

---

# Papyrus Docs Documentation

**Papyrus**

**Jun 08, 2022**



---

## Contents:

---

<b>1</b>	<b>Vision (or Why I should run my dApp on Papyrus instead of Ethereum?)</b>	<b>3</b>
1.1	Which challenges does Papyrus Network solve? . . . . .	3
1.2	How we address these issues . . . . .	4
1.3	The main advantages of Papyrus Network . . . . .	4
<b>2</b>	<b>Governance</b>	<b>5</b>
2.1	Papyrus Network User Agreement . . . . .	5
2.2	Definitions . . . . .	5
2.3	Adding new Authority Node . . . . .	6
2.4	Authority Nodes Management . . . . .	7
2.5	1. Election . . . . .	7
2.6	2. Blacklist . . . . .	7
2.7	Changing BIOS contract parameters . . . . .	8
2.8	Authority Nodes token reward recommendation . . . . .	8
2.9	Governance attack considerations . . . . .	9
<b>3</b>	<b>Comparison of Public Blockchain Networks</b>	<b>11</b>
<b>4</b>	<b>Consensus</b>	<b>13</b>
<b>5</b>	<b>Authority Nodes</b>	<b>15</b>
5.1	<b>Recommended Authority Node configuration:</b> . . . . .	16
5.2	<b>How to deploy Authority Node</b> . . . . .	16
5.3	<b>Expected Authority Node rewards:</b> . . . . .	16
<b>6</b>	<b>Staking</b>	<b>17</b>
6.1	How to stake tokens . . . . .	17
6.2	Code Examples . . . . .	21
<b>7</b>	<b>Network Performance</b>	<b>23</b>
7.1	Results . . . . .	25
<b>8</b>	<b>Papyrus API</b>	<b>29</b>
8.1	BIOS Contract . . . . .	29
8.2	API: Staking . . . . .	30
8.3	API: Voting . . . . .	32
<b>9</b>	<b>Tools</b>	<b>35</b>

<b>10</b>	<b>FAQ</b>	<b>37</b>
10.1	What is Papyrus Network? . . . . .	37
10.2	What is the prospects of Papyrus Network? . . . . .	37
10.3	What differs Papyrus Network from Ethereum, EOS and TRON? . . . . .	37
10.4	What are Authority nodes? . . . . .	38
<b>11</b>	<b>Papyrus Network key facts</b>	<b>39</b>
<b>12</b>	<b>Work in progress</b>	<b>41</b>
<b>13</b>	<b>Indices and tables</b>	<b>43</b>

Papyrus Network is a blockchain and decentralised applications platform, based on the Papyrus protocol, which was created as refined version of Ethereum protocol with Proof-of-Authority consensus for better scalability, security, robustness and user-friendly application interfaces.

Mission of Papyrus Network is to stimulate adoption of decentralized applications in B2C and enterprise segments across the globe by resolving issues, which are slowing down their practical adoption, such as limited network throughput, security risks, volatile transactional fees, complicated user onboarding process.

Papyrus Network is compatible with Ethereum ecosystem of 3rd party software, user wallets, developer libraries and productivity tools. Ethereum dApps can be launched within Papyrus Network, achieving improved stability, scalability, speed and security, while also simplifying user onboarding process by removing barriers like calculation of gas limit and gas price, and enabling removal of transactional fees for users completely.

Major advantages of Papyrus Network are:

- **Proof-of-Authority consensus** with variable Authority Nodes count, which is energy-efficient (no PoW waste), faster (one second blocks interval and fast block finality), cheaper (no need to store thousands of data replicas and involve thousands of nodes to consensus protocol, if protocol have governance model, which deliver same or higher level of security by utilising small number of credible Authority Nodes) and more secure (small scale PoW networks are subject to 51% reorg attacks, dPoS networks are subject to the power of minority (“whales”) of large token holders), than other approaches for the public distributed ledger network consensus.
- **Token staking** as a way to allocate network resources, similar to the one in EOS network, where network resources are tokenised and application developers are able to use subscriptional model to pay for network resources by staking resource tokens. Staking enables developers to have predictable cost on their application operations, ability to have reliable models for their financials, and allows developers to provide free-of-charge transactions for their application users, so that dApp can be accessed even by those with empty wallet, if application smart contract allows it. That is a major step for building user-friendly dApp interfaces and improving user onboarding conversions.
- **Programmed decentralized governance and high network security**, based on Constitution and Network BIOS contract, which splits power between Authority Nodes and token holders, and tries to establish system of checks and balances to ensure that everyone is incentivised to behave in a best way to achieve better Papyrus Network service and wider Papyrus Network adoption, also providing instruments to tolerate attack attempts and maintain efficient self-governance. Papyrus Network sets initial governance vision and implementation and supports evolution of it by BIOS contract upgrades, which might be executed by the community according to certain rules, as likely there will be new things to address as network develops.
- **Support of Ethereum ecosystem** of developer libraries and productivity tools, wallets and Solidity smart contracts. It enables usage of Papyrus Network by the Ethereum developers community.



---

## Vision (or Why I should run my dApp on Papyrus instead of Ethereum?)

---

Papyrus Network is a new Ethereum-based public blockchain designed for ultimate mass adoption of decentralised applications. Main idea of Papyrus Network design was to make a perfect ecosystem, which removes limitations, which slow down growth of the user base in existing public blockchain networks such as Ethereum and EOS.

Papyrus Network have couple of characteristics, which makes it a great choice for application developers, as it reduces network costs, while improving at the same time network robustness, scalability, and ability to serve for user-friendly applications with easy user onboarding.

Major changes to Ethereum, which we implemented in Papyrus Network, are:

- Instead of energy intensive Proof-of-Work a lightweight **Proof-of-Authority consensus** is used, where fixed amount of network nodes called “Authority Nodes” are elected and operated by credible organizations. Papyrus Network has native system of checks and balances, which lead to better and secure governance.
- Instead of gas transaction fee model **token staking model** is implemented, where token supply represents available network throughput and each token represents fraction of it. Application developers need to stake tokens (locking them for specific period of time) to receive access to required network bandwidth and may adjust staked amount from time to time accordingly to their needs. Developers don't need to worry about token price volatility between revisions of their bandwidth requirements.

### 1.1 Which challenges does Papyrus Network solve?

Nowadays there are a lot of challenges of using public blockchains as application platforms:

- Lack of trust (network is usually controlled by anonymous elite, which control largest mining pools or use coalitions of network nodes in own interest).
- Lack of reliability (network is not protected from failures and application developer may experience large losses due to mistakes of others).
- Lack of responsibility (there is no any formal responsibility by network operators for application developers).
- Lack of support (application developers need to find solution for every problem themselves).

- Poor user experience (existing solutions are too complex and push users away by requiring them to use sophisticated wallets and plugins).
- Extreme costs and volatility (application developers need to care about cryptocurrency prices because they are used to pay network fees, they cannot build stable business model because of too high dependence on cryptocurrency market).

## 1.2 How we address these issues

By introducing elements of permissioned blockchain networks to public network setup.

