University of Glasgow | School of Computing Science

Blockchain simulation and visualization

Michal Vinarek, 2148003v

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 28, 2018

Abstract

This paper introduces the reader into the problematic of blockchain: firstly starts with explaining the concept of blockchain and its usages for permanent data storage in distributed systems. It then considers the problems of trust, and describes common proofs of chain validity in a distributed network; and finishes the introduction by considering potential vulnerabilities and problems. Next, it introduces a simulation with a number of revised and new concepts (blockchain chunking, authorization-based access to data blockchain), as to aid the understanding of blockchain-based distributed systems, and puts the simulation into test under common usage scenarios. Furthermore, a real-world system of Electronic Registration of Sales of Czech government is used to indicate the types of systems that might be able to utilize the network in its current form, and performs limited testing. Lastly, future work and limitations are discussed to indicate direction of this project in the near future.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

With an increased media focus on blockchain-based technology (mainly on cryptocurrencies), many do call blockchain 'the technology of the future'; and with the excitement of technological industry and investors, it is paramount to properly describe what blockchain actually is, and what its capabilities and shortcomings might be in a range of use cases. The excitement around the blockchain is understandable, however the concept cannot be used for every system and for every usecase some inventors might come up with. With the great power of blockchain comes the responsibility to understand the underlying technologies and their implications on intended usage.

1.1 Motivation

Various companies and groups have explored the blockchain concept as presented by Bitcoin, and came up with different usages and use-cases: ranging from with store of value in Bitcoin-like networks, distributed computing and smart contracts of Ethereum [6], domain-management using Namecoin, fund raising for companies via ICOs[1], distributed application development through EOS [25], distributed file storage using Storj [46], to enterprise-grade Hyperledger framework from Linux Foundation [28]. These uses can not only be a bit difficult to grasp for the lay public, but it is also harder to predict how the network will behave under certain conditions. Suppose that we would have a simulation of Bitcoin before its prices spiked in the recent past: would we be able to better predict how would these spikes affect the price for using the network to transfer money?

Given the fact that private companies or public bodies require a degree of control over some of their data, they might be more interested in access-controlled blockchain solutions rather than fully-public networks; however given the fact that each institution might require a different implementation of the system and its behaviours, it would be beneficial to have a simple, configurable, and relatively resource-light testing environment.

1.2 Aims

The aim of this project is to develop such a testing environment, specifically a generalized access-based blockchain simulator, which can be tweaked and tailored to investigate network behaviour in different scenarios. Such simulator will be 'naive' in its implementation, and will focus on the interaction between the nodes and emergent

[1] Initial Coin Offering, used to distribute shares for newly-founded companies

behaviour; it will not be concerned with the actual data flowing through the network, nor will include any incentives (like cryptocurrencies) - node engagement in the network is thought to be handled outside of the scope of the simulation, for example by a company requiring all its suppliers to use it.

The simulator will be written in Java, and is able to simulate networks of sizes that would otherwise be impossible to test on one machine; and will include specialized data outputs and graphs to further illustrate the behaviour of the network. This will enable not only programmers, but also less technically experienced decision-makers to better understand the implications and usages of such system.

The abilities and shortcomings of the network will be tested using a real-life scenario of a system, that could potentially benefit from using the access-controlled blockchain.

Chapter 2

Background

The network follows complex behaviours of blockchain, and as such it is important to initially define the different technologies, their strengths, and their weaknesses.

2.1 Blockchain

Blockchain is the technological foundation of all today's cryptocurrencies; in fact, it was initially proposed to be the solution of distributed trust-free system for financial transactions [33]. How exactly does it work?

Every bit of data being sent through network needs to be encapsulated in a 'transaction' object, signed by the sender to establish validity. These transactions are propagated in a network via other nodes, and wait in 'transaction pool' of each node; depending on network requirements, the transaction can be rejected by other nodes if it does not follow validation criteria - for example, in a monetary transaction, if the sender does not have enough funds to satisfy it.

These transactions need to be consolidated in a 'block', which serves as a main storage of the data - it is, by its nature, a linked-list of block objects, each containing a list of transactions. In order to establish such a linked-list, a chain similar to a hash chain is created, illustrated in Figure 2.1.

The data structure allows for a degree of trust to be established; it can be done so by using hashes of previous blocks, as all previous hashes are, in fact, included as well through hash of a block preceding the previous one. Should an attacker want to change data in any of the blocks, the task becomes progressively more difficult: not only would they need to recalculate the hashes of the whole chain succeeding the changed block, but they would

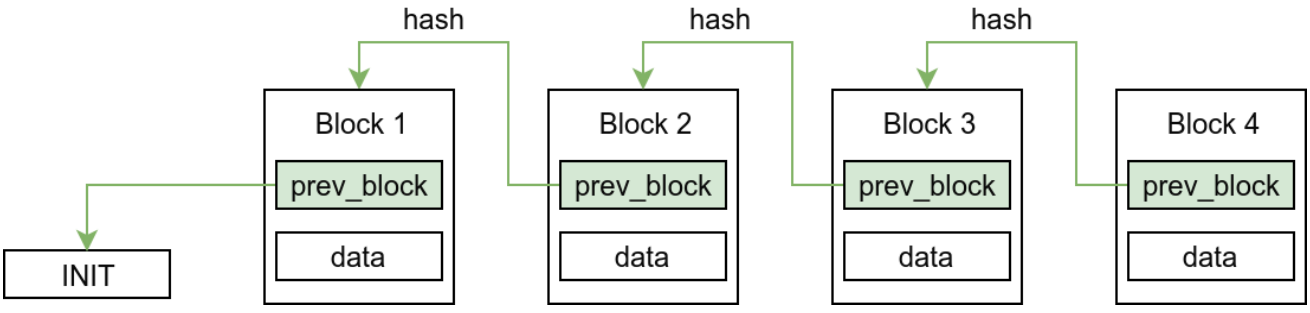

Figure 2.1: Basic blockchain structure

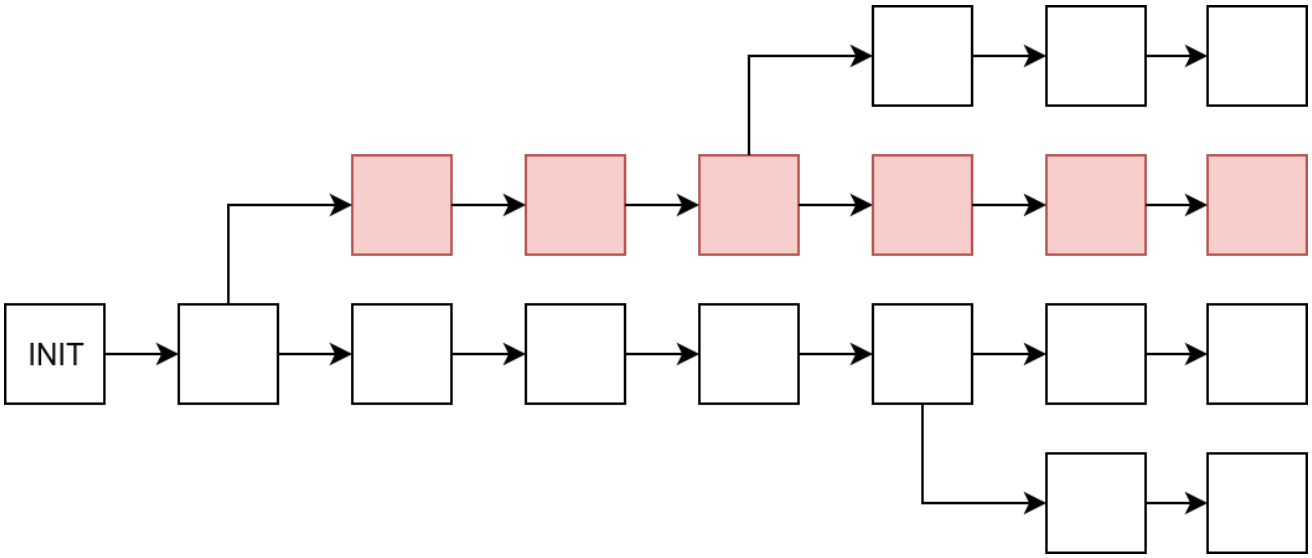

Figure 2.2: Model blockchain splits

also need to make sure other nodes would accept the newly created chain head. In fact, the attacker has now created a ‘side-chain’ in the network, as indicated in Figure 2.2

What is noticeable is the number of splits apparent on the chain - in fact, even the malicious chain can split by a regularly-acting node, and can form a part of a new, legitimate chain. So how do the nodes know which chain to trust?

2.2 Network trust and consensus

Trusting strangers with important data is not easy, let alone trusting tens of thousands of strangers all around the globe to handle the data the way they should be handled. Research suggests that every person is capable of maintaining social circle of between 100 and 230, commonly around 150, people [27] in which they can place a degree of trust – and so it follows that trust is not easily scale-able. The network and blockchain enforce trust through mathematical means and precisely defined algorithms. Motives of people, organizations, and even governments can be a subject to change, but the basic mathematical laws are not changeable.

The first - and already mentioned - source of trust is the hash chain. By including a hash of previous block it is possible to establish a proof of an unchanged chain of blocks. The only trust issue in this system is the initial block, to which all blocks on the network eventually point to (see ’INIT’ in Figure 2.2). This problem can be solved by publishing the source block on the internet; in fact, Blockchain network contains a ‘Genesis block’ with a newspaper title from 3rd January 2009 [5]. By hashing the contents of the block and by pointing every chain to it, the nodes can always be sure that the chain itself is potentially a valid one.

With trust at the origin block established, how does a node determine which chain is the valid one? Upon the creation of Bitcoin, a protocol of ‘proof-of-work’ (PoW) was used for its network. This validation is by no means a new invention - Rai stones can be an approximated example of a proof-of-work protocol. These lime stones were used as a currency on an island of Yap [20], which does not have any natural reserves of limestone; and as such they had to be quarried on another island and brought in on rafts. A larger stone requires more work in order to be quarried, transported, and chiseled, and so its value is more-or-less directly observable. For blockchain, a block nor a chain can be objectively measured other than by their lengths - the longer chain must be the most trusted one, since so many blocks were built on top of it, and so much work was invested creating

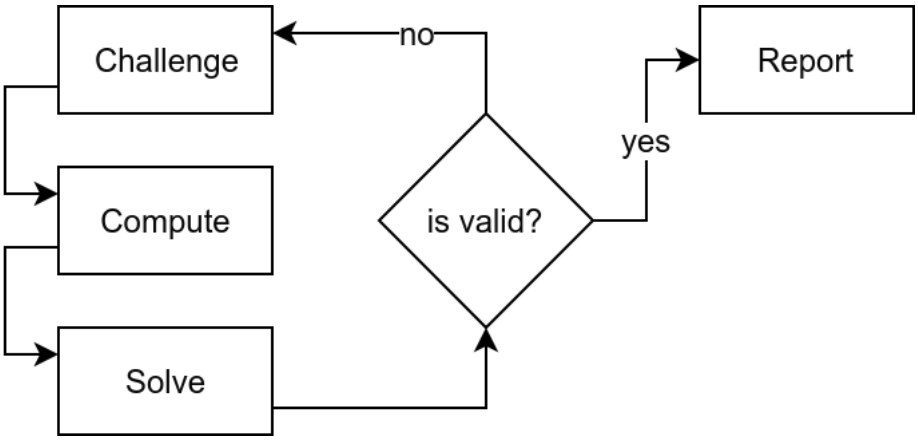

Figure 2.3: Proof-of-work solution-verification model

$$sha256(sha256(blockContents))$$

Figure 2.4: Proof-of-work challenge in Bitcoin network

it. But how to define a metric of 'work investment' for computers? The first cryptocurrencies exploit a PoW protocol of 'solution-verification' [8][29], indicated in Figure 2.3.

The computer is given a pre-defined challenge, computes it, and compares the output to the expected parameters; if they match, the challenge was solved and needs to be reported. The PoW of Bitcoin network uses a version of Hashcash PoW system [2][33], which was originally created to combat spam by forcing computers to create hashes of strings with easily-comparable validations; as such it has taken a computationally-difficult task used to deter attacks against a system, and used it for validation: Bitcoin uses the algorithm described in Figure 2.4 to determine if a challenge has been solved.

Block itself contains a 'nonce', which can be set to any number in order to force hash changes. Validity is checked by matching the resultant hash to a 'difficulty measure', which specifies with how many zeros the hash must start. To ensure relatively-stable block creation, the network decides on the difficulty level so that a block is created roughly every 10 minutes. Any node can easily check if the criteria have been met by looking at the number of zeros at the start of double-hash of the block. Other cryptocurrencies using PoW employ different hashing algorithms and their combinations, however the basis remains the same. As such, a longer chain in a PoW-based blockchain constitutes a larger investment in computing power, and by an extension shows the trust other nodes put into the chain. It is still however possible to see a chain split for a few blocks, as it might happen that more than one node solves the challenge at around the same time before they receive each other's blocks. The issue will be solved by simply waiting for a longer chain to form.

PoW is a computationally-demanding task, and since modern computers can easily compute millions of hashes per second, the network will have to compensate by increasing its difficulty, and causing the nodes to spend more computing power on block creation - in fact, in 2014 a research compared power consumption of the network to that of Republic of Ireland [36]; currently, the network is estimated to use 56 TWh of electricity each year (as of March 20th 2018 [9]), and was estimated to have used only 42 TWh of energy mid-January 2018 [26], which represents almost one third increase in just two months. Due to the economical and environmental impact, many different ways of selecting new block creator were created in response:

Firstly, Proof of stake with coin age, which uses a partial PoW process, but takes in account age of coins held by the creator instead of hashing power of the creator - and the network difficulty is significantly lower than PoW-pure networks. It was the first proof proposed outside of PoW, and used by Peercoin network [30].

Secondly, proof of stake with a randomized block selection, accounting for the amount of coins held in one account; the more coins an account has and the longer since it last created a block, the higher chance it has to gain rights to 'forge' a block. A known implementation can be seen in Nxt network [35].

Thirdly, proof of stake velocity factors in the movement of the coins besides the coin age (hence the 'velocity'). It effectively forcing users to spend the currency to have a higher chance of creating a block; its use can be observed in Reddcoin network [41].

Fourth, proof of burn, which uses a process of 'burning' coins to gain a higher chance of creating a block, an alternative to waiting for coin age to accrue, and is partially used in Slimcoin network [42].

Finally, proof of capacity, which utilizes storage capacity as a measure of the node ability to contribute to the network, and is utilized by Burstcoin network [21].

Further variations and specific approaches exist. These above were selected to show the different proofs a network may use instead of PoW to establish a network consensus. The proofs can be used for a very specific use case; for example, proof of capacity may be extremely useful for a decentralized file storage network.

2.3 Network weaknesses

Even though blockchain provides an easy and verifiable way of confirming chain validity, the network itself is vulnerable due to its decentralized nature.

P2P[1] networks are in danger of experiencing a Sybil attack, which is performed by creating a large number of 'fake' nodes in the network. The attacker can afterwards disrupt communication within the network, isolate clusters of nodes, and send falsified data to other nodes. These attacks are proportional to cost of establishing a node in the network; in fact, a decentralized Tor network, used as a hidden layer of internet and to hide its users IPs, has been reported to have been under attack for about 5 months in 2014 [23], and the perpetrators could have tried to uncover user identities using previously discovered bug [24]. While measuring a Bit Torrent Mainline DHT network (used for P2P file-sharing), they have managed to record a Sybil attack on the network in 2011 [44], and have measured the effects on the number of nodes and load of the network [45]. For PoW blockchain network, such attacks do pose a thread by isolating nodes. Nevertheless, it is not easy to spoof the blockchain of the whole network exactly because of the work that is needed to create it. Bitcoin network transaction propagation susceptibility to Sybil attacks was explored in 2012 [1]. PoS itself is also susceptible to Sybil attacks [40], although different networks may behave differently according to their PoS parameters. Various versions exploiting delayed communication and spoofing can be made, for example by isolating subgroups of balanced mining power, and by that artificially create two or more competing chains [34]. Consequently, highly-fractured networks are at a higher risk than more compact ones.

Another type of attack is an attempt to take over control over PoW network (commonly known as 51% attack). Should an attacker attain more than a half of nodes capable of creating and validating blockchain, the other (and valid) blockchain would be phased-out by superior hashing power of the attacker. Even though the attack is unfeasible for one party attempting to compromise a large network, nodes can (and do) aggregate into large 'mining pools', which appear as one entity for the purposes of block creation. Litecoin network (second oldest cryptocurrency) had F2Pool controlling over 51% hash-rate just last year [18], and Zcash network is largely controlled by Flypool as of March 2018 [47]. Although the pools might not have a reason to tamper with the network (if honest miners comprise the majority), they certainly have the possibility. As a matter of fact, a research in 2014 has suggested that a pool does not necessarily need to attain 51% of the network's hashing

[1] Peer-to-peer networks are distributed networks, where communication pathways go straight between two nodes without using a central server.

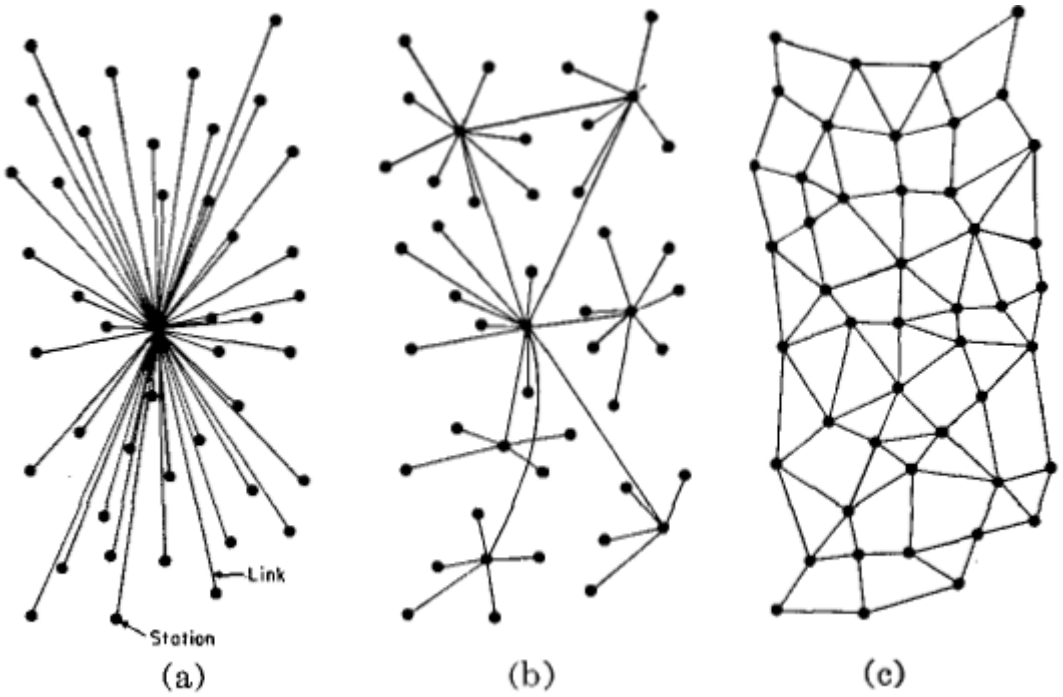

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Figure 2.5: Network types, figure used from Paul Baran's paper [3]

power, but in case of 'selfish mining pools' can be as low as 25% [17]. As such, it is important to point out that this type of attack is feasible, but costly for large networks, and does pose a larger thread to smaller and less active networks.

2.4 Network structure

To better understand the networks, their consensus, and weaknesses, it is important to recognize and define the different types of networks and their behaviour. As described by Paul Baran in 1964 while forming basis of distributed communication models [3], we can distinguish three broad types of networks, as shown in Figure 2.5

By briefly looking at the diagram, one might immediately jump to a conclusion that blockchain must always be a part of a distributed network. Most cryptocurrencies are, by their nature, heavily distributed; however some are very centralized - take Tether network as an example. These cryptocurrency tokens are redeemable into US dollars at 1:1 ration with Tether tokens, all through the company that backs-up the network [43], and in effect acts as a central bank. Therefore, it should not be assumed that blockchain technology cannot be used outside of distributed networks.

Through comparison of the three network approaches, it is possible to see that, firstly in the most apparent measure - their inter-connectivity and communications. Centralized network does have a central point of communication, which can be either a single server, or even an whole new sub-network created to handle requests - a fitting comparison would be a load-balancing server of a popular website, which redirects every client traffic through itself; or even private sub-nets, which have exposed public access points, and an unknown number of network points within. A decentralized solution breaks up the single access point into localized access points, which communicate between each other; and finally distributed solution breaks up communication away from any centralization, and nodes communicate with each other.

The drawbacks and advantages of each network can be relatively easily summed up: the more decentralized the network is, the less control is given to the entity that has created the network in the first place. On the other hand, splitting the network into more independent parts brings an increased redundancy – should one of the localized points fail, the network itself can still function to some degree; all the while the distributed network offers an unprecedented amount of connection and data redundancy: should one node fail, it is theoretically possible to connect to any other node and get the very same information. After all, that is exactly the theory behind Bit Torrent protocol and shared data networks [39] - some nodes contain the same data (an ISO image of Ubuntu, for example), while other nodes are trying to fetch the data. If one of the offering nodes should suffer an outage, the other nodes still hold the data, and the requesting nodes can just switch connection. The only difference in cryptocurrency networks is that every node holds every block ever created. And that may cause a difficulty.

2.5 Data redundancy and ownership

Data redundancy in a more distributed networks seems to be an excellent idea - should a part of the network be taken offline, no data is lost and the network can continue working. However, after a time size of data shared between all nodes becomes a problem. As of March 2018, Bitcoin's blockchain size comes to around 152.0 GB [4], and Ethereum's unpruned blockchain comes to around 325 GB [15] - and using a fast synchronization with pruning[2] comes close to around 59.5 GB [16]. As such, it is clear that increased data redundancy has its drawbacks in the amount of data each node has to store.

Data ownership and control over the data is quite simple in a distributed network - once the data is sent to the network, it might never disappear - a simple example might be the illegal content being shared through P2P networks, as when some nodes have at least parts of the data, it can still be reconstructed. The only solution to deleting data from such network would be to bring all 'compromised' nodes offline to ensure the data will not be transmitted further. Using blockchain in distributed networks points out another problem with data control: once the data is written inside the blockchain, it can technically never be removed, as by removing the data the whole blockchain would need to be recomputed. In fact, recent research has shown there are around 1600 files stored within blockchain, some of which contains more than questionable (and potentially illegal) content [31]. The data is now downloaded by every node that wishes to participate in the network, since it needs a whole copy of the blockchain to fully verify future transactions. As such, compared to P2P file-sharing networks, distributed blockchain does not forget data, and the problem cannot be solved by taking the nodes transmitting the questionable data - it is now a part of the network until the network itself ceases to exist.

Furthermore, if a company would decide to use the a blockchain-based storage (for example to store customer data), privacy laws need to be considered. Once again, blockchain cannot delete data from past blocks unless the whole chain is re-calculated, and should the company store personal data (even locally and without public access), these may have to be deleted under upcoming EU's General Data Protection Regulations under 'Right to erasure' [22]. Consequently, if a company owns the blockchain and allows anyone to create a block with any information contained within, it may be liable for the contents. As such, it is important to discover different levels of control over the contents of the blockchain, while embracing the positive effects of a robust distributed network.

[2] Pruning is a process of cleaning up unused block content without compromising block hashes

Chapter 3

Simulation design

After considering the advantages and disadvantages of blockchain networks, a generalized simulation was created in order to better visualize and investigate behaviours of the network; and the simulation itself proposes some improvements on network design, redundancy, and stability.

The simulation is rather naive in its implementation, as it does not consider the increased security measures that current blockchain-based networks use (for example transaction signatures, etc.), however it does include all the necessary checks for a stable testing environment and blockchain simulation. As it is meant to be generalized as much as possible, the features can be easily added should another programmer express the desire to test specific behaviour or scenarios.

The network is governed by a tick-based clock instead of following internal clock of the machine; and as such it is possible to simulate a large number of nodes even on a regular computer, all without the need to resort to multi-threaded simulations. If the simulation were to assign a new thread to each node, it would be near impossible to simulate a network with tens of thousands nodes: it is likely that communication between threads would suffer from random delays, producing a different result after every iteration of the same simulation. Tick-based system removes the overhead of inter-thread communication, and produces the same results for the same input parameters every time. By extension, it can be assumed that every node is its own machine, and can potentially assign thread limit as high as the host machine, and thus there is a possibility that the simulation would be able to handle only one node at a time. Using tick-based approach the network can simulate a high number of nodes, each with any number of threads "running" inside it.

Furthermore, any random action is guided by a seed random number generator (RNG). Declaring a seed adds a layer of reproducible results, as all random actions use the same RNG; and since the simulation uses its own internal clock and simulates nodes consecutively, the order at which nodes access and use RNG will always remain the same regardless of system variables.

The section will begin by describing basic structures, then will move onto their behaviours and interactions, and will conclude with explanation of emergent network behaviours.

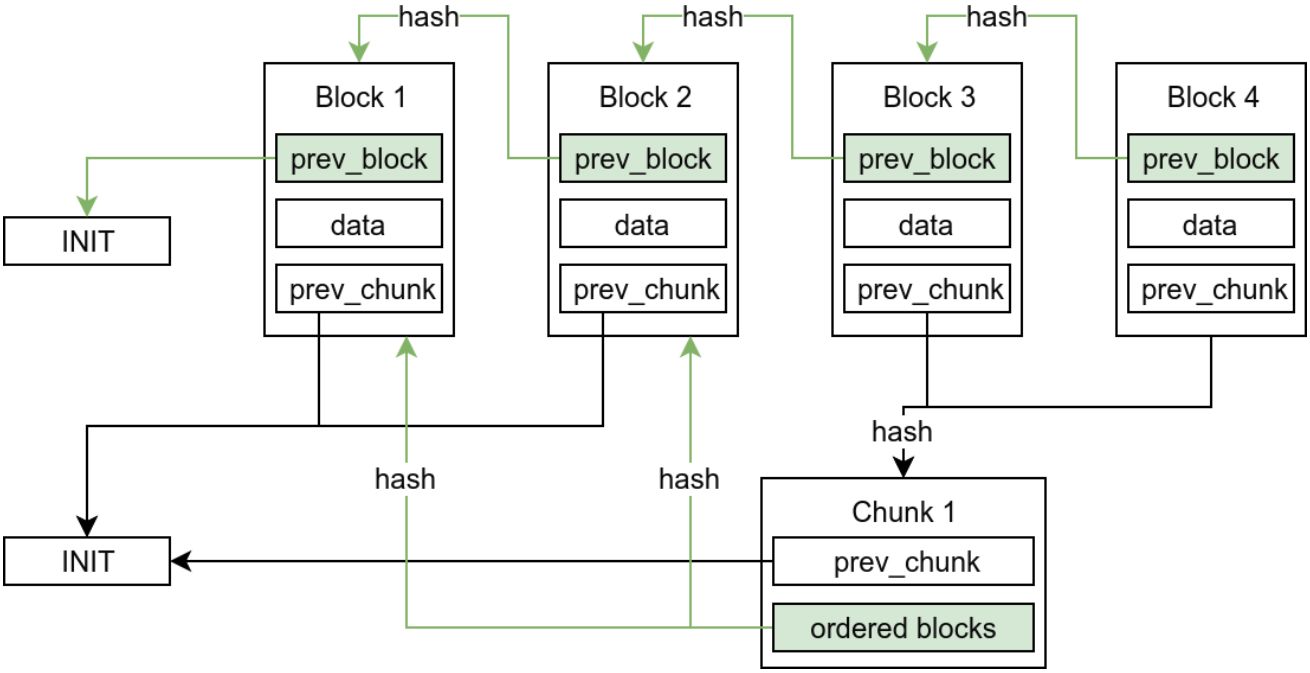

Figure 3.1: Blockchain with block chunks

3.1 Basic structures

3.1.1 Blockchain

The network uses the very same blockchain structure which was mentioned in the Background section; however, the simulation introduces a concept of 'block chunks', as indicated in Figure 3.1

These chunks serve a role of de-facto another blockchain on top of the existing data blockchain, all while the original blocks point to the chunks as well, all with the size of the chunk configurable depending on the needs of the network (e.g. every 100 blocks). This establishes an additional hash chain relationship, and opens up possibilities for a faster block lookup - in fact the implementation or ordered_blocks field through LinkedHashSet allows for a best-case lookup time of O(1), and the worst-case of O(n). This behaviour is important should it be needed to look forward in blockchain: blocks are linked backwards, and thus by searching the chunks and their ordered_blocks field, it is possible to fetch a block hash of the next block much faster, which in turn allows the other nodes a simple lookup into their block databases, instead of the need to iterate backwards over the blockchain.

