Papyrus Docs Documentation

Release

Alexander Shvets

Feb 02, 2018

Papyrus Adtech Solution:

Papyrus is the world's first fully comprehensive and highly scalable decentralized ecosystem for digital advertising which radically improves programmatic advertising stack to provide efficient, transparent and mutually beneficial environment for users, publishers, advertisers and decentralized application (dApp) developers using blockchain architecture.

CHAPTER 1

Overview

1.1 Market issues Papyrus solves

- Absence of transparency, incorrect incentives due to rebates from media companies.
- Long chains of middleman with large margin cuts between advertisers and publishers, only 30 to 40 cents of every digital media dollar are estimated to actually reach publishers and result in an ad showing up, according to ANA.
- Growth of ad-blocking software, 26% of US consumers use some sort of adblocking software in 2017, internationally, the loss of publisher revenue from ad blocking rose to $42 billion - up from $28 billion in 2016.
- Fraud traffic accounts to $6,5 billion losses in 2017 (~9% in desktop display, ~22% in video ads).
- Brand safety issues, for example, P&G cuts $140 million from digital ad spending recently due to brand safety and supply chain concerns.
- Viewability issues, 40%+ of ads served are out of view.

1.2 Blockchain solution for advertising market

- Fix transparency issues by creating decentralized storage of ad campaigns data with permissioned access and incentivizing market participants to store that data, decentralization protects data from manipulation
- Remove excessive and hidden budget cuts by allowing advertisers to make payments in tokens based on established payment conditions fixed in smart contracts, all supply chain participants receive payments according to smart contracts, smart contracts are executed using complete ad campaign information stored in decentralized storage
- Resolve brand safety, viewability and fraud issues by connecting auditors / antifraud vendors / attribution providers that verify ad traffic according to smart contracts and put verification information into decentralized storage, ad campaign payments are made automatically according to verification results, dispute resolution happens automatically

- Transition from ad blockers to value exchange with end users by providing tools to publishers to construct dialogue with users on value exchange, users can turn off ads and pay content subscription fees or engage with ads and receive compensations in the form of content access or tokens
- Less paperwork and organizational expenses on payments reconciliation and disputes resolution due to usage of token for payments and automation of processes via smart contracts

CHAPTER 2

Technology overview

2.1 Basic concept

The base of Papyrus technology is the scalability layer. This layer allows processing large amounts of data (operational logs of the bidding process, ad tracking, inputs from the data processing vendors) while keeping blockchain guarantees. Papyrus uses map-reduce approach to aggregate data from multiple sources into compact lists of output transactions which are posted to the base blockchain. This is somewhat similar to plasma.io approach. But our solution is much simpler because we do not try to build public universal blockchain.

The scalability layer is a combination of:

- State channel nodes responsible for accumulating data and placing it into the distributed storage.
- Distributed offchain storage based on IPFS.
- Validator nodes responsible for aggregating,and processing validating data (though logically distinct, this functionality will be implemented as part of channel nodes).
- Base chain: in the prototype - ethereum blockchain. In production mode - consortium blockchain shared among ecosystem participants (like https://github.com/tendermint/ethermint or bitshares).
- Token exchange protocol between the base chain and public chains like Ethereum or Bitcoin (two-way peg).

2.2 Papyrus node

Papyrus node is a complete set of required software for blockchain and IPFS communication. It consists of business logic, state channel node and validator node.

2.3 Channel blocks

State channel's message log may grow at a rate of thousands messages per second and needs to be compacted before settling to the base blockchain. For this reason messages in each state channel are grouped into blocks. Blocks may have thousands of messages. So we have a hierarchy: base chain→state channel→block→message. We update the base chain on a per-block basis so that blockchain interaction is limited. Only block headers go to the base chain. Block data is stored in the offchain block storage, and retained for a limited period of time.

Messages which need to be processed together (e.g. messages for the same impressionId) must be assembled into the same block and hence need to be assigned the same block number. Participants must agree in advance on how block numbers are generated. For example SSP may generate block number incremented every hour and pass it to other participants during RTB. Each message must include signature, target channel id and block number.

Lifecycle of each channel block has the following stages:

- Collection. Each participant produces messages and sends them to its channel node. Channel node temporarily stores messages in apache kafka for persistence.
- Collation. When collection period is finished (no new messages may be added to the block) each participant:
  - assembles all block messages into a single blob - "block part" (block part is different for each participant, so each block may have up to N block parts, where N - is number of participants);
  - writes the block part to block storage;
  - computes hash of the block part and store to base chain.
- Processing. After each participant have posted its block part, all block parts are merged and processed by validators. As a result of processing validators generate block output. This part is described in Validators section below.
- Settlement. Block output transactions are settled to the base chain.

Notes:

- Different blocks are processed independently. Processing of the next block may be completed earlier than that of the previous. To ensure that all participants have enough time to write their data there is a timeout before the block is considered final. It must account for all possible delays such as the maximum period needed for verifying impressions by the appointed auditor(s).
- To avoid DoS attacks there are limits on the number of messages per block and the message size.
- Data written to the block storage is encrypted with the keys known only to the particular channel participants.
- When assembling their block parts participants may include both messages created by themselves and messages created by other participants. Generally speaking participants are expected to include those messages they are incentivised to store (and most likely skip all other messages), i.e. all the messages that being excluded make the participant risk losing money.

2.4 Validators

Block data processing is performed offchain by validator nodes. Validation algorithm is deterministic so each non-faulty validator given the same input data must return the same result. To make validation process scalable Papyrus uses only a few validator nodes for each block, and only if no consensus can be reached additional validators can be involved for dispute resolution.

Validator nodes act on request from participants. To process the block they need to

- download all the block parts from the storage

- merge block parts - duplicate messages are removed and messages are sorted deterministically
- process result block data using specific algorithm which also may use data from the state channel contract
- as a result of processing a list of output transactions is constructed

Block processing has the following steps:

- Each participant chooses which validators it will delegate block validation to and sends them a signed request containing channel id, block number and block encryption key. Validators use provided information to process block data and post hashed results to the blockchain.
- If all choosen validators produce the same result (normal situation) then processing completes.
- If at least two validators have returned different results then block challenge period is extended and dispute process is used to resolve conflicts.

Participants may use different strategies for choosing validators:

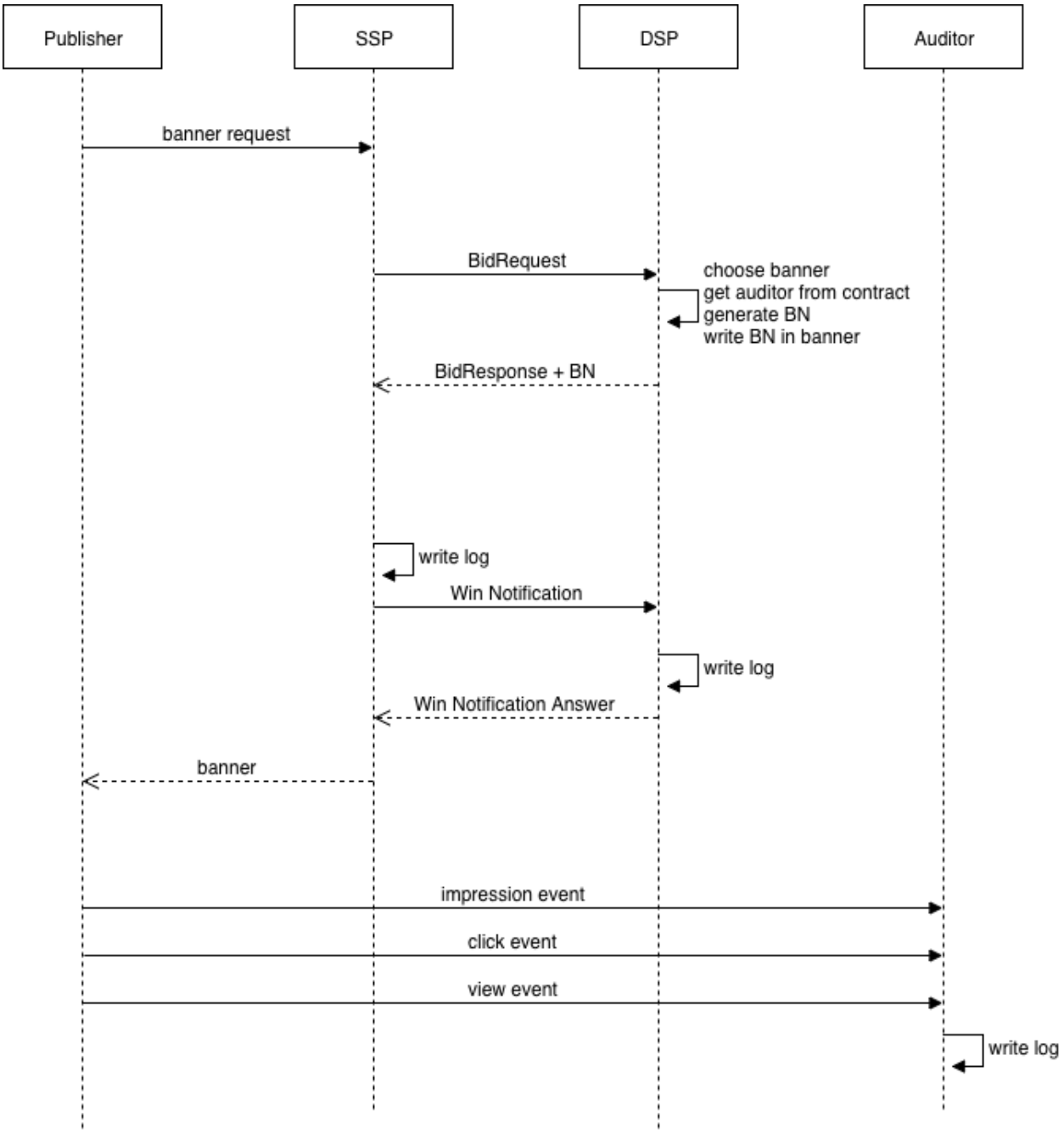
- By running trusted validator node. Install trusted validator node on their own servers and use it for each new block.
- By running coordinator node which selects external validators. One or several randomly choosen validators are used for each new block. Participant still need to run coordinator node to choose and communicate with validators.
- In normal situation the number of validators is equal to the number of participants. But in case of faulty or malicious behaviour of validators the dispute process is used to ensure correctness and additional validators may be used.

