Check has:

- **Nonce** - unique “id” of the check.
- **Coin Symbol** - symbol of coin.
- **Value** - amount of coins.
- **Due Block** - defines last block height in which the check can be used.
- **Lock** - secret to prevent hijacking.
- **Signature** - signature of issuer.

1.6.2 Check hijacking protection

Minter Checks are issued offline and do not exist in blockchain before "cashing". So we decided to use special passphrase to protect checks from hijacking by another person in the moment of activation. Hash of this passphrase is used as private key in ECDSA to prove that sender is the one who owns the check.

TODO: describe algorithm

1.6.3 How to issue a Minter Check

For issuing Minter Check you can use our tool.

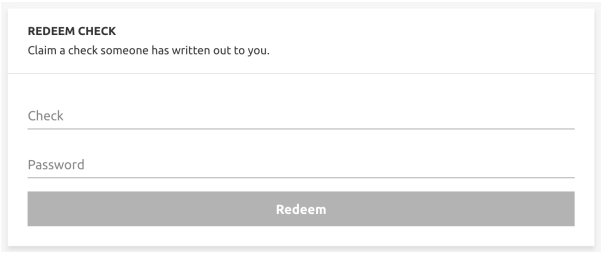
You will need to fill a form:

- **Nonce** - unique "id" of the check.
- **Coin Symbol** - symbol of coin.
- **Value** - amount of coins.
- **Pass phrase** - secret phrase which you will pass to receiver of the check.
- **Private key** - private key of an account with funds to send.

1.6.4 How to cash a Minter Check

To redeem a check user should have:

- Check itself
- Secret passphrase



After redeeming balance of user will increased instantly.

1.6.5 Commission

There is no commission for issuing a check because it done offline. In the moment of cashing issuer will pay standard "send" commission.

1.7 Multisignatures

Minter has built-in support for multisignature wallets. Multisignatures, or technically Accountable Subgroup Multisignatures (ASM), are signature schemes which enable any subgroup of a set of signers to sign any message, and reveal to the verifier exactly who the signers were.

Suppose the set of signers is of size n. If we validate a signature if any subgroup of size k signs a message, this becomes what is commonly reffered to as a k of n multisig in Bitcoin.

Minter Multisig Wallets has 2 main goals:

- Atomic swaps with sidechains
- Basic usage to manage funds within Minter Blockchain

1.7.1 Structure of multisig wallet

Each multisig wallet has:

- Set of signers with corresponding weights
- Threshold

Transactions from multisig wallets are proceed identically to the K of N multisig in Bitcoin, except the multisig fails if the sum of the weights of signatures is less than the threshold.

1.7.2 How to create multisig wallet

TO BE DESCRIBED

1.7.3 How to use multisig wallet

TO BE DESCRIBED

1.8 Commissions

For each transaction sender should pay fee. Fees are measured in “units”.

1 unit = 10^15 pip = 0.001 bip.

1.8.1 Standard commissions

Here is a list of current fees:

Type	Fee
TypeSend	10 units
TypeSellCoin	100 units
TypeSellAllCoin	100 units
TypeBuyCoin	100 units
TypeCreateCoin	1000 units
TypeDeclareCandidacy	10000 units
TypeDelegate	100 units
TypeUnbond	100 units
TypeRedeemCheck	10 units
TypeSetCandidateOnline	100 units
TypeSetCandidateOffline	100 units

Also sender should pay extra 2 units per byte in Payload and Service Data fields.

Multisend transaction requires 1 unit per recipient excluding first one.

1.8.2 Special fees

To issue a coin with short name Coiner should pay extra fee. Fee is depends on length of Coin Symbol.

3 letters – 1 000 000 bips + standard transaction fee
4 letters – 100 000 bips + standard transaction fee
5 letters – 10 000 bips + standard transaction fee
6 letters – 1000 bips + standard transaction fee
7 letters – 100 bips + standard transaction fee
8 letters – 10 bips + standard transaction fee
9-10 letters - just standard transaction fee

1.9 Validators

1.9.1 Introduction

The Minter Blockchain is based on Tendermint, which relies on a set of validators that are responsible for committing new blocks in the blockchain. These validators participate in the consensus protocol by broadcasting votes which contain cryptographic signatures signed by each validator's private key.

