A Modest Comparison Of Blockchain Consensus Algorithms

Ilja Rõžakov
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
i.rozakov@student.utwente.nl

ABSTRACT

The market for blockchain technologies, systems to record transactions in a distributed, verifiable and permanent way, has grown significantly over the past years and still continues to introduce new blockchain-based products into our daily lives. Due to the nature of these systems, it is quite difficult to change the core software that these products are based upon, which in turn causes a strong need for tools and frameworks to quickly analyze and verify such systems, before it becomes nearly impossible to change them on the fly. This paper describes a structured way to model a probabilistic part, the consensus algorithm within a blockchain-backed system, using the Modest Toolset, then proposes a method to use this model for performance measurements of this algorithm, and finally provides a proof of concept by applying this method to the BFT consensus algorithm of the Exonum blockchain framework by Bitfury. The results are remarkable: Even though it is possible to use the constructed model to analyze network behaviour, not every blockchain network can be compared to each other in a trivial way. Instead there are different categories of blockchain networks among which comparisons can be made.

1. INTRODUCTION

When Nakamoto introduced the concept of peer-to-peer electronic cash systems 11 years ago [18], it did not only lead to the creation of the Bitcoin digital currency, but it also introduced the concept of a blockchain. A blockchain is essentially a distributed ledger that can efficiently record transactions between parties in a permanent and verifiable way. In order to make these guarantees, different algorithms are used together in order to record data as mathematical structures (blocks) that are recursively dependent on their historical predecessors (parent blocks).

This overall structure makes it impossible to alter data without having to rewrite all history after the original entry that was altered [18]. It is however of vital importance that the algorithms used to properly assert the permanent and verifiable nature of the entire blockchain actually do guarantee the required mathematical properties, even though some of these algorithms involve probability and chance.

The Bitcoin project itself has been subject to a variety of research projects studying its security and safety [11, 4, 3]. However, with the introduction of many new projects that provide a so called ‘private blockchain’ or ‘managed ledger’ [28], a need arises for new and efficient ways to model the implementations of the core algorithms of blockchain projects in order to be able to verify safety and compare performance of these systems.

To that end, a collaboration was sought with the company Bitfury and one of their projects, Exonum, was used for analysis. Exonum is a framework for creating custom blockchain applications for private use [10]. This research therefore focusses specifically on the BFT consensus algorithm implemented in Exonum.

The main goal of this research is to obtain a quick and structured way of modelling blockchain network consensus algorithms, and provide a proof of concept by applying this structured way to the Exonum blockchain software framework by Bitfury [10].

The main research question to answer is therefore: **“Does the Exonum BFT consensus algorithm provide the same level of performance as the Bitcoin consensus algorithm?”** In order to obtain the answer to this question, the following sub-questions have been identified:

- RQ 1: In what ways does a consensus algorithm relate to blockchain performance?
- RQ 2: Which metrics are needed to measure consensus algorithm performance in a decentralized blockchain based system?
- RQ 3: Which tools and models are needed to describe consensus algorithm performance?
- RQ 4: Is it possible to compare blockchain performance measurements?

This paper is organized as follows: Section 2 discusses the literature research findings, Section 3 discusses the modeling preparations, Section 4 discusses the constructed model, Section 6 discusses the results and findings of this research, and Section 7 concludes the research.

2. BACKGROUND

This section describes existing research, related work and theory upon which the rest of the research is built. The following subsections each introduce existing research in a specific field, itś relevance, and provide a quick summary of the theory. Finally this section concludes with a summary of silimar related work to this research.

2.1 Bitcoin

As mentioned in Section 1, Bitcoin uses a blockchain to provide a decentralized digital currency. In this decentralized network, there is no notion of trust or any assumptions made based upon trust. Instead, the network assumes that the information agreed upon by the majority of the computing power used for the network will be trustworthy. In order to achieve this, Bitcoin uses a system called "Proof-of-Work".