2.5 Event sequence

An important part of the process is how events are recorded. In the prototype ecosystem Papyrus suggests two types of records:

1. Win records. These records are generated by SSP after determining a winner and by DSP after receiving a win notification.
2. Auditor check records. These records are generated by auditor after receiving tracking events from the browser/app.

The placement of record generation is illustrated on the following sequence diagram.


Publisher
SSP
DSP
Auditor
banner request
BidRequest
choose banner
get auditor from contract
generate BN
write BN in banner
BidResponse + BN
write log
Win Notification
write log
Win Notification Answer
banner
impression event
click event
view event
write log

CHAPTER 3

Papyrus scanner

3.1 General

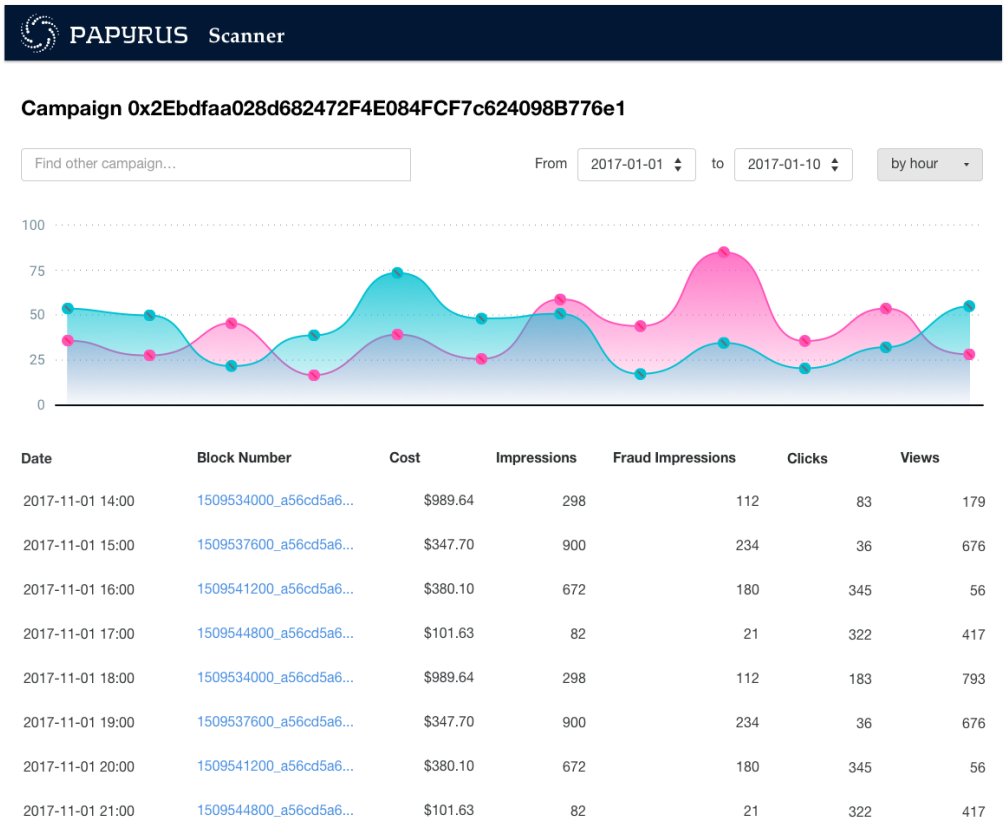
Papyrus scanner is a software solution to view Papyrus blockchain. It enables to any participant (advertisers, publishers, vendors) to see ad campaigns they pariticpate into.

Papyrus scanner provides information about ad campaign smart contracts and aggregates validated statistics information. The scanner shows aggregated statistics blocks where you can find

- actual cost paid to SSP and other vendors;
- amount of fraud impressions;
- actual clicks and views count.

For example, you can see that there were 1,975 impressions, but in fact only 1,545 were real not fraud ones. And you can view real clicks count and impressions. More important that you paid only for these real impressions.

The agency fee and other vendor fees are included in reporting according to the campaign contract, so you can be sure that there are no opaque markups.



3.2 Scanner API

3.2.1 Read campaign list

GET /campaigns
Returns a list of campaigns.

Request example

```
GET /api/v1/campaigns HTTP/1.1
Host: scanner.papyrus.global
Accept: application/json
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824",
    "name": "Campaign name"
    "advertiser_id":
↪"0a526a90a85596dcb3669fd86963422969edbbf7c4752492d780b78e6355d4ee",
    "start_date": "2018-01-01",
```

```
    "end_date": "2018-01-31",
    "budget": "10000000000",
    "maximum_cpm": "10000000",
    "ssps": [
      {
        "id": "007d831ea2e8e1d080b31e33c50b89ea07f9b694bccad998c8cf5cb1a087f889",
        "fee_percent": "0.1"
      }
    ]
    "auditors": [
      {
        "id": "c5a62ce3fa7f6d86af0009389ccd815277691ea64da0c5c98e302bb13dd59248",
        "fee_percent": "0.05"
      }
    ]
  }
]
```

Query Parameters

- **key** (*String*) – participant_key, required
- **campaign_id** (*String*) – campaign filter
- **advertiser_id** (*String*) – advertiser filter
- **dsp_id** (*String*) – dsp filter
- **ssp_id** (*String*) – ssp filter
- **auditor_id** (*String*) – auditor filter

3.2.2 Read campaign statistics

GET /statistics

Returns campaign statistics.

Request example

```
GET /api/v1/statistics?campaign_
↪id=2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 HTTP/1.1
Host: scanner.papyrus.global
Accept: application/json
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: text/javascript

[
  {
    "date": "2017-12-12",
    "block_number": "1511718000_
↪496aca80e4d8f29fb8e8cd816c3afb48d3f103970b3a2ee1600c08ca67326dee"
    "cost": "12340000",
    "impressions": "1234",
    "fraud_impressions": "321",
    "clicks": "56",
```

```
    "views": "77",
    "ssps": [
       {
         "id": "007d831ea2e8e1d080b31e33c50b89ea07f9b694bccad998c8cf5cb1a087f889",
         "fee": "1234000"
       }
    ]
    "auditors": [
       {
         "id": "c5a62ce3fa7f6d86af0009389ccd815277691ea64da0c5c98e302bb13dd59248",
         "fee": "617000"
       }
    ]
  },
  {
    "date": "2017-12-12",
    "block_number": "1511723000_
↪6d0b07ee773591f2a1b492d3ca65afdefc90e1cadfcc542a74048bb0ae7daa27"
    "cost": "43210000",
    "impressions": "4321",
    "fraud_impressions": "789",
    "clicks": "123",
    "views": "135",
    "ssps": [
       {
         "id": "007d831ea2e8e1d080b31e33c50b89ea07f9b694bccad998c8cf5cb1a087f889",
         "fee": "4321000"
       }
    ]
    "auditors": [
       {
         "id": "c5a62ce3fa7f6d86af0009389ccd815277691ea64da0c5c98e302bb13dd59248",
         "fee": "2160500"
       }
    ]
  }
]
```


Query Parameters

- **key** – participant_key, required
- **campaign_id** – campaign filter, required
- **date_from** – date filter
- **date_to** – date filter