Validator candidates can bond their own coins and have coins "delegated", or staked, to them by token holders. The validators are determined by who has the most stake delegated to them.

Validators and their delegators will earn BIP (MNT) as rewards for blocks and commissions. Note that validators can set commission on the rewards their delegators receive as additional incentive.

If validators double sign or frequently offline, their staked coins (including coins of users that delegated to them) can be slashed. The penalty depends on the severity of the violation.

1.9.2 Requirements

Minimal requirements for running Validator's Node in testnet are:

- 4GB RAM
- 200GB SSD
- x64 2.0 GHz 4 vCPUs

SSD disks are preferable for high transaction throughput.

Recommended:

- 4GB RAM
- 200GB SSD
- x64 3.4 GHz 8 vCPUs
- HSM

1.9.3 Validators limitations

Minter Network has limited number of available slots for validators.

At genesis there will be just `16` of them. `4` slots will be added each `518,400` blocks. Maximum validators count is `256`.

1.9.4 Rewards

Rewards for blocks and commissions are accumulated and proportionally (based on stake value) payed once per `12 blocks` (approx 1 minute) to all active validators (and their delegators).

Block rewards are configured to decrease from 333 to 0 BIP (MNT) in ~7 years.

Delegators receive their rewards at the same time after paying commission to their validators (commission value is based on validator's settings).

`10%` from reward going to DAO account.

`10%` from reward going to Developers.

1.9.5 Rules and fines

Validators have one main responsibility:

- Be able to constantly run a correct version of the software: validators need to make sure that their servers are always online and their private keys are not compromised.

If a validator misbehaves, its bonded stake along with its delegators' stake and will be slashed. The severity of the punishment depends on the type of fault. There are 3 main faults that can result in slashing of funds for a validator and its delegators:

- **Double signing**: If someone reports on chain A that a validator signed two blocks at the same height on chain A and chain B, this validator will get slashed on chain A
- **Unavailability**: If a validator's signature has not been included in the last 12 blocks, 1% of stake will get slashed and validator will be turned off

Note that even if a validator does not intentionally misbehave, it can still be slashed if its node crashes, looses connectivity, gets DDOSed, or if its private key is compromised.

1.9.6 Becoming validator in testnet

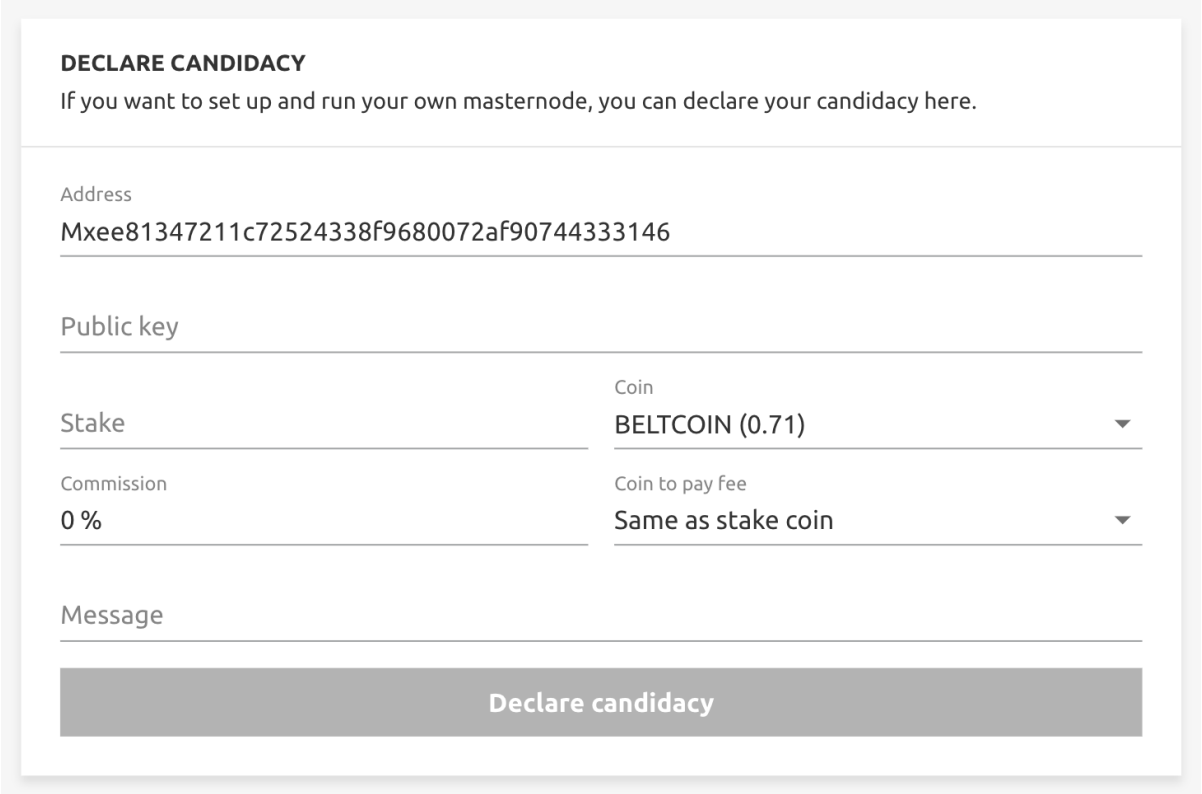
1. **Install and run Minter Full Node.** See *Install Minter*. Make sure your node successfully synchronized.
2. Get your validator's public key from Minter GUI.
3. **Go to Minter Console and send 2 transactions:** Fill and send `Declare candidacy` and `Set candidate online` forms.