Compared to a block creation process, the chunks are not created and transmitted by the creator node, but instead are an emergent object from prev_chunk fields in the main blockchain. Should the node happen to not have a chunk chain list, it can easily reconstruct it using the node chain. The block template the simulation uses is as follows:

```
block_height : creator : previous_block_hash : prev_chunk : payload
payload = transaction(s), separated by '||'

(template does not use spaces around ':' separators)
```

The hash of the block is computed as `sha256(block.toString())` with the .toString() template above.

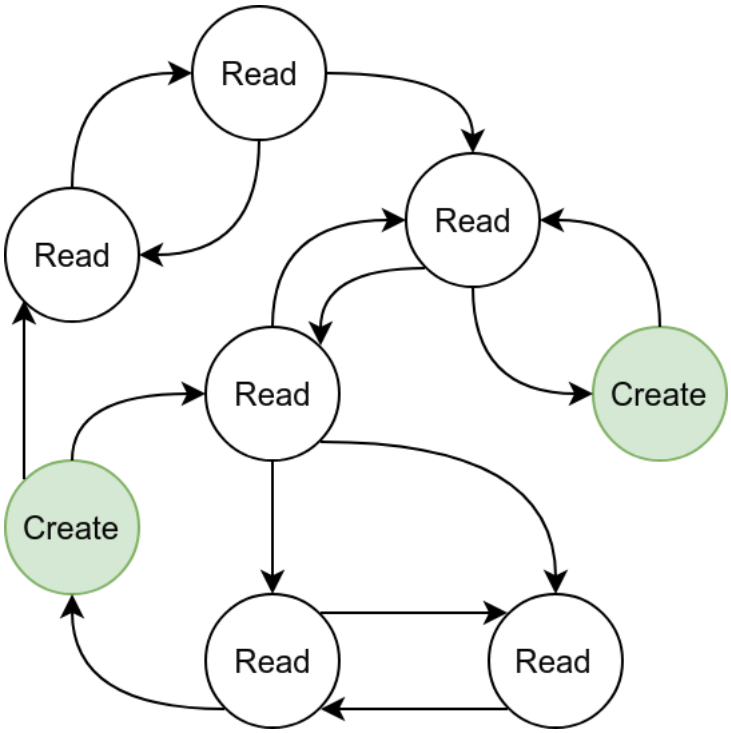

Figure 3.2: Example of a network with different nodes and permissions

3.1.2 Authentication blockchain

The data blockchain is internally protected by the network, and as such a form of authorization needs to be established to control which nodes can commit changes to the blockchain. The simulation achieves this by introducing a secondary 'authentication blockchain', which is publicly-readable. It specifies 4 different levels of access to the internal blockchain:

```
0 - read
1 - publish (transactions)
2 - create (blocks)
3 - create auth (blocks)
```

By using this system the network nodes can determine which nodes have the permission to perform an action on the protected internal blockchain. The network can have a default permission for all nodes, and as such it is not needed to keep a list of all nodes' permissions, but only to keep the higher permissions in record. Any high permission can automatically perform any actions of the lower levels, e.g. 'create' can also 'publish' and 'read'. An example of such a network can be seen in Figure 3.2.

The structure of the blockchain follows a simple one described in Figure 2.1, and contains no block chunks. The block template the simulation uses is as follows:

```
block_height : creator : previous_block_hash : payload
payload = transaction(s), separated by '||'
```

```
(template does not use spaces around ':' separators)
```

The hash of the block is computed as `sha256(block.toString())` with the .toString() template above.

3.1.3 Transactions

The transaction objects are a simple data carrier, and are used both in data and auth blockchains. Every transaction follows the template of:

```
sender_id ; payload

(template does not use spaces around ';' separator)
```

The hash of a transaction is computed as `sha256(transaction.toString())` with the .toString() template above.

The payload cannot contain `||` (double pipes), as it would interfere with the block payload separator, however it can contain any other character including `;`, which is a separator of sender_id and payload. The transaction parser takes this into an account, for example

```
trans = node_1;<xml>hello;world</xml>
    ->
sender = node_1
payload = <xml>hello;world</xml>


trans = node_1;{field1: "text", field2: "hello;world"}
    ->
sender = node_1
payload = {field1: "text", field2: "hello;world"}
```

3.1.4 Signals

Signals serve as communication wrappers in the simulated network, and follow the template of:

```
sender_id forward_id payload

(space is considered as delimited)
```

ACTIVE signal

Serves to signal other nodes the 'liveliness' of a node. Is transmitted on both up-links and down-links of a node.

ACK signal

Serves as a simple acknowledgment signal, and is sent back in response to receiving any regular signal.

NEW signals

A wrapper around Transaction, Block, or Auth Block data, signifies that a node has either created or received respective data from other nodes in the network. These signals are meant to be propagated further.

REQUEST signals

Signals used to fetch Blocks, Block chunks, and Auth Blocks from other nodes. These signals are not expected to receive an ACK signal back, but rather a RESPONSE signal with status.

RESPONSE signals

Signals used to let the state of a REQUEST signal known to requesting node. A node that has received a request signal either reports back with data requested, WAIT signal if it tries to fetch it from its neighbours, or a DENY signal to terminate search through this node.

3.1.5 Nodes

Every node is a self-sustained system within the network, and the simulation tries to emulate such a behaviour. To achieve 'concurrency' between the nodes, the network uses a concept of jobs - these are data wrappers above NodeSignals, and simply state which node should receive them and when. Every time a node needs to send signal to a node, it will create a job through the network, and will assume the signal has been sent correctly.

The node execution process has 3 separate steps, and that is reflected on the structure of the class. For any tick, the network firstly sends any pending signals to nodes (it will have precise send times thanks to the jobs wrapper class). Each node has multiple open connections, and each can have a certain number of threads assigned for processing of incoming signals, the pseudo-code algorithm can be seen in Figure 3.3.

After all the signals have been 'sent', the simulator will try to invoke the main thread of the node. The network class keeps track of next availability of every node thread, and will only execute it when the time comes. The main loop contains the logic outside of signal parsing, which is mostly connection refreshes, ACTIVE signals sending, block clean-ups; most importantly the main thread can have a specific action assigned to a node, so it can create new transactions, blocks, etc.

Lastly, after all the main threads are finished executing, the simulation gathers all signals that need to be sent out from all the nodes - in fact it will gather all created jobs with specific delays, and will use them in subsequent network loops. That also means that every node action is controlled via the network class, which gives the system a high degree of centralized control. All the nodes have to use the network, as they cannot directly call another node.

Each node contains its own configuration, which specify its behaviours and connectivity to other nodes:

```
max upload speed
max download speed
max incoming connections
max outgoing connections

threads per incoming connection
```

```
for each signal received/sent:
   add to queue

for every available socket:
   if has available threads:
      remove job from queue, parse signal
      update thread availability
```

Figure 3.3: Node signal accept loop

```
threads for main loop

connection stale threshold
block suspicion threshold
```

Using these settings it is possible to create vastly differently-behaving nodes, for example a high-capacity node with a large number of active connections, and a small, mobile-like node with minimal amount of connections. It is also possible to influence the behaviour of the network itself, as each node contributes according to its capacities.

The node keeps track of some of network properties, and has limited understanding of its surroundings. The data is usually kept in HashMaps or HashSets, which have a potential to achieve O(1) complexity as a best case (O(n) as worst). Depending on the settings of the auth blockchain, it can have full knowledge of number of nodes in the network (if the network requires ALL nodes to have a READ permission in the auth blockchain); otherwise the node will have only a limited knowledge. It can discover other nodes via the network, and can query the connection costs between those discovered nodes, however it still cannot make any definite assumptions about the network as a whole.

The limited knowledge is the reason why a node must interact with other nodes, and by extension allow the network to exist. These interactions will now be explained, along with why the node contains the specific configurations.

3.2 Behaviour and interactions

3.2.1 Node connections

In order to function properly, each node needs to connect to others to form a loosely-structured network of individual nodes. As explained above, each node will achieve this by calling 'peer discovery' through network wrapper. The network will predictably yet randomly (using seeded RNG) shuffle existing nodes, and will try to establish a connection between them; should it fails so the requesting node will try again as soon as it can. Moreover, each connection will incur a specific cost, which is unique to the two nodes: in fact, when the node was added to the network wrapper class, it was assigned a 'cost' (again, using seeded RNG) within a specified range (default: between 10 and 100 ticks). After a new connection is to be made, a link cost is computed using the mean average of the connections:

$$linkCost = \frac{costNode1 + costNode2}{2} \tag{3.1}$$

```
for each signal received/sent:
   get bit size, add to counter

every preset number of ticks:
   utilization = bit_counter/ticks_since_last_check
   refresh_connections(), reset counters

refresh_connections():
   if max_speed > utilization:
      add new connection
   else if max_speed < utilization:
      remove new connection

   if has stale connection:
      add timeout cooldown, disconnect and replace

   if random and connection_count > min_connections:
      disconnect and replace random connection
```

Figure 3.4: Node connection handler

The node will automatically determine the number of connections it will attain using its internal limits (max connections) and the forced minimum of 2 upload/download connections. It can happen that a node will have its connections dropped at random: this is actually expected behaviour, as nodes try to refresh their connections relatively often.

Every node also contains connection counters, which are used to compute the aggregate utilization of both upload and download connections in between connection refreshes, after which they are reset. Through this the node can determine the optimal number of connections automatically, so as to saturate the connections to optimal levels.

An overview of these behaviours can be seen in pseudo-code form in Figure 3.4.

Why were these behaviours implemented like so? The automated connections help the network through DDoS attacks, as if the node record unusual traffic, it will automatically close some of its connections. Sybil attacks are also less likely, since nodes try to refresh their connections; and for example if the connection does not send ACK signals back, the connection will be deemed to be dead and will be dropped; it also prevents long-term cluster separation, as the connections are dynamic. Using these approaches does not completely remove the risk of attacks occurring, and further limitations are discussed in the chapter 6.

3.2.2 Signal propagation

Each node can maintain full knowledge of some of the states of the network, particularly the transactions and blocks are important to keep track of. In order to inform all nodes as fast as possible, each node will propagate every received signal to its connected neighbours. Of course, simply broadcasting every transaction and block signal would cause the network to quickly flood, as it would in fact perform a DDoS on itself, hence there are safeguards to prevent excessive signal propagation, using the algorithm from Figure 3.5 to determine which signals should be forwarded.

It is of a particular notability that there are different forward rules: new discoveries of transactions and

```
for each signal received:
   send ACK if not REQUEST or ACK

switch signal.type:
   NEW_*: parse data, if valid create_forward()
   ACTIVE: update last_seen of node, if WRITE create_forward()
   REQUEST_*: if has data, create_response(), else handle_request()
   RESPONSE_*: handle_response()
   ACK: handle_ack()

create_forward():
   if not received_in_recent_past and not already_forwarded:
      create forward signal, send

create_response():
   fetch data, create signal and sent

handle_response():
   switch type:
      DATA: if not_already_reported then create_response(DATA)
      WAIT: if all connections WAIT, create_response(WAIT)
      DENY: if all connections DENY, create_response(DENY)

handle_ack():
   switch in_response_to.type:
      RESPONSE: delete_forward_signal() after delay
      NEW_*: mark as received in forward_signal
      ACTIVE: mark node as active
```

Figure 3.5: Node signal handler

blocks get forwarded with no change; ACTIVE signals from nodes with WRITE permissions are also forwarded without a change in order to inform other nodes of the core nodes; and REQUEST signals are not forwarded, but are rather formed as new signals to be broadcast to neighbours.

Even so, it would be rather easy to cause an ever-repeating loop, and as such a behaviour similar to a split-horizon rule [7] is used: no node will forward a signal to node from which it has received the signal in the first place. Furthermore, a temporary cache is established for transaction signals in particular – it could viably happen that the same transaction would be received more than once through different pathways. Since the node does not know how the network looks like, it cannot guess whether the transaction should be received and forwarded or not: it uses a list of transaction hashes, which signify previous delivery of the same transaction; in fact it is possible to see this check in Figure 3.5's method 'create_forward()'. Additions to this list of hashes are cleaned after enough blocks are received, which is by default threshold of 5 blocks. By using this approach it is possible to reduce duplicitous communications in the network.

The nodes do not actually know if their forward signal was forwarded further or not, and that is again simply due to the dynamic nature of the network – if the node were to make assumptions about where to send its forward signals, there is a potential that some nodes would never receive the data.

3.2.3 Network consensus

Given that nodes have limited knowledge of the network, how can they create a network-wide consensus? Since auth blockchain is used, each node will construct its own list of expected block creators, and will follow a round-robin approach towards block acceptance; conversely, every block writer node will do the same check before creating a block (ie. waiting until previous creator's block has arrived). By using this approach the creator's turn is undisputed.

But what if the next creator should happen to go offline? Firstly, this is mitigated by having a central party controlling the access to the data blockchain and by having external incentives. Node can be determined to be usable by requiring a certain uptime guarantees, and granting permissions only if the guarantee is kept. Secondly, each node does have a handler for situations when a creator is offline – that is exactly why ACTIVE signals from WRITE nodes get propagated throughout the whole network. If a node observes a creator did not send ACTIVE signal recently, it can decide to wait for a while until a threshold for creator skip is reached. Should that happen, a node will consider next creator as a valid block creator. Each node also contains handlers for 'possible' blocks, which can potentially form a part of the main chain. Should it happen to receive a block, which does not point to current block's hash but was created by a creator just a few hops ahead in the round-robin list, it will keep the block in the memory. After it receives a valid block to fill the gap (or after the above specified time-out occurs), it can search its memory and advance the chain by multiple blocks at a time. It is important to note that a node will not forward any potential blocks, as some of them may be potentially malicious, given as an example in Figure 3.6. It is possible to see that the first example shows two potential chains to advance the main chain to, and only after receiving the missing link a node can decidedly determine which blocks are valid and which are not, and propagate them to its connections; should it propagate blocks before validating them, connections would be needlessly used to transmit invalid blocks. The second example shows the same behaviour, but where multiple fragments of the chain exist. Again, a node does not know whether the two fragments represent two competing chains, or if they form one chain with some blocks missing in between - and only after it receives those missing links the fragments will get propagated to its connections.

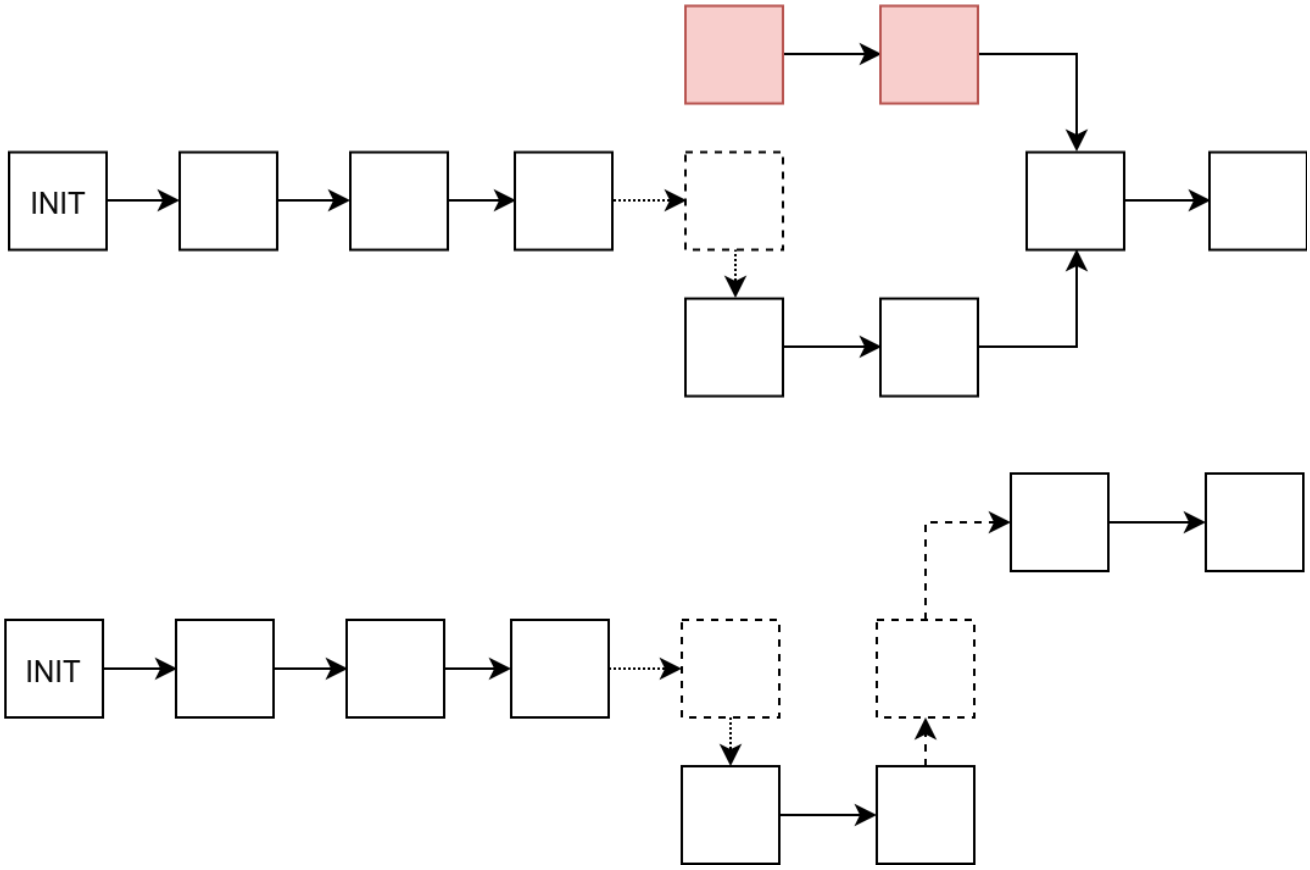

Figure 3.6: Two examples of future chain fragments

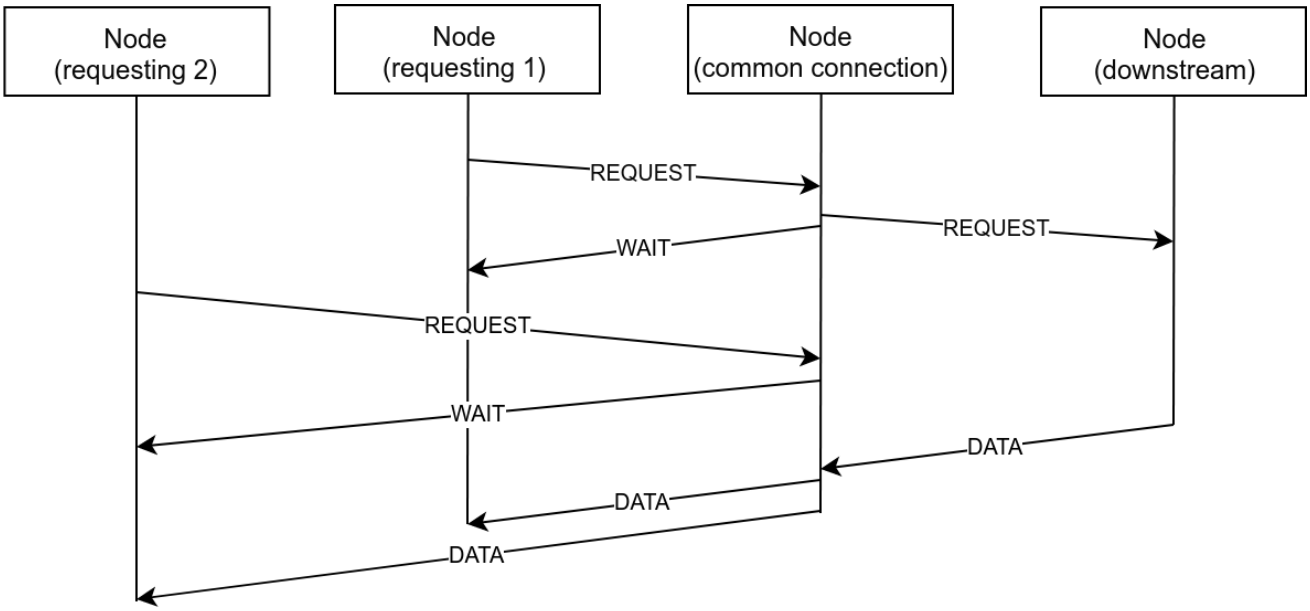

Figure 3.7: Example of signal propagation and aggregation

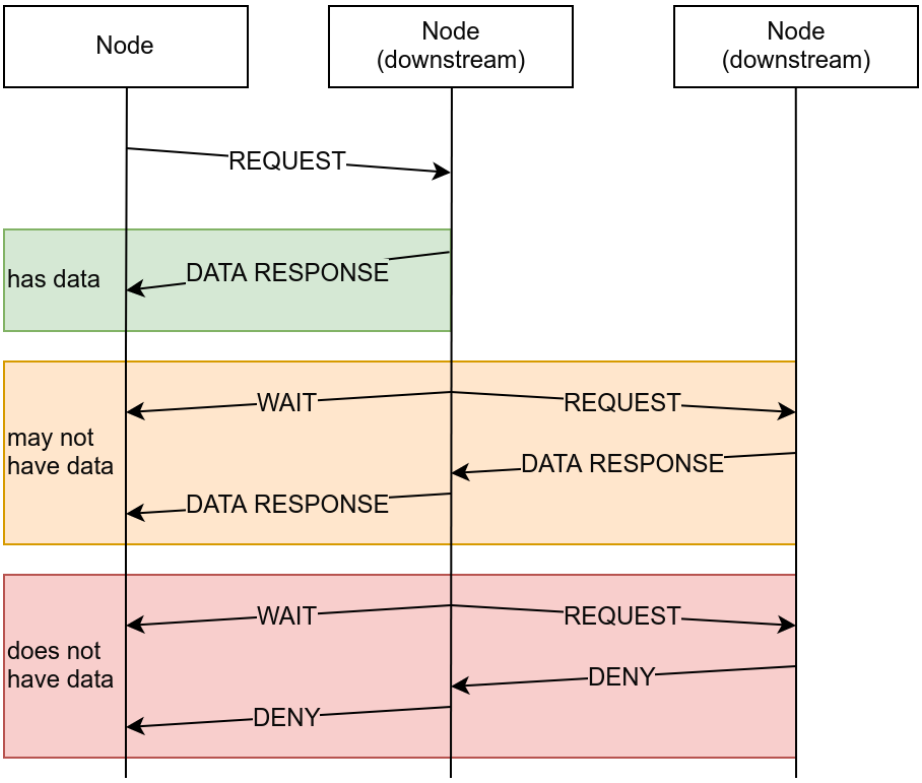

Figure 3.8: Example of a block look-up

3.2.4 Data requests

The request/response signals function on a very similar principle to regular forwarding: a request is never made to the previously-requesting node, and should the node receive the same request from another node, it will treat both requests as a singular entity. As such, it is possible to have multiple different response times for the same data request onto the same node, barring that each requesting node submits their query at different ticks. The overview of this behaviour can be seen in Figure 3.7: by using the aggregation requesting node 2 receives the same data as requesting node 1, and the data is pulled/requested only once for both requests. In fact, the nodes perform a graph search over unknown distances, and aggregate results received.

This approach allows the network to maintain block accessibility while lowering the storage needs - a truly distributed storage with redundancy measures. Potential advancements in this area are discussed in Chapter 6, and a general overview can be seen in Figure 3.8

In fact, it is possible to combine the requests with proposed block chunks to maintain even larger communication savings; since block chunks contain hashes of blocks which have been placed on the chain are were consented by the network as a whole, it is possible to use them as a short cut for new node synchronization, using the following algorithm:

```
after new block received:
    while doesn't have previous_chunk:
        request previous chunk until genesis chunk reached
        set last known block hash (end of chunk chain)

        while doesn't have current block:
            request blocks succeeding last known block hash
```

Of course, the block chunk fetching is only feasible for networks that do not explicitly require transaction checks – cryptocurrency networks need to maintain a state of network to determine if funds can be spent. The approach could be used for distributed logging systems, where a new block or transaction does not need to consider previous data, but simply appends it to current ledger state.

Chapter 4

Evaluation and testing with scenarios

The evaluation is conducted by forcing particular simulation states and by observing output. The following sections are, in fact, runnable as a part of the source code, and substitute functional testing of the network. These small-scale tests allow to establish an understanding on how nodes will behave in a larger setting; after all, since the nodes are supposed to only gather partial data about the network, it is possible to conclude that these tests are, in fact, just testing a subset of a larger system.

Each test defines a scenario, which in turn defines the structure, behaviour, and data states of nodes. As such, these scenarios can be used to create pre-prepared network topologies for further testing. The scenario recipes follow JSON data structure, and as such are lightweight and easily editable.

Each test JSON template can be found in Appendices, along with expected output. All the tests described were run successfully, and their outcomes are described.

4.1 Basic connection, disconnect cool down

Firstly, it is important to test each node can connect to each other, and that it can and will drop inactive connections. A child class of SimulatedNode was created – SimulatedDeafNode – which does not receive or send any signals, but allows other nodes to establish connections.

As is described in Figure 4.1, the expected outcome is to see two nodes disconnect from each other after a time, and by the end of the simulation (5000 ticks) they will be disconnected. As mentioned, every node contains its own 'cooldown' threshold, which was set to 5000 ticks for this simulation: and as such the nodes will not reconnect until after the simulation is finished. In fact, it is possible to easily test node re-connections by setting their 'cooldown' thresholds lower, to 2000 ticks, which will cause the two nodes to disconnect, and re-connect after a brief cooldown period.

4.2 Signal repetitions

4.2.1 Cut off after no ACKs received

Building on top of previous test, we can establish that a node will disconnect a node should it not receive proper ACKs. To recall the reasoning, the behaviour was created in order to curb the possibility of a Sybil attack, or

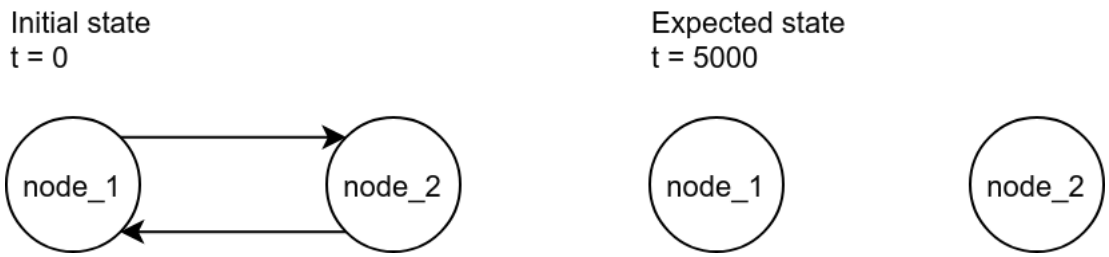

Figure 4.1: Basic connections test

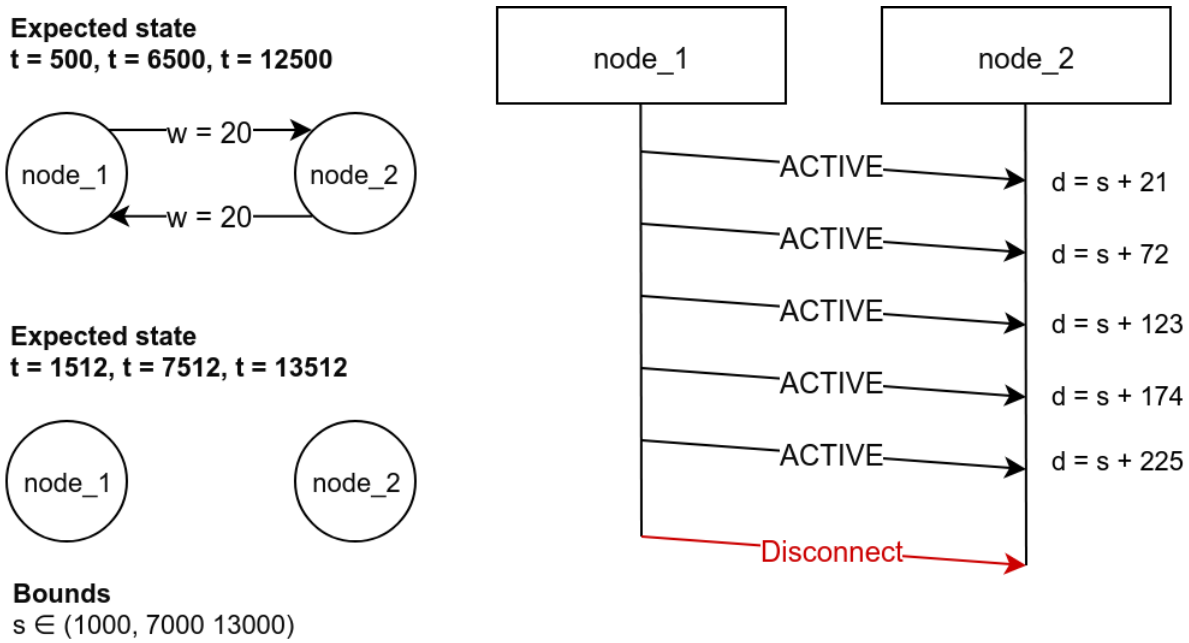

Figure 4.2: Testing disconnect if no ACKs received

even to regulate slow/unresponsive nodes and connections.