- **Trust** – control of the network authority nodes belongs to decentralized consortium of credible organizations with full public disclosure of governance processes, by having multiple different organizations in the consortium network achieves trust that they won't collude and risk their reputation by attacking the network
- **Reliability** – network architecture ensures network resources allocation between registered application developers to avoid network overload
- **Responsibility** – being publicly exposed and having their business and reputation at risk authority nodes owners are accountable for their activities
- **Great user experience** – moving obligation to pay network fees from user to application developer, significantly improve user experience as now users don't need to care about keeping positive crypto wallet balance, calculating transaction fees and confirming them
- **Reasonable and predictable costs** – application developers are able to reserve necessary network bandwidth for specified period of time with reasonable upfront payment, it allows them to provision their expenses and eliminate crypto volatility impact on their business models; overall cost of ownership is low because of limited amount of authority nodes in the network

## 1.3 The main advantages of Papyrus Network

- **Compatibility with existing Ethereum ecosystem** including wallets, smart contracts, and other tools; no need to study new languages and frameworks.
- **Increased Network speed and reliability**: no more worries about application scalability and availability issues.
- **High standard of network security** provided by the authority nodes managed by credible organizations, which cannot be achieved in the environment of anonymous mining pools.
- **An Unprecedented level of user experience** — token staking model with no need to install complex plugins or browsers, pay various fees and make gas calculations; just use your application like any other — Facebook or Amazon.
- **An easy way of making money** without building complex token economies—you can simply sell subscriptions or in-app services to your application users, accepting both traditional payments and cryptocurrencies.
- **Low network resource cost and low volatility**: you can plan your infrastructure expenses in advance thanks to token staking resource allocation model.
- **Network reliability** network is protected from spam and DoS attacks by having staking procedure to consume network resources.
- **Authority node owners are all identified credible organizations** which put their reputation and business at risk in case of abuse.

There are many issues with governance paradigms used in other blockchain solutions, which lead to unacceptable level of centralization. For instance, EOS network is governed by Constitution and 21 Block Producers, which execute EOS protocol and are elected using dPoS mechanism. But voting thresholds to become elected BPs are low due to absence of quorum requirements, and power is consolidated in the hands of few people with significant token holdings and influence over BP decisions. Political systems, where power is consolidated in the hands of richest, are called plutocracy. In another dPoS network - TRON - situation is even worse, as not only BPs are elected by elite token holders with large stakes, but BPs are allowed to buy votes from token holders, and they spend their mining rewards to attract votes, instead of investments in better service and infrastructure. On the opposite, PoW networks have an issue with consolidation of power in large mining pools, controlling a significant percentage of network's hashpower and effectively controlling the network itself.

Our design goal in Papyrus Network was to construct governance protocol, which is aligned with Proof-of-Authority consensus for network scalability reasons, in a way to maintain greater level of decentralization and incentivize benefit of everyone: token holders, miners (authority nodes) and network customers (application developers and users). To achieve this goal we implemented flexible governance protocol, which somewhat reflects how corporations are managed in the modern world by shareholders and the board of directors. We believe that this is the best construction to be used as the network evolves to stimulate its growth.

## 2.1 Papyrus Network User Agreement

## 2.2 Definitions

Papyrus Network User Agreement: This document (PNUA)

Chain ID: 32328

BIOS Contract: An Papyrus Network smart contract with a dynamic permissions structure, which defines network governance procedures.

User: Any person or organization of persons who maintain(s) direct or indirect ownership of an Papyrus Network address, or property connected to an Papyrus Network address.

**Ownership:** Direct or indirect access to an Papyrus Network address through one or more valid permissions checks. Ownership may be partially shared between Users through the use of multi-signature permissions.

**Authority Nodes:** Users who have created new blocks in Papyrus Network.

**On-Chain:** Any transaction, smart contract, or Ricardian contract which is located within a block that is irreversible and appended to the Papyrus Network.

**Papyrus Network-based Property:** Anything that requires a valid permission in order to directly manipulate, alter, transfer, influence, or otherwise effect on the Papyrus Network

**Call:** To submit an action to the Papyrus Network.

**Authorizations & Permissions:** Permissions are arbitrary names used to define the requirements for a transaction sent on behalf of that permission. Permissions can be assigned for authority over specific contract actions.

**Article I - User Acknowledgement of Risks** If User loses access to their Papyrus Network address on chain\_id and has not taken appropriate measures to secure access to their Papyrus Network address by other means, the User acknowledges and agrees that that Papyrus Network address will become inaccessible. Users acknowledge that the User has an adequate understanding of the risks, usage and intricacies of cryptographic tokens and blockchain-based software. The User acknowledges and agrees that the User is using the Papyrus Network at their sole risk.

**Article II-Consent of the PNUA** The nature of the Papyrus Network User Agreement is such that it serves as a description of the current Papyrus Network Mainnet governance functions that are in place. These functions, enforced by code, do not require the consent of Users as these functions are inherent and systemic to the Papyrus Network Mainnet itself.

**Article III-Governing Documents** Current version of PNUA is referred by its SHA256 hash in Papyrus Network BIOS Contract and modifications to the PNUA may be made using BIOS Contract.

**Article IV-Native Unit of Value** The native unit of value on Papyrus Network chain\_id shall be the PPR token as defined by Papyrus Network software.

**Article V-Maintaining the Papyrus Network** Papyrus Network code is published in open GitHub repositories and open source developers community is supposed to maintain the active blockchain codebase which includes, but is not limited to, the implementation of all modifications of all features, optimizations, and upgrades: present and future.

**Article VI-No Fiduciary** No User shall have a fiduciary purpose to support the value of the PPR token. No User can authorize anyone to hold assets, borrow, speak, contract on behalf of other Papyrus Network Users or the Papyrus Network chain\_id collectively. Papyrus Network shall have no owners, managers, or fiduciaries.

**Article VII-User Security** All items pertaining to personal account security, including but not limited to the safekeeping of private keys, is solely the responsibility of the User to secure.

**Article VIII - Authority Nodes Limited Liability** The User acknowledges and agrees that, to the fullest extent permitted by any applicable law, this disclaimer of liability applies to any and all damages or injury whatsoever caused by or related to risks of, use of, or inability to use, the PPR token or the Papyrus Network under any cause of action whatsoever of any kind in any jurisdiction, including, without limitation, actions for breach of warranty, breach of contract or tort (including negligence) and that Authority Nodes shall not be liable for any indirect, incidental, special, exemplary or consequential damages, including for loss of profits, goodwill or data.

## 2.3 Adding new Authority Node

Any Authority Node can make proposal on adding new Authority Node to the network, which follows the approval process. Any Authority Node can vote for proposed Authority Node candidate, if it is not blacklisted in either authority nodes blacklist or stakeholders blacklist. In total each Authority Node can vote for a maximum of 7 other Authority Nodes or candidates. Authority Node can move their votes from one Node to another at any time. Candidate Node becomes Authority Node if it keeps minimum 3 votes and holds in the top 47 Candidate and Authority Nodes by received votes number for continuous 7 days. Only votes from Authority Nodes are counted.

## 2.4 Authority Nodes Management

### 2.5 1. Election

New Authority Nodes are elected by existing Authority Nodes.

Authority Node candidate can be proposed by any of existing Authority Nodes. After Authority Node candidate is proposed, voting for the Authority Node candidate begins and last 14 days.

Any Authority Nodes is able to vote for other Authority Nodes and Authority Node candidates. The following constraints are applied in network BIOS contract:

- Authority Node have maximum of 7 votes to be casted for different Authority Nodes
- Authority Node can't cast more than 1 vote for the same Node
- Authority Node can cast one vote for itself
- Authority Node can withdraw a vote from any node with immediate effect
- Withdrawed votes can be casted again only after 14 days vote cooldown period

After 14 days of voting for a new candidate Authority Node, decision is made based on received votes. If by the end of voting period candidate received minimum of 3 votes from Authority Nodes AND is not added to the blacklist (see below), than candidate becomes Authority Node.

*Additional rule to be implemented by upgrading BIOS contract in the near time:*

If by the end of voting period candidate fits with ALL of the following:

- received minimum of 3 votes from Authority Nodes
- is not added to the blacklist (see below)
- current number of Authority Nodes < 47 OR there is an Authority Node, which have less received votes than candidate node

Than candidate is promoted to Authority Node. If number of Authority Nodes = 47, than simultaneously existing Authority Node with lowest amount of received votes is excluded from Authority Nodes. This logic ensures that maximum number of Authority Nodes is limited with 47.