   P.S. You can receive testnet coins in our telegram wallet @BipWallet_Bot.

   3.1. Declare candidacy Validators should declare their candidacy, after which users can delegate and, if they so wish, unbond. Then declaring candidacy validator should fill a form:

   - Address - You will receive rewards to this address and will be able to on/off your validator.
   - Public Key - Paste public key from step 2 *(Mp...)*.
   - Commission - Set commission for delegated stakes.

- Coin - Enter coin of your stake (i.e. MNT).
- Stake - Enter value of your stake in given coin.



3.2. Set candidate online Validator is **offline** by default. When offline, validator is not included in the list of Minter Blockchain validators, so he is not receiving any rewards and cannot be punished for low availability.

To turn your validator **on**, you should provide Public Key (from step 2 *(Mp...)*).

Note: You should send transaction from address you choose in Address field in step 3.1

4. **Done.** Now you will receive reward as long as your node is running and available.

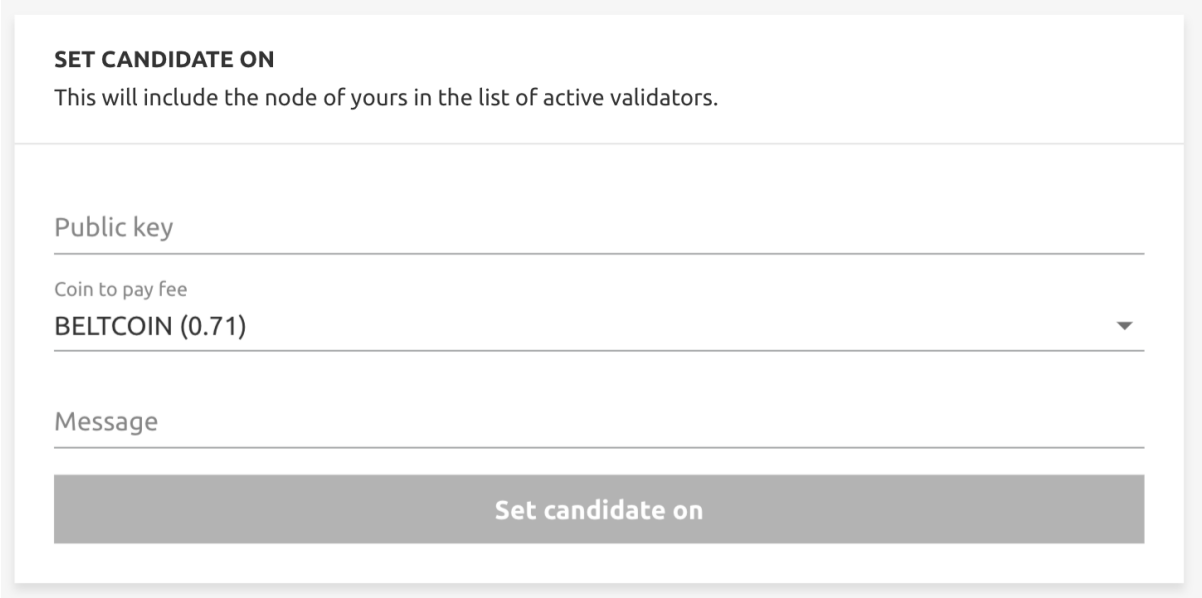
1.9.7 DDOS protection. Sentry node architecture