The scenario is described in Figure 4.2: the left side shows the expected states at various time segments (connection established and connection dropped, for top and bottom respectively). What is different from the previous scenario are the connection weights, which need to be pre-set to a delay of 20 ticks for every signal - the scenario template allows for connection weights to be set to a specific value.

The right-hand side of the Figure 4.2 describes the networking diagram directly before after node_1 establishes connection with node_2, until it disconnects due to un-answered ACTIVE signals. The formula for 'delta' (denoted as 'd') describes the arrival time of the signal to target node, along with precisely set 'offset' (denoted as 's'). The offset describes the starting point of the network diagram in time, and since the behaviour is reproducible *and* is expected to behave exactly the same way regardless of its position in time, the 'template' can be used to describe the whole interaction.

What is noticeable is the time difference between distinct ACTIVE signals, which happens to be exactly 51 ticks. Each node uses formula of

$$lastSent + connectionCost * 2.5 < currentTick \tag{4.1}$$

to determine if a signal should be sent again. The thread delay was set to 1, and as such the final repeat value will be 51 ticks until disconnect.

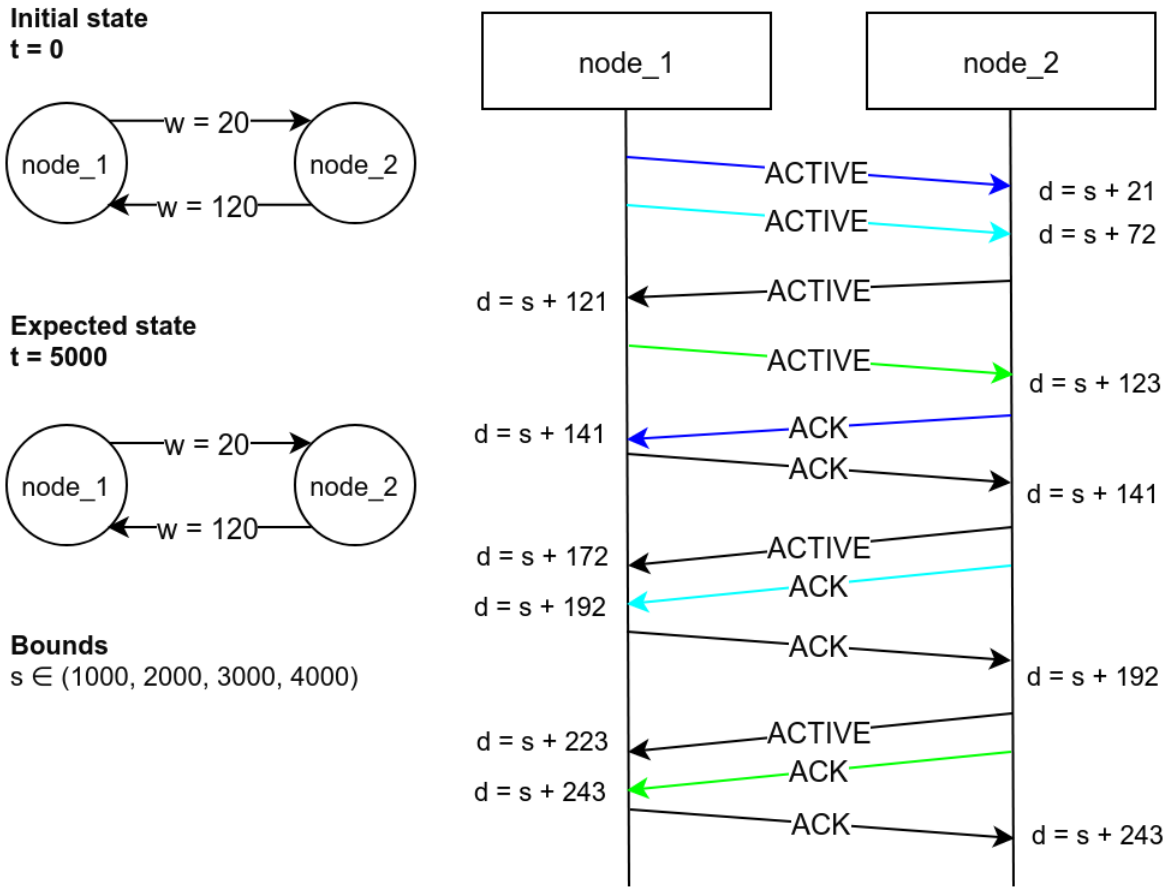

Figure 4.3: Testing signal repetition with delayed response

4.2.2 Delayed response

The scenario describes the behaviour of signal repetitions and delayed response. With reference to Figure 4.3, it is possible to see the network expects to remain the same in terms of its connections. The communication diagram describes the relations between different signals, and is repeated in total of 4 times, once for each new ACTIVE signal reporting. The signals have been colour-coded to illustrate the responses given by node_2 to node_1's signals.

What is particularly interesting about this scenario is the repetition logic - if one ACK is received for one of the ACTIVE signals, all possible repeated signals have been acknowledged: as a matter of fact, the ACK signal received at d = s + 141 by node_1 stops any ACTIVE signal repetition. Node_2 still sends delayed ACK signals, as they have been sent as a response to faster ACTIVE signals. Node_1 will keep sending ACKs to node_2's ACTIVE signals, and once again, due to the delay, two additional ACTIVE signals besides the received one at d = s + 121 will arrive, since they have been sent before the first ACK arrives at d = s + 141.

Having this behaviour working properly means, that the node does not need to wait for all of the delayed ACKs to arrive before disposing of the forwarding signal instructions – suppose the target node would have one of its ACKs lost in transition, and the sending node would try to wait for that one particular acknowledgement before removing the forward instruction. Although the space saving is relatively negligible, it might to prove to be useful for low-resource devices, as they will not need to keep the information needlessly, and thus wasting memory.

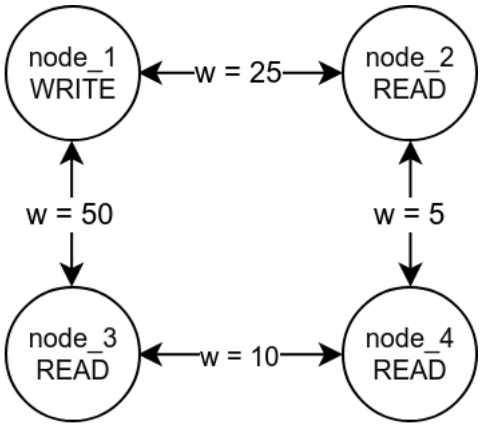

Figure 4.4: Testing ACTIVE signal forwarding of WRITE nodes

4.3 Signal forwarding

4.3.1 WRITE nodes ACTIVE signals

To start, as explained in Section 3.1.2, network uses auth blockchain to ensure only approved nodes create blocks, and by extension reaches distributed consensus. In order for this behaviour to fully work, ACTIVE signals from WRITE nodes need to be transmitted throughout the network; without knowledge of when the next creator node last communicated with the network, a node cannot know if it should wait for a block to arrive or if it should skip the current creator and accept next valid block.

The scenario to test such a behaviour is described in Figure 4.4, and is fairly straight-forward: there are four nodes connected in rectangular network, with various connection costs. What should be seen after the simulation end is that node_4 will know there is a node_1, even despite not being directly connected. It is important to point out that node_4 will try to propagate the signal to 'uninformed node': when receiving forwarded signal from node_3, it will forward to node_2, and vice versa. Both of these nodes will either stop propagation (if they have already received the same signal), or forward it to their connections.

By extension, the presented network structure allows us to see how signal propagation allows nodes to circumvent costly connections, and is in fact very reminiscent of vector-based router mapping; only in this case nodes do not have a perfect knowledge of all the nodes in the map.

4.3.2 Transactions, blocks

After nodes in the network know about WRITE nodes, it is needed to test propagation of transactions and blocks; the network will be built as shown in Figure 4.5, which consists of three nodes of different type. The connections between them serve as both upload and download links.

Transactions will be propagated straight from node_2 to both node_1 and node_3 with appropriate delays, as expected. What is important to notice is how the blocks will propagate - the network was purposefully built to include a shorter route from node_1 to node_3 through node_2: 50 ticks via straight connection from node_1 compared to 30 ticks via node_2. And as expected, the block will arrive *and* will be accepted before node_3 receives 'the original' signal from node_1. This behaviour is crucial, as it allows the network to circumvent possible slowdown attacks by utilizing multiple communication routes for transactions and blocks.

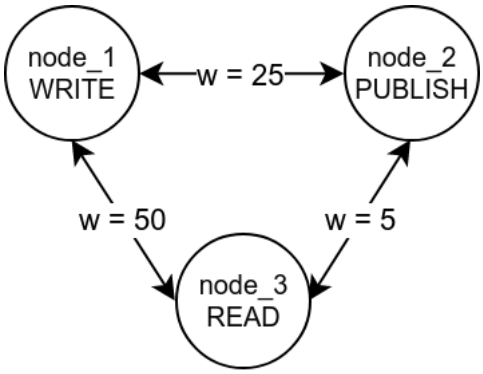

Figure 4.5: Regular WRITE scenario

4.4 Block acceptance

After the network is able to forward signals and change its connections, it must now decide on which blocks and signals are valid and should be accepted, and which ones are not. Since the underlying behaviours were tested in previous scenarios, the following tests will not consider, for example, ACTIVE and ACK signals.

4.4.1 Regular block acceptance

The simulation will use Figure 4.5 structure to test the block acceptance of each node. As was already mentioned during Block propagation testing, this network will have a faster route available from node_1 to node_3 via node_2, and as such is expected to accept the block before it will arrive from node_1 via direct route.

4.4.2 Delayed block acceptance

And now, at last, a test concerning a fault in the system that will require the nodes to keep local storage of chain fragments, as visualized previously in Figure 3.6. The actual structure of the network must be changed to introduce another node to the set, seen in Figure 4.6 and marked in yellow colour. This particular node has WRITE permissions, but will never forward node_1's newly-created blocks, but will still accept and act upon them; in fact it can be assumed that node_3 was compromised, and is now behaving selfishly in a cooperative environment of the private network.

Due to the nature of node_3, node_4 will be disadvantaged, as it will not receive node_1 blocks via the fastest route, and neither will it receive them before node_3 could send their own, 'future' blocks, which were built on top of blocks unknown to node_1.

As such, node_4 will utilize its block storage, and will keep any potential block in its memory to be acted upon after a missing link is found. In fact, right after the node receives a block from node_1, it will be able to validate and append node_3's block without the need to receive the block again from either node_3 or node_2.

4.4.3 Malicious WRITE node

The last test concerns a network with directly malicious node (seen in red in Figure 4.7) – not just selfishly acting node as presented in the previous scenario, but an unpredictable and openly-hostile one. It achieves its status by sometimes appearing and acting as a regular WRITE node, and sometimes spoofing and changing block

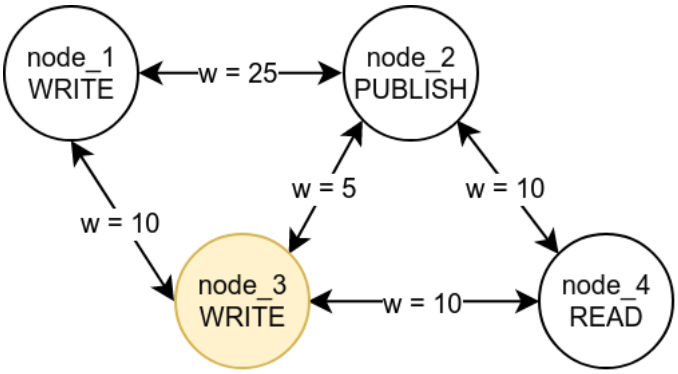

Figure 4.6: Scenario with delayed block acceptance

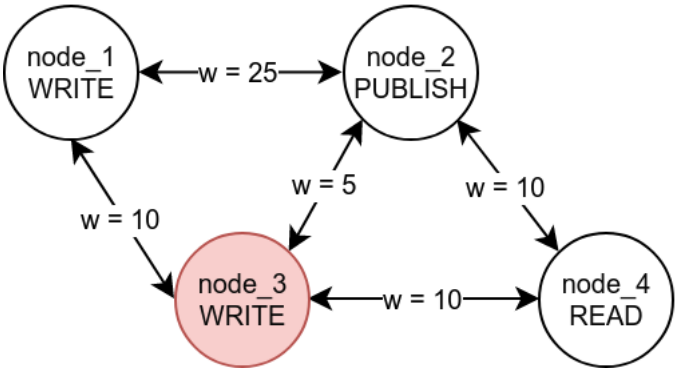

Figure 4.7: Scenario with a malicious WRITE node

contents. The choice is governed by network's RNG, so once again, the behaviour is random but predictable for the purposes of the testing, all performed by changing a few values of validly-created block.

Breaking the hash chain – the first possibility of tampering with blocks. Block's previous_hash will be changed to a non-sensical string. In a real-life scenario, an intruder might choose to instead point to their own chain, perhaps in order to smuggle an invalid or malicious transaction into other nodes' internal dataset.

Changing the creator – the second possibility of changing the block, which is handy way of testing how nodes check their internal authentication blockchain. Suppose a node would not check the block's headers, and would blindly trust that the origin node did send a correct and valid block – how would the blockchain ensure traceability if the creator's identity can be changed in the block data? If the malicious node would like to frame another node for putting illegal content onto private blockchain, it would be almost effortless if no checks would be present.

Changing the height – the third possibility of wreaking havoc. Suppose the network would allow block roll-back up until the genesis block, then a malicious node could simply force it by introducing a 'valid' block of height one, forcing all the nodes to roll-back their blockchains; or even to force the network to halt by introducing blocks heights in such a manner that the blockchain height would never climb past a certain number – after all, the node knows exactly when its turn will be, all thanks to the authorization levels. Disallowing roll-backs and far-future blocks prevents the network from being overwhelmed by fake heights.

Chapter 5

Larger-scale performance testing

With the small-scale integration testing done, it is now time to test the behaviour of a larger network, and its potential throughput. A single node was followed during the tests, and its graphs can be found in Appendix.

5.1 Brief overview, terminology, test aims

Parliament of Czech Republic decreed in 2016 to enact a Registration Sales Act [12] [11] (Elektronická evidence tržeb, in Czech; abbreviated as EET), which has and will see many business needing to submit their sales transactions to the ministry for tax records. The system is fairly simple in its design[19], as the business owner:

- asks the Financial Administration of Czech Republic for an authorization key
- purchases a system capable of communication with EET systems, and generates a signing certificates
- every transaction is automatically submitted to the EET system, and receives a confirmation code to be printed on customer's receipt
- should there be problem during the submission process, the owner will have to submit all non-submitted transactions at a later date

Each transaction contains the information about the sale itself, and specifies the location of the sale, business owner's VAT number, total amount, and different VAT brackets and their tax calculations. The full specifications of each transaction can be found in the Appendix, along with their English translations. The system uses SOAP as its communication protocol with XML objects being passed from and to the point of sale system [10].

Why this system? Since its inception, it was met by a wide-spread critique, which culminated by a recent Constitutional Court case submitted by 41 MPs to fully revoke the Registration Sales Act [13]. The Court has made changes to the law itself, removing controversial parts and stopping tier-based roll-out of the system, which was supposed to include most of sole traders and business using cash-like transactions. As of now, the system has gone through 2 of these tiers: first one included CZ-NACE [1] 55 and 56 business (58,612 businesses as of 2015 [37]), and the second one included CZ-NACE 45, 46, and 47 businesses (241,572 businesses as of 2015 [38]); and since the tier exclude some businesses, there are 167,049 registered businesses, with 207,803 authentication keys and 293,018 signing certificates [14]. Data released about the project show a maximum of about 15,000,000 transactions being settled every day, or translated to hourly peaks, around 420 transactions at the busiest times [2].

[1] NACE is a statistical classification of economic activities with European community [32].

[2] Data extracted from publicly available data sets [14].

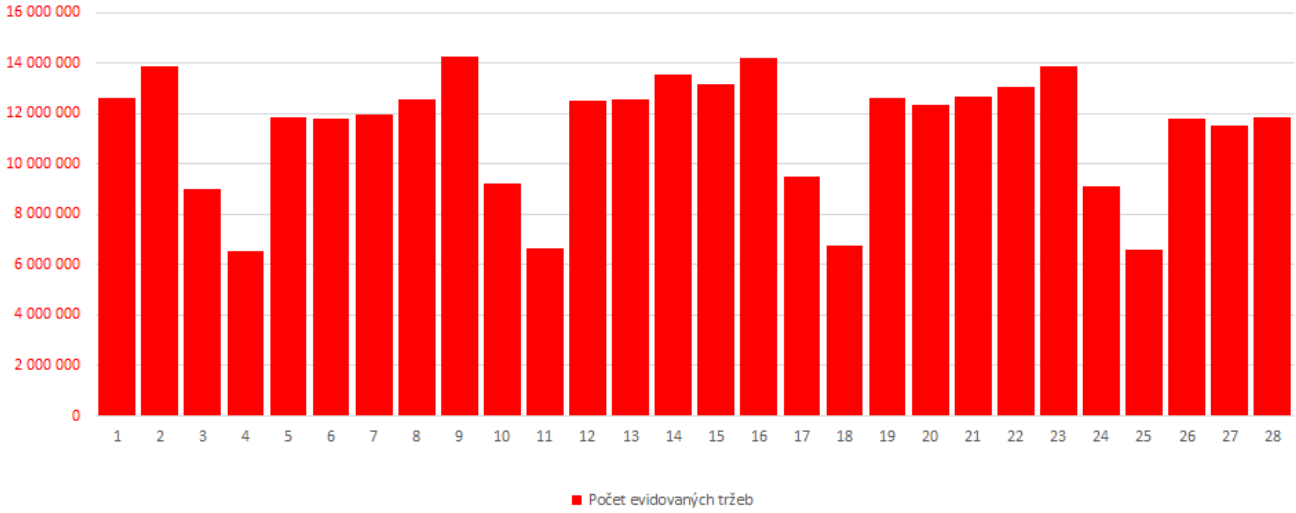

Figure 5.1: EET system, February 2018 [14]

Since this is such a large-scale system, and because it is controlled by a single entity which already owns the core network, it might prove insightful to investigate whether the proposed blockchain could be used to leverage increased data redundancy and network resistance.

5.2 Parameters

It would be near-impossible to test a full system using the simulation, and as such an approximation will be made, along with later extrapolation of results to larger systems. The Financial Administration maintains 215 offices around Czech republic; and given the sensitivity of the data it is assumed the only WRITE nodes will be those of the Administration. We can then assume the following:

```
215 WRITE nodes
293,018 PUBLISH nodes
=> 1,363 publish nodes for each write node
```

The number of publish nodes per write node was rounded up to 1500, to account for potential outages, networking problems, etc. Given the highest amount of hourly transaction in latest available month (February 2018) was reached on 9th February (seen in Figure 5.1, and Figure 5.2):

```
1,298,068 transactions per hour
=> 360 transactions per second
```

There was even a higher peak of 510 transactions per second reached on 22nd December 2017 (Figures 5.3 and 5.4), but given the fact it was the last working day before Christmas and it was Friday, the peak can be thought of as an outlier. Either way, the simulation was ran assuming production of 450 transactions per second, and depending on settings may be able to handle much higher loads.

Since the EET system uses XML, the tests were performed using mock data conforming to the exact specifications; and JSON objects were generated from the same data in order to compare and contrast network performance with lighter data structures.

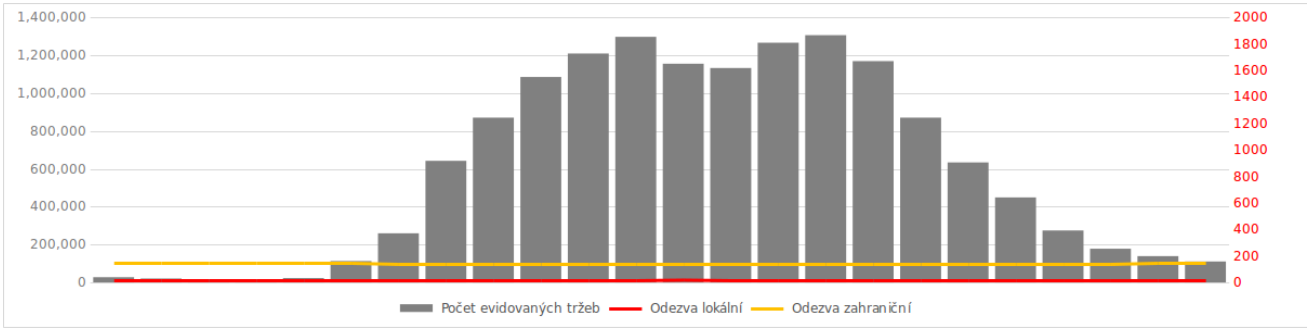

Figure 5.2: EET system, 9th Debruary 2017 [14]

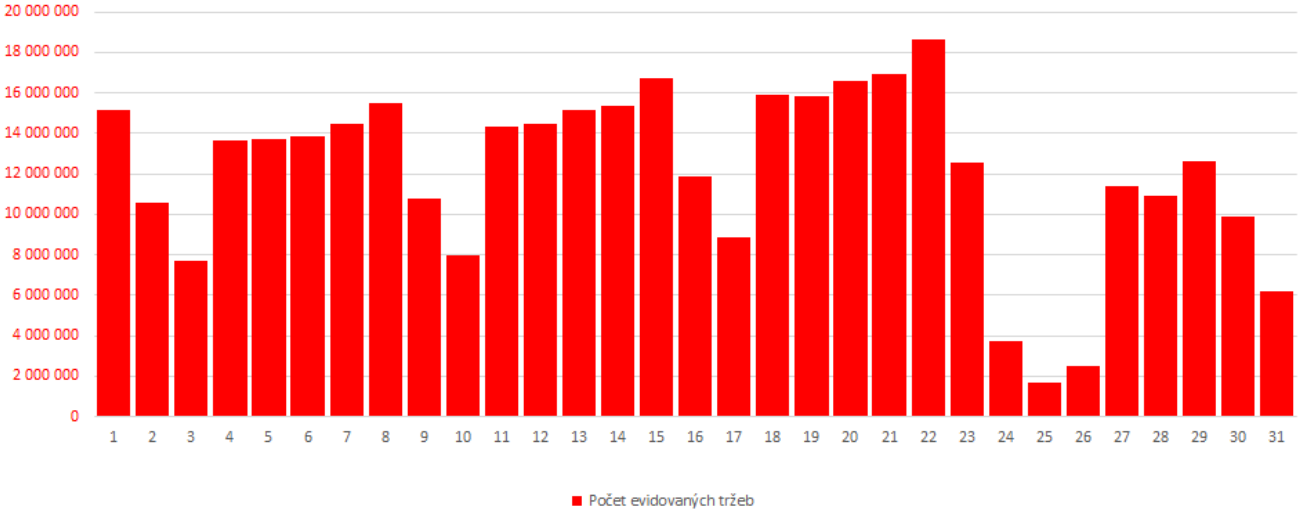

Figure 5.3: EET system, December 2017 [14]

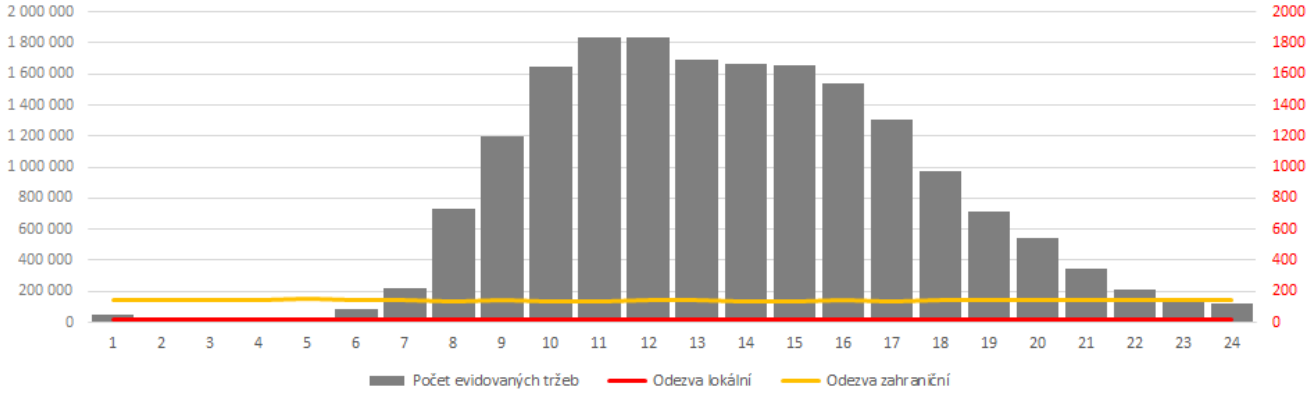

Figure 5.4: EET system, 22nd December 2017 [14]

5.3 Test 1

The first test tested the network under scaled parameters:

```
1500 publishing nodes
1 write node
2 transactions per second (scaled from 450 for the whole network)
15,000 ticks
min. 2 transactions per block
```

and produced following results:

[XML transactions]	mean	mode	median	min	max
Verified transactions	21	21	21	21	21
Blocks accepted	7	7	7	7	7
Transactions per block	3	3	3	3	3
Block size [bits]	3344	3457	3319	3238	3457

[JSON transactions]	mean	mode	median	min	max
Verified transactions	21	21	21	21	21
Blocks accepted	7	7	7	7	7
Transactions per block	3	3	3	3	3
Block size [bits]	3122	3010	3097	3010	3242

5.4 Test 2

The second test tested the network under scaled parameters:

```
150 publishing nodes (10% of 1500 nodes)
1 write node
450 transactions per second (100% of expected load)
15,000 ticks
min. 450 transactions per block
```

and produced following results:

[XML transactions]	mean	mode	median	min	max
Verified transactions	5904	5910	5910	5007	5910
Blocks accepted	12	13	13	11	13
Transactions per block	454	451	451	451	497
Block size [bits]	497,121	494,343	493,413	491,443	543,948

[JSON transactions]	mean	mode	median	min	max
Verified transactions	5864	5867	5867	5416	5867
Blocks accepted	12	13	13	12	13
Transactions per block	451	451	451	451	453
Block size [bits]	460,327	461,223	460,414	457,699	462,982

5.5 Test 3

The third test tested the network under scaled parameters:

```
150 publishing nodes (10% of 1500 nodes)
3 write node
450 transactions per second (100% of expected load)
15,000 ticks
min. 450 transactions per block
```

and produced following results:

[XML transactions]	mean	mode	median	min	max
Verified transactions	5865	5865	5865	5865	5865
Blocks accepted	13	13	13	13	13
Transactions per block	451	451	451	451	452
Block size [bits]	493,754	493,318	493,504	492,850	495,372

[JSON transactions]	mean	mode	median	min	max
Verified transactions	5690	5702	5702	4800	6167
Blocks accepted	11	12	12	10	13
Transactions per block	474	451	451	451	728
Block size [bits]	484,427	461,094	461,094	458,757	742,220

5.6 Test 4

The third test tested the network under scaled parameters:

```
300 publishing nodes (20% of 1500 nodes)
3 write node
450 transactions per second (100% of expected load)
10,000 ticks
min. 450 transactions per block
```

and produced following results:

[XML transactions]	mean	mode	median	min	max
Verified transactions	3600	3608	3608	3157	3608
Blocks accepted	7	8	8	7	8
Transactions per block	451	451	451	451	451
Block size [bits]	493,750	494,774	494,237	491,036	495,936

[JSON transactions]	mean	mode	median	min	max
Verified transactions	3609	3612	3160	3612	6167
Blocks accepted	7	8	8	7	8
Transactions per block	451	451	452	451	452
Block size [bits]	460,881	462,527	461,375	458,986	462,527

5.7 Results and discussion

Firstly, why were the test parameters chosen the way they were?

- Test 1 provided data with exactly scaled network, including transactions/second
- Test 2 provided data with peak 450 transactions/second on 10% scaled network
- Test 3 provided data with peak 450 transactions/second, on 10% scaled network with 3 WRITE nodes
- Test 4 provided same data as Test 3, only with 20% scaled network

As such, it is possible to infer insights into what a system based on proposed blockchain structure would look like, and how it compares to the current system; thanking to the use of seeded RNG, the results are fully reproducible too.