CHAPTER 4

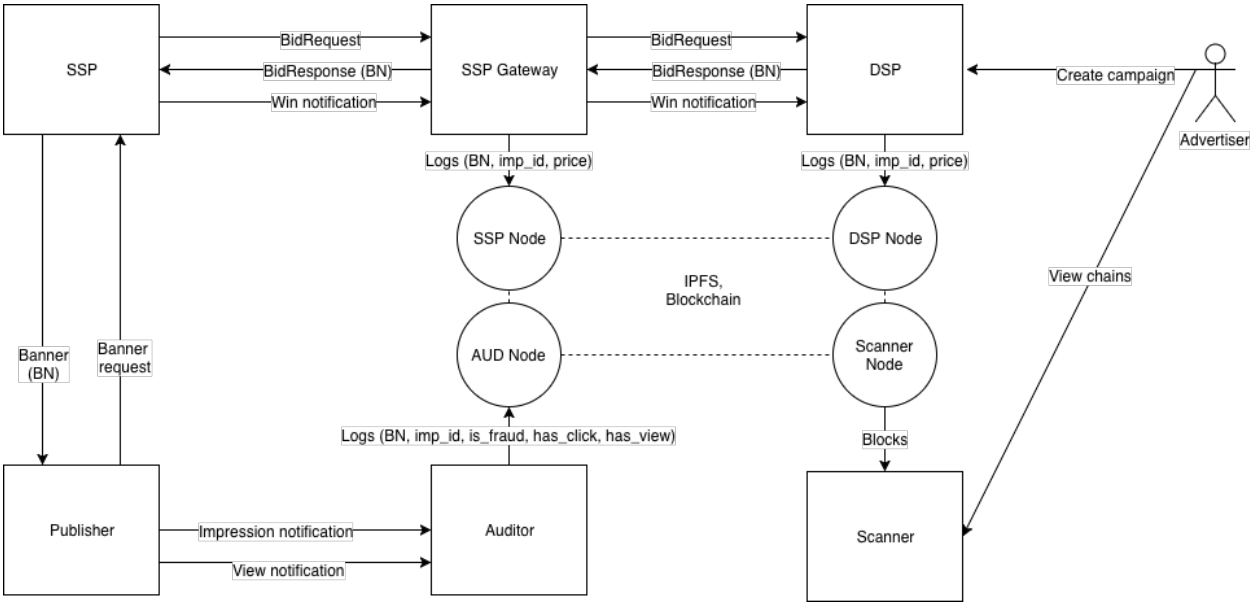
SSP integration

The most easiest participant for an integration into the Papyrus ecosystem is SSP.

There are 3 ways of SSP integrations.

1. Papyrus deploys SSP gateway with its own blockchain node and SSP connects to this gateway through API.
2. Papyrus installs the node and SSP sends logs to this node.
3. SSP installs its own node and communicates with it internally.

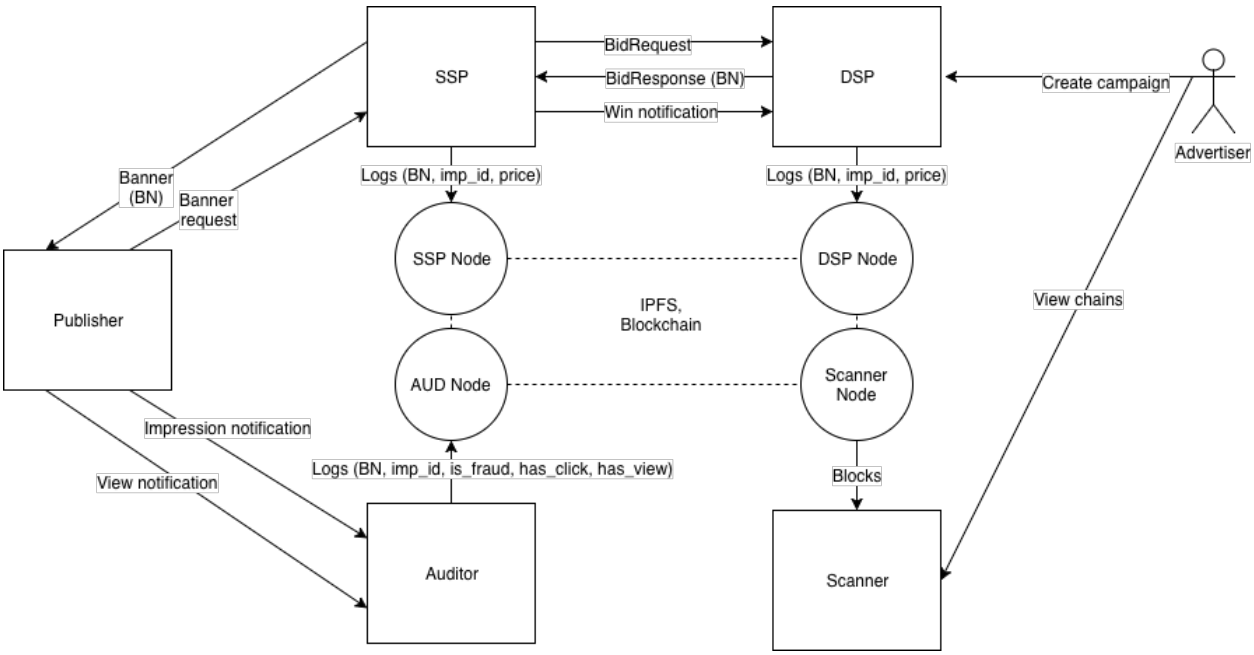
4.1 Papyrus SSP gateway



For basic integration Papyrus can deploy SSP gateway by its own team. In this case SSP can connect SSP gateway like any other DSP.

SSP gateway is the system which passes Bid Requests and Win Notifications from integrated SSP to connected DSPs. SSP just sends all requests to SSP gateway instead of DSP and gateway does all necessary job to put log records into decentralized storage. The auction is holded by SSP itself and gateway just records the winner record. SSP Gateway has to send log record to Channel Node after receiving Win Notification. The record contains block number, impression_id and the winning price.

4.2 Papyrus node



This case is similar to previous, but doesn't require SSP gateway. Papyrus team just deploys Papyrus blockchain node and SSP sends log records into this node through gRPC.

The main difficulty that SSP has get block number from Bid Response. This block number is written in *ext.blocknumber* field of Bid object.

SSP has to send message on Win Notification generation. The format of gRPC message is presented below.

```
// Main channel interface
service StateChannel {
    // Creates or updates outgoing channel with given participant
    rpc RegisterTransaction(RegisterTransactionRequest) returns␣
↪(RegisterTransactionResponse);
}

// Registers transaction
message RegisterTransactionRequest {
    // sender address in HEX, from config
    string sender = 1;
    // block_number, from Bid Response
    int64 block = 3;
    // encoded message, format below
    bytes data = 4;
```

```
    // EC signature by sender's key, from congif
    bytes signature = 5;
}


message PapyrusWinNotification {
    string imp_id = 1;
    // price in token * 10^18
    int64 price = 2;
}
```

4.3 SSP node

This case is similar to previous, but in this case SSP has to install its own blockchain node. Papyrus team distibutes SSP node as docker image with instruction provided. The link to the distro will be published later.

CHAPTER 5

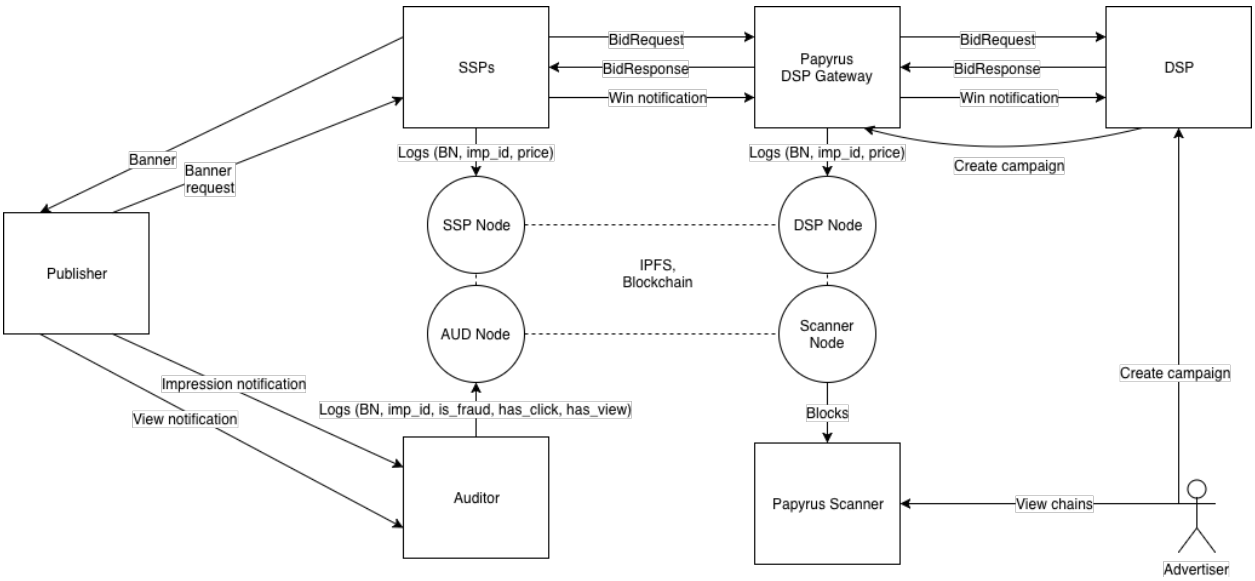
DSP integration

DSP is the key participant in the ecosystem because DSP is in charge of creating ad campaign smart contracts

There are 3 ways of DSP integrations.