Bitcoin transactions are combined into "blocks", which reference the hash of the previously last known block, and a proof the block creator had access to a certain amount of computing power. This is done by searching for partial hash collisions with the hash of the previous block and embedding the match, resulting in a proof the block creator actually spent time and resources on this specific block. The longest chain of blocks will be considered the proper blockchain, and in case a transaction ended up on a shorter branch of blocks, it will simply be considered as illegitimate. Unfortunately, in this assumption also rests a significant source of possible vulnerabilities [6]. A transaction can only be considered legitimate by an observer when this observer is sure the transaction ended up in the longest chain [5]. However, due to the decentralized nature of the blockchain, it is impossible to guarantee it actually did end up in this chain, for there may be some (yet) unknown longer chain of blocks which is, either artificially or by accident, unknown to the observer [24]. Therefore, it is only possible to denote the chance of a transaction being in the largest chain of blocks, by taking a look at the amount of succeeding blocks that occurred after the block containing the transaction. As the amount of succeeding blocks increases, the chances of the transaction being illegitimate decreases. There is however no amount of blocks for which the chance becomes completely nonexistent.

The amount of blocks after which a transaction is deemed "most likely legitimate" is determined by the receiver and can vary on a case to case basis. If the transaction amount was low, for instance the purchase of a pizza, a relatively low amount like one or two blocks suffices to mitigate the risks [20]. On the other hand, if the transaction involved a high value, for instance a mortgage, a relatively high number of blocks will be needed such as ten in order to mitigate the risks. Most public systems deem a transaction valid when the number of succeeding blocks reaches six [21]. Bitcoins are also prone to different kind of attacks, such as private key theft [25] and timejacking[23].

2.2 Byzantine Fault Tolerance

The Byzantine Fault Tolerance concept originates in the Byzantine Generals Problem[16]. Although several variations of this problem exist referring to this made-up historical event, the gist of the story is always the following. The Byzantine army has surrounded an enemy city. The army is led by a number of Byzantine generals and each of the generals leads their own division. However, there are traitors among the generals who are interested in the Byzantine army failing.

In order for the battle to reach a successful or a neutral outcome, all the loyal generals must agree on the same plan of actions. The problem formulation allows for two possible actions on their behalf: attack or retreat. In case only a part of the divisions led by loyal generals attacks, and the other loyal generals retreat, the entire Byzantine army will be defeated by the enemy city forces. This is why it is essential for the loyal Byzantine generals to reach a consensus regarding their battle plan.

Lamport has shown [16] an algorithm exists for loyal generals to always reach a consensus if the number of the loyal generals is strictly larger than $\frac{2}{3}$ of all generals. This holds for a system in which messages between the generals may be lost.

In general, a distributed system is deemed Byzantine fault tolerant if it is able to properly function in the presence of arbitrarily behaving or malicious participants[9].

2.3 Exonum by Bitfury

Bitfury is a blockchain technology company that offers both software and hardware products. Exonum is one of such software products. It is a software framework for creating private blockchain-based projects for custom use cases, mainly targeted at companies and governments. Example use cases provided by Bitfury include E-voting, P2P lending, E-auction and a government registry[10].

Exonum provides the permissioned kind of blockchain [7]. This means that blockchain participants can be identified, moreover different blockchain participants can be assigned different set of permissions. In practice this means that certain nodes of a blockchain have the right to participate in decision-making (consensus), while other nodes are simply observers. The details depend on the specific implementation of the framework.

Unlike in the Bitcoin network, where no assumptions about trust between anonymous participants are made, the Exonum framework assumes an environment of semi-trust. This is due to the application specifics of the Exonum framework. Since the framework finds its main use cases in the corporate/government environment where the blockchain network nodes are pre-approved and can be identified, some level of trust can be assumed. However, the Exonum framework does takes into account the fact that a node might malfunction or might be compromised and thus acts in a malicious way. According to its specifications, Exonum-based systems can withstand up to strictly fewer than $\frac{1}{3}$ improperly behaving nodes out of the total number of nodes in a network, and still reach a correct consensus under such conditions[27]. This makes the Exonum consensus algorithm Byzantine Fault Tolerant.