Verified transactions

The scenarios ran under regular settings, and as such the transaction generation started after connections were made at 500 ticks. Taking the extra delta into account, the transactions/second throughput of the scenarios is as follows:

test	generated/second	verified [XML]	verified [JSON]
Test 1	2	1	1
Test 2	450	394	391
Test 3	450	391	411
Test 4	450	360	617

It is expected to have a lower amount of verified transactions, as the generated/second is only an indicator for a *chance* to generate a new transaction.

Block sizes

Looking at block sizes in bits, it is possible to see and compare the average (mode) block sizes of each test:

test	block size [XML]	block size [JSON]
Test 1	3457	3010
Test 2	494,343	461,223
Test 3	493,318	461,094
Test 4	494,774	462,527

As is evident between XML and JSON, the block sizes can differ to a large degree - in fact JSON implementation saves around 7% of transfered data compared to XML. It is also clear to see that the simulation is very consistent in block sizes, the deviations are caused by slightly different order of calls to RNG, and hence transactions are created with small differences.

Comparison to EET

When comparing the simulation to EET system, it must be noted that a much larger simulation must be made in order to fully understand how the system would behave; however, based on the results obtained, the simulation was relatively consistent in the rate of verification of transactions. It must also be noted that the EET system was specifically built for the purpose of sales transactions verification, while as the simulation presented is only a generalized system with customized transaction content.

Chapter 6

Limitations and future work

The simulation presented in this dissertation is a fairly large project, and its focus had to be narrowed down constantly; as such there are inherent limitations that had to be introduced to keep the requirements manageable for a single-person project. The EET system, which the simulation tries to emulate, was built by a team of top-notch engineers, and so expectations over the breadth of the work must be equally scaled.

The first limitation that is apparent is the speed of the simulation, and its slow progression in larger networks. The testing on EET system has shown that networks of around 300 publishing nodes and 3 write nodes, with up to 450 transactions per second flowing through present a heavy load on the hosting system. Combined with its single-core implementation, the simulation has room for improvements in streamlining some of its parts. It must, however, be noted, that Java 8 does perform extraordinarily well, and was in fact delegating work between all cores of the host machine when testing larger networks.

Concerning data storage and access times, there might be a room for improvements as well. Most of the data structures in the network use HashMaps or HashSets, which have O(1) best-case access times, and only O(n) worst-case access times. However, blockchain is by its nature a backwards-linked list, and any searches through it will follow O(n) complexity at best, `O(n^2)` complexity at worst – all because previous_hash points to a block stored in a HashMap. The block chunking is not fully used in the simulation either, but is an interesting concept worthy of mentioning and explaining, as it might prove to be useful in network synchronization times, and even a block lookup. In fact, should the network require extremely quick fetch times paired with high volumes of blocks, one could entertain the idea of chaining block chunks themselves, and forming more compact chunk chains to be fetched quicker, with a reduction in times for each level added.

Tying in to block chunking, the idea of 'ant colony' behaviour also comes to mind. The ant colony has an emergent behaviour of classes: for example, if a worker ant did not see any soldier and for a while, it will become a soldier; and conversely, if a soldier ant saw too many other soldier ants, it will become a worker. This self-balancing behaviour could be used together with REQUEST signals to maintain a degree of dependence on the network, all while lowering the amount of blocks a node needs to store. In fact, this behaviour was one of the out-of-scope ideas that was intended to be made.

Connection balancing is another area of limitation and improvements: the current system manages to balance links thanks to a counter, which is updated every few ticks. The limitation is, that there is no upper limit that would cause 'connection overflow', where connections would have to wait to be served due to link saturation; as such it has happened that nodes with limit of 10,000 bits/tick would see traffic climbing over the limit before number of connections was curbed. A future improvement could introduce a better-fairing algorithm and improved implementation of these connections.

Connection predictability and further look into Sybil attack vulnerabilities could be of interest as well. The

network has increased protection against such an attack by changing connections, however there are possible attack vectors by exploiting 'slow ACKs' approach, as presented in one of the integration tests. Further focus can be given to such a problem of slow nodes, which are affecting forwarding rates in the network, perhaps by introducing a threshold for slow connections, just the same as threshold for unanswered signals already exists.

All-in-all, although the simulation managed to show its smaller-scale solutions of common problems of P2P and distributed networks of today, there is still an immense amount of research and testing to be done; after all, many of these vulnerabilities presented smaller problems for networks with fluid data, however with blockchain and its 'write once, remember forever' structure it is even more important to make sure these vulnerabilities are accounted for.

Appendices

Appendix A

Running the program

The program is a Maven-controlled source code, so in order to run it you must have both Maven and Java 8 installed. Afterwards, you can simply run

```
mvn clean compile assembly:single
```

to compile and prepare the source code. All required files will be packed in a jar file without the need to put anything in JAVA path, in:

```
target/HybridChunkedBlockchainTickSimulation-1.0.0-jar-with-dependencies.jar
```

To run the program, you can use the following command:

```
java -jar <jar_file>
```

You will have to point the jar_file to the file created previously.

To run the program with a scenario, or with a scenario and a compare-to case, use following syntax:

```
java -jar <jar_file> <scenario_file.json> <compare_to.txt>
```

Do note that not proving compare_to file will force output into console. Not not providing scenario_file will cause a fallback to *default.json*, described further in appendices.

Appendix B

Scenarios

B.1 Template and guidance

The JSON scenarios offer a wide range of potential configurations, for actual usages refer to test and eet scenarios.

Test scenarios can be called by using template of

```
tests/[id]_test.json

e.g. test/1_test.json
```

Comparison scenarios can be called by using template of

```
tests/[id]_test.json tests/[id]_expected.txt

e.g. test/1_test.json tests/1_expected.txt
```

You can also run all tests with comparators using provided bash script

```
bash run_tests.sh
```

Basic structure

```
{
   "name": String,
   "seed": long,
   "start": int,
   "end": int,
   "speed": int [ms per tick],
   "nodes": <nodes>,
   "many_nodes": <many_nodes>,
   "connections": <connections>,
   "logging": <logging>,
```

```
   "graphs": <graphs>
}
```

nodes

```
{
   "id": String,
   "class": <nodeClass>,
   "permission": int[0-3],
   "connectionCooldown": int,
   "connectionStale": int,
   "maxUploadConnections": int,
   "maxDownloadConnections": int,
   "repetitionDisconnectThreshold": int,
   "connectionCost": int,
   "ticksPerTransaction": int (only for SimulatedTransactionNode)
},
...
```

many_nodes

```
{
   "id_prefix": String (used in form "id_prefix"+_+id, eg."publish_0"),
   "class": <nodeClass>,
   "permission": int[0-3],
   "count": int,
   "speedUpload": int,
   "speedDownload": int,
   "transactionsPerSecond": int (only for SimulatedSalesTransactionNode),
   "minTransactions": int (only for SimulatedBlockWriteNode),
   "style": String[xml|json] (only for SimulatedSalesTransactionNode)
},
...
```

nodeClass

```
SimulatedReadNode
- only reads

SimulatedDeafNode
- does not reply to any signal

SimulatedDelayNode
- replies with delay of 100 ms more than expected

SimulatedTransactionNode
- creates basic DummyTransaction transactions

SimulatedSalesTransactionNode
- creates fake SalesTransaction transactions
```

```
SimulatedBlockWriteNode
- regular block creator

SimulatedDelayedBlockWriteNode
- does not forward new blocks from other nodes

SimulatedMaliciousBlockWriteNode
- randomly wreaks havoc on the network
```

connections

```
{
   ["node_1 node_2 direction cost"],
   ...
}

direction
   U = upload
   D = download
   UP = both upload and download
```

logging

```
{
   "signals": <signals>,
   "nodes": <node_ids>,
   "special": <special_cases>
}

signals:
   ACTIVE
   NEW_BLOCK
   NEW_AUTH_BLOCK
   NEW_TRANSACTION
   ACK
   BLOCK_REQUEST
   BLOCK_CHUNK_REQUEST,
   AUTH_BLOCK_REQUEST
   REQUEST_DENY
   REQUEST_WAIT
   BLOCK_RESPONSE

special_cases:
   NODE_KNOWLEDGE
   ACK_TRANSACTION
   ACK_BLOCK
   BLOCK_HEIGHT
   TRANSACTION_POOL_LEN
   BLOCK_ACCEPTANCE
   BLOCK_CREATION
```

graphs

```
{
   "network": [graphs_types...],
   "node": ["node_id", graph_types...],
}

graph_types:
   transaction_pool
   forward_signals
   connections
   signals_received
   signals_sent
   waiting_signals
   connection_speeds
   blocks_accepted
   transactions_verified
```

B.2 Tests

B.2.1 01 - Basic connection

```
{
      "name": "Basic connection",
      "seed": 123456789,
      "start": 0,
      "end": 5000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedDeafNode",
                  "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 500
      },
      {
                  "id": "node_2",
                  "class": "SimulatedDeafNode",
            "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 500
      }
      ],
      "logging": {
            "nodes": ["node_1", "node_2"]
      }
}
```

Expected output

```
500: node_2 New DOWN connection from node_1, delay 42
500: node_2 New UP connection to node_1, delay 42
1132: node_1 closed DOWNLOAD from node_2
1132: node_1 closed UPLOAD to node_2
1532: node_2 closed DOWNLOAD from node_1
1532: node_2 closed UPLOAD to node_1
```

B.2.2 02 - Basic connection

```
{
	"name": "Basic connection",
	"seed": 123456789,
	"start": 0,
	"end": 5000,
	"speed": 1,
	"nodes": [
		{
			"id": "node_1",
			"class": "SimulatedDeafNode",
			"permission": 0,
			"connectionCooldown": 2000,
			"connectionStale": 500
	},
	{
			"id": "node_2",
			"class": "SimulatedDeafNode",
		"permission": 0,
			"connectionCooldown": 2000,
			"connectionStale": 500
	}
	],
	"logging": {
		"nodes": ["node_1", "node_2"]
	}
}
```

Expected output

```
500: node_2 New DOWN connection from node_1, delay 42
500: node_2 New UP connection to node_1, delay 42
1132: node_1 closed DOWNLOAD from node_2
1132: node_1 closed UPLOAD to node_2
1532: node_2 closed DOWNLOAD from node_1
1532: node_2 closed UPLOAD to node_1
3500: node_1 New DOWN connection from node_2, delay 42
3500: node_1 New UP connection to node_2, delay 42
4500: node_2 closed DOWNLOAD from node_1
4500: node_1 terminated UP to node_2
4532: node_1 closed UPLOAD to node_2
```

B.2.3 03 - ACK signal disconnect

```
{
      "name": "ACK signals disconnect",
      "seed": 123456789,
      "start": 0,
      "end": 15000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedReadNode",
                  "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 10000,
                  "activeDisconnectThreshold": 5,
                  "connectionCost": 20
      },
      {
                  "id": "node_2",
                  "class": "SimulatedDeafNode",
            "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 10000,
                  "repetitionDisconnectThreshold": 10,
                  "connectionCost": 20
      }
      ],
      "logging": {
            "nodes": ["node_1", "node_2"]
      }
}
```

Expected output

```
500: node_2 New DOWN connection from node_1, delay 20
500: node_2 New UP connection to node_1, delay 20
1021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1174: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1225: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1257: node_1 closed DOWNLOAD from node_2
1257: node_1 closed UPLOAD to node_2
```

```
1276: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1512: node_2 closed DOWNLOAD from node_1
1512: node_2 closed UPLOAD to node_1
6500: node_1 New DOWN connection from node_2, delay 20
6500: node_1 New UP connection to node_2, delay 20
7021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7174: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7225: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7257: node_1 closed DOWNLOAD from node_2
7257: node_1 closed UPLOAD to node_2
7276: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
7512: node_2 closed DOWNLOAD from node_1
7512: node_2 closed UPLOAD to node_1
12500: node_1 New DOWN connection from node_2, delay 20
12500: node_1 New UP connection to node_2, delay 20
13021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
13072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
13123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
13174: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
13225: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
13257: node_1 closed DOWNLOAD from node_2
13257: node_1 closed UPLOAD to node_2
13276: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
```

```
13512: node_2 closed DOWNLOAD from node_1
13512: node_2 closed UPLOAD to node_1
```

B.2.4 04 - Delayed AKC signals

```
{
      "name": "Delayed ACK signals",
      "seed": 123456789,
      "start": 0,
      "end": 5000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedReadNode",
                  "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 10000,
                  "activeDisconnectThreshold": 5,
                  "connectionCost": 20
      },
      {
                  "id": "node_2",
                  "class": "SimulatedDelayedNode",
            "permission": 0,
                  "connectionCooldown": 5000,
                  "connectionStale": 10000,
                  "repetitionDisconnectThreshold": 10,
                  "connectionCost": 20
      }
      ],
      "logging": {
            "nodes": ["node_1", "node_2"]
      }
}
```

Expected output

```
500: node_2 New DOWN connection from node_1, delay 20
500: node_2 New UP connection to node_1, delay 20
1021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
1121: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
1123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
```

```
1141: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
1141: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
1172: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
1192: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
1192: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
1223: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
1243: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
1243: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
2021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
2072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
2121: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
2123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
2141: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
2141: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
2172: node_2 -> node_1 (sig 0) hash: 63
```

```
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
2192: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
2192: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
2223: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
2243: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
2243: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
3021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
3072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
3121: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
3123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
3141: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
3141: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
3172: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
3192: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
3192: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
```

```
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
3223: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
3243: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
3243: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
4021: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
4072: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
4121: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
4123: node_1 -> node_2 (sig 0) hash: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860,
   ↪ payload: null
4141: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
4141: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
4172: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
4192: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
4192: node_1 -> node_2 (sig 4) hash:
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
4223: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
4243: node_2 -> node_1 (sig 4) hash:
   ↪ c520ac12331f0f6c927bed96d7326883506acc26a8f1903801b488d97d9ed764,
   ↪ payload: 000
   ↪ b662105bd1aed101b54133afea000566b73cea332990f7e02352d0135f860
4243: node_1 -> node_2 (sig 4) hash:
```

```
   ↪ dec2ad2d9b17b845383c95cadb1406ea31640f3a17f82ac6bbe4356cdc8d627d,
   ↪ payload: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d
```

B.2.5 05 - ACTIVE forwarding for WRITE nodes

```
{
      "name": "ACTIVE forwarding for WRITE nodes",
      "seed": 123456789,
      "start": 0,
      "end": 2000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_2",
                  "class": "SimulatedReadNode",
            "permission": 0,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_3",
                  "class": "SimulatedReadNode",
            "permission": 0,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_4",
                  "class": "SimulatedReadNode",
            "permission": 0,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      }
      ],
      "connections": [
            "node_1 node_2 UD 25",
            "node_1 node_3 UD 50",
            "node_2 node_4 UD 5",
            "node_3 node_4 UD 10"
```

```
      ],
      "logging": {
            "signals": ["ACTIVE"],
            "special": ["NODE_KNOWLEDGE"]
      }
}
```

Expected output

```
6: node_2 -> node_4 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
6: node_4 {node_2=6, node_3=0}
6: node_4 -> node_2 (sig 0) hash: 24381838191069
   ↪ ae25a120a3015bb55f7e16f185c289f0f86899f743c211812c, payload: null
6: node_2 {node_1=0, node_4=6}
11: node_3 -> node_4 (sig 0) hash: 9950
   ↪ a90fb7b773293e48988b4b567b33cd009349c3a303d896fb4f6e617764e4, payload
   ↪ : null
11: node_4 {node_2=6, node_3=11}
11: node_4 -> node_3 (sig 0) hash: 24381838191069
   ↪ ae25a120a3015bb55f7e16f185c289f0f86899f743c211812c, payload: null
11: node_3 {node_1=0, node_4=11}
26: node_1 -> node_2 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
26: node_2 {node_1=26, node_4=6}
26: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
26: node_1 {node_2=26}
51: node_1 -> node_3 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
51: node_3 {node_1=51, node_4=11}
51: node_3 -> node_1 (sig 0) hash: 9950
   ↪ a90fb7b773293e48988b4b567b33cd009349c3a303d896fb4f6e617764e4, payload
   ↪ : null
51: node_1 {node_2=26, node_3=51}
1006: node_2 -> node_4 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
1006: node_4 {node_2=1006, node_3=11}
1006: node_4 -> node_2 (sig 0) hash: 24381838191069
   ↪ ae25a120a3015bb55f7e16f185c289f0f86899f743c211812c, payload: null
1006: node_2 {node_1=26, node_4=1006}
1011: node_3 -> node_4 (sig 0) hash: 9950
   ↪ a90fb7b773293e48988b4b567b33cd009349c3a303d896fb4f6e617764e4, payload
   ↪ : null
1011: node_4 {node_2=1006, node_3=1011}
1011: node_4 -> node_3 (sig 0) hash: 24381838191069
   ↪ ae25a120a3015bb55f7e16f185c289f0f86899f743c211812c, payload: null
```

```
1011: node_3 {node_1=51, node_4=1011}
1026: node_1 -> node_2 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1026: node_2 {node_1=1026, node_4=1006}
1026: node_2 -> node_1 (sig 0) hash: 63
   ↪ cc96d584ecb739c8fa63e5407d39933a8a778071cdc7e4a7e3cb790f5d643d,
   ↪ payload: null
1026: node_1 {node_2=1026, node_3=51}
1031: node_2 -> node_4 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1031: node_4 {node_1=1031, node_2=1006, node_3=1011}
1036: node_4 -> node_2 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1036: node_2 {node_1=1036, node_4=1006}
1041: node_4 -> node_3 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1041: node_3 {node_1=1041, node_4=1011}
1051: node_1 -> node_3 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1051: node_3 {node_1=1051, node_4=1011}
1051: node_3 -> node_1 (sig 0) hash: 9950
   ↪ a90fb7b773293e48988b4b567b33cd009349c3a303d896fb4f6e617764e4, payload
   ↪ : null
1051: node_1 {node_2=1026, node_3=1051}
1051: node_3 -> node_4 (sig 0) hash: 0
   ↪ adf15224d729244e7c3e99fb30c0b1fdad68f7ce902bd6eddefb4238477d095,
   ↪ payload: 1
1051: node_4 {node_1=1051, node_2=1006, node_3=1011}
```

B.2.6 06 - BLOCK acceptance, single WRITE node

```
{
      "name": "BLOCK acceptance, single WRITE node",
      "seed": 123456789,
      "start": 0,
      "end": 1000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
```

```
      {
            "id": "node_2",
            "class": "SimulatedTransactionNode",
      "permission": 1,
            "connectionCost": 20,
            "maxUploadConnections": 2,
            "maxDownloadConnections": 2,
            "ticksPerTransaction": 1000
      },
      {
            "id": "node_3",
            "class": "SimulatedReadNode",
      "permission": 0,
            "connectionCost": 20,
            "maxUploadConnections": 2,
            "maxDownloadConnections": 2
      }
      ],
      "connections": [
            "node_1 node_2 UD 25",
            "node_1 node_3 UD 50",
            "node_2 node_3 UD 5"
      ],
      "logging": {
            "signals": ["NEW_TRANSACTION"],
            "special": ["ACK_TRANSACTION"]
      }
}
```

Expected output

```
5: node_3 ACK for signal with hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a
5: node_2 -> node_3 (sig 3) hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a,
   ↪ payload: node_2;-1040170699
25: node_1 ACK for signal with hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a
25: node_2 -> node_1 (sig 3) hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a,
   ↪ payload: node_2;-1040170699
55: node_1 ACK for signal with hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a
55: node_3 -> node_1 (sig 3) hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a,
   ↪ payload: node_2;-1040170699
85: node_3 ACK for signal with hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a
85: node_1 -> node_3 (sig 3) hash: 7
   ↪ f7a8d0e948a0e38d8ae879d5a80b307278d30a4bbc377279ec9f84258f9e93a,
   ↪ payload: node_2;-1040170699
```

B.2.7 07 - BLOCK acceptance, single WRITE node

```
{
      "name": "BLOCK acceptance, single WRITE node",
      "seed": 123456789,
      "start": 0,
      "end": 1000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_2",
                  "class": "SimulatedTransactionNode",
            "permission": 1,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2,
                  "ticksPerTransaction": 1000
      },
      {
                  "id": "node_3",
                  "class": "SimulatedReadNode",
            "permission": 0,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      }
      ],
      "connections": [
            "node_1 node_2 UD 25",
            "node_1 node_3 UD 50",
            "node_2 node_3 UD 5"
      ],
      "logging": {
            "signals": ["NEW_BLOCK"],
            "special": ["ACK_BLOCK", "BLOCK_HEIGHT"]
      }
}
```

Expected output

```
25: node_1 blockchain height: 1
60: node_2 ACK for signal with hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277
```

```
60: node_1 -> node_2 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
60: node_2 blockchain height: 1
65: node_3 ACK for signal with hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277
65: node_2 -> node_3 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
65: node_3 blockchain height: 1
71: node_2 ACK for signal with hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277
71: node_3 -> node_2 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
87: node_3 ACK for signal with hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277
87: node_1 -> node_3 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
```

B.2.8 08 - BLOCK acceptance, single WRITE node

```
{
      "name": "BLOCK acceptance, single WRITE node",
      "seed": 123456789,
      "start": 0,
      "end": 5000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_2",
                  "class": "SimulatedTransactionNode",
            "permission": 1,
                  "connectionCost": 20,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2,
                  "ticksPerTransaction": 1000
      },
      {
                  "id": "node_3",
                  "class": "SimulatedReadNode",
            "permission": 0,
```

```
                "connectionCost": 20,
                "maxUploadConnections": 2,
                "maxDownloadConnections": 2
        }
        ],
        "connections": [
                "node_1 node_2 UD 25",
                "node_1 node_3 UD 50",
                "node_2 node_3 UD 5"
        ],
        "logging": {
                "special": ["BLOCK_HEIGHT", "TRANSACTION_POOL_LEN"]
        }
}
```

Expected output

```
0: node_2 transaction pool: 1
5: node_3 transaction pool: 1
25: node_1 transaction pool: 1
25: node_1 blockchain height: 1
60: node_2 blockchain height: 1
65: node_3 blockchain height: 1
1000: node_2 transaction pool: 1
1005: node_3 transaction pool: 1
1025: node_1 transaction pool: 1
1025: node_1 blockchain height: 2
1060: node_2 blockchain height: 2
1065: node_3 blockchain height: 2
2000: node_2 transaction pool: 1
2005: node_3 transaction pool: 1
2025: node_1 transaction pool: 1
2025: node_1 blockchain height: 3
2060: node_2 blockchain height: 3
2065: node_3 blockchain height: 3
3000: node_2 transaction pool: 1
3005: node_3 transaction pool: 1
3025: node_1 transaction pool: 1
3025: node_1 blockchain height: 4
3060: node_2 blockchain height: 4
3065: node_3 blockchain height: 4
4000: node_2 transaction pool: 1
4005: node_3 transaction pool: 1
4025: node_1 transaction pool: 1
4025: node_1 blockchain height: 5
4060: node_2 blockchain height: 5
4065: node_3 blockchain height: 5
```

B.2.9 09 - BLOCK acceptance, delayed BLOCKS

```
{
```

```
	"name": "BLOCK acceptance, delayed BLOCKS",
	"seed": 123456789,
	"start": 0,
	"end": 5000,
	"speed": 1,
	"nodes": [
		{
			"id": "node_1",
			"class": "SimulatedBlockWriteNode",
			"permission": 2,
			"maxUploadConnections": 2,
			"maxDownloadConnections": 2
	},
	{
			"id": "node_2",
			"class": "SimulatedTransactionNode",
		"permission": 1,
			"maxUploadConnections": 3,
			"maxDownloadConnections": 3,
			"ticksPerTransaction": 1000
	},
		{
			"id": "node_3",
			"class": "SimulatedDelayedBlockWriteNode",
			"permission": 2,
			"maxUploadConnections": 3,
			"maxDownloadConnections": 3
	},
	{
			"id": "node_4",
			"class": "SimulatedReadNode",
		"permission": 0,
			"maxUploadConnections": 2,
			"maxDownloadConnections": 2
	}
	],
	"connections": [
		"node_1 node_2 UD 25",
		"node_1 node_3 UD 10",
		"node_2 node_3 UD 5",
		"node_3 node_4 UD 10",
		"node_2 node_4 UD 10"
	],
	"logging": {
		"special": ["BLOCK_HEIGHT"],
		"signals": ["NEW_BLOCK"]
	}
}
```

Expected output

```
15: node_1 blockchain height: 1
```

```
35: node_1 -> node_3 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
35: node_3 blockchain height: 1
45: node_3 -> node_4 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
45: node_4 blockchain height: 1
50: node_1 -> node_2 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
50: node_2 blockchain height: 1
55: node_4 -> node_2 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
56: node_2 -> node_3 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
56: node_4 -> node_3 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
60: node_2 -> node_4 (sig 1) hash: 563
   ↪ c1555e5ab592eb7221d2a5c49a084865cad85a049b649faa0485ca8b68277,
   ↪ payload: 1:node_1:INIT:INIT:node_2;-1040170699
1005: node_3 blockchain height: 2
1020: node_3 -> node_2 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1020: node_2 blockchain height: 2
1025: node_3 -> node_1 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1025: node_1 blockchain height: 2
1025: node_3 -> node_4 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1025: node_4 blockchain height: 2
1031: node_2 -> node_4 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1035: node_4 -> node_2 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
```

```
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1045: node_2 -> node_1 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1051: node_1 -> node_2 (sig 1) hash: 8348738959751
   ↪ cd3fe8b3a9b8b2d768ed1230e5fa8e7a6e9ddc2c878ab5451ab, payload: 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
2015: node_1 blockchain height: 3
2035: node_1 -> node_3 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2035: node_3 blockchain height: 3
2045: node_3 -> node_4 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2045: node_4 blockchain height: 3
2050: node_1 -> node_2 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2050: node_2 blockchain height: 3
2055: node_4 -> node_2 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2056: node_2 -> node_3 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2056: node_4 -> node_3 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2060: node_2 -> node_4 (sig 1) hash: 8
   ↪ cd41a6de0cf5039daf40dccce812675ce2081817ce17532a2b3ae0a21b36b14,
   ↪ payload: 3:node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
```

```
3005: node_3 blockchain height: 4
3020: node_3 -> node_2 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3020: node_2 blockchain height: 4
3025: node_3 -> node_4 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3025: node_4 blockchain height: 4
3027: node_3 -> node_1 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3027: node_1 blockchain height: 4
3031: node_2 -> node_4 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3035: node_4 -> node_2 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3045: node_2 -> node_1 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3053: node_1 -> node_2 (sig 1) hash: 36
   ↪ bb767271509226c75f5c8d67df8ec1eae9d607ba70bf38178bccfd8a03f820,
   ↪ payload: 4:node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
4015: node_1 blockchain height: 5
4035: node_1 -> node_3 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
4035: node_3 blockchain height: 5
4045: node_3 -> node_4 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
```

```
4045: node_4 blockchain height: 5
4050: node_1 -> node_2 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
4050: node_2 blockchain height: 5
4055: node_4 -> node_2 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
4056: node_2 -> node_3 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
4056: node_4 -> node_3 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
4060: node_2 -> node_4 (sig 1) hash:
   ↪ db225397a7209fec6cec8f4d12100a8a7cec43ddfd9a08343dbe449183703c98,
   ↪ payload: 5:node_1:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-260042
```

B.2.10 10 - BLOCK acceptance, malicious BLOCKS

```
{
      "name": "BLOCK acceptance, malicious BLOCKS",
      "seed": 123456789,
      "start": 0,
      "end": 20000,
      "speed": 1,
      "nodes": [
            {
                  "id": "node_1",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
                  "maxUploadConnections": 2,
                  "maxDownloadConnections": 2
      },
      {
                  "id": "node_2",
                  "class": "SimulatedTransactionNode",
            "permission": 1,
                  "maxUploadConnections": 3,
                  "maxDownloadConnections": 3,
                  "ticksPerTransaction": 1000
```

```
	},
		{
			"id": "node_3",
			"class": "SimulatedMaliciousBlockWriteNode",
			"permission": 2,
			"maxUploadConnections": 3,
			"maxDownloadConnections": 3
	},
	{
			"id": "node_4",
			"class": "SimulatedReadNode",
		"permission": 0,
			"maxUploadConnections": 2,
			"maxDownloadConnections": 2
	}
	],
	"connections": [
		"node_1 node_2 UD 25",
		"node_1 node_3 UD 10",
		"node_2 node_3 UD 5",
		"node_3 node_4 UD 10",
		"node_2 node_4 UD 10"
	],
	"logging": {
		"special": ["BLOCK_ACCEPTANCE", "BLOCK_CREATION"],
		"signals": ["BLOCK_REQUEST"]
	}
}
```