1. Papyrus deploys DSP gateway with its own blockchain node and DSP connects to this gateway through API.
2. Papyrus installs the node and DSP sends logs to this node. Also DSP has to create smart contracts through this node.
3. DSP installs its own node and communicates with it internally.

5.1 Papyrus DSP gateway



To make integration easier Papyrus team can deploy DSP gateway to include DSP into Papyrus ecosystem. This gateway acts like any additional SSP, but actually it does all necessary job necessary for blockchain working and resends messages to connected SSPs and in other way.

Integration of DSP gateway differs from SSP integration in 2 points:

1. DSP has to create smart contract in Papyrus blockchain for every ad campaign.
2. DSP has to generate block number for every impression it put bid on.

DSP gateway provides an API method to creae smart contracts.

POST /campaigns

Creates a smart contract for campaign.

Request example

```
POST /api/v1/campaigns HTTP/1.1
Host: scanner.papyrus.global
Accept: application/json
Content-type: application/json

{
        "name": "Campaign name"
    "advertiser_id":
↪"0a526a90a85596dcb3669fd86963422969edbbf7c4752492d780b78e6355d4ee",
    "start_date": "2018-01-01",
    "end_date": "2018-01-31",
    "budget": "10000000000",
    "maximum_cpm": "10000000",
    "ssps": [
      {
        "id": "007d831ea2e8e1d080b31e33c50b89ea07f9b694bccad998c8cf5cb1a087f889",
        "fee_percent": "0.1"
      }
    ]
    "auditors": [
      {
        "id": "c5a62ce3fa7f6d86af0009389ccd815277691ea64da0c5c98e302bb13dd59248",
        "fee_percent": "0.05"
      }
    ]
}
```

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
  }
]
```

Advertisers, SSP and Aufitor IDs will be provided by decentralized registry in the future, now this registry is provided by Papyrus team and should be saved locally.

The result ID is the smart contract ID and should be saved for the further usage.

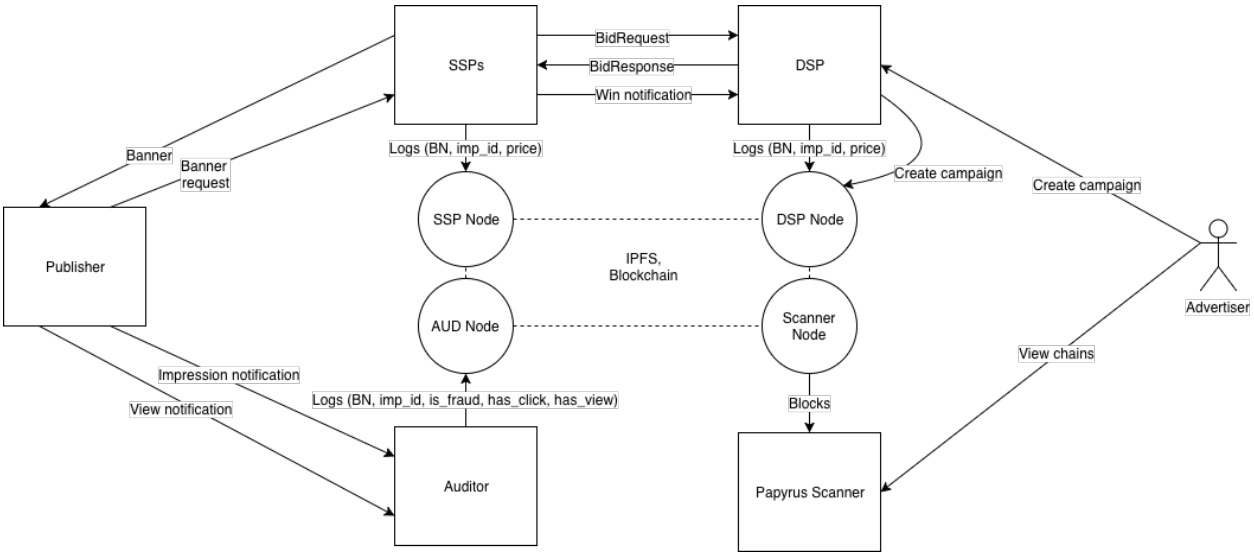
The contract ID is used in block generation.

block_number = current_timestamp + '_' + contract_id

A new block number should be generated for the contract every hour. This time window can be smaller in case of big number of messages in block.

Generated block number has to be included in *ext.blocknumber* field of Bid object.

5.2 Papyrus node



This case is similar to previous, but doesn't require DSP gateway. Papyrus team just deploys Papyrus blockchain node and DSP sends log records into this node through gRPC.

DSP has to send log message right after receiveing Win Notification. The format of gRPC message is presented below.

```
// Main channel interface
service StateChannel {
    // Creates or updates outgoing channel with given participant
    rpc RegisterTransaction(RegisterTransactionRequest) returns␣
↪(RegisterTransactionResponse);
}

// Registers transaction
message RegisterTransactionRequest {
    // sender address in HEX
    string sender = 1;
    // block_number
    int64 block = 3;
    // encoded message
    bytes data = 4;
    // EC signature by sender's key
    bytes signature = 5;
}


message PapyrusWinNotification {
    string imp_id = 1;
    // price in token * 10^18
```