Denial-of-service attacks occur when an attacker sends a flood of internet traffic to an IP address to prevent the server at the IP address from connecting to the internet.

An attacker scans the network, tries to learn the IP address of various validator nodes and disconnect them from communication by flooding them with traffic.

One recommended way to mitigate these risks is for validators to carefully structure their network topology in a so-called sentry node architecture.

Validator nodes should only connect to full-nodes they trust because they operate them themselves or are run by other validators they know socially. A validator node will typically run in a data center. Most data centers provide direct links the networks of major cloud providers. The validator can use those links to connect to sentry nodes in the cloud. This shifts the burden of denial-of-service from the validator's node directly to its sentry nodes, and may require new sentry nodes be spun up or activated to mitigate attacks on existing ones.



Sentry nodes can be quickly spun up or change their IP addresses. Because the links to the sentry nodes are in private IP space, an internet based attacked cannot disturb them directly. This will ensure validator block proposals and votes always make it to the rest of the network.

It is expected that good operating procedures on that part of validators will completely mitigate these threats.

Practical instructions

To setup your sentry node architecture you can follow the instructions below:

Validators nodes should edit their `config.toml`:

```
# Comma separated list of nodes to keep persistent connections to
# Do not add private peers to this list if you don't want them advertised
persistent_peers = [list of sentry nodes]

# Set true to enable the peer-exchange reactor
pex = false
```

Sentry Nodes should edit their `config.toml`:

```
# Comma separated list of peer IDs to keep private (will not be gossiped to other␣
↪peers)
private_peer_ids = "ipaddress of validator nodes"
```

1.10 Delegator FAQ

1.10.1 What is a delegator?

People that cannot, or do not want to run validator operations, can still participate in the staking process as delegators. Indeed, validators are not chosen based on their own stake but based on their total stake, which is the sum of their own stake and of the stake that is delegated to them. This is an important property, as it makes delegators a safeguard

against validators that exhibit bad behavior. If a validator misbehaves, its delegators will move their staked coins away from it, thereby reducing its stake. Eventually, if a validator's stake falls under the top addresses with highest stake, it will exit the validator set.

Delegators share the revenue of their validators, but they also share the risks. In terms of revenue, validators and delegators differ in that validators can apply a commission on the revenue that goes to their delegator before it is distributed. This commission is known to delegators beforehand and cannot be changed. In terms of risk, delegators' coins can be slashed if their validator misbehaves. For more, see Risks section.

To become delegators, coin holders need to send a "Delegate transaction" where they specify how many coins they want to bond and to which validator. Later, if a delegator wants to unbond part or all of its stake, it needs to send an "Unbond transaction". From there, the delegator will have to wait 30 days to retrieve its coins.

1.10.2 Directives of delegators

Being a delegator is not a passive task. Here are the main directives of a delegator:

- Perform careful due diligence on validators before delegating. If a validator misbehaves, part of its total stake, which includes the stake of its delegators, can be slashed. Delegators should therefore carefully select validators they think will behave correctly.
- Actively monitor their validator after having delegated. Delegators should ensure that the validators they're delegating to behaves correctly, meaning that they have good uptime, do not get hacked and participate in governance. If a delegator is not satisfied with its validator, it can unbond and switch to another validator.

1.10.3 Revenue

Validators and delegators earn revenue in exchange for their services. This revenue is given in three forms:

- Block rewards
- Transaction fees: Each transaction on the Minter Network comes with transactions fees. Fees are distributed to validators and delegators in proportion to their stake.

1.10.4 Validator's commission

Each validator's staking pool receives revenue in proportion to its total stake. However, before this revenue is distributed to delegators inside the staking pool, the validator can apply a commission. In other words, delegators have to pay a commission to their validators on the revenue they earn.