Expected output

```
15: node_1 Regular block from node_1, height 1, new chain height of 1 1:
   ↪ node_1:INIT:INIT:node_2;-1040170699
35: node_3 Regular block from node_1, height 1, new chain height of 1 1:
   ↪ node_1:INIT:INIT:node_2;-1040170699
40: node_2 Regular block from node_1, height 1, new chain height of 1 1:
   ↪ node_1:INIT:INIT:node_2;-1040170699
45: node_4 Regular block from node_1, height 1, new chain height of 1 1:
   ↪ node_1:INIT:INIT:node_2;-1040170699
1005: node_3 Regular block from node_3, height 2, new chain height of 2 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1005: node_3 Will create a new block: Regular
1020: node_2 Regular block from node_3, height 2, new chain height of 2 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
1025: node_1 Regular block from node_3, height 2, new chain height of 2 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
```

```
1025: node_4 Regular block from node_3, height 2, new chain height of 2 2:
   ↪ node_3:
   ↪ eb35a2b8525739a23270d43caf47a1394dd306b60ce8ebb7f5413ac4d142750f:INIT
   ↪ :node_2;-1040170
2015: node_1 Regular block from node_1, height 3, new chain height of 3 3:
   ↪ node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2035: node_3 Regular block from node_1, height 3, new chain height of 3 3:
   ↪ node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2040: node_2 Regular block from node_1, height 3, new chain height of 3 3:
   ↪ node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
2045: node_4 Regular block from node_1, height 3, new chain height of 3 3:
   ↪ node_1:
   ↪ a29efccf4f4a183e94558e24c3e3091673ad105490ec91f9f9cf986f5e410d6f:INIT
   ↪ :node_2;-520085
3005: node_3 Will create a new block: Invalid height -> too far in the
   ↪ future
3016: node_3 Will create a new block: Invalid height
3027: node_3 Will create a new block: Invalid creator
3038: node_3 Will create a new block: Invalid creator
3049: node_3 Will create a new block: Invalid creator
3060: node_3 Will create a new block: Invalid Hash chain
3065: node_2 Regular block from node_3, height 4, new chain height of 4 4:
   ↪ node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:8426
   ↪ fe6df433aa66a723428b5c6776572f7e7ff4af42b69831a99898225cfd1f:node_2
   ↪ ;-346723
3070: node_1 Regular block from node_3, height 4, new chain height of 4 4:
   ↪ node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:8426
   ↪ fe6df433aa66a723428b5c6776572f7e7ff4af42b69831a99898225cfd1f:node_2
   ↪ ;-346723
3070: node_4 Regular block from node_3, height 4, new chain height of 4 4:
   ↪ node_3:37
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:8426
   ↪ fe6df433aa66a723428b5c6776572f7e7ff4af42b69831a99898225cfd1f:node_2
   ↪ ;-346723
3071: node_3 Will create a new block: Invalid Hash chain
3082: node_3 Will create a new block: Invalid height -> too far in the
   ↪ future
3093: node_3 Will create a new block: Invalid creator
3104: node_3 Will create a new block: Invalid height -> too far in the
   ↪ future
3115: node_3 Will create a new block: Invalid Hash chain
3126: node_3 Regular block from node_3, height 4, new chain height of 4 4:
   ↪ node_3:37
```

```
   ↪ b711f6027a7a3fe752b8575b530ec24e8b420c1f01676b7885bd35d74c2eab:INIT:
   ↪ node_2;-346723
3126: node_3 Will create a new block: Regular
4015: node_1 Regular block from node_1, height 5, new chain height of 5 5:
   ↪ node_1:
   ↪ fd0f738a4ea5d5e3f1c71cf858e22b45cd28b4aa28b70abe89f450b4e816b8a9:INIT
   ↪ :node_2;-260042
4050: node_2 Regular block from node_1, height 5, new chain height of 5 5:
   ↪ node_1:
   ↪ fd0f738a4ea5d5e3f1c71cf858e22b45cd28b4aa28b70abe89f450b4e816b8a9:INIT
   ↪ :node_2;-260042
4060: node_4 Regular block from node_1, height 5, new chain height of 5 5:
   ↪ node_1:
   ↪ fd0f738a4ea5d5e3f1c71cf858e22b45cd28b4aa28b70abe89f450b4e816b8a9:INIT
   ↪ :node_2;-260042
5128: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
5128: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
5128: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
6017: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6017: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6052: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6052: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6052: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6062: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
6062: node_4 -> node_2 (sig 5) hash: 08587749
```

```
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
7129: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
7129: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
7129: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
8018: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8018: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8053: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8053: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8053: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8063: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
8063: node_4 -> node_2 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
9130: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
9130: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
9130: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
```

```
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
10019: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10019: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10054: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10054: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10054: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10064: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
10064: node_4 -> node_2 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
11131: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
11131: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
11131: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
12020: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
12020: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
12055: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
```

```
12055: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
12055: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
12065: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
12065: node_4 -> node_2 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
13132: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
13132: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
13132: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
14021: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14021: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14056: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14056: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14056: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14066: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
14066: node_4 -> node_2 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
```

```
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
15132: node_3 Shifting forger
15132: node_3 Future block received from node_1, height 5
15133: node_3 -> node_2 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
15133: node_3 -> node_1 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
15133: node_3 -> node_4 (sig 5) hash:
   ↪ b806341b44d87822d8ad4c03849e55b07f6f1cd312870d53c77bf5ef294e774e,
   ↪ payload: 414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2
15133: node_3 Will create a new block: Invalid height -> too far in the
   ↪ future
15133: node_3 Future block received from node_1, height 5
15144: node_3 Regular block from node_3, height 5, new chain height of 5
   ↪ 5:node_3:414
   ↪ b79a23caf6c9eb1016fab3999d75763a6ecbd94ba216ab29c03ab378917c2:INIT:
   ↪ node_2;-148595||node_2;-130021||node_2;-260042||node_2;-208034||
   ↪ node_2;-115574||node_2;-74297||node_2;-80013||node_2;-94560||node_2
   ↪ ;-104017||node_2;-86680||node_2;-173361||node_2;-69344
15144: node_3 Will create a new block: Regular
16021: node_1 Shifting forger
16022: node_1 -> node_3 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16022: node_1 -> node_2 (sig 5) hash:
   ↪ f689bc5dbd619872d12a57cffb7dba8d37818490d216a199668f63d20bb277b2,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16022: node_1 Regular block from node_1, height 6, new chain height of 6
   ↪ 6:node_1:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f:INIT
   ↪ :node_2;-148595||node_2;-130021||node_2;-208034||node_2;-115574||
   ↪ node_2;-74297||node_2;-65010||node_2;-80013||node_2;-94560||node_2
   ↪ ;-104017||node_2;-86680||node_2;-173361||node_2;-69344
16056: node_2 Shifting forger
16057: node_2 -> node_3 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16057: node_2 -> node_4 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16057: node_2 -> node_1 (sig 5) hash:
   ↪ a916ea8c067dbe60b1197e50d65d90a0cf6a6a8e6dd302bd120b71654ce57022,
```

```
   ↪ payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16057: node_2 Regular block from node_1, height 6, new chain height of 6
   ↪ 6:node_1:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f:INIT
   ↪ :node_2;-148595||node_2;-130021||node_2;-208034||node_2;-115574||
   ↪ node_2;-74297||node_2;-65010||node_2;-80013||node_2;-94560||node_2
   ↪ ;-104017||node_2;-86680||node_2;-173361||node_2;-69344
16066: node_4 Shifting forger
16067: node_4 -> node_3 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16067: node_4 -> node_2 (sig 5) hash: 08587749
   ↪ aa9f05637da2a752a6b065d1d66b901f931ebb022d9023593f818774, payload:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f
16067: node_4 Regular block from node_1, height 6, new chain height of 6
   ↪ 6:node_1:
   ↪ eb97b6015a2134717b907e13a9a2389c793337e03dd8a1db1829024e3bd7d27f:INIT
   ↪ :node_2;-148595||node_2;-130021||node_2;-208034||node_2;-115574||
   ↪ node_2;-74297||node_2;-65010||node_2;-80013||node_2;-94560||node_2
   ↪ ;-104017||node_2;-86680||node_2;-173361||node_2;-69344
17146: node_3 -> node_2 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
17146: node_3 -> node_1 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
17146: node_3 -> node_4 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
18024: node_1 -> node_3 (sig 5) hash:
   ↪ e362e85acc522fae4bc036d84c4c716b8718d02f783cf1aa478bdea7cc889b4e,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18024: node_1 -> node_2 (sig 5) hash:
   ↪ e362e85acc522fae4bc036d84c4c716b8718d02f783cf1aa478bdea7cc889b4e,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18059: node_2 -> node_3 (sig 5) hash: 4
   ↪ d5cb0dd314744edf4c30c7c43d5e46c3b0c6681846b107f7c4c1adbbf1eba3e,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18059: node_2 -> node_4 (sig 5) hash: 4
   ↪ d5cb0dd314744edf4c30c7c43d5e46c3b0c6681846b107f7c4c1adbbf1eba3e,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18059: node_2 -> node_1 (sig 5) hash: 4
   ↪ d5cb0dd314744edf4c30c7c43d5e46c3b0c6681846b107f7c4c1adbbf1eba3e,
```

```
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18069: node_4 -> node_3 (sig 5) hash:
   ↪ d162921b9c6e9ab5bdbf4d90516d2beba9aee1eefd5305f3021fc29c95c382f1,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
18069: node_4 -> node_2 (sig 5) hash:
   ↪ d162921b9c6e9ab5bdbf4d90516d2beba9aee1eefd5305f3021fc29c95c382f1,
   ↪ payload: 1
   ↪ d4d3902995e7c22ae88a534f011cb0fae1bd82fb275fb7d38f04ceda9bb06df
19147: node_3 -> node_2 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
19147: node_3 -> node_1 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
19147: node_3 -> node_4 (sig 5) hash: 9
   ↪ ac9221759f0b43c140f9626823ca3903687e0cfb107adf8f1cc49a31ad467c6,
   ↪ payload: 3
   ↪ bebde26b026a34a1dd1d879910e4f0a414c5832eb36d622d18a5ece33f5143c
```

Appendix C

EET testing

All EET tests are placed under eet/folder, ie.

```
eet/eet_[id]_[type].json

e.g. eet/eet_2_xml.json
e.g. eet/eet_3_json.json
```

C.1 Test 1

C.1.1 Scenario

```
{
      "name": "EET, scenario 1: 1500 publish, 1 write, 2 transactions per
         ↪ second, JSON",
      "seed": 123456789,
      "start": 0,
      "end": 15000,
      "speed": 1,
      "many_nodes":
      [
            {
                  "id_prefix": "publish",
                  "class": "SimulatedSalesTransactionNode",
                  "permission": 1,
                  "count": 1500,
                  "speedUpload": 10000,
                  "speedDownload": 10000,
                  "transactionsPerSecond": 2,
                  "style": "json"
            },
            {
                  "id_prefix": "write",
                  "class": "SimulatedBlockWriteNode",
                  "permission": 2,
```

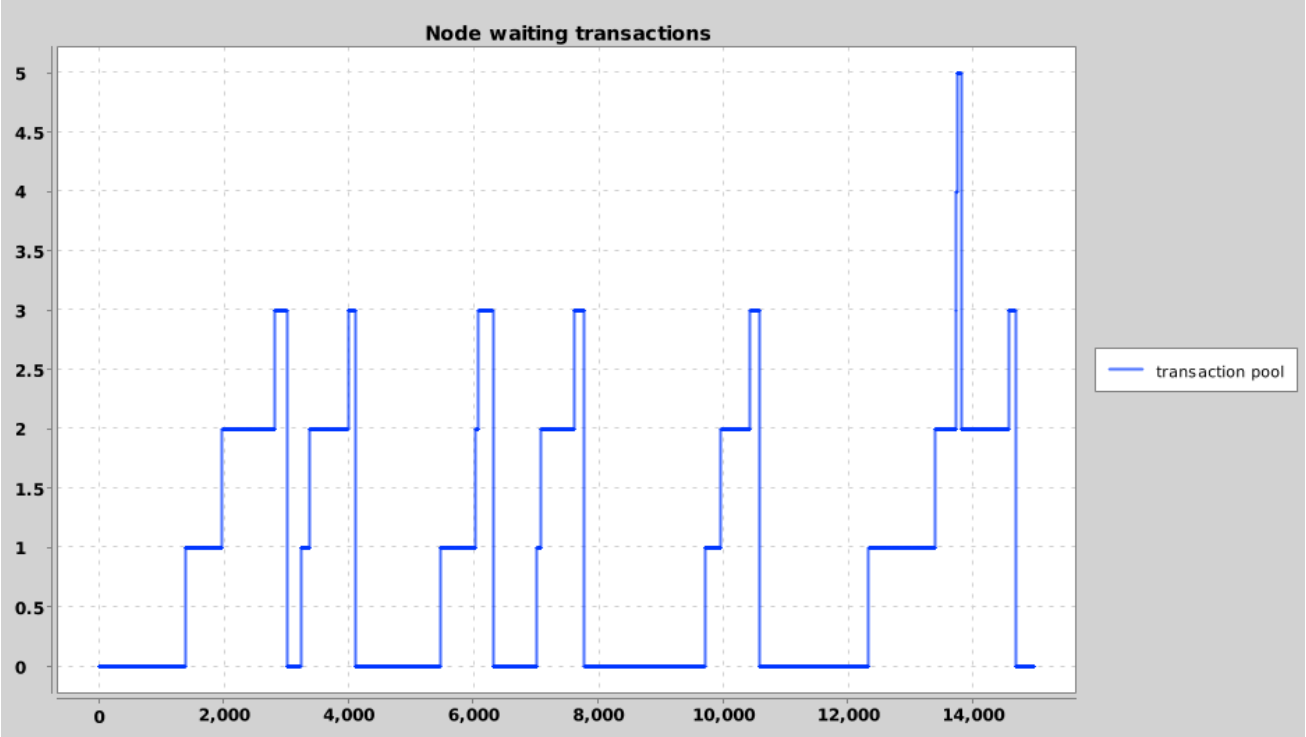

Figure C.1: Scenario 1, JSON

```
                    "count": 1,
                    "speedUpload": 10000,
                    "speedDownload": 10000,
                    "minTransactions": 2
              }
        ],
        "graphs": {
              "node": ["publish_0", "transaction_pool", "
                 ↪ transactions_verified", "connection_speeds", "
                 ↪ waiting_signals"]
        }
}
```

C.1.2 Graphs

C.2 Test 2

C.2.1 Scenario

```
{
        "name": "EET, scenario 2: 150 publish, 1 write, 450 transactions per
           ↪  second, JSON",
        "seed": 123456789,
        "start": 0,
        "end": 15000,
        "speed": 1,
```