\*\*\*

Otherwise, candidate node is rejected and not promoted to Authority Node.

### 2.6 2. Blacklist

Authority Node and candidate for Authority Node can be blacklisted by existing Authority Nodes.

To add candidate or Authority Node to the blacklist, any Authority Node can create blacklist proposal and initiate proposal voting. Voting period for blacklist proposal is 5 days, which enables ability of blacklisting for Authority Node candidates before candidate voting period of 14 days ends. Only Authority Nodes can cast votes in the blacklist voting. Each Authority Node can cast 1 vote for the proposal.

After 5 days of blacklist proposal voting, proposal is deemed successful, if:

- amount of votes for proposal is > 50% of Authority Nodes count

Otherwise, proposal is rejected.

If proposal is successful, Authority Node or Authority Node candidate is added to the blacklist. Any node included into the blacklist can't be Authority Node.

*Community blacklist to be implemented by upgrading BIOS contract in the near time:*

Authority Node and candidate for Authority Node can be blacklisted by owners of PPR token stakes. Blacklist formed based on PPR token stake holders voting is called community blacklist.

To add candidate or Authority Node to the blacklist, any owner of PPR token stake can create community blacklist proposal and initiate proposal voting by staking minimum amount of PPR tokens towards the proposal (minimum is to be determined).

Voting period for community blacklist proposal is 5 days, which enables ability of blacklisting for Authority Node candidates before candidate voting period of 14 days ends. Only owners of PPR token stakes can cast votes in the community blacklist voting. Each owner of PPR token stake can cast vote proportional to their token stake. In case of voting for blacklist proposal stake withdrawal period for stake owner is increased to 5 days starting at the time of voting, so he can never vote twice in the same voting using the same stake. Each vote for community blacklist can be either positive (for) or negative (against).

After 5 days of blacklist proposal voting, proposal is deemed successful, if: - amount of positive votes is bigger, than amount of negative votes - total amount of votes is  $> 10\%$  of total token stake owner votes possible in the network based on existing network-wide amount of token stakes (quorum)

If proposal is successful, Authority Node or Authority Node candidate is added to the community blacklist. Any node included into the community blacklist can't be Authority Node.

\*\*\*

## 2.7 Changing BIOS contract parameters

BIOS contract refers to Papyrus Network User Agreement using its SHA-256 hash code, linking network operations with the agreement.

Parameters such as maximum amount of Authority Nodes or mining rewards are configured in BIOS contract as well.

Upon network launch Papyrus have ownership rights on BIOS contract and can override / reconfigure it in case of network issues. In the future Papyrus will surrender ownership of BIOS contract so that no party will be controlling it.

To achieve decentralised governance, BIOS contract may be upgraded by supermajority decision of Authority Nodes, which is not objected by voting of the community of token stake owners.

Implementation of this voting will be deployed to BIOS contract in the near time.

## 2.8 Authority Nodes token reward recommendation

To incentivize Authority Nodes participation, they shall receive token rewards for each block, which they include in the blockchain. With 1 seconds block interval it is recommended to set block reward at  $1.5 * K$  PPR tokens per block, where  $K = \{AMOUNT\ OF\ AUTHORITY\ NODES\} / 47$ . It will keep annual inflation of PPR token supply under 5% for the network with 47 Authority Nodes, and it will avoid Authority Node reward dilution due to new nodes joining the network. As rewards aren't diluted, Authority Nodes will be incentivized to propose new nodes inclusion to increase trust and adoption of the network, influencing token value.

## 2.9 Governance attack considerations

Network governance and resistance to attacks is considered sufficient, assuming that >50% of Authority Nodes are controlled by honest owners at all times. When amount of Authority Nodes in the network is between 5 and 47, three or more nodes can collude to include more their allies as nodes into the network with the idea of eventually getting control over 50%+ Authority Nodes and performing network attack. Assuming that honest Nodes represent at least 50% of the Authority Nodes at the moment of attack preparation suspicion, they shall blacklist proposed node candidates to tolerate potential attack. In case of very unlikely situation, where network attack such as double spending is made by attackers, which managed to get control over more than 50% of Authority Nodes, token stake owners together with honest Authority Nodes can make hard fork of the blockchain and use media to distribute incident information and guides on necessary updates for network customers.





Comparison of Public Blockchain Networks

	Ethereum	EOS	TRON	Papyrus Network
Energy consumption	19.07 TWh / year (0.09% of world's electricity consumption)	Negligible	Negligible	Negligible
TPS	15	3900	750	1500
Block Time	12s	0.5s	3s	3s
Consensus	Proof-of-Work	Delegated Proof-of-Stake	Delegated Proof-of-Stake	Proof-of-Authority
Resource Management	Gas	Token staking	Gas and Token Staking	Token staking
Average tx fee	~ \$0.13 USD per average transaction	~ \$0.1 USD to reserve CPU for average transaction; ~ \$0.159 USD - Cost to persist 1 Kb of data in RAM; ~ \$0.1 USD/ms/Day - Cost to reserve 1 ms/Day CPU Bandwidth; ~ \$0.001 USD/Kb/Day - Cost to reserve 1 Kb/Day Network Bandwidth	~ \$0.7 USD to reserve bandwidth for average transaction; ~ \$0.14 USD/ms/Day - Cost to reserve 1 ms/Day CPU Bandwidth; ~ \$3.5 USD/Kb/Day - Cost to reserve 1 Kb/Day Network Bandwidth	< 0.001 USD
Smart Contracts	EVM/Solidity	WASM	TVM/Solidity	EVM/Solidity
Usage complexity	Very difficult for both app developers and users	Token staking significantly improves app developers and user experience	Token staking significantly improves app developers and user experience	Token staking significantly improves app developers and user experience
Developer community	Strongest community globally, estimated as 250 000 developers by ConsenSys	Growth stage, but much smaller than in Ethereum	Potentially equal to Ethereum community, but TRON have many differences and require more work for Ethereum applications migration	Equal to Ethereum community, as network is fully compatible with Ethereum applications
Network availability	Not guaranteed, network clogging can paralyze all applications	Ensured by having active and reserve block producers	Ensured by having active and reserve super representatives	Ensured by having Authority Node eligibility criteria

Papyrus Network is a distributed computing system, where network nodes has to agree on transactions, before they are included in the distributed ledger, to maintain synchronized copies of the ledger and avoid inappropriate transactions. How network nodes reach the agreement is determined by consensus protocol, deisgned with the goal to maintain system reliability in the presence of a number of faulty processes.

Assumptions used in Papyrus Network design

- Cost of network protocol manipulation should exceed potential benefit for an attacker;
- If total amount of value stored in the network is X, than cost of an attack on the network should be  $>X$ ;
- Economy processes within the network is complemented with economy processes outside the network:
- poor behaviour of network node owned by some business entity, may be covered in the media and cause reputational damage to the entity's other businesses, leading to its financial losses;
- attack on the network by a group of network nodes will cause their removal from the network and possible hard fork, meaning that they will lose ability to receive mining rewards in the network;
- There are governance rules, initially established as constitutional for the network, which require agreement not only between network nodes, but also between token holders, to change protocol;
- Technically collusion/agreement between minimum of  $>50\%$  of active network nodes should be required to manipulate/change the protocol, but it is not enough to run a successful attack, as it will be detected and considered as constitution violation, if change was not approved by token holders according to established policy.