2.4 Stochastic Modelling

Modelling a consensus algorithm requires, due to the distributed nature of blockchain networks, a modelling toolset which is capable of simulating parallel processes, and is also able to deal with stochastic modelling – i.e., modelling under probabilistic uncertainty. For instance, this applies to packet loss, or a malicious attack occurring, since such events are subject to probability and chance.

2.5 Modest & The Modest Toolset

The Modest modelling language and Toolset [8, 14, 15] is an example of a toolset that fulfills the above requirements. Modest is a high-level compositional modelling language. The Modest Toolset supports the modelling of real-time and stochastic systems, among other features[13]. Being able to model check a real-time stochastic system is relevant to measuring blockchain performance, since both time and probability-based events are key factors in the functioning of a blockchain.

The Modest Toolset includes modelling support for a variety of model formalisms such as Markov automata, stochastic timed automata, Markov decision processes and labelled transition systems. The above formalisms can be seen as special cases of SHA – stochastic hybrid automata, the modelling of which is supported by Modest[13].

The Modest Toolset includes tools to perform two types of model verification: exhaustive model checking and statistical model checking. The tool **mcsta** is an exhaustive model checker that performs the full state space exploration of a model. The tool **modes** is a statistical model checker that bases its results on the statistical measurements obtained after a specified number of model runs.

2.6 Related Work

Significant research into BFT consensus performance already exist. In this research [22] a Stochastic-Reward-Network [17] is used to perform performance analysis of the BFT consensus algorithm of the Hyperledger Fabric [2]. The current research will perform a different approach to this, as a Modest model can be used to perform different types of analysis, among which the possibility exists to use SRNs [12] to compare Exonum performance to Hyperledger performance by comparing the results of these two researches. Research into the differences between PoW consensus and BFT consensus also already exists. Abraham [1] provided a comparison that introduces analogies and connections between PoW-based consensus protocols, and BFT-based consensus algorithms, making it possible to partially bridge between these two paradigms. Finally, Nguyen [19] provided a survey of different types of consensus algorithms, providing a taxonomy which can be used to segment performance comparisons.

3. TOWARDS A MODEST MODEL

3.1 Consensus in Exonum

Operating in a semi-trust environment with pre-approved blockchain participants, the Exonum framework utilizes a BFT consensus algorithm as the means to reach agreement between blockchain nodes regarding every block that is proposed by a blockchain participant node. Depending on the outcome of a multiple-round vote, the proposed block is either added to the blockchain or discarded. When a block is committed to the blockchain as a result of consensus, all of the transactions contained in that block are deemed valid and will never be rescinded.

More specifically, the consensus algorithm in Exonum is implemented as follows[27]. The Exonum network consists of two types of participant nodes with different pre-approved permission sets. **Auditor** nodes have read-only rights and do not participate in consensus reaching. **Validator** nodes, on the other hand, are the ones that participate in the consensus algorithm. In this paper, validator nodes are referred to as simply "nodes", since only validator nodes are a part of the consensus algorithm.