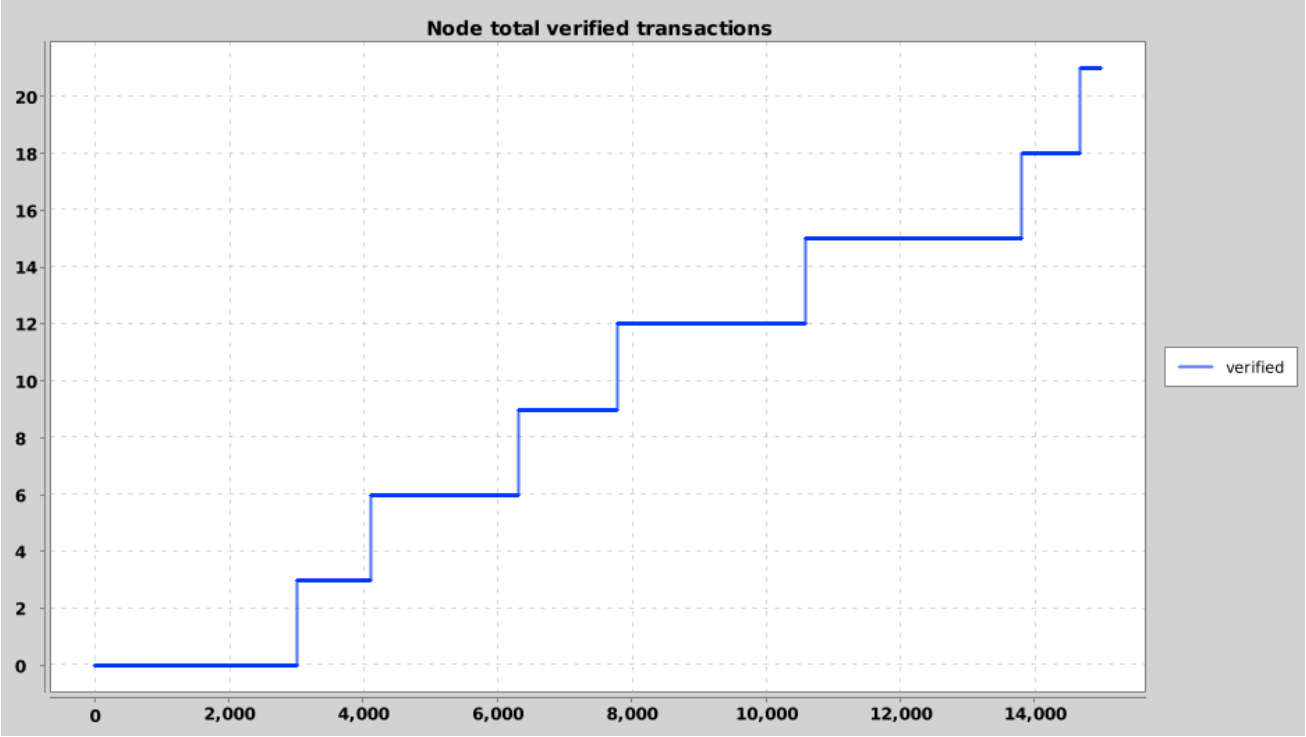

Figure C.2: Scenario 1, JSON

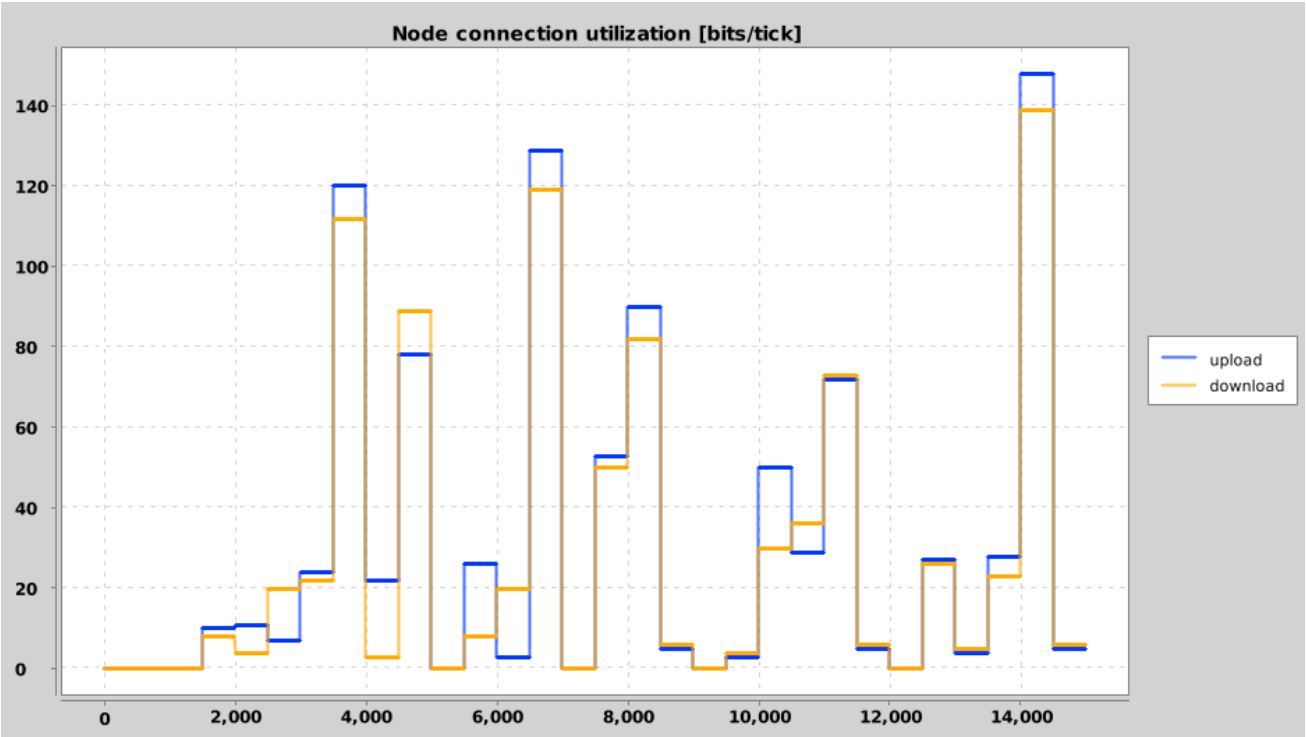

Figure C.3: Scenario 1, JSON

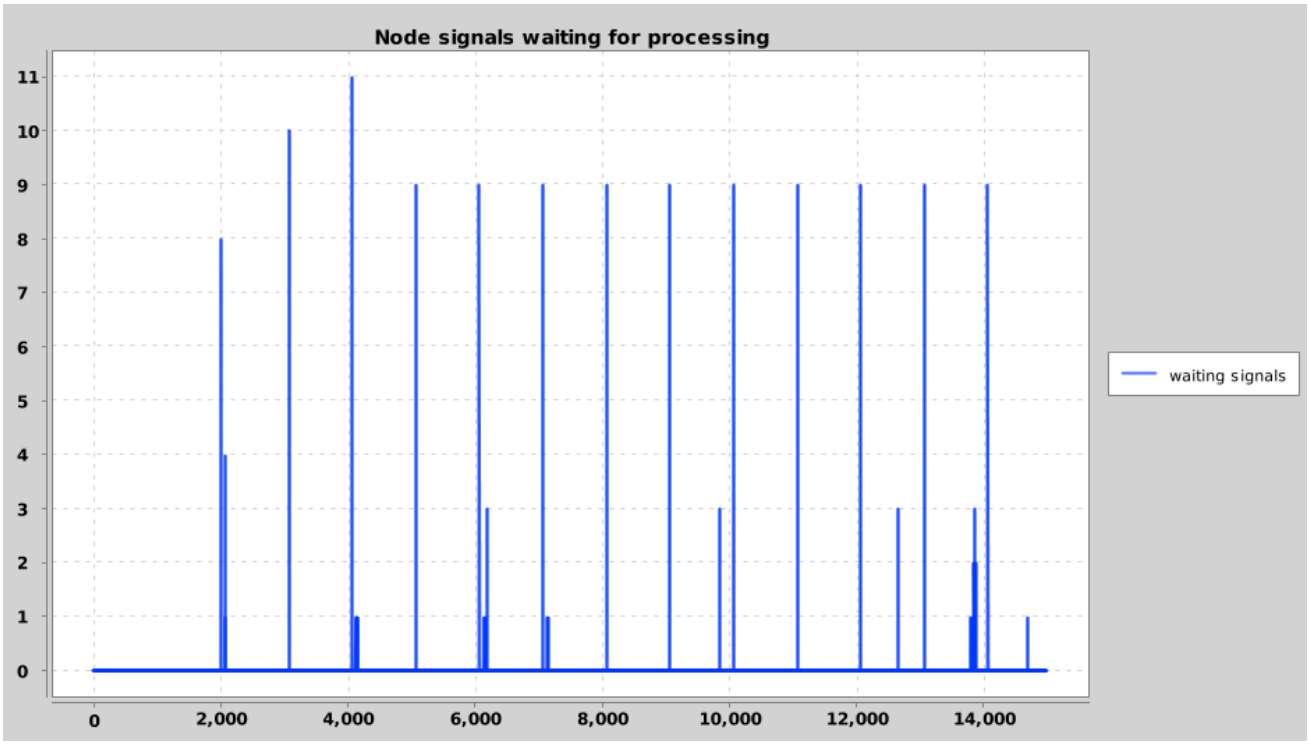

Figure C.4: Scenario 1, JSON

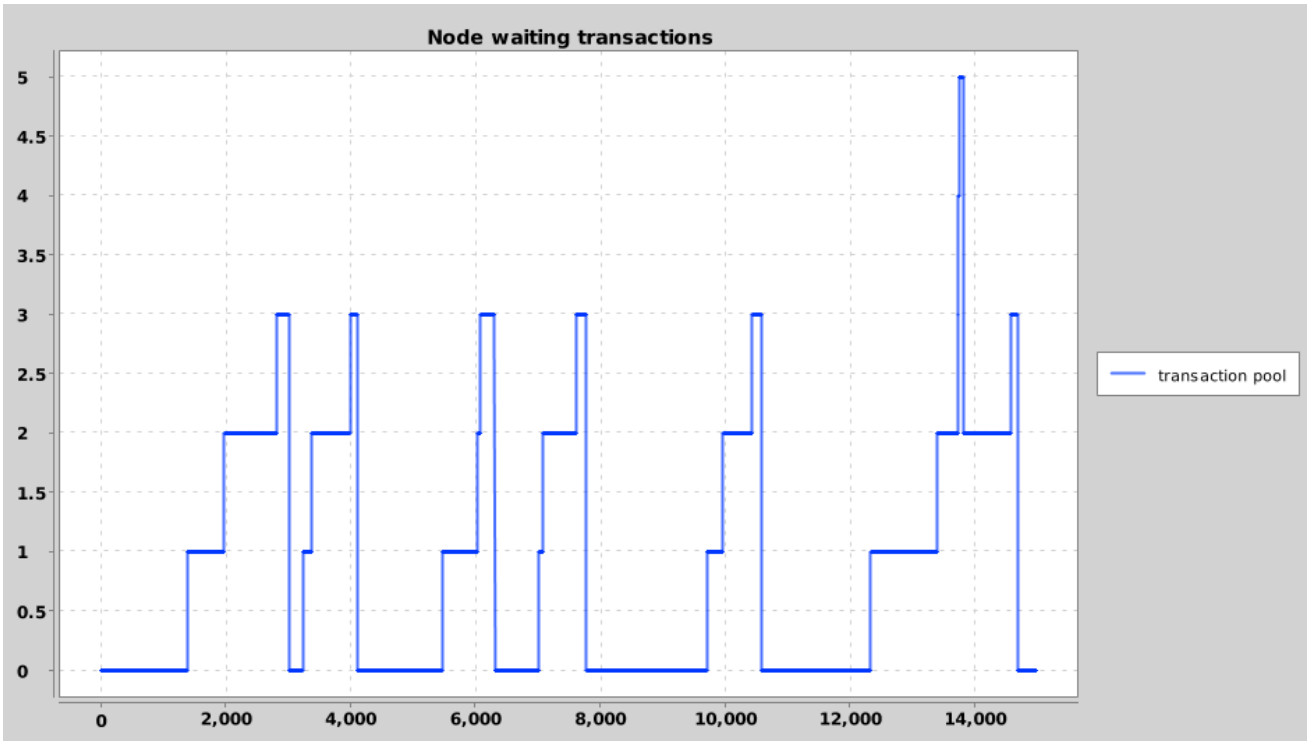

Figure C.5: Scenario 1, XML

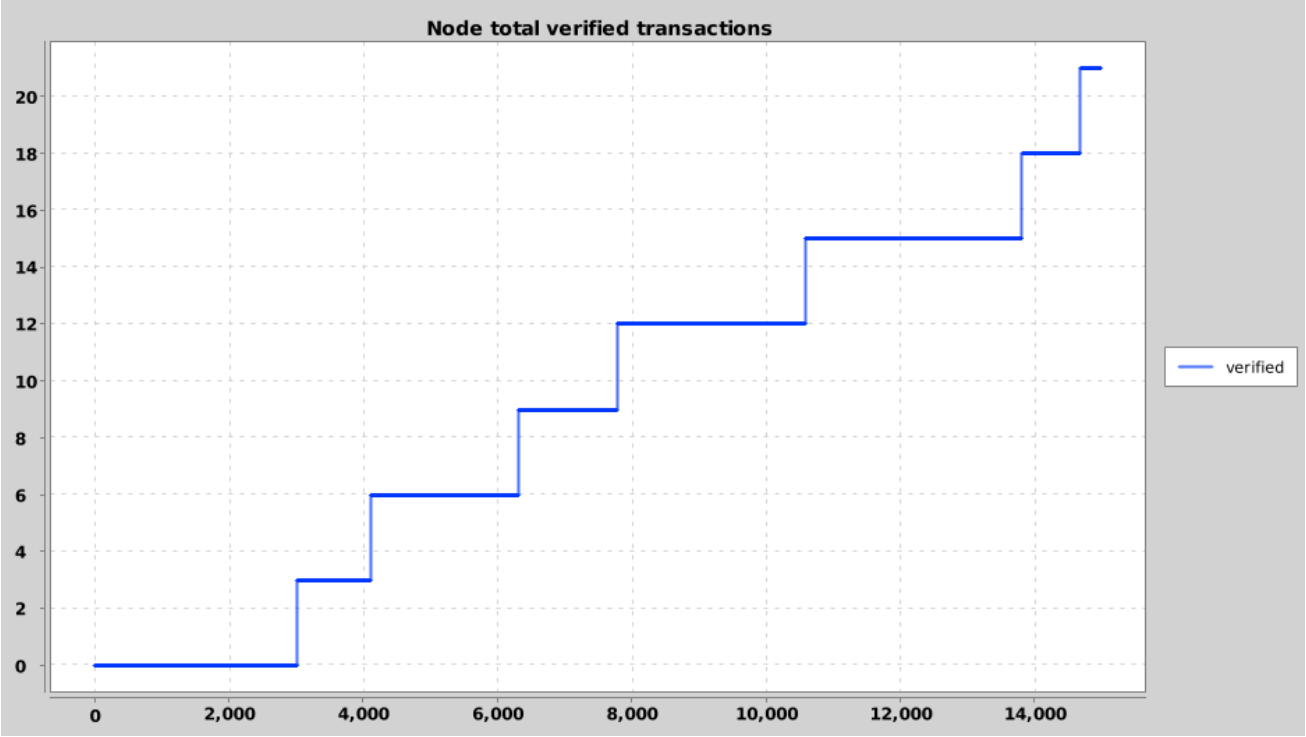

Figure C.6: Scenario 1, XML

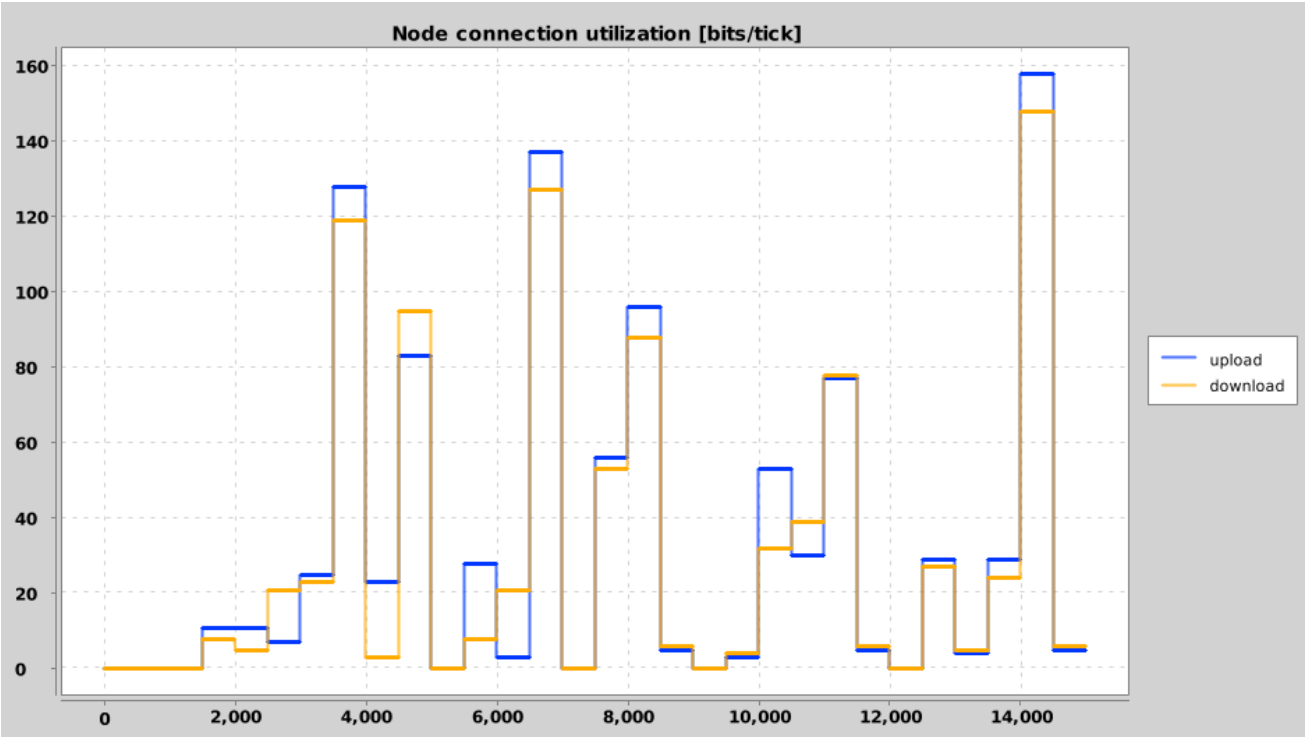

Figure C.7: Scenario 1, XML

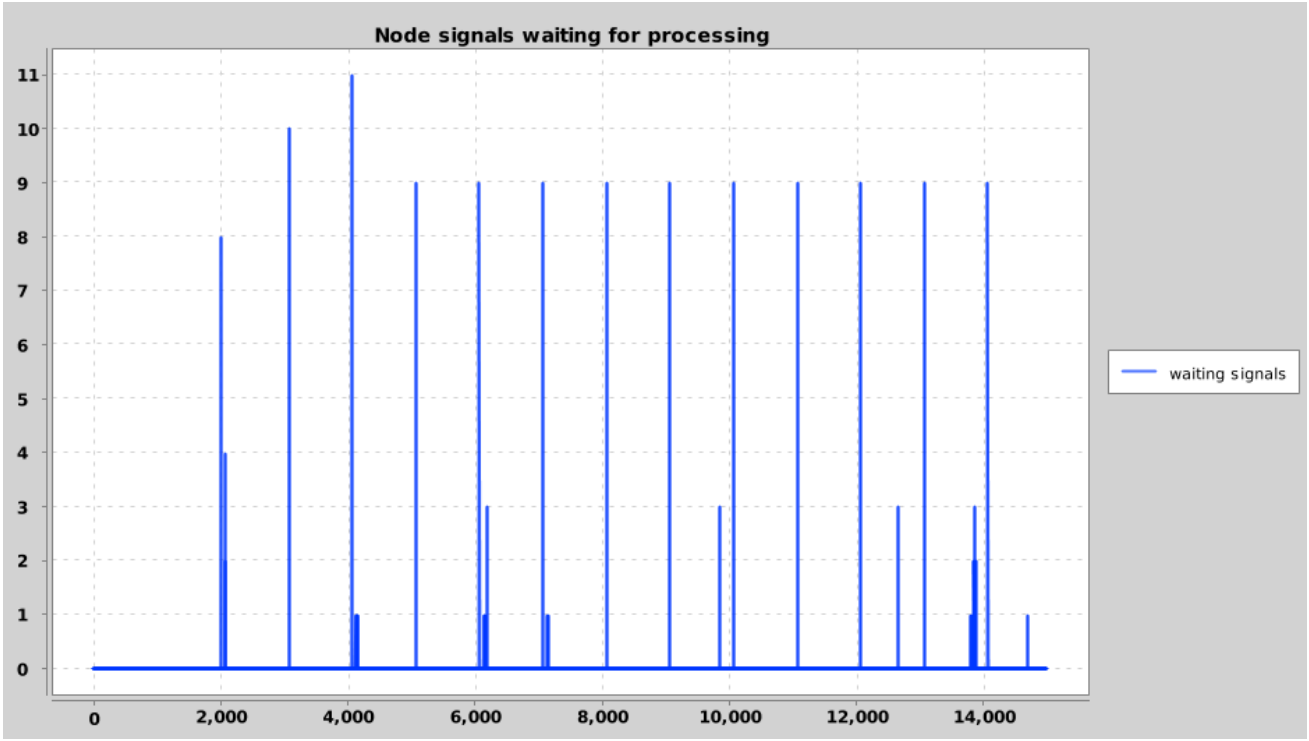

Figure C.8: Scenario 1, XML

```
        "many_nodes":
        [
                {
                        "id_prefix": "publish",
                        "class": "SimulatedSalesTransactionNode",
                        "permission": 1,
                        "count": 150,
                        "speedUpload": 10000,
                        "speedDownload": 10000,
                        "transactionsPerSecond": 450,
                        "style": "json"
                },
                {
                        "id_prefix": "write",
                        "class": "SimulatedBlockWriteNode",
                        "permission": 2,
                        "count": 1,
                        "speedUpload": 10000,
                        "speedDownload": 10000,
                        "minTransactions": 450
                }
        ],
        "graphs": {
                "node": ["publish_0", "transaction_pool", "
                   ↪ transactions_verified", "connection_speeds", "
                   ↪ waiting_signals"]
        }
}
```

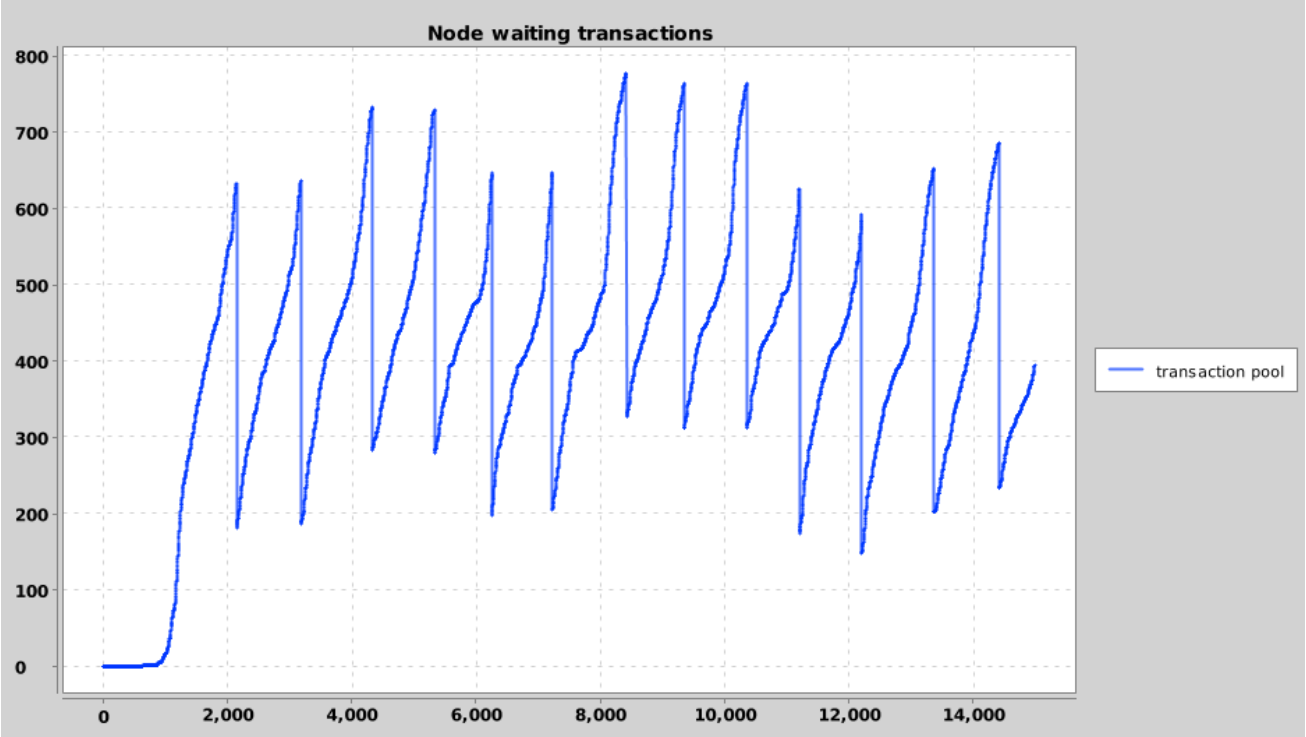

Figure C.9: Scenario 2, JSON

C.2.2 Graphs

C.3 Test 3

C.3.1 Scenario

```
{
	"name": "EET, scenario 3: 150 publish, 3 write, 450 transactions per
	   ↪ second, JSON",
	"seed": 123456789,
	"start": 0,
	"end": 15000,
	"speed": 1,
	"many_nodes":
	[
		{
			"id_prefix": "publish",
			"class": "SimulatedSalesTransactionNode",
			"permission": 1,
			"count": 150,
			"speedUpload": 10000,
			"speedDownload": 10000,
			"transactionsPerSecond": 450,
			"style": "json"
		},
		{
			"id_prefix": "write",
```