Keeping in mind these assumptions we developed a Proof-of-Authority consensus for Papyrus Network, which have the following features:

- Network nodes allowed to participate in consensus protocol for network transactions confirmation are called Authority Nodes;
- Amount of Authority Nodes is allowed to be in a range of 5 to 47 nodes, upper limit of 47 is chosen as reasonable to avoid excessive infrastructure cost, while keeping network security at high level;
- We use the same logic as in EIP225 ( <https://eips.ethereum.org/EIPS/eip-225> ) Clique consensus to reach an agreement between Authority Nodes, but change the voting logic to elect Authority Nodes (see [http://docs.papyrus.network/en/latest/doc/network\\_architecture.html](http://docs.papyrus.network/en/latest/doc/network_architecture.html) for more details).



---

## Authority Nodes

---

Papyrus Network is operated by Authority Nodes, which provide their hardware capacities to run Papyrus Network software, which implements Papyrus Network protocol. Authority Nodes are operated by identified business entities and are elected by other Authority Nodes according to established network governance model (see [http://docs.papyrus.network/en/latest/doc/network\\_architecture.html](http://docs.papyrus.network/en/latest/doc/network_architecture.html) for more details).

Papyrus Network protocol allows up to 47 Authority Nodes to be a part of the network.

In exchange for maintaining network operations Authority Nodes get mining rewards equal to  $1.5 \cdot K$  PPR tokens per block, where  $K = \{\text{AMOUNT OF AUTHORITY NODES}\} / 47$ . This formula ensures that existing Authority Nodes will keep receiving same amount of daily/monthly rewards with the growth of amount of Authority Nodes in the network. In fact, Authority Nodes are incentivised to make network more decentralised and invite new nodes to the network without losing own rewards.

PPR token is a native token of Papyrus Network required to allocate network resources to execute transactions. There are 1 000 000 000 PPR tokens distributed at the genesis block, and according to mining rewards schedule with 1 second block interval annual inflation is limited with a maximum amount of new tokens equal to  $(60 \text{ blocks per minute}) \cdot 60 \cdot 24 \cdot 365 \cdot 1.5 \cdot (47/47) = 47\,304\,000$  PPR, ie ~ 4.7% of total emission.

Mining rewards are subject to change only based on community voting (see Governance documentation).

Initially Papyrus Network mainnet is launched with 5 Authority Nodes chosen by Papyrus, few of them will be initially operated by Papyrus.

After the launch Authority Nodes election procedures are activated, network governance becomes decentralised and existing Authority Nodes are able to elect new Authority Nodes. To make proper decisions as Authority Node owner Papyrus will audit each current and potential node owner entity's mission, executive leadership, net annual revenue, number of full-time employees, years in business, organizational structure, and the industry they are a part of, to ensure that existing network Authority Nodes are reliable enough.

Following network launch Papyrus team will be pursuing full network decentralisation, where all Authority Nodes will be operated by other organisations without Papyrus involvement. To achieve that Papyrus will be removing own nodes from the network, when new Authority Nodes are joining it.

It is strongly recommended in the interest of network reliability that all existing Authority Nodes do a regular audit of current and potential Authority Node owner entity's mission, executive leadership, net annual revenue, number of full-time employees, years in business, organizational structure, and the industry they are a part of. In case of doubt in

other Authority Node reliability, Authority Nodes should use BIOS Contract to set a vote on adding suspicious Nodes to the Blacklist. Papyrus Wallet and Papyrus Explorer support necessary functionality.

If you would like to run an Authority Node and receive mining rewards in Papyrus Network, you can apply here: <https://papyrusglobal.typeform.com/to/Va9wX5>

Instruction on how to set up a node can be found here: <https://github.com/papyrusglobal/papyrus>

## 5.1 Recommended Authority Node configuration:

CPU - Intel® Core™ i7 or more powerful

RAM: 64Gb+

Hard drive: 512Gb+ (SSD recommended)

Connection: 100 MBit/s+ port

## 5.2 How to deploy Authority Node

Prerequisites You need have Docker and Docker Compose installed.

To start a node, run the following docker command. It will download the image and start it. Optionally, you may add keys such as `-rpc` or `-ws` (see the 'geth' command line options) to the end of the command.

```
docker run -d --name=my-node -p 33309:33309 -p 33309:33309/udp -v my-node-volume:/
↪root/.ethereum --restart unless-stopped papyrusglobal/geth-papyrus:latest --port_
↪33309 --ethstats='my-node-public-name:ante litteram@status-server.papyrus.
↪network:3800'
```

Additionally, if you are authority node: You will then need to copy your `<account.json>` to the container where node runs.

```
docker cp account.json my-node:/root/.ethereum/keystore/
docker exec -it my-node ./console.sh 'personal.unlockAccount(eth.accounts[0], "<<
↪<passphrase>>", 0) '
docker exec -it my-node ./console.sh 'miner.setEtherbase(eth.accounts[0]); miner.
↪start()'
```

## 5.3 Expected Authority Node rewards:

Expected monthly PPR reward for active Authority Node is equal to  $60*60*24*30*(K/47)*(1.5/K) = \sim 82,723$  PPR

### Why we implemented token staking in Papyrus Network?

Token staking is the method to allocate network bandwidth to specific user or application developer needs. We use idea of tokenisation to manage network resources and avoid network flood and sybil attacks on the network.

There is a fixed amount of network bandwidth tokens issued at every point in time (every block formed in the network). Imagine that to get right to use  $X\%$  of network bandwidth you have to freeze  $X\%$  of issued tokens, making them unavailable to transfers and any other usage. You can always unfreeze them by requesting that, but you will have to wait 3 days and will lose ability to use  $X\%$  of network bandwidth. That is the simple illustration, how network bandwidth may be tokenised with fixed amount of tokens.

In Papyrus Network native token is called PPR and it is used for network bandwidth tokenisation. Rules around token staking/unstaking and resource allocation are slightly more complex in the real Papyrus Network and are described below. Also Papyrus Network supports delegation of allocated bandwidth, meaning that application developer can make own stake to allocate network bandwidth for his application and then allow to use it for free for their smart contract users.

If existing stake provides bandwidth, that is not enough to run required transactions for the user or smart contract, it will result in transaction failure. Still transaction can be retried again, as bandwidth allocation for the stake becomes refilled with the time. Application developers and users should plan their bandwidth allocation in advance and increase/decrease their stakes as necessary.

Ultimately stake-based resource management allows application developers to plan network resources cost in advance and build reliable business models for their dApps. As an example, dApp developer could charge users off-chain (via Google Play or Apple App Store) and put them into on-chain whitelist to execute smart contract functions for free using developer's stake, thus significantly simplifying their onboarding to dApp.

## 6.1 How to stake tokens

The easiest way to stake/unstake tokens is to use [Papyrus Wallet](#). Papyrus Wallet works with MetaMask extension and gives you a particularly user-friendly interface for all features of Papyrus Network. To stake your tokens just chose staking tab and then fill the fields with stakes or gaslimit and press Stakes button. By default - stake will be processed

onto your address. To stake on smartcontract - just fill its address to the field and push stake button! Remember that only one stake is available to the one smartcontract. Main Papyrus Wallet interface you could see on the screenshot:



STAKING

VOTING



0x4a2D0F1EBb52E130d4C5ebF9d59563E0F7129734

0 PPR-wei

My stake	My limit	All stakes
0 PPR-wei	0 gas	0 PPR-wei

Stake      Unstake

PPR-wei at stake: 100      Gas you receive: 18168576307200

It will **increase** your gas limit at **18168576307200 gas** per 3 days

To address: 0x4a2D0F1EBb52E130d4C5ebF9d59563E0F7129734

STAKE

You can easily unstake your tokens, just go to the Unstake tab and enter value for unstaking. When unstaking is done your tokens are frozen and you have to wait for some time to process with withdraw.