`10%` from reward going to DAO account.

`10%` from reward going to Developers.

Lets consider a validator whose stake (i.e. self-bonded stake + delegated stake) is 10% of the total stake of all validators. This validator has 20% self-bonded stake and applies a commission of 10%. Now let us consider a block with the following revenue:

- 111 Bips as block reward (after subtraction taxes of 20%)
- 10 Bips as transaction fees (after subtraction taxes of 20%)

This amounts to a total of 121 Bips to be distributed among all staking pools.

Our validator's staking pool represents 10% of the total stake, which means the pool obtains 12.1 bips. Now let us look at the internal distribution of revenue:

- Commission = 10% * 80% * 12.1 bips = 0.69696 bips

- Validator's revenue = 20% * 12.1 bips + Commission = 3.11696 bips
- Delegators' total revenue = 80% * 12.1 bips - Commission = 8.98304 bips

Then, each delegator in the staking pool can claim its portion of the delegators' total revenue.

1.10.5 Risks

Staking coins is not free of risk. First, staked coins are locked up, and retrieving them requires a 30 days waiting period called unbonding period. Additionally, if a validator misbehaves, a portion of its total stake can be slashed (i.e. destroyed). This includes the stake of their delegators.

There are 2 main slashing conditions:

- **Double signing**: If someone reports on chain A that a validator signed two blocks at the same height on chain A and chain B, this validator will get slashed on chain A
- **Unavailability**: If a validator's signature has not been included in the last 12 blocks, 1% of stake will get slashed and validator will be turned off

This is why delegators should perform careful due diligence on validators before delegating. It is also important that delegators actively monitor the activity of their validators. If a validator behaves suspiciously or is too often offline, delegators can choose to unbond from it or switch to another validator. Delegators can also mitigate risk by distributing their stake across multiple validators.

1.11 Minter Node API

Minter Node API is based on JSON format. JSON is a lightweight data-interchange format. It can represent numbers, strings, ordered sequences of values, and collections of name/value pairs.

If request is successful, Minter Node API will respond with `result` key and code equal to zero. Otherwise, it will respond with non-zero code and key `log` with error description.

1.11.1 Status

This endpoint shows current state of the node. You also can use it to check if node is running in normal mode.

```
curl -s 'localhost:8841/status'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "version": "0.8.0",
    "latest_block_hash":
↪"171F3A749F85425147986DD90EA0C397440B6A3C1FEF8F30E5E5F729DA174CC2",
    "latest_app_hash":
↪"55E75C9860E56AF3DEB8DD55741185F658569AB43C084436DDDB69CBFB06CC63",
    "latest_block_height": "4",
    "latest_block_time": "2018-12-03T13:18:42.50969Z",
    "tm_status": {
      "node_info": {
        "protocol_version": {
          "p2p": "4",
```


(continues on next page)

(continued from previous page)


```
          "block": "7",
          "app": "0"
        },
        "id": "9d5eb9f8fb7ada3ff6228841c3500f39e3121901",
        "listen_addr": "tcp://0.0.0.0:26656",
        "network": "minter-test-network-27-local",
        "version": "0.26.4",
        "channels": "4020212223303800",
        "moniker": "MacBook-Pro-Daniil-2.local",
        "other": {
          "tx_index": "on",
          "rpc_address": "tcp://0.0.0.0:26657"
        }
      },
      "sync_info": {
        "latest_block_hash":
↪"171F3A749F85425147986DD90EA0C397440B6A3C1FEF8F30E5E5F729DA174CC2",
        "latest_app_hash":
↪"55E75C9860E56AF3DEB8DD55741185F658569AB43C084436DDDB69CBFB06CC63",
        "latest_block_height": "4",
        "latest_block_time": "2018-12-03T13:18:42.50969Z",
        "catching_up": false
      },
      "validator_info": {
        "address": "AB15A084DD592699812E9B22385C1959E7AEFFB8",
        "pub_key": {
          "type": "tendermint/PubKeyEd25519",
          "value": "4LpQ40aLB/u8EnhAlT649P5X1ugWLfk7rv159dW8K5c="
        },
        "voting_power": "100000000"
      }
    }
  }
}
```


1.11.2 Candidate

This endpoint shows candidate's info by provided public_key. It will respond with `404` code if candidate is not found.