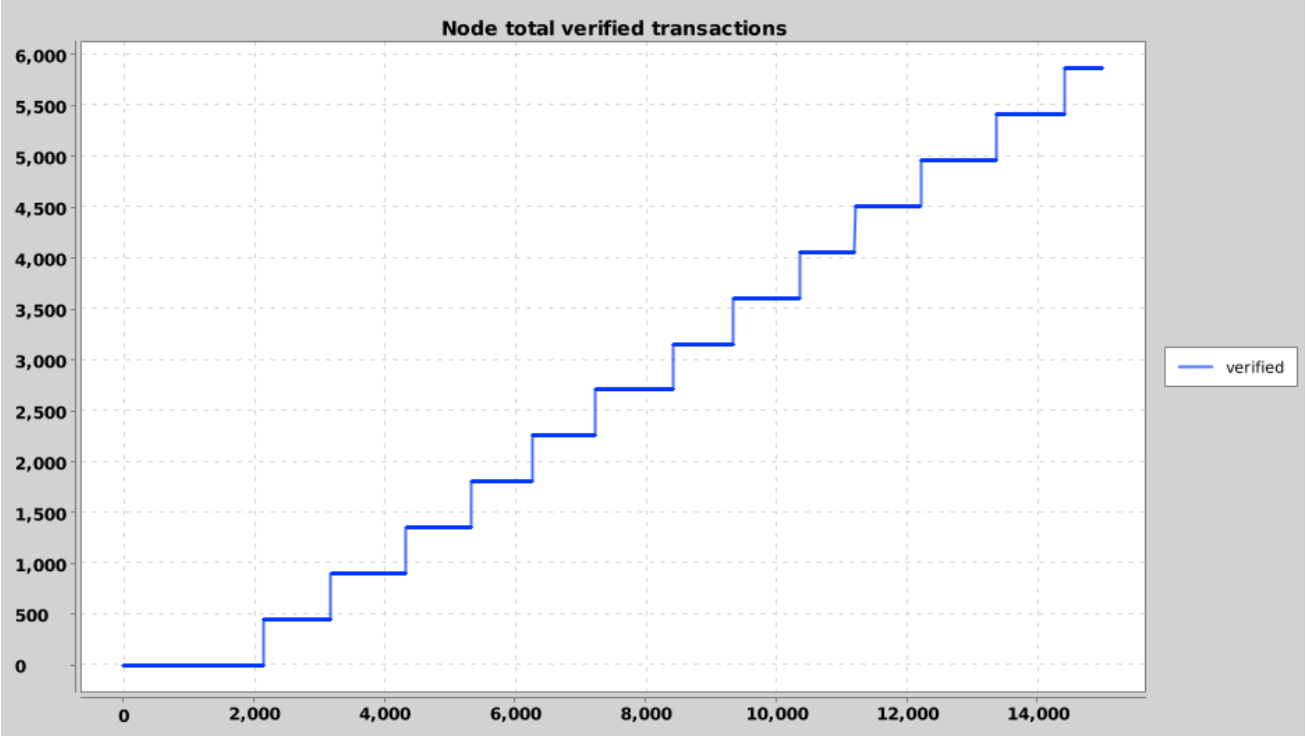

Figure C.10: Scenario 2, JSON

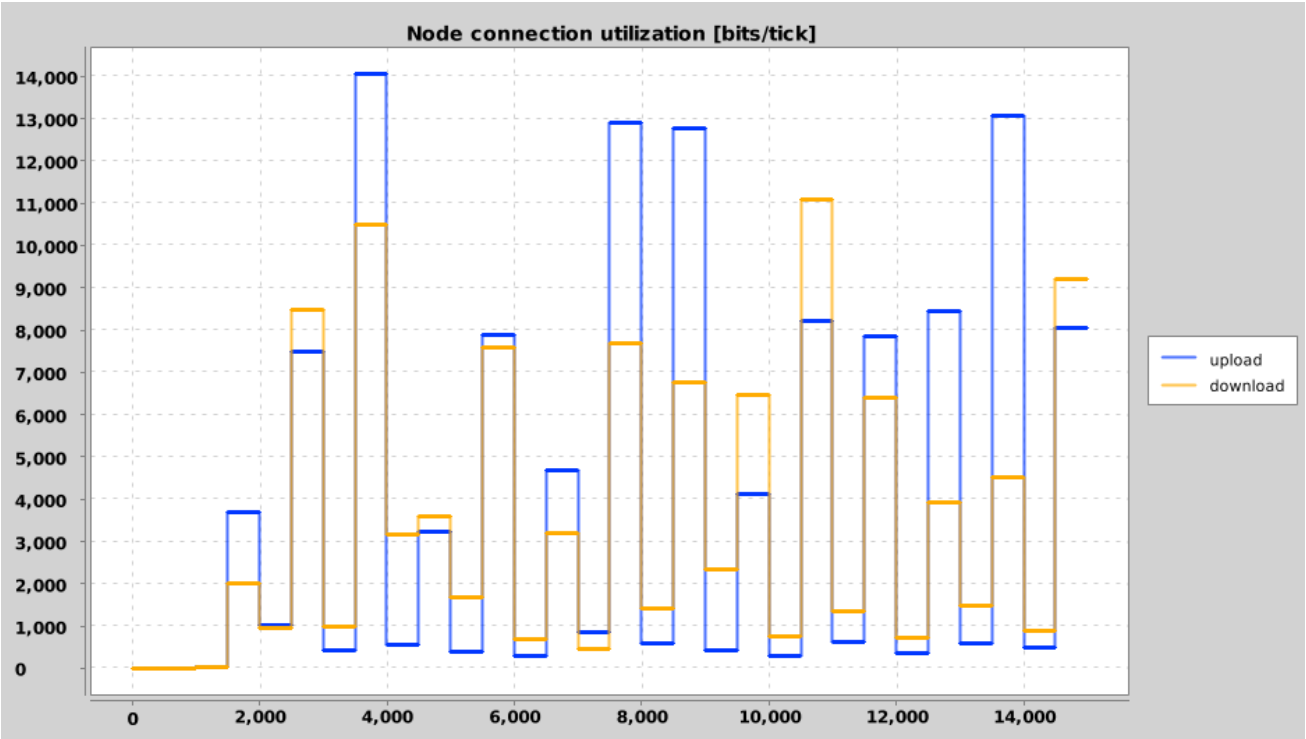

Figure C.11: Scenario 2, JSON

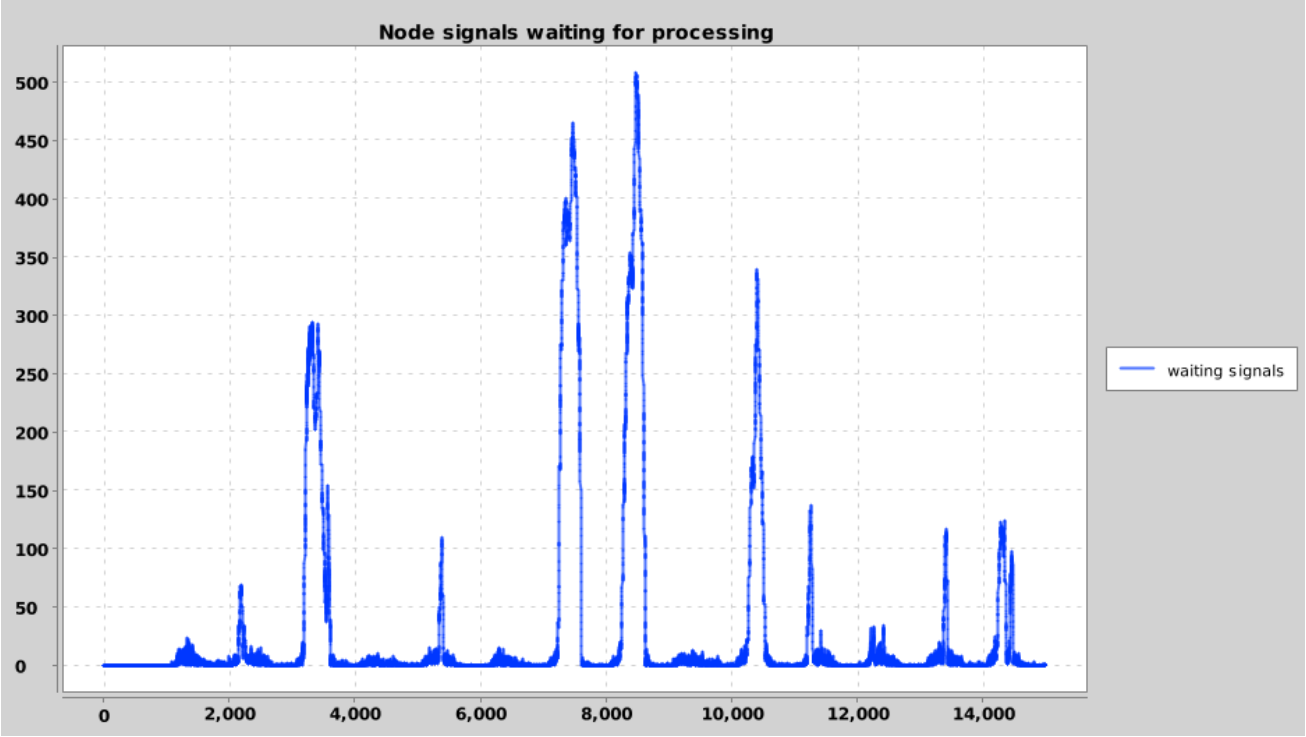

Figure C.12: Scenario 2, JSON

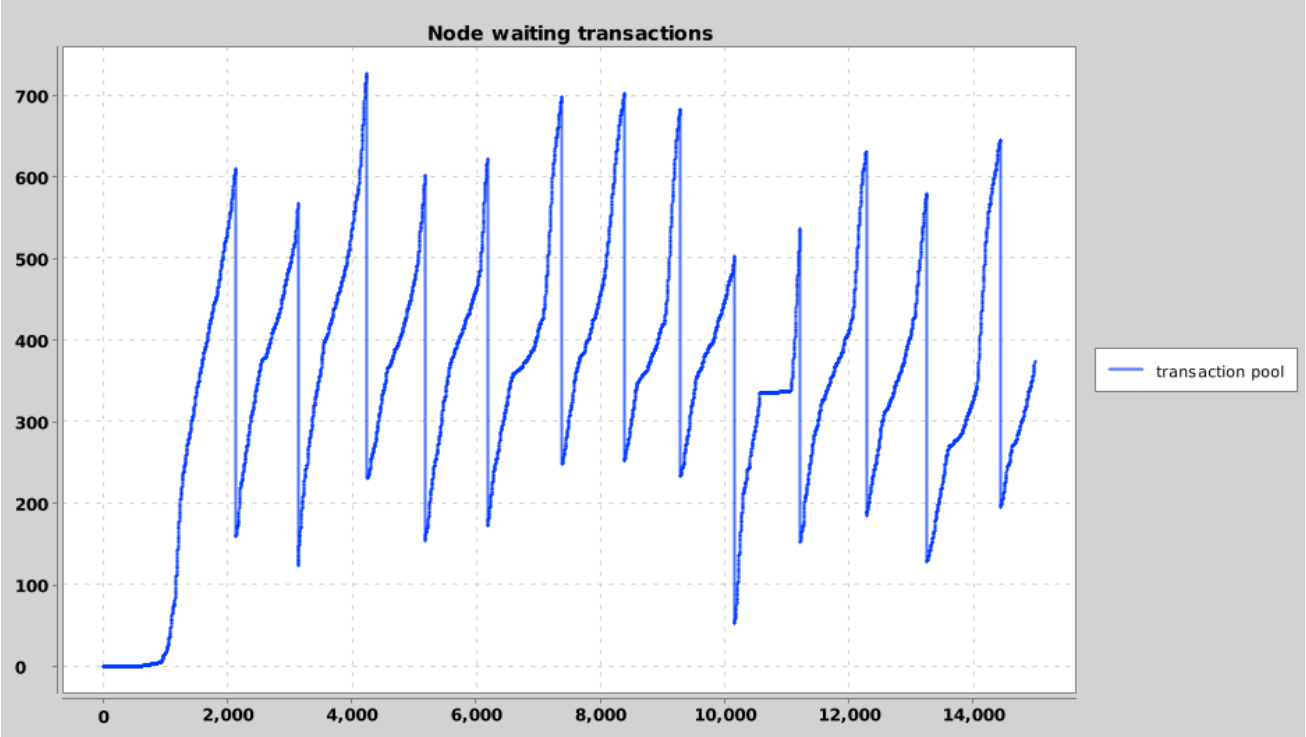

Figure C.13: Scenario 2, XML

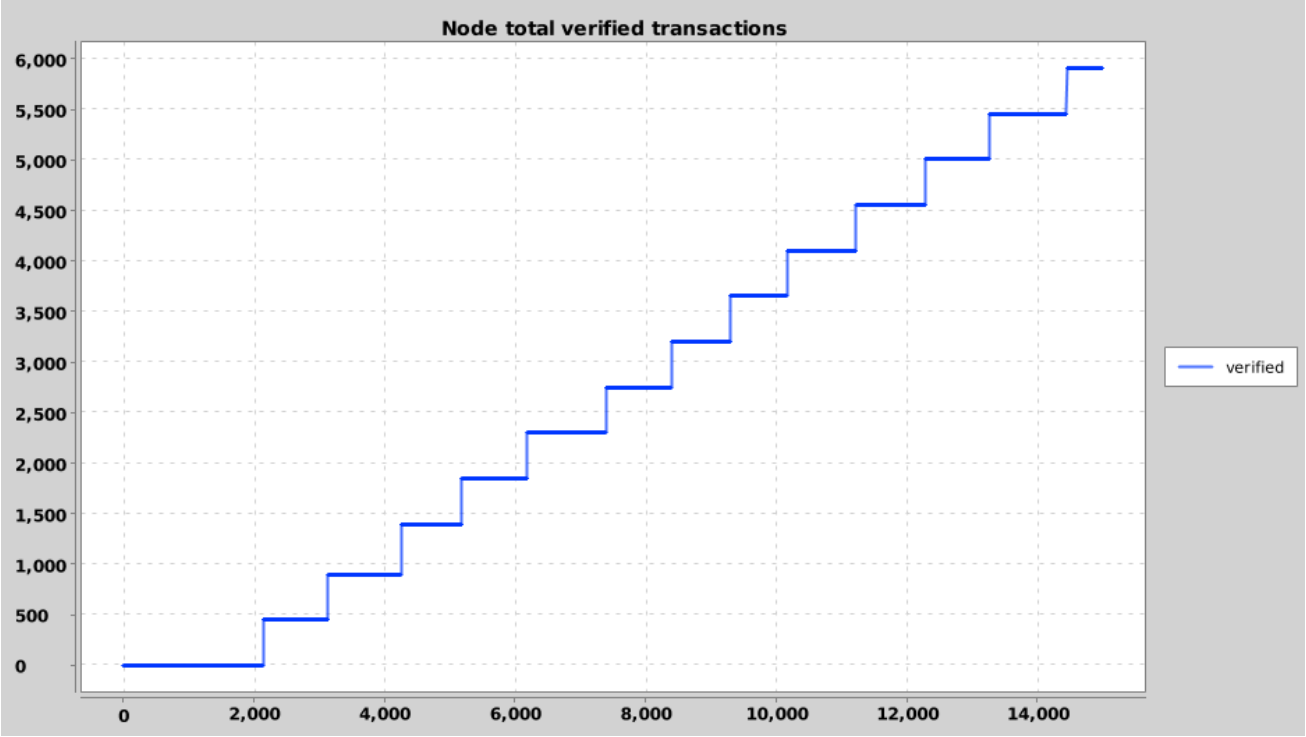

Figure C.14: Scenario 2, XML

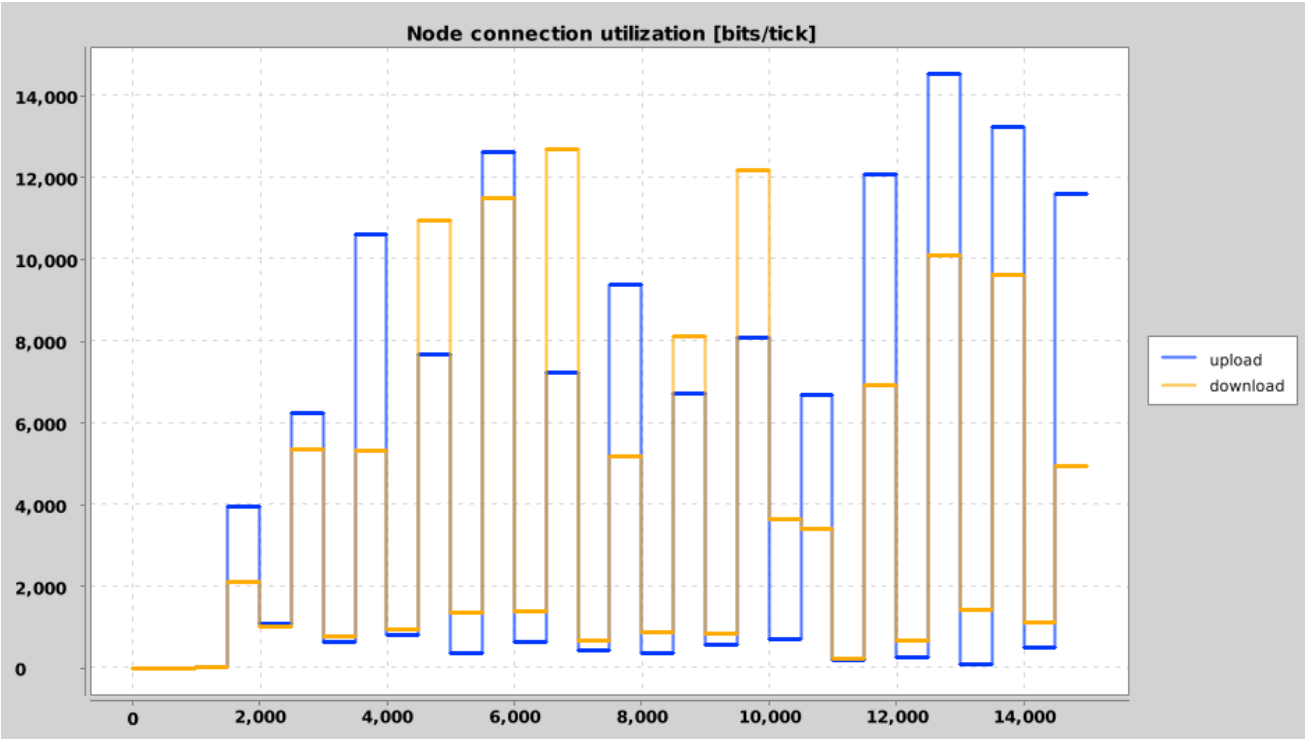

Figure C.15: Scenario 2, XML

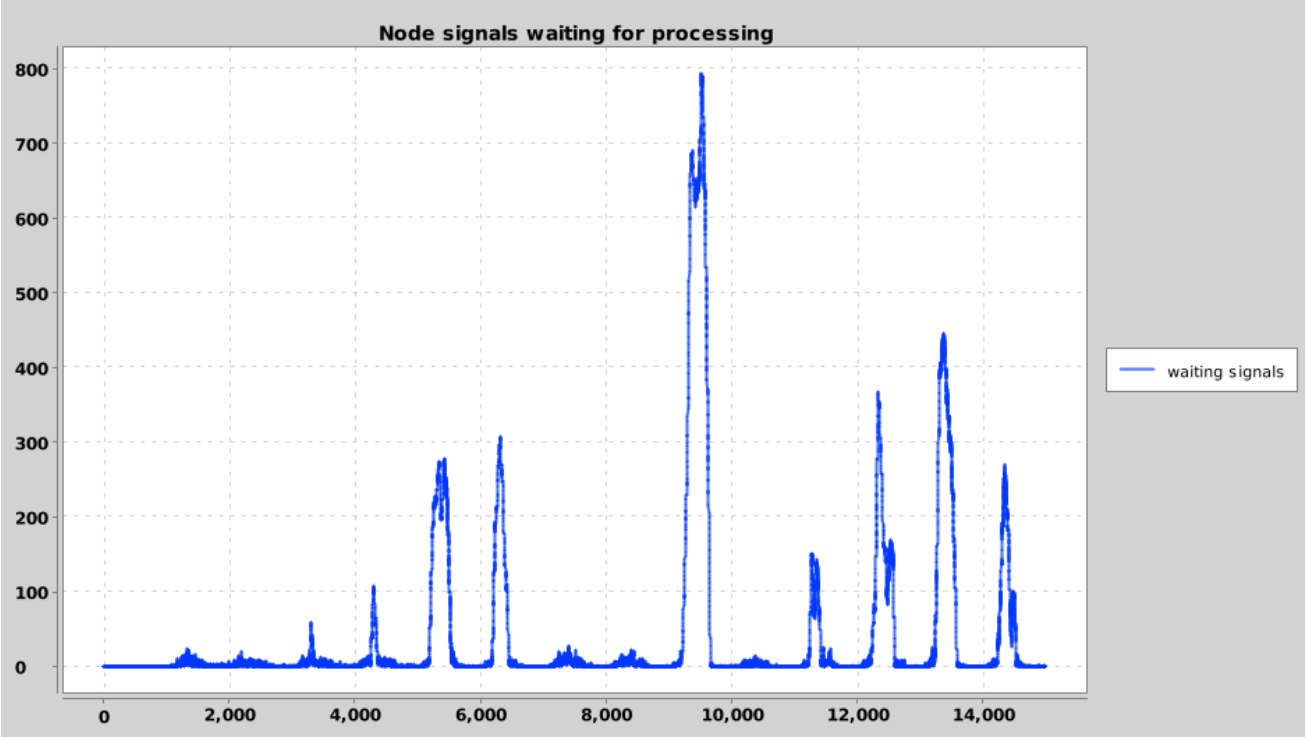

Figure C.16: Scenario 2, XML

```
                "class": "SimulatedBlockWriteNode",
                "permission": 2,
                "count": 3,
                "speedUpload": 10000,
                "speedDownload": 10000,
                "minTransactions": 450
            }
        ],
        "graphs": {
            "node": ["publish_0", "transaction_pool", "
               ↪ transactions_verified", "connection_speeds", "
               ↪ waiting_signals"]
        }
}
```

C.3.2 Graphs

C.4 Test 4

C.4.1 Scenario

```
{
        "name": "EET, scenario 4: 300 publish, 3 write, 450 transactions per
           ↪  second, JSON",
        "seed": 123456789,
        "start": 0,
```

Figure C.17: Scenario 3, JSON

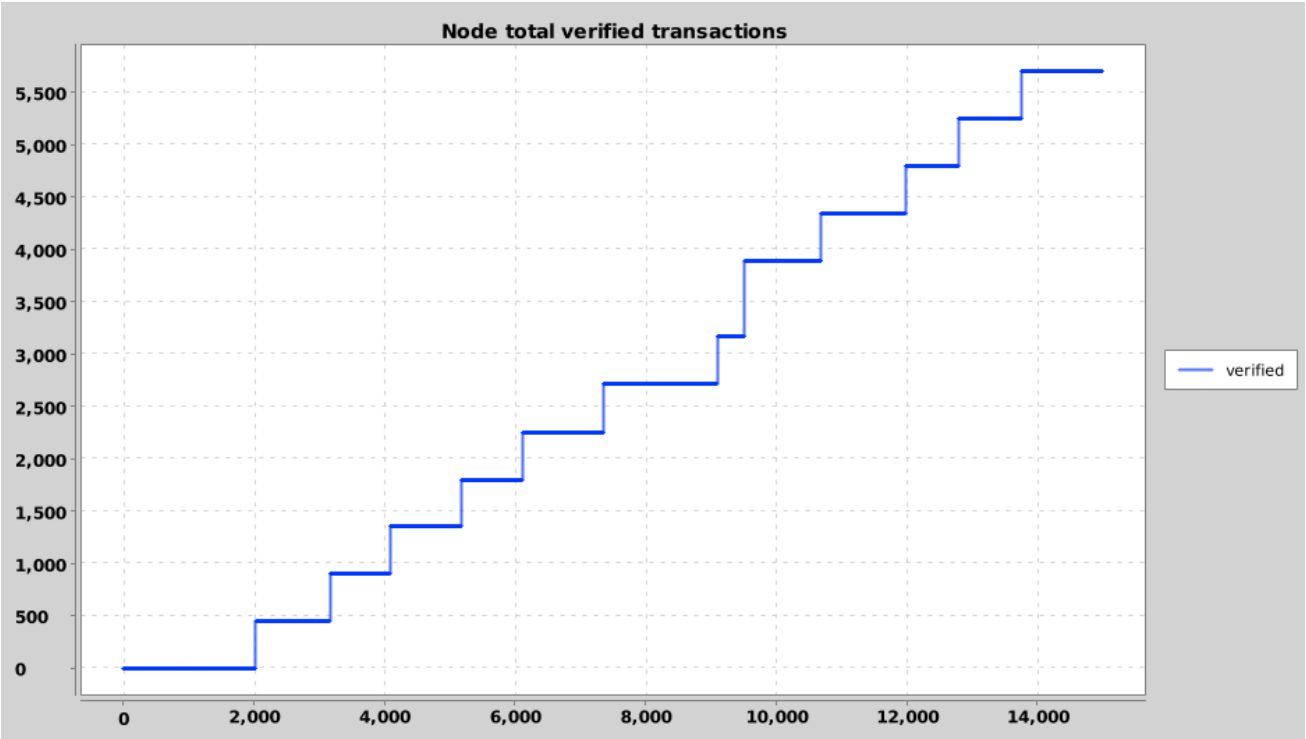

Figure C.18: Scenario 3, JSON

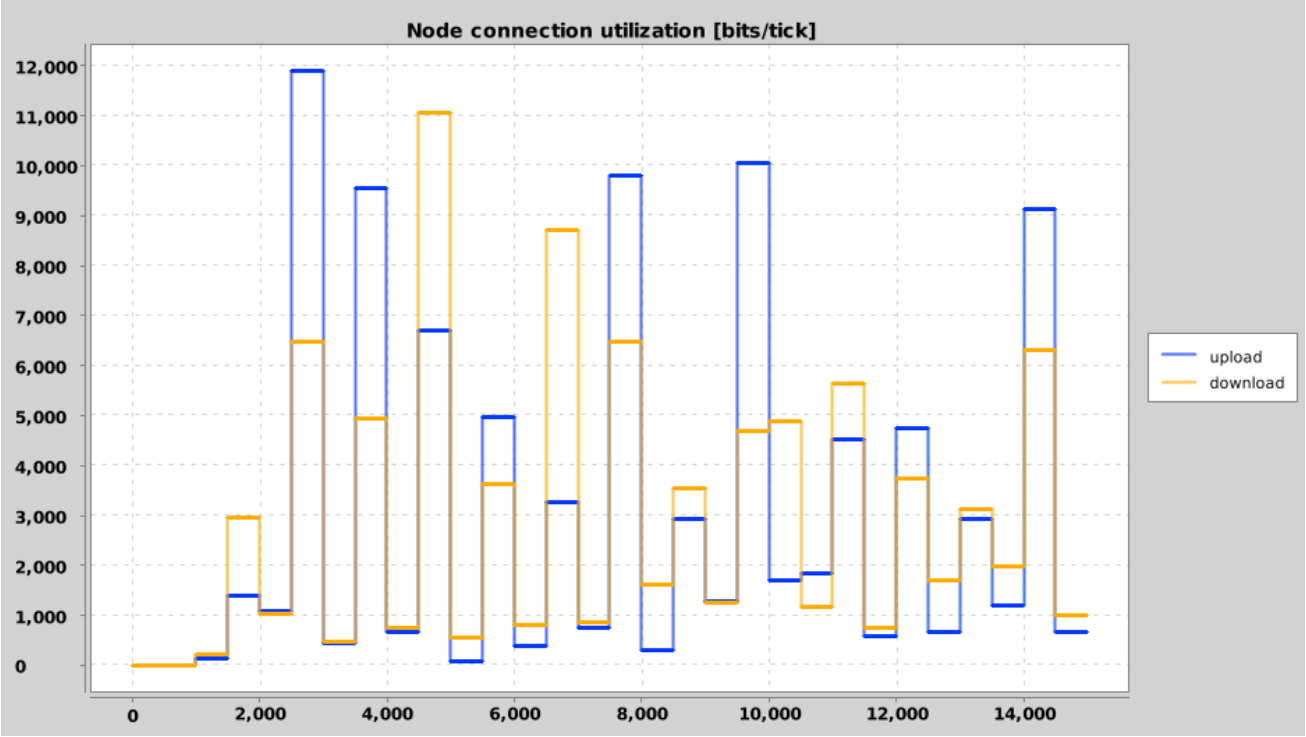

Figure C.19: Scenario 3, JSON

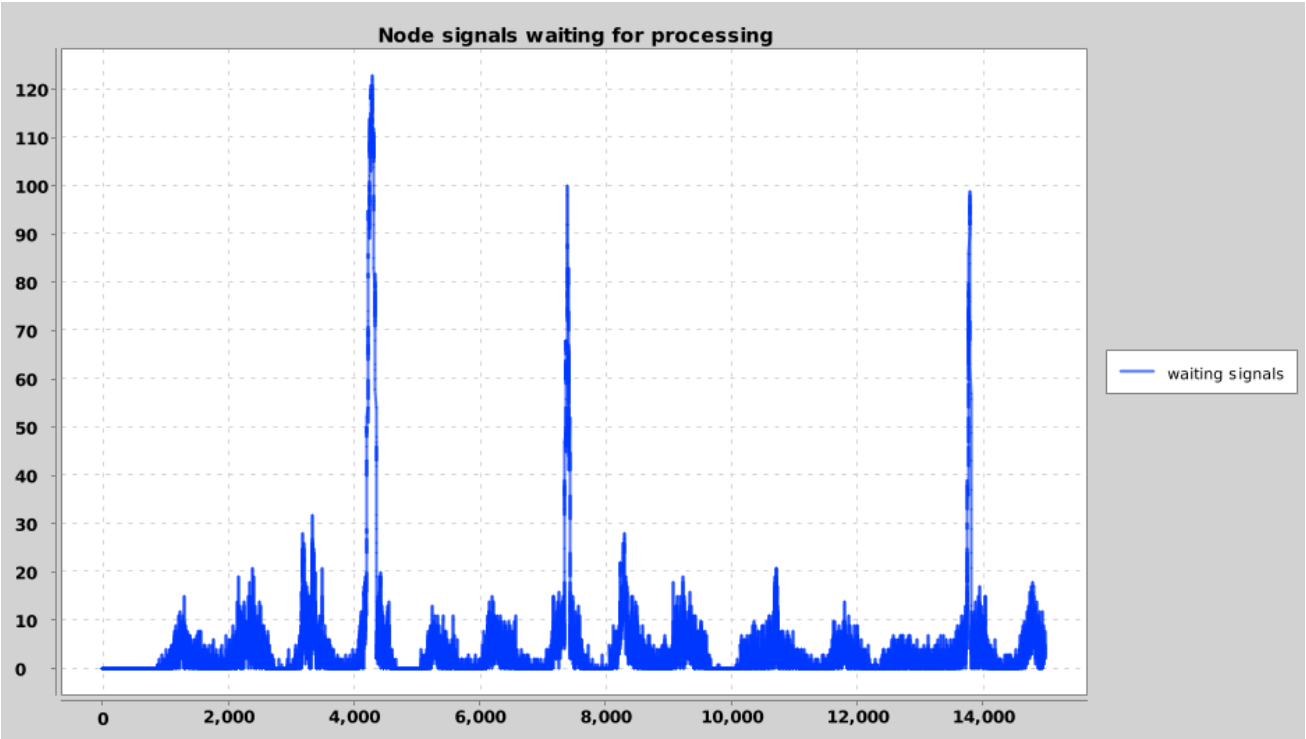

Figure C.20: Scenario 3, JSON

Figure C.21: Scenario 3, XML

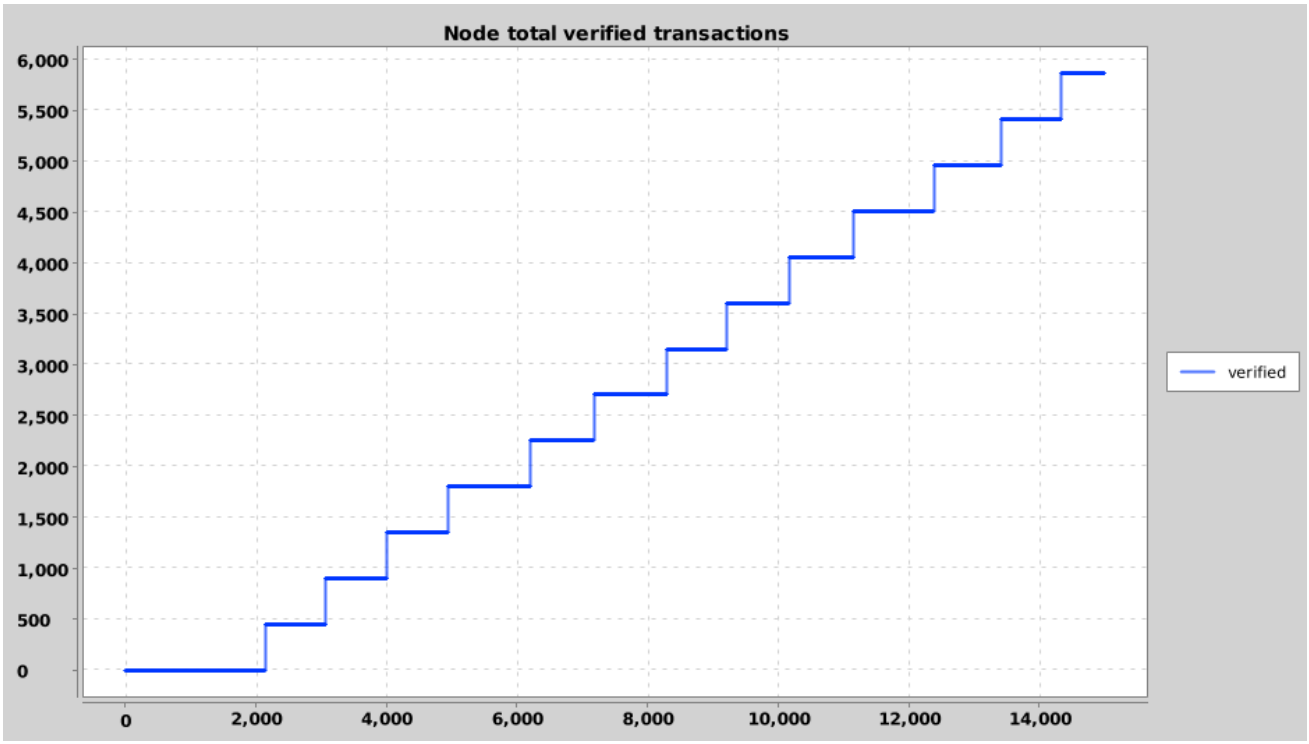

Figure C.22: Scenario 3, XML

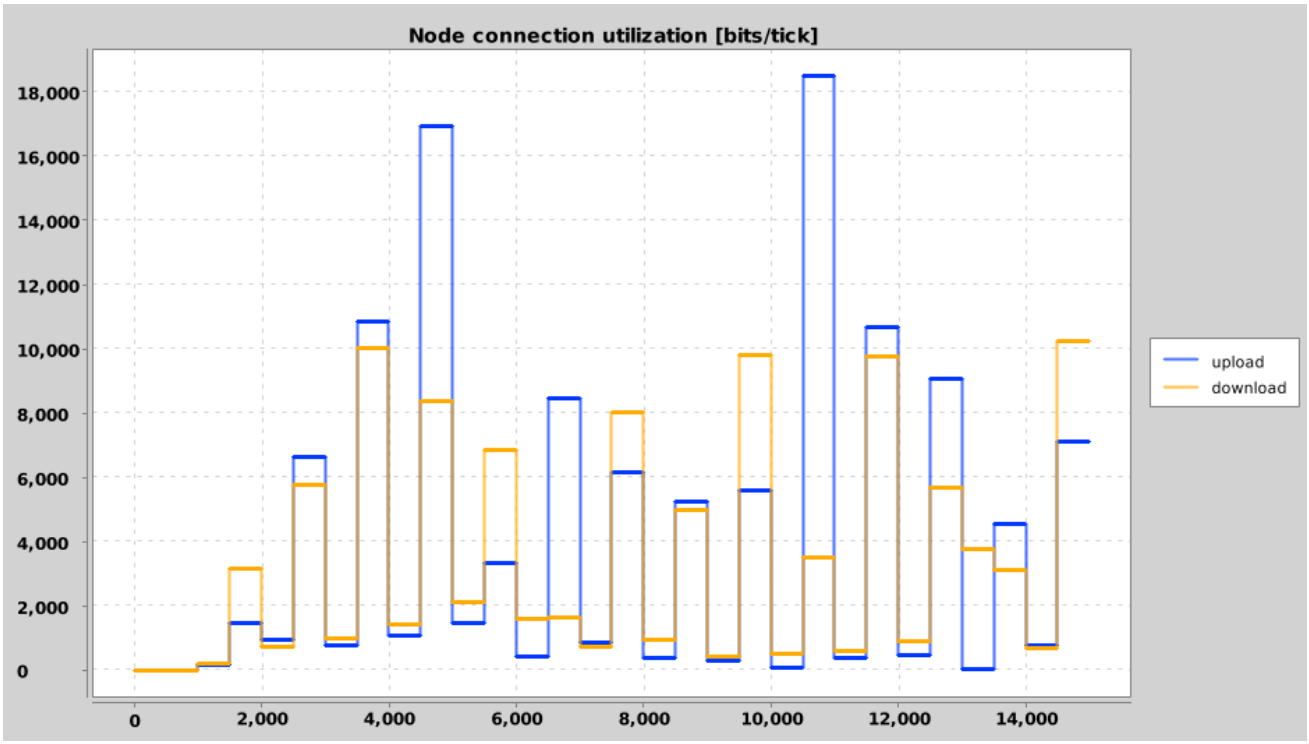

Figure C.23: Scenario 3, XML

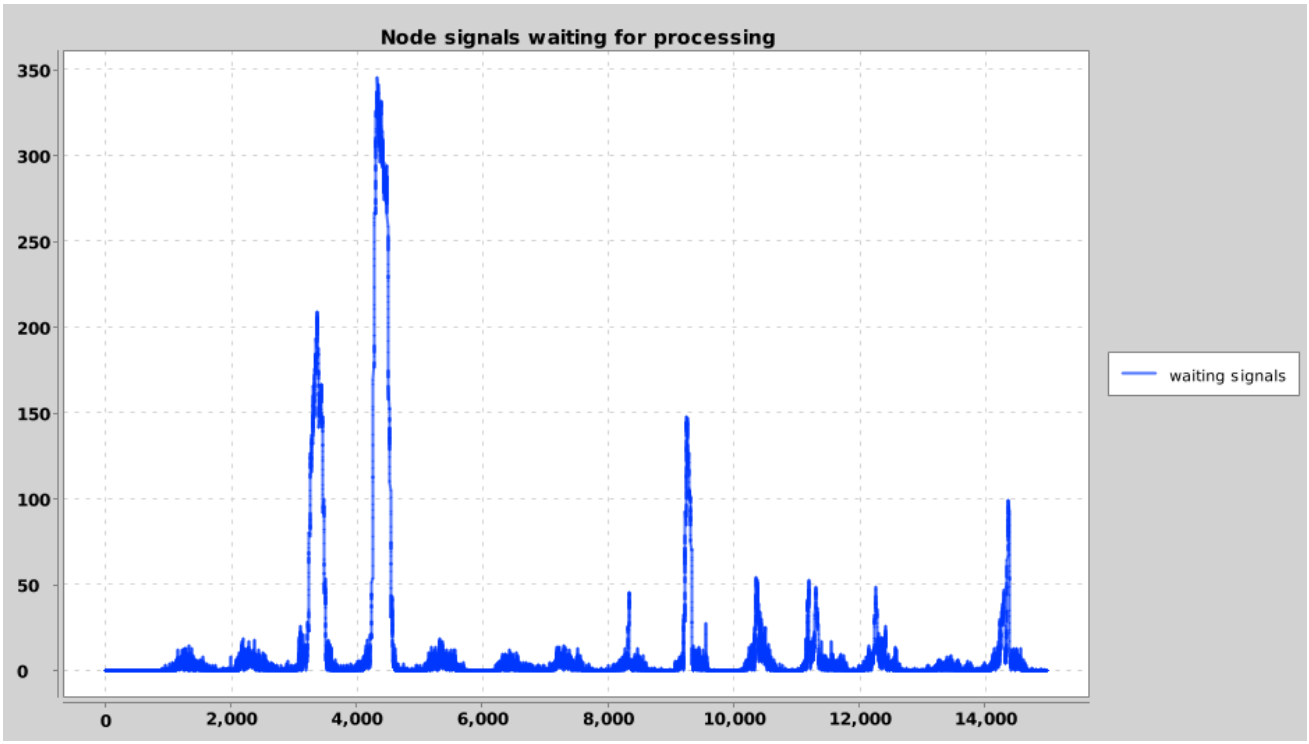

Figure C.24: Scenario 3, XML

```
	"end": 10000,
	"speed": 1,
	"many_nodes":
	[
		{
			"id_prefix": "publish",
			"class": "SimulatedSalesTransactionNode",
			"permission": 1,
			"count": 300,
			"speedUpload": 10000,
			"speedDownload": 10000,
			"transactionsPerSecond": 450,
			"style": "json"
		},
		{
			"id_prefix": "write",
			"class": "SimulatedBlockWriteNode",
			"permission": 2,
			"count": 3,
			"speedUpload": 10000,
			"speedDownload": 10000,
			"minTransactions": 450
		}
	],
	"graphs": {
		"node": ["publish_0", "transaction_pool", "
		   ↪ transactions_verified", "connection_speeds", "
		   ↪ waiting_signals"]
	}
}
```