**PAPYRUS**  
NETWORK

**STAKING** **VOTING**



**0x4a2D0F1EBb52E130d4C5ebF9d59563E0F7129734**

0 PPR-wei

My stake	My limit	All stakes
0 PPR-wei	0 gas	0 PPR-wei

Stake **Unstake**

PPR-wei to unstake Gas you lose

To address  
0x4a2D0F1EBb52E130d4C5ebF9d59563E0F7129734

**UNSTAKE**

## 6.2 Code Examples

Examples - how to use stakes in your DApp you could find in the API section



---

## Network Performance

---

Papyrus network designed as high loaded solution that should be able to process more than 1000 tps. To achieve these performance results we reimplement batching and buffers to allow them to aggregate more data. To do that first of all we had to increase potential number of transaction in queue.

- `peer.go` class that contains constants that are responsible for queue sizes.

We conducted a series of load tests and, after internal modeling, we stopped at the following set of values:

```

// maxQueuedTxs is the maximum number of transaction lists to queue up before
// dropping broadcasts. This is a sensitive number as a transaction list might
// contain a single transaction, or thousands.
maxQueuedTxs = /*128*/ 16384

// maxQueuedProps is the maximum number of block propagations to queue up before
// dropping broadcasts. There's not much point in queueing stale blocks, so a
↪few
// that might cover uncles should be enough.
maxQueuedProps = /*4*/ 32

// maxQueuedAnns is the maximum number of block announcements to queue up before
// dropping broadcasts. Similarly to block propagations, there's no point to
↪queue
// above some healthy uncle limit, so use that.
maxQueuedAnns = /*4*/ 32

handshakeTimeout = 5 * time.Second
)

```

Next step was in increasing up to ten times size of transaction chain. For that reason few more classes were tuned by changing its constant values

- `sync.go` - here where have overridden result size of our transactions pack.

```
// This is the target size for the packs of transactions sent by txsyncLoop.
// A pack can get larger than this if a single transactions exceeds this size.
txsyncPackSize = /*100 * 1024*/ 1000 * 1024
```

In the next three classes - ethstats.go, server.go, worker.go we need to increase the size of chain head size.

- ethstats.go

```
// The number is referenced from the size of tx pool.
txChanSize = 4096
// chainHeadChanSize is the size of channel listening to ChainHeadEvent.
chainHeadChanSize = /*10*/ 100
```

- server.go

```
func (pm *ProtocolManager) blockLoop() {
    pm.wg.Add(1)
    headCh := make(chan core.ChainHeadEvent, /*10*/ 100)
    headSub := pm.blockchain.SubscribeChainHeadEvent(headCh)
```

- worker.go

```
txChanSize = 4096

// chainHeadChanSize is the size of channel listening to ChainHeadEvent.
chainHeadChanSize = /*10*/ 100
```

- tx\_pool.go class that contains most of logic for the transaction pool. As in previous classes - we had to override chain size and after that, according to our model, we significantly reworked the sizes of the slots

```
1 DefaultTxPoolConfig = TxPoolConfig{
2     PriceLimit: 1,
3     PriceBump: 10,
4
5     AccountSlots: /*16*/ 8192,
6     GlobalSlots: /*4096*/ 131072,
7     AccountQueue: /*64*/ 4096,
8     GlobalQueue: /*1024*/ 32768,
9 }
```

After that we added transaction batching and overhauled buffers. This kind of transaction packaging lets us to drastically increase network performance without compromising security. In order to unlock the full potential of this approach, we also had to rework the queue sizes for pending and queued transactions. While these changes implied new changes in parameters of the maximum number of permissible transactions, and, most importantly, in the total number of transactions.

- tx\_pool.go

```
// feedLoop continuously sends batches of txs from the txFeedBuf to the txFeed.
func (pool *TxPool) feedLoop() {
    defer pool.wg.Done()

    const batchSize = 1000
    for {
        select {
            case <-pool.chainHeadSub.Err():
```

(continues on next page)

(continued from previous page)

```

        return
    case tx := <-pool.txFeedBuf:
        var event NewTxEvent
        event.Txs = append(event.Txs, tx)
        for i := 1; i < batchSize; i++ {
            select {
                case tx := <-pool.txFeedBuf:
                    event.Txs = append(event.Txs, tx)
                default:
                    break
            }
        }
        pool.txFeed.Send(event)
    }
}

// feedSend queues tx to eventually be sent on the txFeed.
func (pool *TxPool) feedSend(tx *types.Transaction) {
    select {
        case pool.txFeedBuf <- tx:
            return
        default:
            go func() { pool.txFeedBuf <- tx }()
    }
}

```

The result is a tenfold increase in performance. For multiple tests, we used a specific utility to load the network with 1500 transactions every second. The test results showed that the network successfully handles 1500 transactions per second and works stably at such a load for a long period of time. Below you can find the logs of the utility and the logs of the node. The logs show that all 1500 transactions fall into a block, which is generated every second. During the test, we used the type of configuration, suggesting the entire load to be applied to one gateway node, while the gateway-node is not engaged in the generation of blocks. Blocks are generated by several sealer nodes.

## 7.1 Results

Utility logs:

```

1. aholodov@Ubuntu-1804-bionic-64-minimal: ~/load (ssh)
...mat: ~/load (bash) 3E1  ...minimal: ~/load (ssh) 3E2  ...at: ~/restart (ssh) 3E3
2019/01/11 19:01:28 start sender loop
2019/01/11 19:01:29 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:30 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:31 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:32 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:33 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:34 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:35 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:36 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:37 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:38 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:39 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:40 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:41 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:42 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:43 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:44 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:45 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:46 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:47 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:48 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:49 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:50 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:51 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:52 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:53 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:54 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:55 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:56 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:57 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:58 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:01:58 Report - total      dur=30s txs=43500 errs=0 tps=1450
2019/01/11 19:01:58 Report - recent    dur=30s txs=43500 errs=0 tps=1450
2019/01/11 19:01:58 Report - latest    dur=30s txs=43500 errs=0 tps=1450
2019/01/11 19:01:59 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:02:00 send bathc signal for accounts: batchSize: 1500
2019/01/11 19:02:01 send bathc signal for accounts: batchSize: 1500