```
    int64 price = 2;
}
```

Also, DSP has to create smart contract using the node API. This API will be specified later.

5.3 DSP node

This case is similar to previous, but in this case DSP installs its own blockchain node. Papyrus team distibutes blockchain nodes as docker images with instruction provided. The link to the distro will be published later.

CHAPTER 6

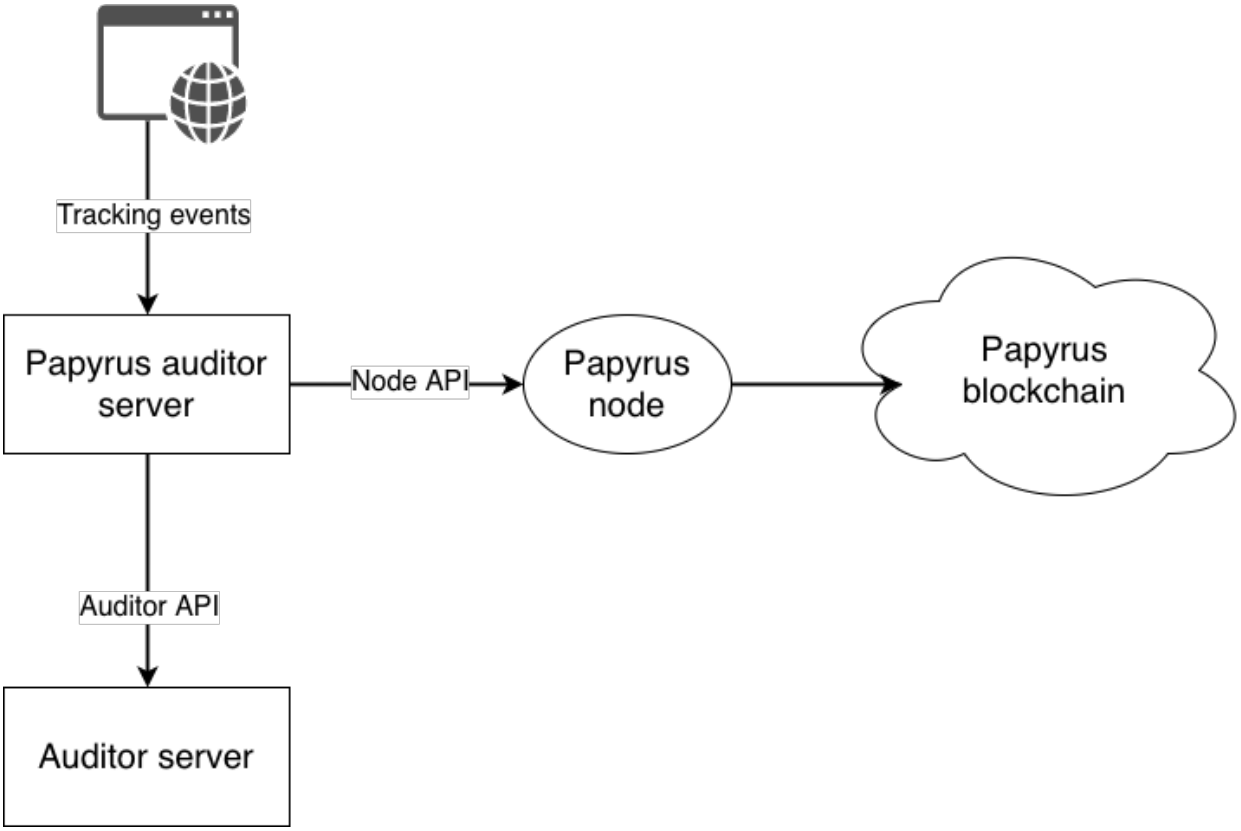
Auditor integration

Auditor integration is the major feature of the Papyrus ecosystem. It enables to make payments only on real impressions and other events.

There are several ways of auditor integrations.

1. Papyrus deploys log server with its own blockchain node and connects to auditor through API.
2. Papyrus installs the node and auditor sends logs to this node.
3. Auditor installs its own node.

6.1 Papyrus log server and Papyrus node



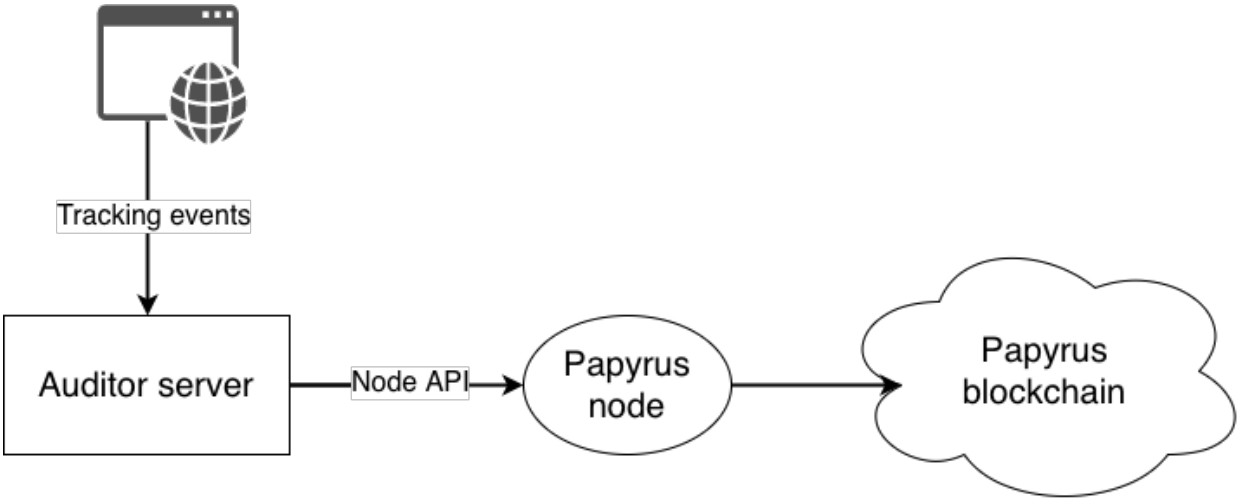
This is the easiest way to connect auditor to Papyrus ecosyste, because Papyrus development team setup and maintain this integration on their side.

But in this case auditor has to provide an API to check requests.

API can vary for different vendors, but it must support at least the following parameters:

- request type
- user IP
- user agent
- page URL
- campaign ID

6.2 Auditor log server and Papyrus node



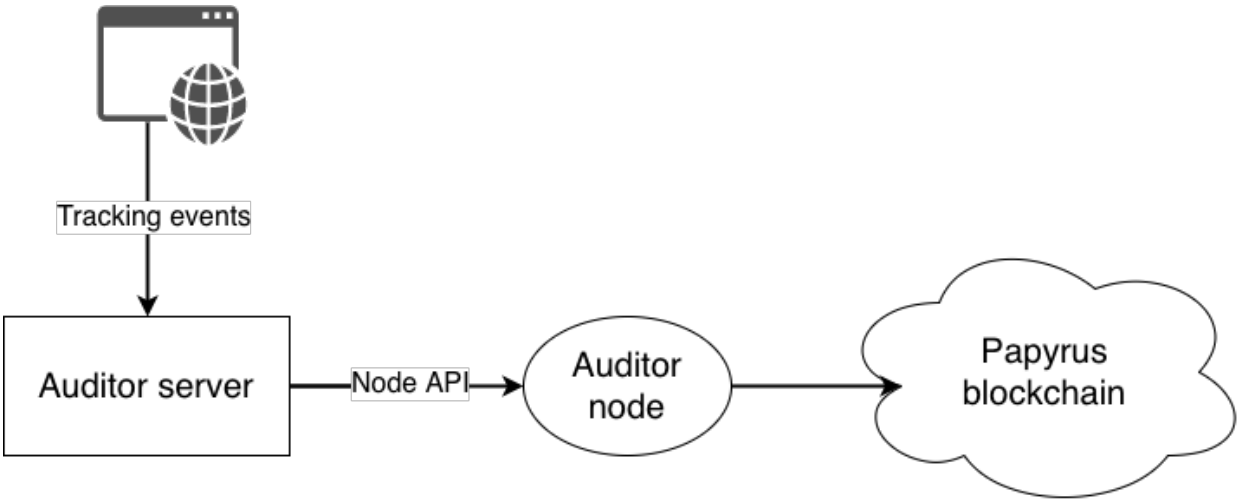
This case has medium difficulty for auditor. It requires that auditor process events itself and works with blockchain node, but blockchain node is maintained by Papyrus team.

Papyrus blockchain node has gRPC API with the following message structure

```
// Main channel interface
service StateChannel {
    // Creates or updates outgoing channel with given participant
    rpc RegisterTransaction(RegisterTransactionRequest) returns␣
↪(RegisterTransactionResponse);
}

// Registers transaction
message RegisterTransactionRequest {
    // sender address in HEX
    string sender = 1;
    // channel contract address in HEX
    string channel = 2;
    // block_number
    int64 block = 3;
    // encoded message
    bytes data = 4;
    // EC signature by sender's key
    bytes signature = 5;
}

message RegisterTransactionResponse {
}

message PapyrusAuditorFeedback {
    string imp_id = 1;
    bool flags = 3;
    enum Flags {
        EMPTY=0;
        FRAUD=1;
        VIEW=2;
        CLICK=4;
    }
```