The process of reaching block consensus consists of multiple rounds, which might run in parallel. Within each of the rounds, a new block is considered for addition to the blockchain. Each round has a leader node which is elected for one round only using a separate algorithm. The leader sends a *proposal* (a block) consisting of transactions that are needed to be added to the blockchain. The validator nodes check if said transactions are valid and have not been committed yet. If certain transactions from the *proposal* do not make sense to the nodes, they ignore the *proposal.* In case all transactions in the *proposal* are valid, the *full proposal* stage is reached, and the nodes start a *prevote* on the *proposal.* A supermajority, defined as strictly more than $\frac{2}{3}$ *prevotes*, is needed to reach the stage called *availability of* $\frac{2}{3}+$ *prevotes.* When that stage is reached, a *lock* occurs. A lock means that nodes who have the information about *availability of* $\frac{2}{3}+$ *prevotes* regarding a *proposal*, lock on that *proposal* and do not cast any other *prevotes* in favor of *proposals* from other ongoing rounds. Finally, the node sends a *precommit* message. Similarly to *prevotes*, there must be strictly more than $\frac{2}{3}$ *precommits.* When that happens, then nodes reach the *commit* stage, which means that the block (*proposal*) has been successfully committed to the blockchain.

It is important to note that all messages are simply broadcast across the network, meaning that they can arrive out of order or be lost due to network errors. In case a certain message does not make sense to a node, that node is able to request missing information from the peer nodes.

3.2 Exonum vs Bitcoin Consensus

In the Bitcoin network, participating nodes are not pre-approved [18]. Anyone is able to join the Bitcoin blockchain, moreover, all the participants are equal in terms of their permissions. Since all the participant nodes are anonymous, a zero-level of trust is assumed here. To solve the problem of no trust, a different (compared to Exonum) approach to consensus is utilized in the Bitcoin blockchain. Unlike in Exonum, no explicit voting takes place, instead, Bitcoin relies on the "Proof-of-Work" type of consensus. Unlike in Exonum, blocks with false transactions can in principle be added to the Bitcoin blockchain. However, since every Bitcoin block references the hash of the previous block, it is in practice impossible for one entity to keep indefinitely finding new blocks and maintaining the fraudulent chain. Compared to Exonum, the Bitcoin network requires significantly more computational power from the participants in order to maintain integrity.

3.3 Metrics

In order to perform analysis and comparisons of the consensus algorithms, Metrics must be identified that influence the overall network performance, together with factors which may influence these metrics. These factors should be easy to modify in the model, in order to make it possible to quickly execute new measurements with variations in these factors.

Exonum BFT performance can be measured in amount of actions per node before a block becomes committed. This could, for instance, be influenced by network packet loss, percentage of badly behaving nodes, network size, and by timeout duration. Bitcoin however, does not have a clear moment at which a block officially becomes committed, due to the fact that it requires multiple new block commits to know if a committed block was used and referenced in a newer block and thus accepted [21].

Therefore, it is not possible to measure Bitcoin performance based on time to commit a block. Instead, for Bitcoin performance modeling, it is needed to define a minimal amount of blocks to be committed after a block commit. Unfortunately, this produces an initially unforeseen side-effect; a single block commit is independent of other nodes, and therefore always achieves better performance compared to Exonum's BFT consensus algorithm on large scale networks, but adding the requirement of multiple succeeding blocks, adds a higher-order complexity to the analysis, which therefore influences any performance comparison to such an extend, that the outcome will depend entirely on the number of required additional blocks.

Because of this, it is important to make proper comparisons between consensus algorithms while keeping the type of consensus algorithm in mind. Proof-of-Work consensus, should not be compared with BFT consensus, for the required assumptions will entirely determine the outcome of the comparison.

4. A MODEST MODEL

This section presents the Modest model of the Exonum BFT consensus algorithm. The subsections describe the parts of this model including *actions*, *processes* and *parallelization* instances that are used to represent different elements of the consensus algorithm. Furthermore, this section explains the assumptions and the design choices that were made when creating the model.