```

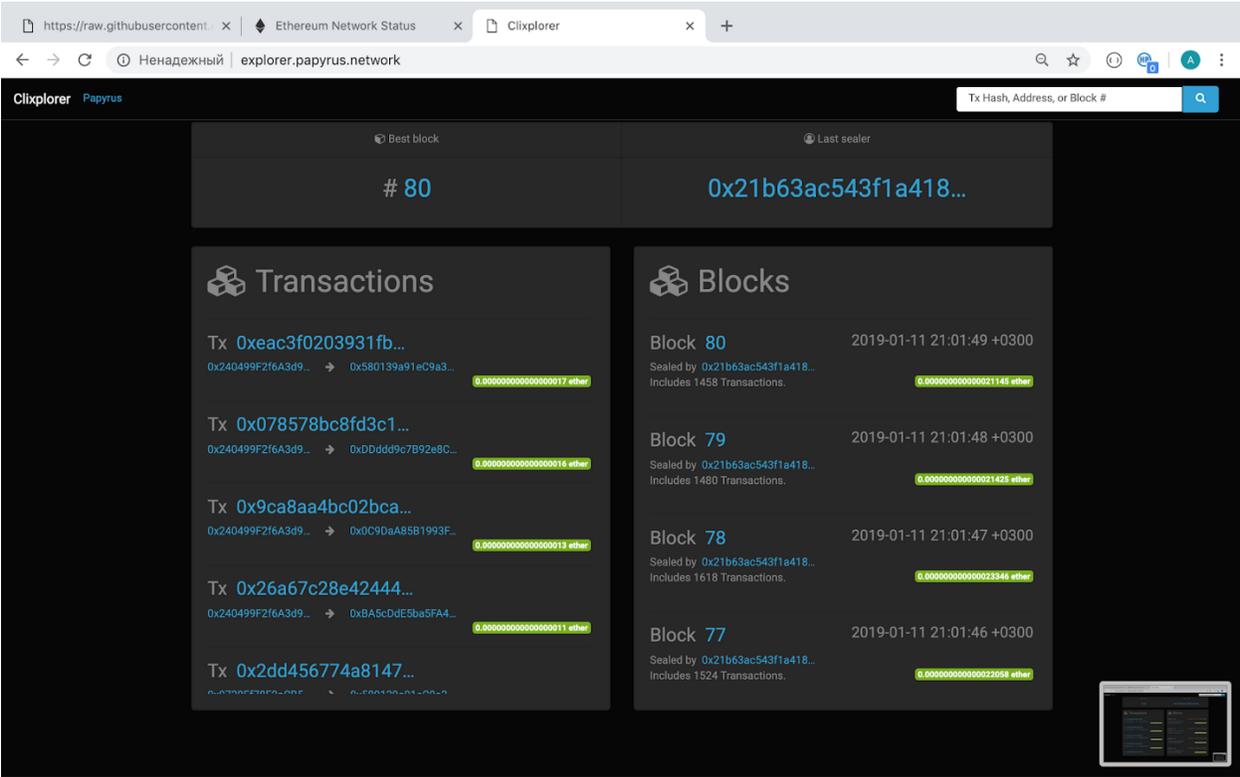
Node logs:

```

...mat: ~/load (bash) 3E1  ...minimal: ~/load (ssh) 3E2  ...at: ~/restart (ssh) 3E3
INFO [01-11-18:01:42.016] ⚡ mined potential block      number=73 hash=55dc96_f0ec96
INFO [01-11-18:01:42.026] Commit new mining work      number=74 sealhash=60b4b6_83d6fe uncles=0 txs=0 gas=0 fees=0 elapsed=5.615ms
INFO [01-11-18:01:42.148] Commit new mining work      number=74 sealhash=3d3825_82aa7d uncles=0 txs=1462 gas=30702000 fees=3.76278e-10 elapsed=127.557ms
INFO [01-11-18:01:43.012] Successfully sealed new block number=74 sealhash=3d3825_82aa7d hash=4bccc2_759473 elapsed=864.475ms
INFO [01-11-18:01:43.012] ⚡ block reached canonical chain number=67 hash=85c3d3_88abac
INFO [01-11-18:01:43.012] ⚡ mined potential block      number=74 hash=4bccc2_759473
INFO [01-11-18:01:43.017] Commit new mining work      number=75 sealhash=7f2780_11353f uncles=0 txs=0 gas=0 fees=0 elapsed=4.446ms
INFO [01-11-18:01:43.082] Commit new mining work      number=75 sealhash=15919_a2c3e2 uncles=0 txs=1414 gas=29694000 fees=3.62271e-10 elapsed=69.717ms
INFO [01-11-18:01:44.012] Successfully sealed new block number=75 sealhash=15919_a2c3e2 hash=82e0fd_32bb39 elapsed=931.241ms
INFO [01-11-18:01:44.012] ⚡ block reached canonical chain number=68 hash=1433e0_af460a
INFO [01-11-18:01:44.013] ⚡ mined potential block      number=75 hash=82e0fd_32bb39
INFO [01-11-18:01:44.023] Commit new mining work      number=76 sealhash=1a8c25_bd78c6 uncles=0 txs=0 gas=0 fees=0 elapsed=4.947ms
INFO [01-11-18:01:44.100] Commit new mining work      number=76 sealhash=70bb5_86b954 uncles=0 txs=1457 gas=30597000 fees=3.72792e-10 elapsed=82.250ms
INFO [01-11-18:01:45.017] Successfully sealed new block number=76 sealhash=70bb5_86b954 hash=6cf92c_91d894 elapsed=917.368ms
INFO [01-11-18:01:45.017] ⚡ block reached canonical chain number=69 hash=2198a0_5c2b515
INFO [01-11-18:01:45.017] ⚡ mined potential block      number=76 hash=6cf92c_91d894
INFO [01-11-18:01:45.029] Commit new mining work      number=77 sealhash=1731aa_50e2b0 uncles=0 txs=0 gas=0 fees=0 elapsed=8.369ms
INFO [01-11-18:01:45.152] Commit new mining work      number=77 sealhash=0b45e0_aa233c uncles=0 txs=1524 gas=32004000 fees=3.92427e-10 elapsed=131.503ms
INFO [01-11-18:01:46.011] Successfully sealed new block number=77 sealhash=0b45e0_aa233c hash=d71d9e_f26122 elapsed=859.951ms
INFO [01-11-18:01:46.011] ⚡ block reached canonical chain number=70 hash=768e50_981a0b
INFO [01-11-18:01:46.011] ⚡ mined potential block      number=77 hash=d71d9e_f26122
INFO [01-11-18:01:46.021] Commit new mining work      number=78 sealhash=495f45_ae006a uncles=0 txs=0 gas=0 fees=0 elapsed=5.291ms
INFO [01-11-18:01:46.122] Commit new mining work      number=78 sealhash=e93c90_641341 uncles=0 txs=1618 gas=33970000 fees=4.10445e-10 elapsed=105.899ms
INFO [01-11-18:01:47.012] Successfully sealed new block number=78 sealhash=e93c90_641341 hash=000f6f_cb2a69 elapsed=890.809ms
INFO [01-11-18:01:47.012] ⚡ block reached canonical chain number=71 hash=08683c_905096
INFO [01-11-18:01:47.012] ⚡ mined potential block      number=78 hash=000f6f_cb2a69
INFO [01-11-18:01:47.136] Commit new mining work      number=79 sealhash=b667c3_ee5234 uncles=0 txs=0 gas=0 fees=0 elapsed=5.466ms
INFO [01-11-18:01:48.011] Successfully sealed new block number=79 sealhash=d6b816_29f068 hash=272694_dd3d54 elapsed=876.654ms
INFO [01-11-18:01:48.011] ⚡ block reached canonical chain number=72 hash=2ad32e_803996
INFO [01-11-18:01:48.011] ⚡ mined potential block      number=79 hash=272694_dd3d54
INFO [01-11-18:01:48.017] Commit new mining work      number=80 sealhash=21f6d1_db04ba uncles=0 txs=0 gas=0 fees=0 elapsed=5.994ms
INFO [01-11-18:01:48.158] Commit new mining work      number=80 sealhash=730ac8_5efc1c uncles=0 txs=1458 gas=30618000 fees=3.72267e-10 elapsed=146.238ms
INFO [01-11-18:01:49.014] Successfully sealed new block number=80 sealhash=730ac8_5efc1c hash=53e571_b70c58 elapsed=856.990ms
INFO [01-11-18:01:49.014] ⚡ block reached canonical chain number=73 hash=55dc96_f0ec96
INFO [01-11-18:01:49.014] ⚡ mined potential block      number=80 hash=53e571_b70c58
INFO [01-11-18:01:49.019] Commit new mining work      number=81 sealhash=41ef27_9a7f9d uncles=0 txs=0 gas=0 fees=0 elapsed=5.808ms
INFO [01-11-18:01:49.114] Commit new mining work      number=81 sealhash=8e052d_9d2213 uncles=0 txs=1484 gas=31164000 fees=3.8262e-10 elapsed=99.718ms
INFO [01-11-18:01:50.014] Successfully sealed new block number=81 sealhash=8e052d_9d2213 hash=1f3605_7ce0b3 elapsed=901.199ms
INFO [01-11-18:01:50.014] ⚡ block reached canonical chain number=74 hash=4bccc2_759473
INFO [01-11-18:01:50.014] ⚡ mined potential block      number=81 hash=1f3605_7ce0b3
INFO [01-11-18:01:50.026] Commit new mining work      number=82 sealhash=11bd72_a4f09d uncles=0 txs=0 gas=0 fees=0 elapsed=5.023ms
INFO [01-11-18:01:50.155] Commit new mining work      number=82 sealhash=58c813_d31e8d uncles=0 txs=1568 gas=32920000 fees=4.06308e-10 elapsed=133.513ms
WARN [01-11-18:01:50.916] Invalid stats history request rsg=false
INFO [01-11-18:01:51.015] Successfully sealed new block number=82 sealhash=58c813_d31e8d hash=b9611e_3f3468 elapsed=861.489ms

```

A visual representation, which can be seen on our monitor explorer (screenshot):



The test shows, that 1500 transactions fall into a block every second it is generated.