```
}
```

6.3 Auditor log server and node



This case is the most diffucult for auditor, but it is the most effective solution. Auditor has to setup its own blockchain node and send logs to its API as in the previous case.

The Papyrus node is distibuted as docker container. We will describe installation guide soon.

CHAPTER 7

Advertiser and publisher nodes

To ensure that the process is valid other participants like advertisers and publishers can install their own nodes. This section will describe installation process for these participants.

CHAPTER 8

Papyrus blockchain

8.1 What is Papyrus

8.1.1 History

Bitcoin was first cryptocurrency platform and public use of blockchain technology. In Bitcoin's case the distributed database is conceived of as a table of account balances, a ledger, and transactions are transfers of the bitcoin token to facilitate trustless finance between individuals. But as bitcoin began attracting greater attention from developers and technologists, novel projects began to use the bitcoin network for purposes other than transfers of value tokens. Many of these took the form of "alt coins" - separate blockchains with cryptocurrencies of their own which improved on the original bitcoin protocol to add new features or capabilities.

Ethereum is first a programmable blockchain. Rather than give users a set of pre-defined operations, Ethereum allows users to create their own operations of any complexity they wish. In this way, it serves as a platform for many different types of decentralized blockchain applications, including but not limited to cryptocurrencies. Ethereum introduced the ability to build application-specific logic upon a blockchain network. This enabled new capabilities beyond transactions to incorporate state, business logic, and multi-party contracts to be stored and executed on a blockchain and written to an immutable ledger.

Still Ethereum as a platform has serious limitations that prevents it from being widely used to run real-world applications.

- Price. Blockchain computation and storage are many orders of magnitude more expensive than in traditional computing. As platform gains popularity transaction commissions (gas price) increase. As of writing single ETH transfer transaction costs more than $1. More complex transactions like contract deployment already cost $10 and more. Cost of 1KB of storage is about 0.032 ETH (with gas price=50gwei). With price of ETH $1000 it gives us $32 per 1KB or insane $32 mln dollars per 1Gb. This is 5 orders of magnitude more expensive than traditional storage. Prices will only increase in future.
- Speed. To protect security of Ethereum network available computing power (measured in gas) per period is limited. This essentially creates performance bottleneck limiting whole network processing power to several simple transactions per second. More complex transactions could easily consume great part of block gas limit leading to increased waiting time.

- Privacy. Anyone can read data stored in Ethereum blockchain. But there are many uses cases where data should be kept private. This includes user's personal data, financial transaction and other use cases. It is possible to introducing privacy directly to blockchain using zero-knowledge proof algorithms (aka zk-Snarks in zcash). Ethereum introduced support for zk-Snarks in Byzantine hardfork. But in practice zkSnarks for arbitrary smart contracts on ethereum are still not possible due to high gas costs.

8.1.2 Blockchain for big data

There are many proposals how to make existing blockhain more scalable and more private. This includes

- Increasing block size and/or block generation period.
- Alternative consensus algorithms like PoS, dPoS or BFT.
- Private or consortium blockchains.
- State Channels.
- Offchain storages
- Sharding.
- zkSnarks.

To implement highload service like ad exchange server, blockchain needs handle hunders of thousands requests per second and terabytes of data per day. This requires improvement by million or even billion times.

Papyrus takes another approach. Instead of taking classic blockchain and try to make it million times faster we begin with solution which may handle high load and add blockchain functionality on top of it while keeping performance reasonable. Doing this we forced to sacrifice some properties of classic blockchains but still maintain acceptable performance and privacy characteristic. Also it is possible to make compromise decisions balance between blockchain guarantees, performance and privacy.

Papyrus enables processing large amount of data while keeping blockchain consistency and BFT guarantees. Papyrus uses map-reduce approach to aggregate data to compact lists of output transactions which are posted to blockchain. This is similar to plasma.io approach. But papyrus solution is much simplier because it is not universal public blockchain.

To save costs raw input data is stored outside of main chain in low cost distributed storage. Only hashes and processed compact results are posted to blockchain.

Papyrus blockchain provides base layer which allow multiple highload application protocols to coexist on top of it.

8.1.3 Requirements

We designed papyrus with following requirements in mind:

- Storage. Ability to store raw transactions[*] for period at least 3 month. Aggregates could be stored longer.
- Immutability. Once written transactions cannot be modified.
- Processing: 10-100 messages/s for single channel.
- Privacy. Transactions are encrypted and accessible only to selected participants. Participants provide keys allowing to read specific blocks of transactions.

8.2 Scalability

We address blockchain scalability in two directions: performance and storage.

8.2.1 Performance

Due to nature of their consensus algorithms blockchains like Ethereum ant Bitcoin require every computation to be handled on every node in network. Malicious adversary needs to operate more than 50% (or 33%) of nodes to make an attack. Blockchain scaling obviously requires that every node process only some portion of network transactions. There are proposals like sharding which suggest spliting blockchain into parts which may be processed in parallel. This introduces security issue because attacker now needs smaller number of nodes to attack consensus of single shard.

Papyrus uses multilevel consensus to provide same level of security. Papyrus uses channels for scalability. Normally there are small number of nodes controlling each channel. But in case of consensus failure any node may request help from other nodes in the network. So attacker may only cause delay in channel progress but not break it's consensus. Also attacker nodes will lose their deposits making an attack economically infeasible.

8.2.2 Storage

Blockchains like Ethereum and Bitcoin store full chain on every node. This ensure great data availability due to very high level of replication. But cost for that is very poor scalability. Papyrus uses distributed storage that provide balance between availability and scalability. Storage nodes are getting paid for storing data. Proof of spacetime is used to detect and penalize cheating nodes.

8.3 Comparison

8.3.1 Hyperledger Fabric

is a platform for distributed ledger solutions. Hyperledger provides scalability and privacy by introducing concept of channels. Channel application logic is encapsulated in chaincode. Different part of system are used to execute chaincode and convert it to transaction proposal and to apply transaction proposal to actual ledger. To avoid race conditions it uses lock-free optimistic concurrency, with rollback in case of dirty read/writes. However for systems with high contention this may lead to serious performance degradation due to transaction rollbacks and introduce another layer of compexity of retry logic. In Papyrus there is no need to do optimistic locking because application code which processes big data is performed in single step and only compact results are applied to blockchain.

8.3.2 IOTA

IOTA differs from other cryptocurrencies because direct acyclic graph instead of blockchain. It allows for much higher parallelisation and total throughput comparing to classic blockchain solutions. But still performance is not good enough for real world highload applications. Current limit is arounf 1000 tx/s. IOTA lacks contract functionality at this moment which limits area where it may be used.

8.3.3 EOS.io

8.3.4 Ethereum

8.3.5 Cardano

8.3.6 Telegram Open Network

8.4 Getting Started

TODO

8.5 Architecture

8.5.1 Deployment Layers

Papyrus uses Cosmos Network https://cosmos.network/ as base blockchain solution and ads it's own unique features.

Cosmos Hub

Papyrus will be connected to main cosmos network hub. This will add additional security by using large set of hub validators. Also this will allow to integrate Papyrus chain with other blockchains which will be connected to Hub like Ethereum mainnet. For example it will be possible to transfer PPR tokens from Papyrus chain to Ethereum mainnet.

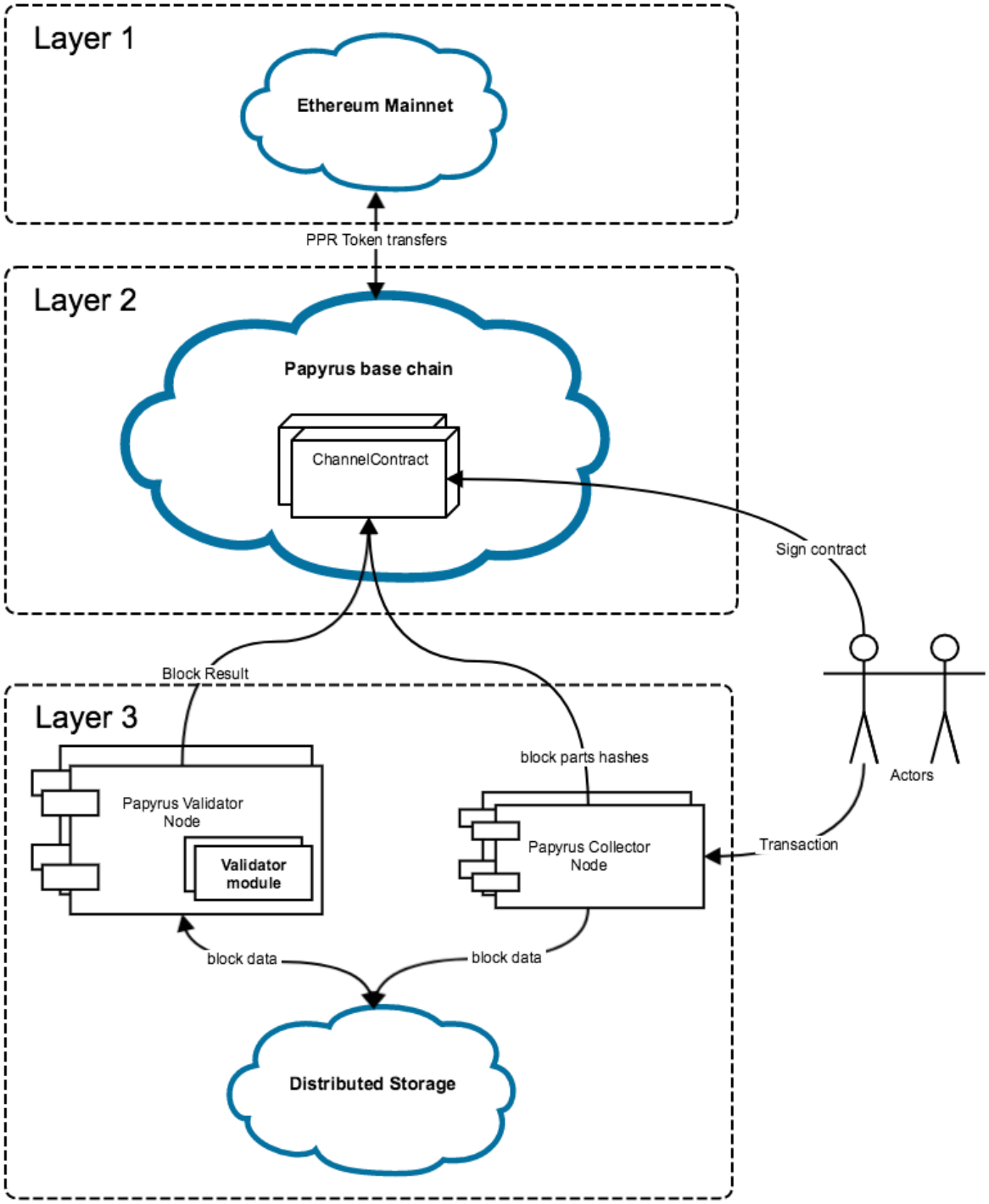
Papyrus chain

Papyrus chain is implemented as Cosmos Zone connected to Cosmos Hub. Code is based on (https://github.com/tendermint/ethermint) with several additional functions. Papyrus chain is consortium blockchain shared among ecosystem participants. It is used for contracts implementation and papyrus channels settlement.

Channel layer

Used for big data processing. Consists of:

- Channel nodes.
- Distributed offchain storage.
- Validator nodes (functionality is implemented as part of channel nodes) with protocol modules.



8.5.2 Channels

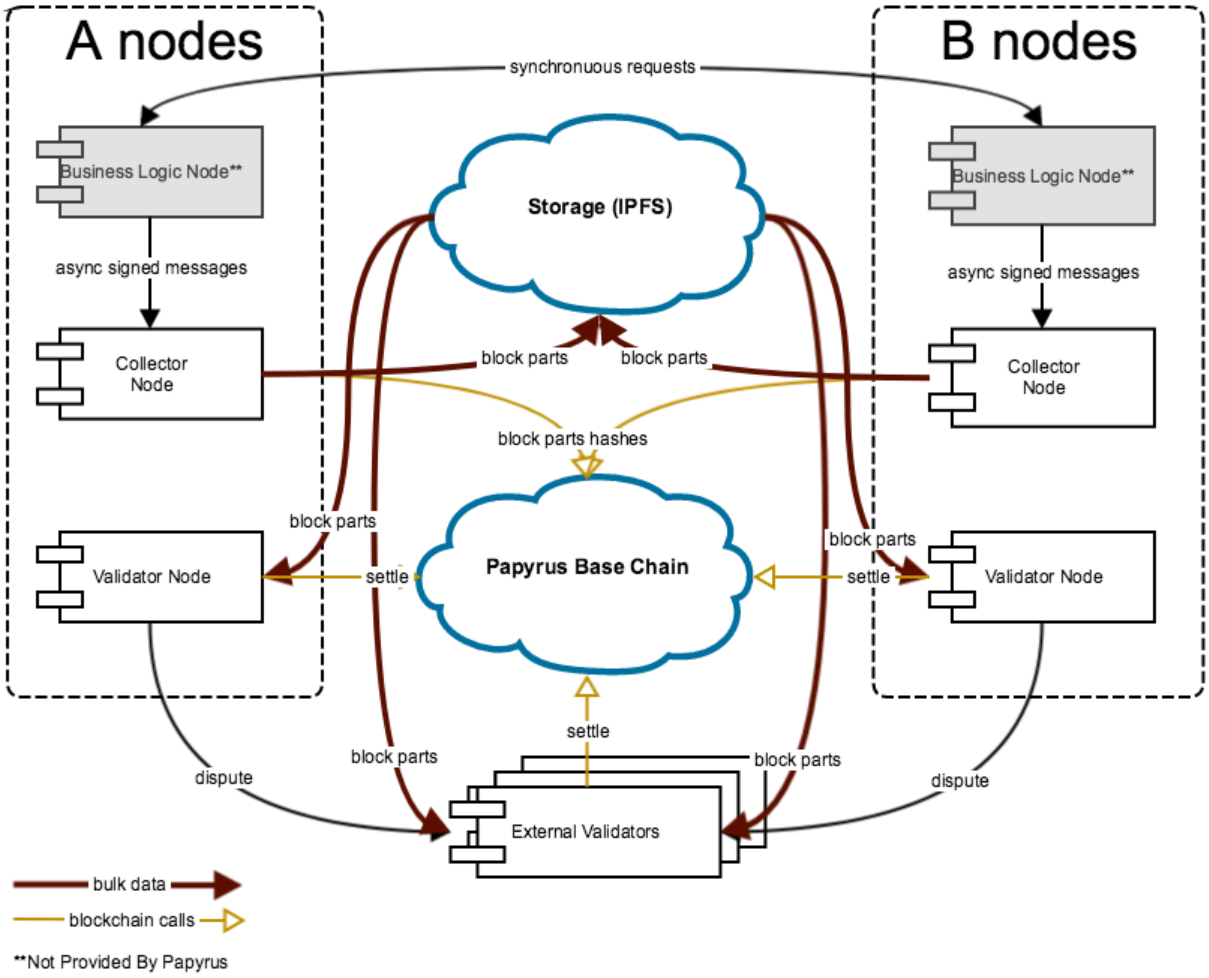
Papyrus channels are similar to Hyperledger channels. Papyrus channels allow 2 or more participants. Participants exchange transactions off chain using any protocol. Transactions are grouped into blocks. Each block is processed independently and result is posted to base chain.

Channel may be viewed as a separate blockchain with custom transaction processing logic. Channel transaction

processing logic is encapsulated in validator module (similar to hyperledger's chaincode). Job of validator module is to aggregate transactions in block data to compact result that is possible to settle to blockchain.

Validators use consensus algorithm to decide on result of each block. Possible consensus algorithms are: - BFT. More than X% of predefined validators should agree on a result. - Stake voting. Validators may bet on a result by putting stake in tokens. More than X% of validators by total stake should agree on a result.

Papyrus channel example diagram for two participants A and B:



State channel's transaction log may contain thousands transactions per second and needs to be compacted before settling to base blockchain. For this reason transactions in each state channel are grouped into blocks. Block may have thousands of transactions. So we have hierarchy: base chain→state channel→block→transaction. We update base chain on per-block basis so blockchain interaction is limited. Only block headers go to base chain. Block data is stored in offchain block storage for some period.

Transactions that need to be processed together (e.g. transactions for same impressionId) must get same block number. Participants must negotiate in advance over how block numbers are generated. For example SSP may generate block number incremented every hour and pass it to other participants during RTB. Each transaction must include signature, target channel id and block number.

Lifecycle of each channel block has following stages:

- Collection. Each participant produce transactions and sends it to his collector node. Collector node temporarily store transactions in local database (on filesystem for now, later will be implemented using apache kafka).

- Collation. When collection period is finished (no new transactions may be added to the block) each participant: - gather all block transactions single blob - "block part" (block part is different for each participant, so each block may have up to N block parts, where N - is number of participants). - writes block part to block storage - compute hash of block part and store to base chain
- Processing. After each participant have posted his block part, all block parts are merged and processed by validators. As a result of processing validators generate block output. This part is described in Validators.
- Settlement. Block result are settled to base chain.

Notes:

- Different blocks are processed independently. Processing of next block may be completed earlier than previous.
- To ensure that all participants have enough time to write their data there is a timeout before block is considered final. It must accont for all possible delays like verifying impressions by auditor.
- To avoid DoS attacks there are limits on number of transactions in block and each transaction size.
- Data written to block storage is encrypted by key known only to channel participants.
- Participants may include in their block parts as transactions created by them and also transactions created by other participants. In other words participants should include transactions they are incentivised to store and skip other transactions, i.e. transactions that being excluded make participant risk losing money.

8.5.3 Restrictions

Papyrus doesn't enforce any upfront deposits requirements for channels. But this requirements may be implemented on top of papyrus channel in the application layer.

It is not possible to add or remove participants or alter configuration of open channel. In such cases new channel must be created.

8.5.4 Validators

Block data processing is performed off-chain by validator nodes. Validation algorithm is deterministic so each non-faulty validator given same input data must return same result.

To make validation process scalable Papyrus uses few validator nodes for each block.

Validator nodes act by request from participants. To process block they need to

- download all block parts from storage
- merge block parts - duplicate transactions are removed and all transactions are sorted deterministically
- process result block data using specific algorithm which also may use data from state channel contract
- as a result of processing list of output transactions is constructed

Block processing has following steps:

- Each participant choose which validators it will delegate block validation and send them signed request containing channel id and block number and block encryption key.
- Validators use provided information to process block data and post hashed result to blockchain.