4.1 Exonum Round Process

The first thing that requires modelling is the actual internal workings of a validator node for a single round. Parallelization can then be used to extend the process into multiple rounds or multiple nodes. A node can be in several stages for a specific round:

- Full proposal
- Availability of $\frac{2}{3}+$ prevotes
- Lock
- Commit

Figure 1 shows a code excerpt in which the *process* Node represents the set of actions a properly functioning validator node triggers throughout the consensus reaching process, depending on the internal state and on the vote counters. The Node *process* has a parameter that indicates the id number of each node.

```
process Node(int id)
{

 int(0..3) state;
 int(0..number_of_nodes) prevotes;
 int(0..number_of_nodes) precommits;

 do {
  :: when(state == 0) rcv_proposal {= state++ =}
  :: when(state == 1) send_prevote {=  =}
  :: when(state == 1) rcv_prevote {= prevotes++ =}
  :: when(state == 1 && prevotes >= supermajority)
     tau {= state++ =}
  :: when(state == 2) send_precommit {= =}
  :: when(state == 2) rcv_precommit {= precommits++ =}
  :: when(state == 2) rcv_prevote {= prevotes++ =}
  :: when(state == 2 && precommits >= supermajority)
     tau {= state++, block_commited[id]=true =}
 }
}
```

Figure 1. Internal workings of a node

State 0 represents the starting stage before receiving a *proposal*. State 1 represents the *Full proposal* stage. In that state, a node is able to send and receive *prevotes*. When a *prevote* is received, the node updates the counter. When the number of *prevotes* reaches supermajority, the node goes into the next state.

State 2 corresponds to both *Availability of* $\frac{2}{3}+$ *prevotes* stage and the *Lock* stage. In that state, the node is able to send and receive *precommits*. It is also able to receive *prevotes*. Although this leads to no progress regarding reaching consensus, it represents the ability of nodes to receive messages out of order. When the recorded *precommits* number reaches supermajority, the state is incremented once again, and the block is deemed commited for this particular node. State 3 corresponds to the *Commit* stage.

It is important to note that in the do-block of the Node *process* a non-deterministic choice occurs between all the listed *actions*. However, the state-related constraints limit the non-determinism to the stage-level.

4.2 Exonum Message Validation

The second thing to model, is the validity of network messages. By modelling the message validation separately, it becomes easy to model bad behaviour. The inner workings of a node can be changed into anything, but the parallelization of the *processes* makes sure only “legal” messages are received and processed.

Figure 2 depicts a code excerpt in which the NetworkNode *process* represents the public side of a node. This *process* safeguards the network behaviour: only the messages that were sent can be received.

```
process NetworkNode(int PER)
{
 int(0..2) sent_pv_state;
 int(0..2) sent_pc_state;

 do {
  :: when(sent_pv_state == 0) send_prevote palt {
     :100-PER: {= sent_pv_state++ =}
     :PER:     {=  =}}
  :: when(sent_pv_state == 1) rcv_prevote {= sent_pv_state++ =}

  :: when(sent_pc_state == 0) send_precommit palt {
     :100-PER: {= sent_pc_state++ =}
     :PER:     {=  =}}
  :: when(sent_pc_state == 1) rcv_precommit {= sent_pc_state++ =}
  :: rcv_proposal
 }
}
```

Figure 2. Behavior of a node within the network

Moreover, the NetworkNode process simulates the network’s property of being able to lose sent messages. The extent of this property can be changed with the parameter “PER” – packet error rate. This parameter can be set to integer values from 0 to 100 representing the error rate from 0% up to 100%.

The *process* has two counters: one for *prevotes* and one for *precommits*. The palt keyword is used to indicate a probabilistic alternative: either a sent message will be successfully received by the nodes or it will be dropped with the probability specified as the parameter PER. Furthermore, the *process* allows for receiveing a *proposal* at any time.

4.3 Exonum Network

By taking multiple round processes, multiple message validation processes, and creating a parallelism of these, a network of nodes can be simulated.

Figure 3 shows code representing a Network *process* of four nodes. Unlike the Node and NetworkNode *processes*, Network has no states of its own. Instead, it connects network behaviours of the nodes by means of *relabeling* and *parallelization*. After all the generic *actions* in NetworkNode have been *relabeled* with Node-specific *action* names and the NetworkNode *processes* are *parallelized*, the receiving-related *actions* are *relabeled* back with generic names. This is due to the fact that it does not matter for a node from which specific node a vote is received to be counted.