As a result, we got the desired and unique combination of a quality network.



The following are the core APIs that could be used in the development of Papyrus Dapps:

## 8.1 BIOS Contract

### 8.1.1 Overview

`Bios.sol` - is the main kernel smartcontract with all the logic about consensus, staking and voting. Its based on the `QueueHelper` that brings queue implementation code. Latest version of the Bios contact you can always find in our [repo](#)

### 8.1.2 BIOS Addresses

To track the current address of the Bios contract, there is a `Versioner` contract located at fixed address `0x0000000000000000000000000000000000000022`. To query it for the Bios contract address, use `bios` public method. Here is an example in javascript:

```
const versionerAbi = [
  {
    constant: true,
    inputs: [],
    name: 'bios',
    outputs: [{ name: '', type: 'address' }],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  }
];
const versionerAddress = '0x0000000000000000000000000000000000000022';
const versioner = new web3.eth.Contract(versionerAbi, versionerAddress);
const biosAddress = await versioner.methods.bios().call({ from: account });
```

Note that the resulting address may be zero, this means that the Bios contract is not yet installed.

### 8.1.3 BIOS Usages examples

Let's take a look at the simple example of Javascript code that will get all authorities nodes from our BIOS contract.

```
const { eth } = require('./web3relay');
const ABI = require('./abi/bios');
const { getConfig } = require('./utils');

const config = getConfig();
if (!config.biosAddress) throw new Error('Setup config.biosAddress');
const contract = new eth.Contract(ABI, config.biosAddress);

module.exports = function (req, res) {
  if (typeof contract.methods.getAuthorities !== 'function') {
    console.error('Contract method \'getAuthorities\' not found', err);
    res.send([]);
    res.end();
  }
  contract.methods.getAuthorities().call()
    .then(authorities => {
      res.send(authorities);
      res.end();
    })
    .catch(err => {
      console.error('Can\'t get authorities from contract. Silently return empty_
→array', err);
      res.send([]);
      res.end();
    })
};
```

If you are interested to see more examples how to call BIOS contract API you could check our API documentation sections for example - [Voting API](#)

## 8.2 API: Staking

### 8.2.1 Contract overview

Bios.sol - is the main kernel smartcontract with all the logic about consensus and staking. Its based on the QueueHelper that brings queue implementation code.

### 8.2.2 Functions

- **function freeze() payable public**

Stake the specified amount of tokens. The value is on the contract account and thus inaccessible to the sender. Input parameter: "msg.value" the value to be staked. Output parameter: none

- **function freezeForContract(address contract\_) payable public**

Stake the specified amount of money to the given contract account. The value is on the contract account and thus inaccessible to the sender. (address `contract_`) payable public { Input parameter: "msg.value" the value to be staked. Input parameter: address - the contract to stake for. Output parameter: none

- **function melt(uint224 val) public**

Unstake the specified value of tokens. The value is put to the melting queue and can be withdrawn after `freezeGap`. Input parameter: "val" - value to unstake. Output parameter: none

- **function meltFromContract(address contract\_, uint224 val) public**

Function to unstake the specified value of money from the contract account. The value is put to the melting queue and can be withdrawn after `kFreezeStake`. Input parameter: "val" - value to unstake. Input parameter: address - the contract to unstake from. Output parameter: none

- **function withdraw() public**

Withdraw the previously unstaked amount of tokens provided the `freezeGap` time had passed since its unstake. Every 'unstake' call must match 'withdraw' call. Takes the latest melting queue element and transfers its tokens amount to the sender's account. Input parameter: none Output parameter: none

- **function getFreeMeltingSlots() view public returns (uint8)**

Service function, calculates the number of queue elements (slots) is available in the melting conveyer for the sender's account. Every unstake call consumes a slot, every withdrawal releases it. Input parameter: none Output parameter: uint8 - slots number

- **function getMeltingHead() view public returns (uint224 stake, uint32 timestamp)**

Service function, calculates the latest melting conveyer slot to be withdrawn first. Return Stake and timestamp pair, where stake is the amount of money unstaked and timestamp is the time of the unstake call. Input parameter: none Output parameter: uint8 - stake Output parameter: uint8 - timestamp

- **function getFreeMeltingSlots() view public returns (uint8)**

Service function, calculates the number of queue elements (slots) is available in the melting conveyer for the sender's account. Every unstake call consumes a slot, every withdrawal releases it. Input parameter: none Output parameter: uint8 - number of free slots that could be used for the unstaking

- **function getMeltingHead() view public returns (uint224 stake, uint32 timestamp)**

Service function, calculates the latest melting conveyer slot to be withdrawn first. Input parameter: none Output parameter: uint224 stake, uint32 timestamp - pair, where stake is the amount of money unstaked and timestamp is the time of the unstake call.

### 8.2.3 API Usage Example

Below you can see typical JS example of usage `Bios.sol` smartcontract.

```
const gatewayUrl = 'http://148.251.152.112:18545/'; // url to the Papyrus testnet
const biosAddress = '0x142ac51e2b05a107c1482f4832b73c5bc55b6fd5'; // Address of the
↳Bios contract in the network

const ether = 10 ** 18;
let contract;
let account;
let web3;

web3 = new Web3(window.web3.currentProvider);
const accounts = await web3.eth.getAccounts();
```

(continues on next page)

```
account = accounts[0];
const netId = await web3.eth.net.getId();
const balance = await web3.eth.getBalance(account);
contract = new web3.eth.Contract(abi, biosAddress);

//lets freeze some tokens
contract.methods.freeze().send({ from: account, gas: 0, value })

//lets unfreeze some tokens
contract.methods.melt(value).send({ from: account, gas: 0 })

//After freeze gap time - lets unfreeze our tokens
contract.methods.withdraw().send({ from: account, gas: 100000 })
```

## 8.3 API: Voting

### 8.3.1 Contract overview

Bios.sol - is the main kernel smartcontract with all the logic about consensus, staking and voting. Its based on the QueueHelper that brings queue implementation code.

Voting mechanism consists of two parts - first is "proposal call" that initiates possibility for voting for any change for other participants (for instance - adding new Authority node or Bios contract changing etc). After that voting begins and in the in a certain period of time Authority Nodes could vote for any active initiatives.