- **candidate_address** - Address of a candidate in minter network. This address is used to manage candidate and receive rewards.
- **total_stake** - Total stake calculated in base coin (MNT or BIP).
- **commission** - Commission for delerators. Measured in percents. Can be 0..100.
- **accumulated_reward** - Reward waiting to be sent to validator and his delegators. Reward is payed each 12 blocks.
- **stakes** - List of candidate's stakes.
- **created_at_block** - Height of block when candidate was created.
- **status** - Status of a candidate.
  - `1` - Offline
  - `2` - Online

- **absent_times** - How many blocks candidate missed. If this number reaches 12, then candidate's stake will be slashed by 1% and candidate will be turned off.

```
curl -s 'localhost:8841/candidate?pubkey={public_key}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "candidate_address": "Mxee81347211c72524338f9680072af90744333146",
    "total_stake": 0,
    "pub_key": "Mpe0ba50e3468b07fbbc127840953eb8f4fe57d6e8162df93baefd79f5d5bc2b97",
    "commission": "100",
    "stakes": [
      {
        "owner": "Mxee81347211c72524338f9680072af90744333146",
        "coin": "MNT",
        "value": "100000000000000000000000",
        "bip_value": "100000000000000000000000"
      }
    ],
    "created_at_block": "1",
    "status": 2
  }
}
```


1.11.3 Validators

Returns list of active validators.

```
curl -s 'localhost:8841/validators'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": [
    {
      "pubkey": "Mpddfadfb15908ed5607c79e66aaf4030ef93363bd1846d64186d52424b1896c83",
      "voting_power": "100000000"
    }
  ]
}
```


1.11.4 Address

Returns the balance of given account and the number of outgoing transaction.

```
curl -s 'localhost:8841/address?address={address}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
```


(continues on next page)

(continued from previous page)


```
    "balance": {
      "MNT": "10001048950000000000000000"
    },
    "transaction_count": "0"
  }
}
```


Result->balance: Map of balances. CoinSymbol => Balance (in pips).
Result->transaction_count: Count of transactions sent from the account.

1.11.5 Send transaction

Sends transaction to the Minter Network. **Note:** tx should start with 0x prefix.

```
curl -s 'localhost:8841/send_transaction?tx={transaction}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "code": 0,
    "data": "",
    "log": "",
    "hash": "C6C6B5008AF8077FB0CE817DDB79268D1C66B6B353AF76778CA5A264A80069DB"
  }
}
```


Result: Transaction hash.

1.11.6 Transaction

```
curl -s 'localhost:8841/transaction?hash={hash}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "hash": "C6C6B5008AF8077FB0CE817DDB79268D1C66B6B353AF76778CA5A264A80069DB",
    "raw_tx":
↪"f88701018a4d4e540000000000000001aae98a4d4e540000000000000094ee81347211c72524338f9680072af907443331
↪",
    "height": "387",
    "index": 0,
    "from": "Mxee81347211c72524338f9680072af90744333146",
    "nonce": "1",
    "gas_price": "1",
    "gas_coin": "MNT",
    "gas_used": "18",
    "type": 1,
    "data": {
```


(continues on next page)

(continued from previous page)


```
      "coin": "MNT",
      "to": "Mxee81347211c72524338f9680072af90744333146",
      "value": "1200000000000000000"
    },
    "payload": "VGVzdA==",
    "tags": {
      "tx.coin": "MNT",
      "tx.type": "01",
      "tx.from": "ee81347211c72524338f9680072af90744333146",
      "tx.to": "ee81347211c72524338f9680072af90744333146"
    }
  }
}
```