- If all choosen validators got same result (normal situation) then processing completes.
- If at least two validators have returned different results then block challenge period is extended and dispute process used to resolve conflicts.

Participants may use different strategies for choosing validators:

- By running trusted validator node. Install trusted validator node on their own servers and use it for each new block.
- By running coordinator node which select external validators. One or several randomly choosen validators are used for each new block. Participant still need run coordinator node to choose and communicate with validators.
- In normal situation number of validators is equal to number of participants. But in case of faulty or malicious behaviour of validators dispute process is used to ensure correctness and additional validators may be used.

8.5.5 Dispute process

Dispute process is triggered when validators have processed block data and came to different results. During dispute process validators vote for particular result by making deposit.

- Dispute process has challenge period.
- During challenge period participants may ask additional validators to verify computation and post their result to blockchain.
- After challenge period have completed top result received most number of votes wins. All validators that produced different results lose their deposits.

Notes:

- Challenge period is negotiated in advance. (Possible it should be allowed to extend challenge period by providing more deposit. For example when participant have not enough time to collect additional validator votes.)
- Since block data is encrypted participants must provide keys for validators who download data from storage. Additional validators should be trusted enough to give them keys to encrypted channel data.
- Other participants may wish to ban participant whose validator shows faulty or malicious behaviour.

8.5.6 Economics

Validators are paid by participants for processing blocks. Validators get paid after block is successfully validated and possible dispute process is won.

Validators lose their deposits when they lose dispute process. Deposit is distributed among winner validators.

Size of payment/penalty is negotiated directly between participant and validator.

Successfull block generation is recorded in blockchain. Participants may derive all neccessary reputation from this information. We will provide open source reference implementation which uses reputation information and auction scheme to choose validators.

To prevent sybil attacks validator must lock tokens for period at least 30 days. Only after 30 days these tokens can be used to make deposits in dispute processes.

8.5.7 Block encryption

To generate block encryption keys hierarchical deterministic keys are used. For each state channel participants choose key pair (PubMasterKey,PrivateMasterKey) and random Seed. Key pair for each block is generated as:

- Offset := sha3(contract_id + block_number + Seed)
- Pub(Offset) := public key generated using Offset as private key
- PrivateBlockKey := PrivateMasterKey + Offset
- PubBlockKey = PubMasterKey + Pub(Offset)

Participants should encrypt block data with PrivateBlockKey. PubBlockKey can be used to decrypt block data. Seed, PrivateMasterKey and PubMasterKey must be kept in secret.

Participants also may choose to rotate PrivateMasterKey and PubMasterKey from time to time.

8.5.8 Storage

IPFS will be used initially for storage solution. Later we may switch to Filecoin or Storj. Due to block validation process participants are incentivised to store their block parts at least until validation process is finished. If there will be demand for durable logs storage we expect that storage service providers will appear as part of natural economical process.

8.6 Contracts

State channel contract is an agreement stored in base blockchain which defines how messages are processed.

Contract consists of blockchain part (smartcontract) and offchain module. Offchain module is part of validator nodes. Initially validators will have prerefined offchain contract logic but in future we consider support plugin architecture. Validators will load contract module from some offchain storage using cryptographic reference provided in smartcontract.

To reduce gas cost in base chain most heavy logic should be moved to offchain module.

Example of contract logic is rule that convert following messages:

- SSP: BidRequest(impId:123)
- DSP: BidResponse(impId:123,bid:2)
- SSP: WinNotification(impId:123,price:1.8)
- Auditor: Approve(impId:123)

To token transfers:

- DSP→SSP 1.79 PRP
- DSP→Auditor 0.01 PRP

Rules could be complex and include slashing conditions for participants for misbehaviour. For example SSP may be punished for having sent two distinct win notifications for single impression to two different DSPs.

8.6.1 Channel contract

ChannelContract stores processed papyrus block results.

Public view functions

- function module() public view returns (string). Name and version of off-chain validator module. This is used by validator to determine which code to execute. In future this may contain reference to contract code.
- function participantCount() public view returns (uint64). Participants which need sign contract.
- function participant(uint64 index) public view returns (address participantAddress, address validatorAddress)
- function closed() public view returns (uint256) - returns block number after which contract will become inactive

Lifecycle functions

- approve(address validator_address). Called by each participant defined during construction. Each participant provide address of validator node.
- block_part(uint64 bn, bytes reference). Called by participant when block part is ready. May be called by each participant only once per block. bytes reference format is: first byte define protocol (currently only 0 - IPFS), other bytes are protocol specific. For IPFS other bytes are hash of ipfs link.
- block_result(uint64 bn, uint256 result_hash, uint256 stake). Called by each validator once per block. stake field is used to weight validators desicions in case of dispute.
- block_finalize(uint64 bn, bytes result). Called by any participant with sha3(result) == result_hash. Require block_result to have been called by each registered validator with same result_hash.

8.6.2 Protocol-specific settlement contract

ChannelContract by itself is not very useful as it only stores block results. Settlement contracts may be created which could take block results and execute payments based on computed block results.

In most cases settlement contract is a contract which:

Allows deposits to be made by participants. # Have function settle_block(channel_contract, block_number) which: extracts block result from channel_contract and make payments according to defined algorithm. Set of settled blocks should be remembered to ensure that every block is settled only once.

Settlement contracts are not provided by papyrus. They are created and managed by participants. They may use ChannelContract's api to retrieve block results and execute settlement logic. Single settlement contract may be used for multiple ChannelContracts. It allow single deposit to be shared across multiple channels.

8.6.3 Lifecycle

- Creation. contract is created in blockchain with parameters:
  - module
  - participants
  - any other parameters required for onchain settlement logic (like StandardToken token, or addresses of other contracts)
- Signing: each participant signs contract by invoking it's approve method.
- Processing blocks
- Closing

8.7 Security

In channels faulty or malicious participant could be identified and blocked by other participants. So damage in most cases is reduced to one or several channel blocks. For example if DSP's validator failed to verify blocks SSP may ban it. It is up to each participant to choose proper thresholds and policies. This also applies to contract deposits used for payment settlement. Participants should decide when deposit is enough to continue accept request for other participants.

8.7.1 Verifier's dilemma

During dispute process verifiers may instead of validating block themselves just vote on other's verifiers result. Incentive skipping block validation increases with block size. Economically rational validators may start avoiding costly validation and betting on some other's results. Without additional incentives rational validators gain unfair advantage over honest validators. Because they do not spend computing resources they may participate in many disputes maximizing their profits.

Additional validators involved in dispute resolution require payment for their services. This payment comes from participant's or validator's stakes that lose dispute. But when all participants are honest there are no disputes and there is no incentive to own validator node. So we need to provide predictable revenue stream for validators even in cases when there are no disputes.

8.7.2 Verification game

In each dispute every validator may secretly receive 'lying ticket'. Lying ticket allow this validator to provide wrong result and not to be punished. Validators will not know which or if any other validator hold such ticket until dispute is finished and ticket holder reveal it.

Existence of lying tickets incentivise validators to actually validate blocks and not just put stake on result of others. If someone tries to put stake on lying ticket holder's result then all his stake will go to ticket holder.

Lying ticket is assigned by pseudo random algorithm with some fixed probability. So this generate guaranteed stream of revenue that keeps validators motivated to well-behave.

8.8 Privacy

All block data is encrypted. Keys are known only to channel participants and block validator nodes.

Channel configuration parameters may be also encrypted. Actually only hash of configuration could be written to blockchain. Actual configuration may be included in each block's header so it will be available to validators which may compare hash with that was written to blockchain.

In papyrus channel data transactions are processed and by validator nodes which include participant's nodes and nodes involved in dispute resolution. Participants may choose dispute resolution algorithm which defines how conflicts are resolved. This may include criteria for choosing additional validators, staking rules, timeouts and so on. Dispute resolution algorith is implemented as a smart contract in base blockchain. By choosing appropriate contract participants may balance between data privacy and processing consistency. Participants may even decide to not use dispute resolution at all in case they do not want to open their channel data to third parties.

8.8.1 Shielded transactions

Since ethereum already supports on-chain ZK proof verification it is possible to construct shielded zcash-like ZPPR token inside base blockchain. Block validators may generate zk proofs as result of block processing which will be settled to blockchain and trigger shielded transactions in ZPPR tokens. Conversion from PPR to ZPPR ant vice versa also may be performed by constructing appropriate ZK proofs and submitting them to blockchain. Since on-chain ZK-proof verification is expensive participants may decide whether transaction privacy worth increased commissions.

8.8.2 ZK-STARK

Truly privacy preserving computation would require algorithms like zero knowledge proofs, homomorphic encryption, or secured multiparty computations. These algorithms are still too expensive to apply them to individual transactions.

For ZK systems to be used with Big Data, it is required that verification process scales sublinearly in data size. Recently it was shown to be possible with ZK-STARKs to construct relatively small (200-400Kb in size) ZK proofs for computation over millions of entries with verification time <50ms with about 10x overhead comparing to plain computation. This approach may be used for resolving disputes without revealing private data.

Participants may construct zk-proofs of fact that processing of some block data will return some result without revealing that data or result. Since it is not possible to construct fake zk-proofs only one node needed to construct such proof. This zk-proof can be quickly verified by any validator in network. When enough validator signatures are collected this result may be settled to blockchain.

For this approach to work block processing algorithm requires to be transformed to special form from which it may be compiled to two programs: prover and verifier. Prover is used by one of participant's validator nodes to construct zk-proofs from raw data. Verifier is used on validators to check this proof. We expect some tool like ZoKrates to be developed in future for compilation of provers and verifiers for arbitrary algorithms.

8.9 Token Economy

PPR token is used for:

- Payments to validators in Papyrus chain for validating blocks.
- Participants may stake PPR tokens in Papyrus chain as part of Tendermint dPoS consensus and receive percentage of rewards.
- In channel layer: stakes and payment to validators during dispute resolution process.
- Could be used as payment token between Papyrus ecosystem participants

8.9.1 Stable Papyrus PPUSD

Stable token PPUSD is guaranteed to have value 1:1 ratio to USD by backing 100% of issued tokens by same amount of fiat currency. Functionality is similar to Tether USD.

PPUSD may be used as alternative payments token to protect from volatility.

8.10 Token Transfer

To transfer between papyrus blockchain and ethereum mainnet tokens we implement token exchange mechanisms. Special bridge nodes are used to monitor transfer requests in one blockchain and transfer it to another.

To initiate transfer in one blockchain participant needs to lock token by making transfer to special contract. It creates pending transaction. Bridge nodes monitor pending transactions and use lightweight BFT-based consensus algorithm to decide which transactions need to be delegated to other chain. When there are enough transactions collected they are submitted to other blockchain along with signatures of all validators. To cover gas costs and bridge node expenses each transfer included some fee that is split between bridge nodes.

To ensure transaction finality, the bridge may require additional block confirmations; typically major exchanges use 120 (half an hour) for transaction confidence.

8.11 Channel Node API

Channel node accepts messages by following gRPC service:

```
// Main channel interface
service StateChannel {
    // Creates or updates outgoing channel with given participant
    rpc RegisterTransaction(RegisterTransactionRequest) returns␣
↪(RegisterTransactionResponse);
}

// Registers transaction
message RegisterTransactionRequest {
    // sender address in HEX
    string sender = 1;
    // channel contract address in HEX
    string channel = 2;
    // block_number
    int64 block = 3;
    // encoded message
    bytes data = 4;
    // EC signature by sender's key
    bytes signature = 5;
}

message RegisterTransactionResponse {
}
```

Signature is calculated by following pseudoode

```
signature := sign(sender_pk,keccak256(channel,block,data))
```

Where sender_pk is sender's private key, channel - address of channel contract, block - is 64-bit channel block indentifier. In other words, sender's address should be recoverable by following solidity code

```
sender == ECRecovery.recover(signature,keccak256(channel,block,data))
```

Field "data" is serialized message according to channel type specification.

8.11.1 Message data serialization

For every channel message encoding format must be defined. Validators use information from smart contract to detect which message data serialization format (and validation logic) is used in given channel.

One of good approaches is to use google protobuf which provide efficient serialization and is easy to use from many languages. Protobuf description may be used on client side for message serialization and on validator side to implement validation logic.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

HTTP Routing Table

/campaigns

/statistics