### 8.3.2 Proposal Functions

- **function proposeNewAuthority(address participant) public**

Propose a poll for a new authority. Input parameter: "address participant" - the address of the new Authority Node. Output parameter: none

- **function proposeBlacklistAuthority(address participant) public**

Propose a poll for blacklisting the authority to the authority black list. Input parameter: "address participant" - the address of the Authority Node that should be added to the black list. Output parameter: none

### 8.3.3 Proposal Examples

Below you can see JS example of usage Bios.sol smartcontract from the Papyrus Wallet implementation:

```
async proposeNewAuthority(address, callbacks = {}) {
  return this.process(
    this.contract.methods.proposeNewAuthority(address).send({
      from: this.account,
      gas: 100000
    }),
    callbacks
  );
}
```

### 8.3.4 Voting Functions

- **function voteForNewAuthority(uint slot, address participant) public**

Function that could be called by existing Authority Vote for the voting for the new Authority Node Input parameter: “slot” - number of voting slot to bet. Input parameter: “participant” - address of the proposed authority. Output parameter: none

- **function voteForBlackListAuthority(address participant) public**

Function that could be called by existing Authority Vote for the voting for the adding another Authority Node to the blacklist Input parameter: “slot” - number of voting slot to bet. Input parameter: “participant” - address of the proposed authority. Output parameter: none

- **function handleClosedPolls() public**

Handle all pollings where time is up. Anybody could call this function. Input parameter: none Output parameter: none

### 8.3.5 Voting Examples

```
import Web3 from 'web3';
import abi from '@abis/abi.json';

const noop = () => {};
const cbCaller = function(fn, ...args) {
  if (fn && typeof fn === 'function') {
    fn(...args);
  }
};

export class Web3Service {
  web3 = null;
  contract = null;
  provider = null;
  account = null;

  constructor(provider) {
    this.provider = provider;
    this.web3 = new Web3(provider);
    this.contract = new this.web3.eth.Contract(
      abi,
      process.env.VUE_APP_BIOS_ADDRESS
    );
  }

  async voteForNewAuthority(votes, address, callbacks = {}) {
    return this.process(
      this.contract.methods.voteForNewAuthority(votes, address).send({
        from: this.account,
        gas: 100000
      }),
      callbacks
    );
  }

  async voteForBlackListAuthority(address, callbacks = {}) {
```

(continues on next page)

(continued from previous page)

```
return this.process(  
  this.contract.methods.voteForBlackListAuthority(address).send({  
    from: this.account,  
    gas: 100000  
  }),  
  callbacks  
);  
}
```

There are many tools created for developers and users of Papyrus Network.

### **1. Papyrus Explorer**

MainNet - <https://explorer.papyrus.network>

TestNet - <https://explorer-testnet.papyrus.network>

Repository - <https://github.com/papyrusglobal/explorer>

### **2. Papyrus Network Status**

MainNet - <https://status.papyrus.network>

TestNet - <https://status.papyrus.network>

### **3. Papyrus Wallet**

MainNet - <https://wallet.papyrus.network>

TestNet - <https://wallet-testnet.papyrus.network>

Repository - <https://github.com/papyrusglobal/wallet>

### **4. Papyrus-Ethereum Bridge**

MainNet<->Ethereum MainNet - <https://bridge.papyrus.network>

TestNet<->Rinkeby Bridge - <https://bridge-testnet.papyrus.network>

Backend repository - <https://github.com/papyrusglobal/papyrus-token-bridge>

Frontend repository - <https://github.com/papyrusglobal/papyrus-token-bridge-web>

### **5. Papyrus Roulette dApp demo**

TestNet - <https://roulette-testnet.papyrus.network/>

Repository - <https://github.com/papyrusglobal/roulette>



### 10.1 What is Papyrus Network?

Papyrus Network is a decentralized blockchain platform for supporting smart contracts and high throughput designed for mass adoption and enterprise usage. Papyrus Network is an operating system which will allow developers to deploy their own decentralized applications and will allow companies integrate custom blockchain for their business needs.

### 10.2 What is the prospects of Papyrus Network?

We aim to build a Papyrus Network ecosystem that everyone can participate in and benefit from. Some of the features of Papyrus Network are:

- Users are able to reduce costs, enjoy convenience, and build fortunes by using different DApp functions deployed in Papyrus Network. Instead of gas transaction fee model token staking model is implemented to allocate network processing power and storage facilities.
- Developers have a vast range of rights including deploying DApp in Papyrus Network net, expanding business and gaining traction as influencers and authority leaders.
- Authority node holders demonstrate support for Papyrus Network and its abundant returns.

Everyone will devote their energy to the construction of the Papyrus Network ecosystem, and will benefit from helping to build Papyrus Network regardless of their role as users, developers or Papyrus Network Authority node holders.

### 10.3 What differs Papyrus Network from Ethereum, EOS and TRON?

Papyrus Network combines the best features of Ethereum and EOS avoiding their weaknesses and having something different inside:

- *Consensus Module*

Instead of energy intensive Proof-of-Work lightweight **Proof-of-Authority consensus and Separation of Powers** are realised, where fixed amount of network nodes called Authority Nodes are operated by credible organizations with public exposure. As a result resource wastage can be eliminated inherent to Proof-of-Work and ensure reasonable transaction costs.

- *Token Staking*

Instead of gas transaction fee model **token staking model** is implemented, where token supply represents total available network throughput and each token represents fraction of it. Application developers need to stake tokens (locking them for specific period of time) for their dApp contracts to receive access to required network bandwidth and may adjust staked amount from time to time accordingly to their needs. Developers don't need to worry about token price volatility between revisions of their bandwidth requirements.

For more information on what differs Papyrus Network from others, go check Comparison page in the Docs.

## 10.4 What are Authority nodes?

They are active network nodes, which participate in Proof-of-Authority consensus. Initial set of Authority nodes was appointed by Papyrus team. After network is launched Authority nodes self-elect themselves from current set of eligible node candidates. Inclusion or exclusion of Authority nodes is based on votes from other Authority nodes. Maximum amount of Authority nodes is fixed as 47, as in our view it is good tradeoff between network speed, scalability and resistance to attacks.

# CHAPTER 11

---

## Papyrus Network key facts

---

This section presents key variables of Papyrus Network setup, which dApp developers need to know.

- ChainId = 32328
- Block interval = 1 second
- Native token for resource allocation = PPR
- Native token emission at genesis block = 1000000000 PPR
- Mining rewards =  $(1.5 * A) / 47$  PPR per mined block, where A equals to the count of Authority Nodes

- 
- Block gas limit = 210284448
  - Unstaking lock period =  $3 * 24$  hours (3 days)
  - Gas refill allocation for an address with X PPR stake =  $X \text{ in wei} * \text{blockGasLimit} * 60 * 60 / \text{totalStake}$  in wei, where blockGasLimit = block gas limit, totalStake = total amount of PPRs staked in the network with X included (implementation notice: for the first stake created for the address gas is allocated immediately, for subsequent stakes gas is allocated once in an hour by gas refill process)
  - Gas refill interval =  $60 * 60$  seconds (1 hour)
  - Maximum gas allocation limit for an address with X PPR stake =  $3 * 24 * X \text{ in wei} * \text{blockGasLimit} * 60 * 60 / \text{totalStake}$  in wei, where blockGasLimit = block gas limit, totalStake = total amount of PPRs staked in the network with X included

- 
- New Authority Node voting period = 14 days
  - Authority Node blacklist voting period = 5 days
  - Minimum votes required for Authority Node candidate approval = 3
  - Maximum number of Authority Nodes and candidates each Authority Nodes can vote for = 7
  - Vote withdrawal lock period = 14 days



## CHAPTER 12

---

Work in progress

---

Papyrus Network further development directions:

1. Implementation of Community blacklist voting.
2. Implementation of 47 Authority Nodes limit and governance improvements.
3. Research on Ethereum wallets compatibility and usability.
4. Support for dApp developers.
5. R&D: native entropy beacon.
6. R&D: blockchain history pruning via state snapshots and epochs to reduce data storage requirements for blockchain nodes.
7. R&D: performance optimisations.



# CHAPTER 13

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`