1.11.7 Block

Returns block data at given height.

```
curl -s 'localhost:8841/block?height={height}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "hash": "0B1226C12783373BB2FFB451A104FF2BE47F59B8E7B6690B7712AADBA197D2FC",
    "height": "9",
    "time": "2018-12-05T09:14:57.114925Z",
    "num_txs": "1",
    "total_txs": "1",
    "transactions": [
      {
        "hash": "Mt0e765f48042683160d33c610a90845aeef5f8e0d71cab60e01895f8bd973d614",
        "raw_tx":
↪"f8a701018a4d4e54000000000000006b84df84b94ee81347211c72524338f9680072af90744333146a021e1d043c6d9c0
↪",
        "from": "Mxee81347211c72524338f9680072af90744333146",
        "nonce": "1",
        "gas_price": "1",
        "type": 6,
        "data": {
          "address": "Mxee81347211c72524338f9680072af90744333146",
          "pub_key":
↪"Mp21e1d043c6d9c0bb0929ab8d1dd9f3948de0f5ad7234ce773a501441d204aa9e",
          "commission": "10",
          "coin": "MNT",
          "stake": "100000000000000000"
        },
        "payload": "",
        "service_data": "",
        "gas": "10000",
        "gas_coin": "MNT",
        "gas_used": "10000",
        "tags": {}
      }
    ],
```


(continues on next page)

(continued from previous page)


```
    "block_reward": "333000000000000000000",
    "size": "1230",
    "proposer": "Mpddfadfb15908ed5607c79e66aaf4030ef93363bd1846d64186d52424b1896c83",
    "validators": [
      {
        "pubkey": "Mpddfadfb15908ed5607c79e66aaf4030ef93363bd1846d64186d52424b1896c83
↪",
        "signed": true
      }
    ]
  }
}
```


1.11.8 Events

Returns events at given height.

```
curl -s 'localhost:8841/events?height={height}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "events": [
      {
        "type": "minter/RewardEvent",
        "value": {
          "role": "DAO",
          "address": "Mxee81347211c72524338f9680072af90744333146",
          "amount": "367300000000000000000",
          "validator_pub_key":
↪"Mp4d706464666164666231353930386564353630376337396536366161663430333065663933333633626431383436643
↪"
        }
      },
      {
        "type": "minter/RewardEvent",
        "value": {
          "role": "Developers",
          "address": "Mx444c4f1953ea170f74eabef4eee52ed8276a7d5e",
          "amount": "367300000000000000000",
          "validator_pub_key":
↪"Mp4d706464666164666231353930386564353630376337396536366161663430333065663933333633626431383436643
↪"
        }
      },
      {
        "type": "minter/RewardEvent",
        "value": {
          "role": "Validator",
          "address": "Mxee81347211c72524338f9680072af90744333146",
          "amount": "293840000000000000000",
          "validator_pub_key":
↪"Mp4d706464666164666231353930386564353630376337396536366161663430333065663933333633626431383436643
↪"
```


(continues on next page)

(continued from previous page)

```
        }
      }
    ]
  }
}
```

1.11.9 Candidates

Returns full list of candidates.

```
curl -s 'localhost:8841/candidates?height={height}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": [
    {
      "candidate_address": "Mxee81347211c72524338f9680072af90744333146",
      "total_stake": "10000000000000000000000000",
      "pubkey": "Mpddfadfb15908ed5607c79e66aaf4030ef93363bd1846d64186d52424b1896c83",
      "commission": "100",
      "created_at_block": "1",
      "status": 2
    },
    {
      "candidate_address": "Mxee81347211c72524338f9680072af90744333146",
      "total_stake": "9900000000000000000",
      "pubkey": "Mp21e1d043c6d9c0bb0929ab8d1dd9f3948de0f5ad7234ce773a501441d204aa9e",
      "commission": "10",
      "created_at_block": "9",
      "status": 1
    }
  ]
}
```


1.11.10 Coin Info

Returns information about coin.

Note: this method **does not** return information about base coins (MNT and BIP).

```
curl -s 'localhost:8841/coin_info?symbol={SYMBOL}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "name": "TEST",
    "symbol": "TESTCOIN",
    "volume": "100000000000000000000",
    "crr": "100",
    "reserve_balance": "100000000000000000000"
```


(continues on next page)

(continued from previous page)

```
  }
}
```

Result:

- **Coin name** - Name of a coin. Arbitrary string.
- **Coin symbol** - Short symbol of a coin. Coin symbol is unique, alphabetic, uppercase, 3 to 10 letters length.
- **Volume** - Amount of coins exists in network.
- **Reserve balance** - Amount of BIP/MNT in coin reserve.
- **Constant Reserve Ratio (CRR)** - uint, from 10 to 100.
- **Creator** - Address of coin creator account.