C.4.2 Graphs

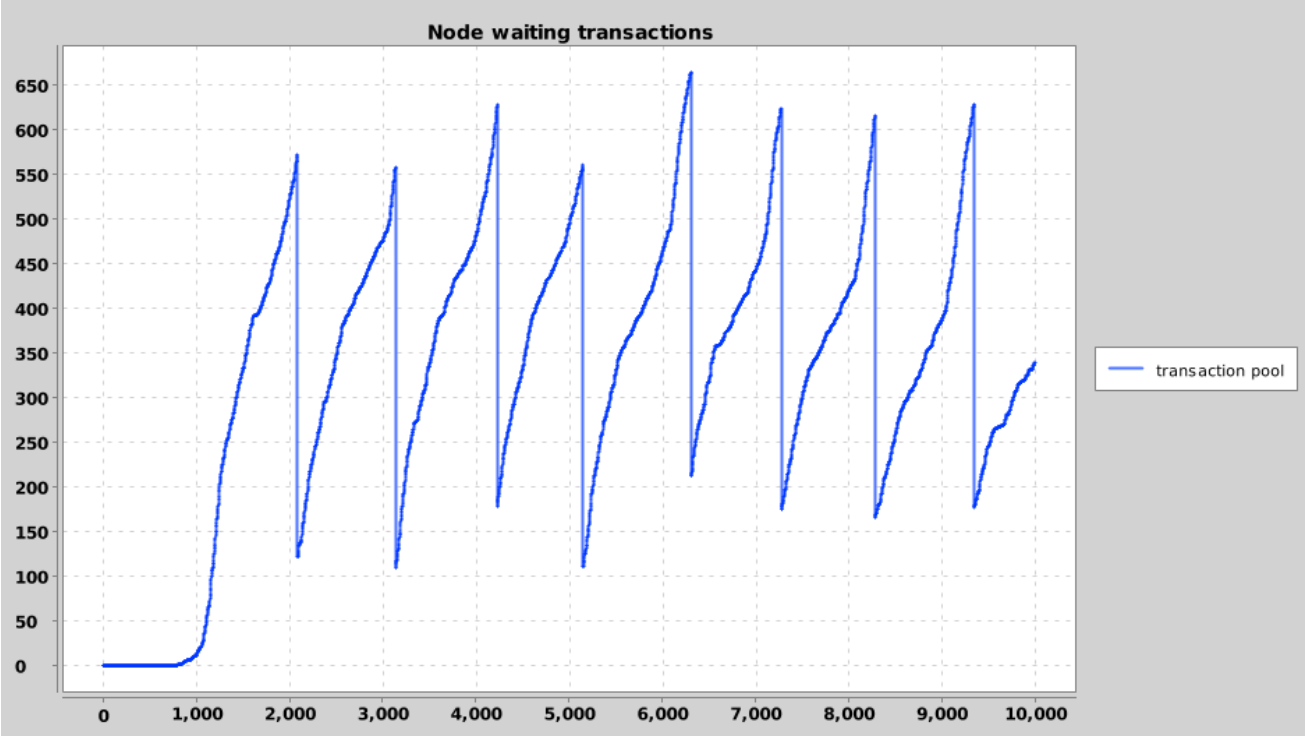

Figure C.25: Scenario 4, JSON

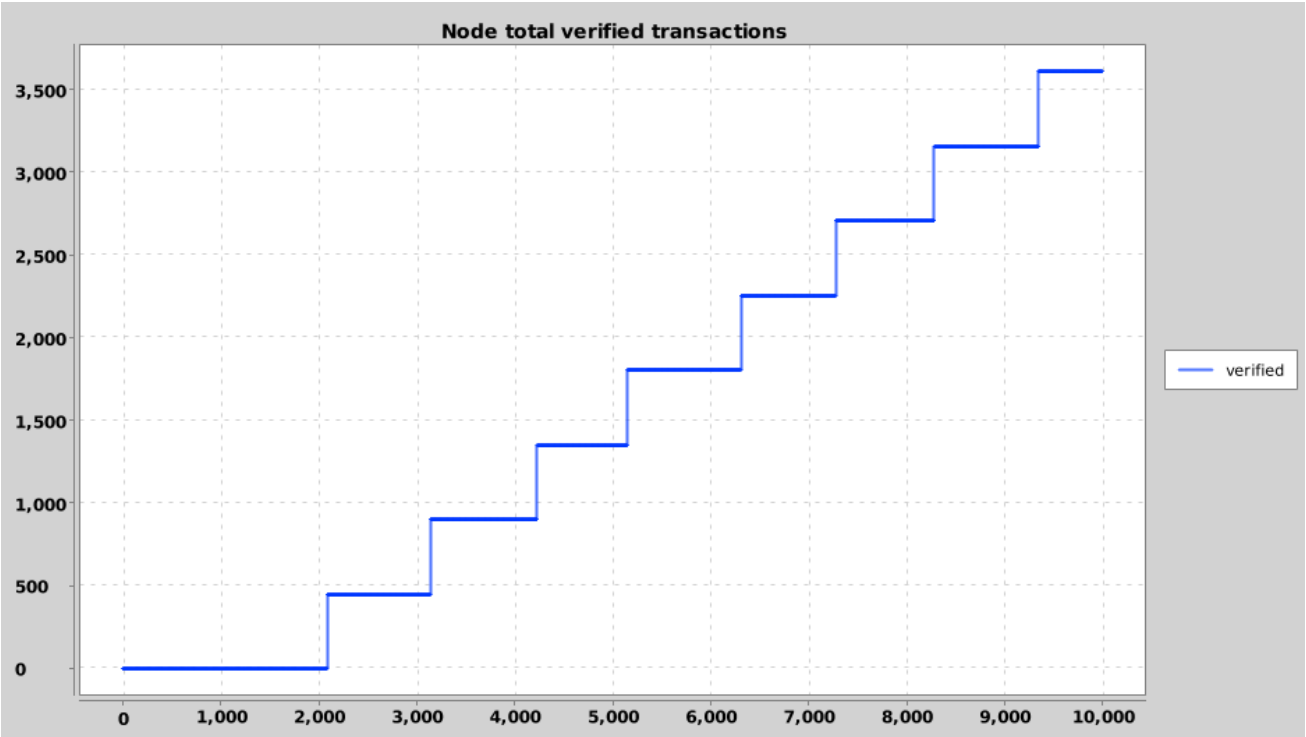

Figure C.26: Scenario 4, JSON

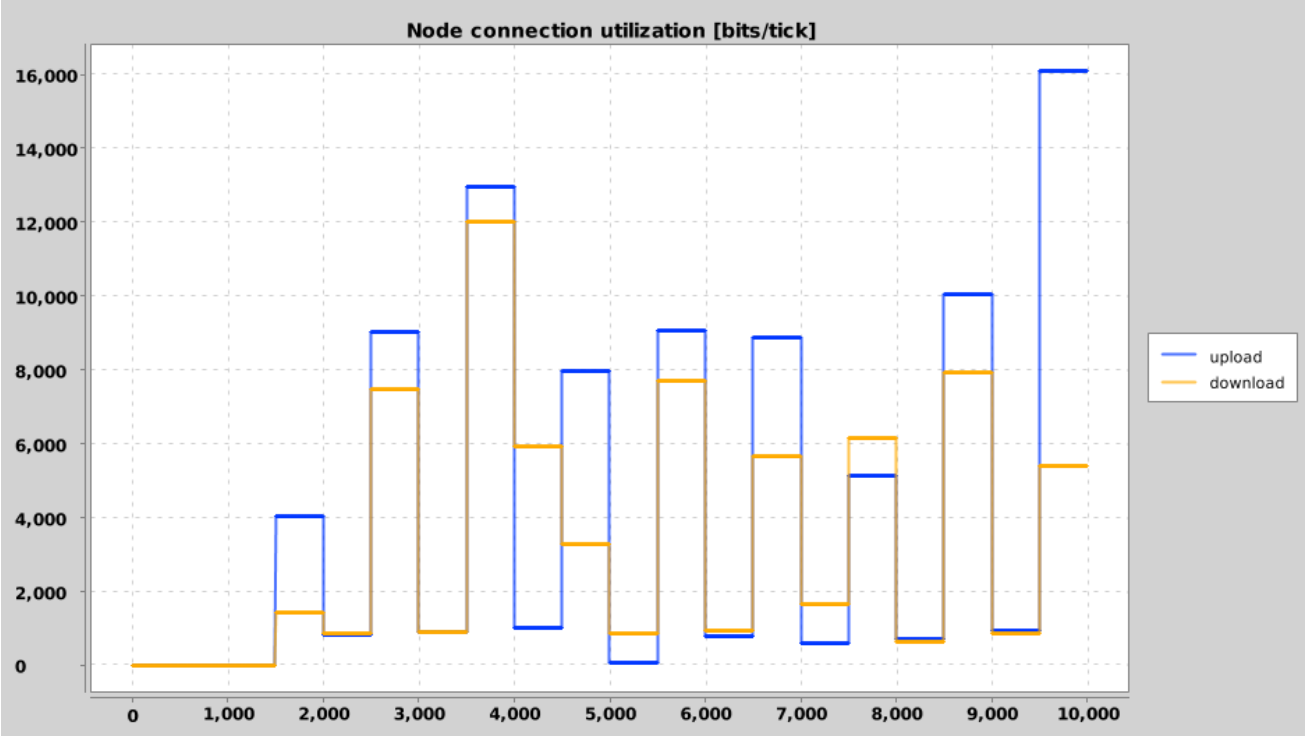

Figure C.27: Scenario 4, JSON

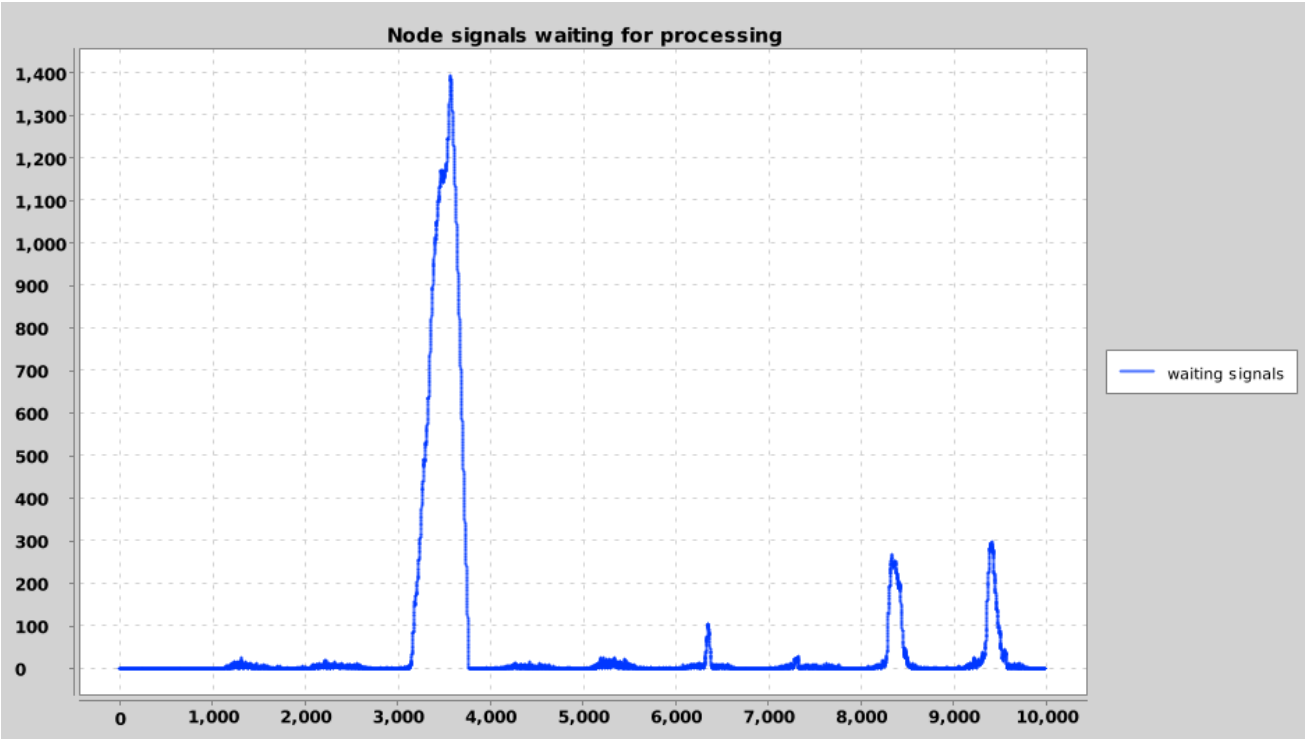

Figure C.28: Scenario 4, JSON

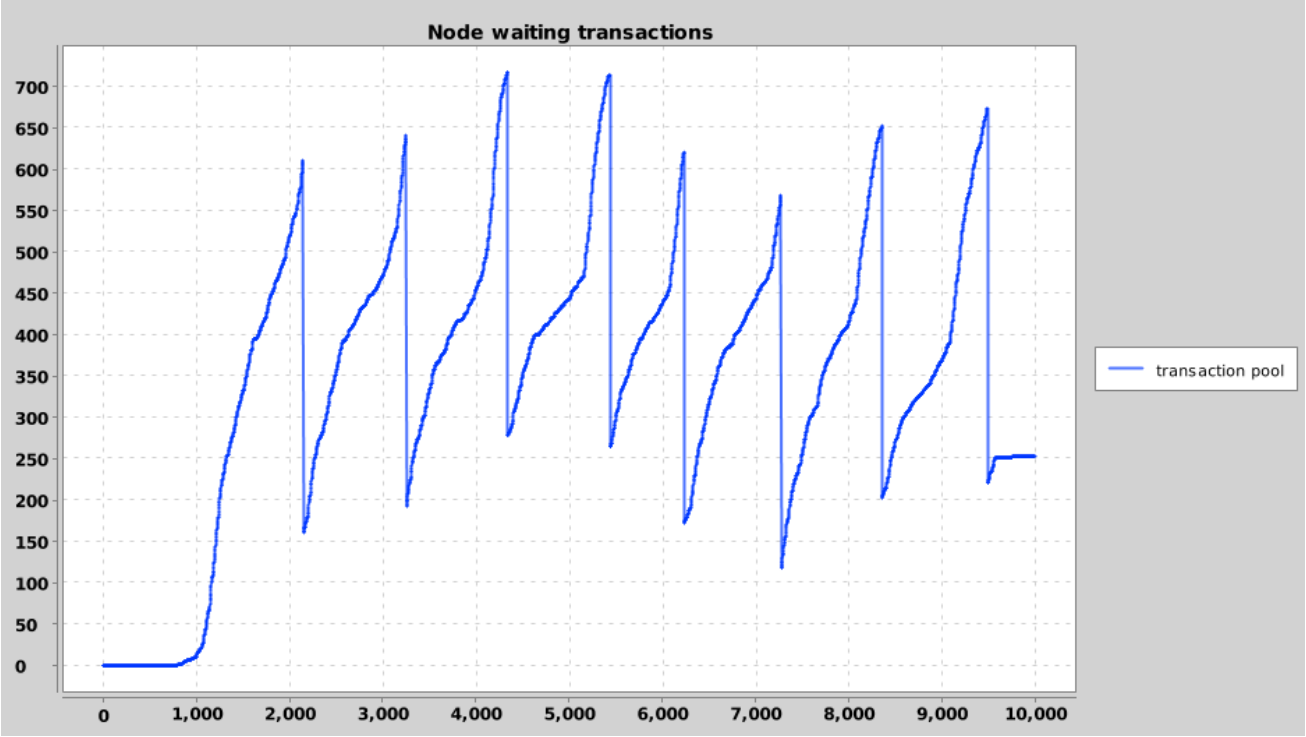

Figure C.29: Scenario 4, XML

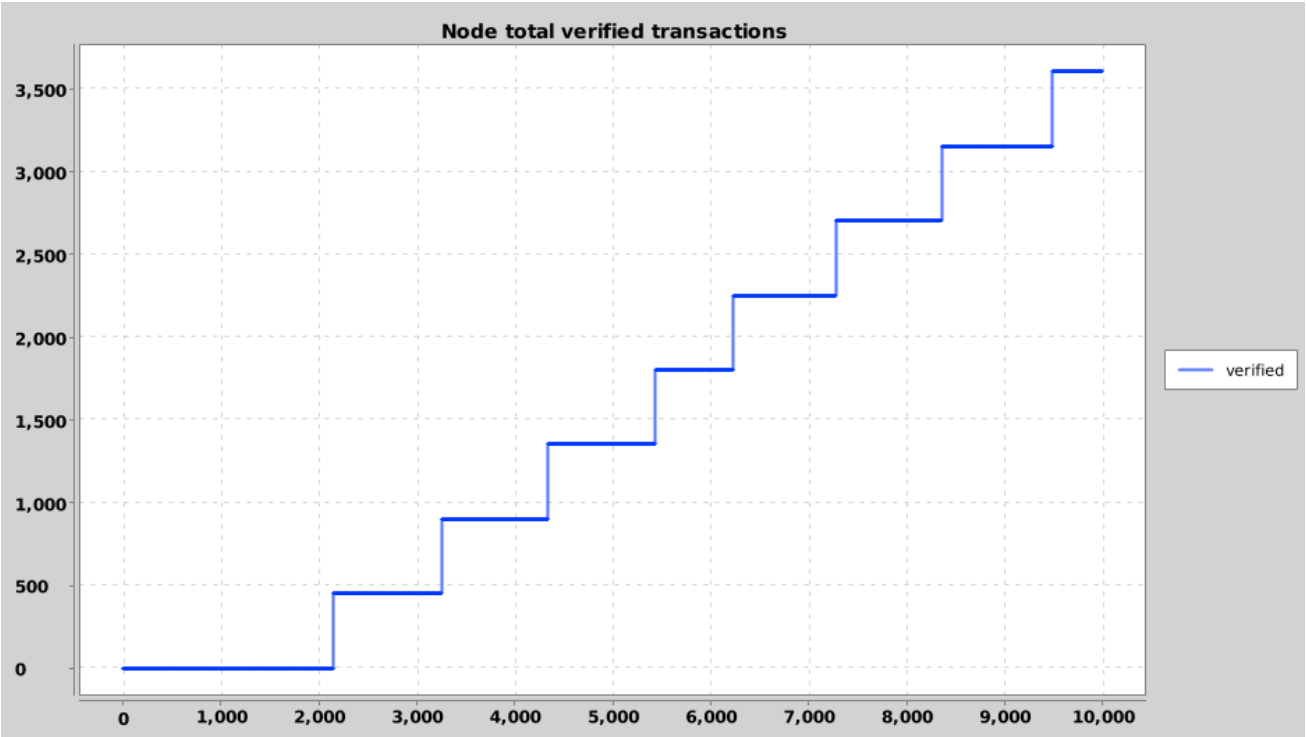

Figure C.30: Scenario 4, XML

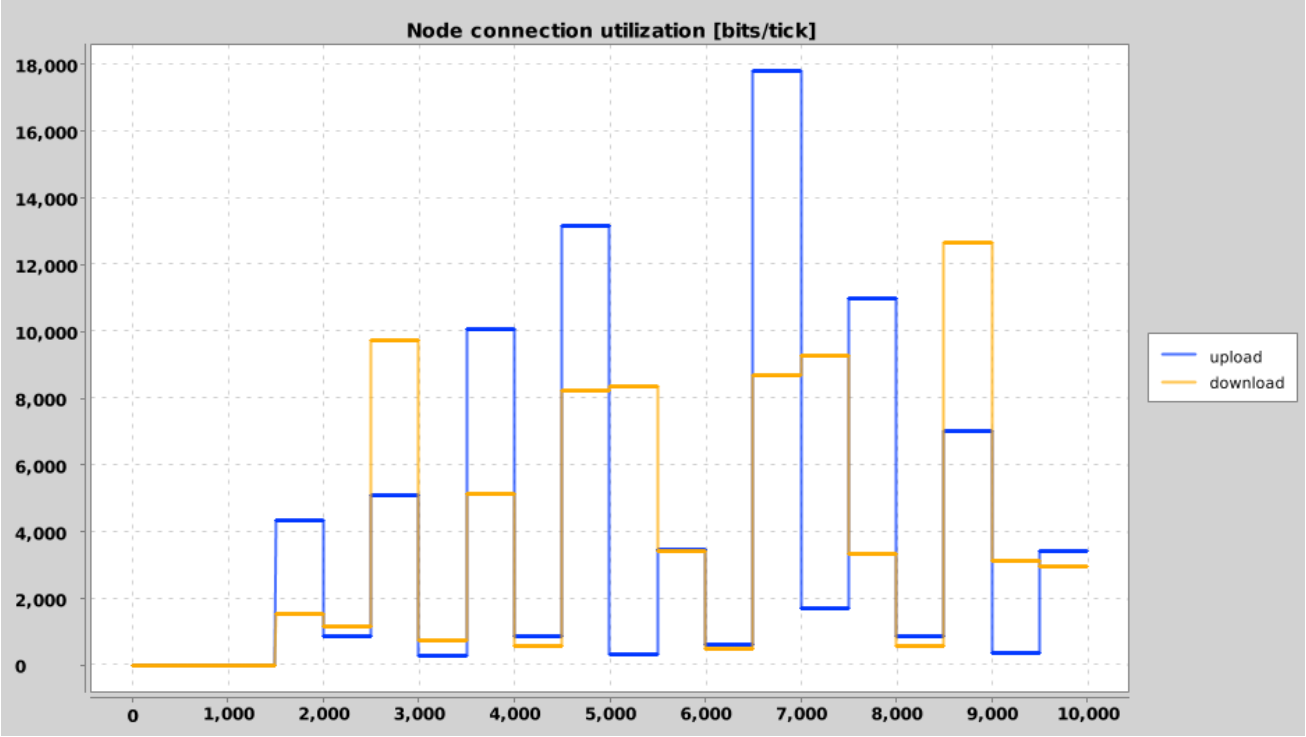

Figure C.31: Scenario 4, XML

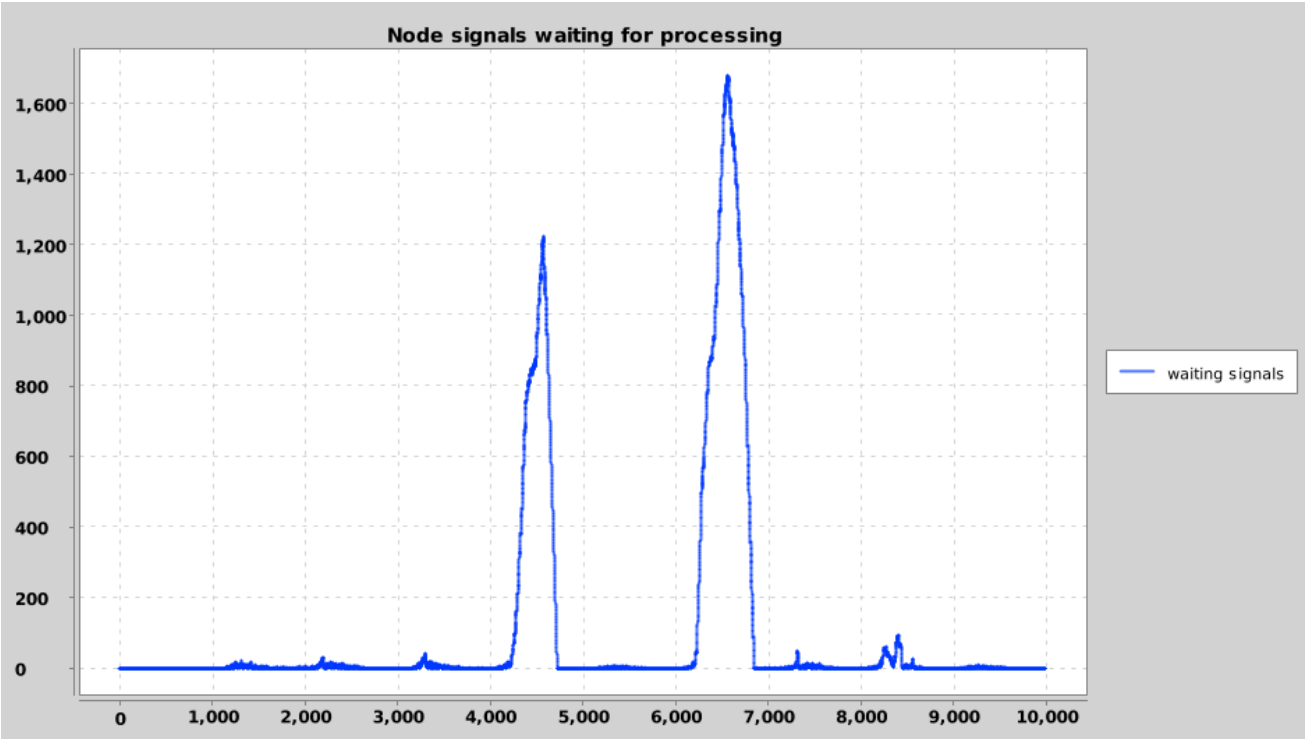

Figure C.32: Scenario 4, XML

Bibliography

[1] Moshe Babaioff, Shagar Dobzinsky, Sigal Oren, and Aviv Zohar. On bitcoin and red baloons. *EC '12 Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 56 – 73, Jun 2012.

[2] Adam Back. Hashcash - A Denial of Service Counter-Measure, Aug 2002.

[3] Paul Baran. Introduction to Distributed Communications Networks, Aug 1964. Memorandum RM-3420-PR.

[4] Blockchain Size — Bitcoin.com charts. `https://charts.bitcoin.com/chart/blockchain-size`. Accessed: 21 Mar 2018.

[5] Blockchain core source code. `https://sourceforge.net/p/bitcoin/code/133/tree/trunk/main.cpp#l1630`. Accessed: 28 Mar 2018.

[6] Vitalik Buterin. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform, 2013.

[7] T. Cegrell. A routing procedure for the tidas message-switching network. *IEEE Transactions on Communications*, 23(6):575–585, Jun 1975.

[8] Fabien Coelho. An (Almost) Constant-Effort Solution-Verification Proof-of-Work Protocol Based on Merkle Trees. *Lecture Notes in Computer Science*, 5023:80 – 93, 2008.

[9] Digiconomist: Bitcoin Energy Consumption Index. `https://digiconomist.net/bitcoin-energy-consumption`. Accessed: 28 Mar 2018.

[10] Electronic Registration of Sales - Description of the data interface for receipt of registered sale data messages (EN version), Oct 2016. Released 10th Oct 2016. Version 3.1.1.

[11] Registration Sales Act, 112/2016 Coll. (EN version). `http://www.etrzby.cz/assets/cs/prilohy/Act-on-Registration-of-Sales-No-112-2016-Coll.pdf`, Mar 2016.

[12] Zákon o evidenci tržeb, 112/2016 Sb. *Sbírka zákonů ČR*, 43:1978 – 1986, Mar 2016.

[13] Nález Ústavního soudu, Pl. ÚS 26/16, Dec 2017. Received by Court on 1st Jun 2016. Actionable from 12th Dec 2017.

[14] Informace o provozu - Statistická data o evidovaných tržbách. `http://www.etrzby.cz/cs/Informace-o-provozu`, 2018. Latest known data from 21 Mar 2018. Accessed 28 Mar 2018.

[15] Ethereum / Ether (ETH) statistics - Price, Blocks Count, Difficulty, Hashrate, Value. `https://bitinfocharts.com/ethereum/`. Accessed: 21 Mar 2018.

[16] Ethereum ChainData Size (Geth w/Fast Sync). `https://etherscan.io/chart2/chaindatasizefast`. Accessed: 21 Mar 2018.

[17] Ittay Eyal and Emin Gun Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. *Financial Cryptography and Data Security. FC 2014. Lecture Notes in Computer Science*, 8437:436 – 454, 2014.

[18] F2Pool Controls Over Half of the Litecoin Network's Hashrate Right Now. https://www.newsbtc.com/2017/06/08/f2pool-controls-half-litecoin-networks-hashrate-right-now/, Jun 2017.

[19] Generální finanční ředitelství. Evidence tržeb - Metodický pokyn k aplikaci zákona o evidenci tržeb, Aug 2018. Released 1st Mar 2018. Version 4. Č.j.:14332/18/7100-10114-205170.

[20] Milton Friedman. The Island of Stone Money. *Working Papers in Economics*, E-91-3, February 1991.

[21] Sean Gauld, Franz von Ancoina, and Robert Stadler. The Burst Dymaxion: An Arbitrary Scalable, Energy Efficient and Anonymout Transaction Network Based on Colored Tangles. https://www.burst-coin.org/wp-content/uploads/2017/07/The-Burst-Dymaxion-1.00.pdf, Dec 2017.

[22] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1 – 88, Apr 2016.

[23] Dan Goodin. Active attack on Tor network tried to decloak users for five months. https://arstechnica.com/information-technology/2014/07/active-attack-on-tor-network-tried-to-decloak-users-for-five-months/, July 2014. Accessed: 28 Mar 2018.

[24] Dan Goodin. Tor developers vow to fix bug that can uncloak users. https://arstechnica.com/information-technology/2014/07/tor-developers-vow-to-fix-bug-that-can-uncloak-users/, July 2014. Accessed: 28 Mar 2018.

[25] Ian Grigg. EOS - An Introduction, 2017.

[26] Alex Hern. Bitcoin's energy usage is huge – we can't afford to ignore it. https://www.theguardian.com/technology/2018/jan/17/bitcoin-electricity-usage-huge-climate-cryptocurrency, Jan 2018. Accessed: 28 Mar 2018.

[27] A. Hernando, D. Villuendas, C. Vesperinas, M. Abad, and A. Plastino. Unravelling the size distribution of social groups with information theory in complex networks. *The European Physical Journal B*, 76:89 – 97, July 2010.

[28] The Linux Foundation - Hyperledger. https://www.hyperledger.org. Accessed: 28 Mar 2018.

[29] M. Jakobsson and A. Juels. Proofs of Work and Bread Pudding Protocols (Extended Abstract). *IFIP - The International Federation for Information Processing*, 23:258 – 272, 1999.

[30] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. https://peercoin.net/assets/paper/peercoin-paper.pdf, Aug 2012.

[31] Roman Matzutt, Jens Hiller, Martin Henze, Jan Henrik Ziegeldorf, Dirk Mullman, Oliver Hihlfeld, and Klaus Wehrle. A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin. *To appear in Proc. 22nd International Conference on Financial Cryptography and Data Security 2018*, Feb 2018.

[32] Glossary:Statistical classification of economic activities in the European Community (NACE), Feb 2016. Eurostat regional yearbook. ISSN 2443-8219.

[33] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Aug 2008.

[34] Christopher Natoli and Vincent Gramoli. The Balance Attack Against Proof-Of-Work Blockchains: The R3 Testbed as an Example, Dec 2016.

[35] Nxt Whitepaper. https://nxtplatform.org/get-started/developers/, July 2014. Revision 4.

[36] K.J. O'Dwyer and D. Malone. Bitcoin Mining and its Energy Footprint. *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pages 280 – 285, 2014.

[37] Czech Statistical Office. Selected indicators of hotels and restaurants, 2015. Latest available data. OBU04 / 16.

[38] Czech Statistical Office. Selected indicators of trade, 2015. Latest available data. OBU03 / 15.

[39] Johan Pouwelse, Pawel Garbacki, Dick Epema, and Henk Sips. The Bittorrent P2P File-Sharing System: Measurements and Analysis. *Peer-to-Peer Systems IV. IPTPS 2005. Lecture Notes in Computer Science*, 3640:205 – 216, 2005.

[40] Proof of Stake versus Proof of Work. http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf, Sep 2015.

[41] Larry Ren. Proof of Stake Velocity: Building the Social Currency of the Digital Age. https://www.reddcoin.com/papers/PoSV.pdf, April 2014.

[42] Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-burn. https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf, May 2014.

[43] Tether: Fiat currencies on the Bitcoin blokchain. https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf, Jun 2016.

[44] Liang Wang and Jussi Kangasharju. Real-world sybil attacks in BitTorrent mainline DHT. *IEEE Global Communications Conference (GLOBECOM)*, Dec 2012.

[45] Liang Wang and Jussi Kangasharju. Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT. *13-th IEEE International Conference on Peer-to-Peer Computing*, Sep 2013.

[46] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, James Prestwich, Gordon Hall, Patrick Gerbes, Phillip Hutchins, and Chris Pollard. Storj: A Peer-to-Peer Cloud Storage Network. https://storj.io/storj.pdf, 2016.

[47] Mining Pools Comparison: ZEC. https://poolstats.org/ZEC. Accessed: 21 Mar 2018.