1.11.11 Estimate sell coin

Return estimate of sell coin transaction

```
curl -s 'localhost:8841/estimate_coin_sell?coin_to_sell=MNT&coin_to_buy=TESTCOIN&
↪value_to_sell=1'
```

Request params:

- **coin_to_sell** – coin to give
- **value_to_sell** – amount to give (in pips)
- **coin_to_buy** - coin to get

```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "will_pay": "1",
    "commission": "100000000000000000"
  }
}
```

Result: Amount of “to_coin” user should get.

1.11.12 Estimate buy coin

Return estimate of buy coin transaction

```
curl -s 'localhost:8841/estimate_coin_buy?coin_to_sell=MNT&coin_to_buy=TESTCOIN&value_
↪to_buy=1'
```

Request params:

- **coin_to_sell** – coin to give
- **value_to_buy** – amount to get (in pips)
- **coin_to_buy** - coin to get

```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "will_pay": "1",
    "commission": "100000000000000000"
  }
}
```


Result: Amount of “to_coin” user should give.

1.11.13 Estimate tx commission

Return estimate of buy coin transaction

```
curl -s 'localhost:8841/estimate_tx_commission?tx={transaction}'
```


```
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "commission": "1100000000000000000"
  }
}
```


Result: Commission in GasCoin.

1.12 Minter SDKs

1.12.1 JavaScript SDK

- minter-js-sdk – communicate with the Minter blockchain through its API
- minter-js-org – communicate with the my.minter user database
- minterjs-wallet – BIP0032 HD Wallet implementation
- minterjs-tx – create, manipulate and sign Minter transactions
- minterjs-util – a collection of utility functions for Minter

1.12.2 iOS SDK

- minter-ios-core – create, manipulate and sign Minter transactions
- minter-ios-explorer – communicate with the Minter blockchain through Explorer
- minter-ios-my - communicate with the my.minter user database

1.12.3 PHP SDK

- minter-php-sdk – a pure PHP SDK for working with Minter blockchain

1.12.4 Android SDK

CHAPTER 2

Minter 102

2.1 Running in production

2.1.1 DOS Exposure and Mitigation

Validators are supposed to setup Sentry Node Architecture to prevent Denial-of-service attacks. Read more about it.

P2P

The core of the Tendermint peer-to-peer system is `MConnection`. Each connection has `MaxPacketMsgPayloadSize`, which is the maximum packet size and bounded send & receive queues. One can impose restrictions on send & receive rate per connection (`SendRate`, `RecvRate`).

RPC

Endpoints returning multiple entries are limited by default to return 30 elements (100 max).

Rate-limiting and authentication are another key aspects to help protect against DOS attacks. While in the future we may implement these features, for now, validators are supposed to use external tools like NGINX or traefik to achieve the same things.

2.1.2 Monitoring Tendermint

Each Tendermint instance has a standard */health* RPC endpoint, which responds with 200 (OK) if everything is fine and 500 (or no response) - if something is wrong.

Other useful endpoints include mentioned earlier */status*, */net_info* and */validators*.

We have a small tool, called tm-monitor, which outputs information from the endpoints above plus some statistics.

2.1.3 Monitoring Minter

Each Minter instance has a standard */api/status* endpoint, which responds with 200 (OK) if everything is fine and 500 (or no response) - if something is wrong.

2.1.4 What happens when my app dies?

You are supposed to run Tendermint and Minter under a process supervisor (like systemd or runit). It will ensure Tendermint and Minter is always running (despite possible errors).

2.1.5 Signal handling

We catch SIGINT and SIGTERM and try to clean up nicely. For other signals we use the default behaviour in Go: Default behavior of signals in Go programs.

2.1.6 Hardware

Processor and Memory

Minimal requirements are:

- 2GB RAM
- 100GB of disk space
- 1.4 GHz 2v CPU

SSD disks are preferable for high transaction throughput.

Recommended:

- 4GB RAM
- 200GB SSD
- x64 2.0 GHz 4v CPU

Operating Systems

Tendermint and Minter can be compiled for a wide range of operating systems thanks to Go language. List of $OS/$ARCH pairs.

While we do not favor any operation system, more secure and stable Linux server distributions (like Centos) should be preferred over desktop operation systems (like Mac OS).

2.1.7 Configuration parameters

...