4.4 Remaining Model Details

In addition to local *actions* contained inside the *processes* depicted in Figures 1 to 3, the model also includes actions declared globally due to them being utilized by multiple *processes*. These are depicted in Figure 4.

Finally, the final composition of the model processes is depicted in Figure 5. Here, a *process* for each of the participating nodes is started, with sending-related *actions relabeled* by node-specific names.

```
process Network()
{

 action rcv_prevote0, rcv_prevote1,
        rcv_prevote2, rcv_prevote3;
 action rcv_precommit0, rcv_precommit1,
        rcv_precommit2, rcv_precommit3;

 do {
  :: relabel
  {rcv_prevote0, rcv_prevote1,
   rcv_prevote2, rcv_prevote3,
   rcv_precommit0, rcv_precommit1,
   rcv_precommit2, rcv_precommit3}
  by
  {rcv_prevote, rcv_prevote,
   rcv_prevote, rcv_prevote,
   rcv_precommit, rcv_precommit,
   rcv_precommit, rcv_precommit}
  {
   par {
    :: relabel {send_prevote, send_precommit,
                rcv_prevote, rcv_precommit}
       by {send_prevote0, send_precommit0,
           rcv_prevote0, rcv_precommit0} NetworkNode(PER)

    :: relabel {send_prevote, send_precommit,
                rcv_prevote, rcv_precommit}
       by {send_prevote1, send_precommit1,
           rcv_prevote1, rcv_precommit1} NetworkNode(PER)

    :: relabel {send_prevote, send_precommit,
                rcv_prevote, rcv_precommit}
       by {send_prevote2, send_precommit2,
           rcv_prevote2, rcv_precommit2} NetworkNode(PER)

    :: relabel {send_prevote, send_precommit,
                rcv_prevote, rcv_precommit}
       by {send_prevote3, send_precommit3,
           rcv_prevote3, rcv_precommit3} NetworkNode(PER)
   }
  }
 }
}
```

Figure 3. The network parallelization

```
action rcv_proposal, rcv_prevote, rcv_precommit;
action send_prevote, send_precommit;

action send_prevote0, send_prevote1,
       send_prevote2, send_prevote3;
action send_precommit0, send_precommit1,
       send_precommit2, send_precommit3;
```

Figure 4. Global *actions*

As mentioned previously, this model includes exactly four participating nodes, for which a Network *process* is started. By using *parallelization*, the internal workings of a node round are synchronized with a network that safeguards allowed messages and is capable of losing messages with a specified probability.

4.5 Modelling the Misbehaving Node

Naturally, malicious and malfunctioning nodes are not an intended part of the Exonum consensus algorithm. However, according to the specifications, the Exonum consensus algorithms is able to withstand up to strictly fewer than $\frac{1}{3}$ nodes that function improperly, and still reach a correct decision regarding a block.

An attempt was made to model a misbehaving node that participates in the Exonum consensus algorithm. It is depicted in Figure 6. The behavior of the misbehaving node is chosen to be as generalized as possible, meaning that the misbehaving node selects *actions* at random. At all times, this node is able to send any kind of message or internally register a block as committed.

```
par {
 :: relabel {send_prevote, send_precommit} by
            {send_prevote0, send_precommit0} Node(0)
 :: relabel {send_prevote, send_precommit} by
            {send_prevote1, send_precommit1} Node(1)
 :: relabel {send_prevote, send_precommit} by
            {send_prevote2, send_precommit2} Node(2)
 :: relabel {send_prevote, send_precommit} by
            {send_prevote3, send_precommit3} Node(3)
 :: Network()
}
```

Figure 5. The final composition

```
process BadNode(int id) {
 do {
  :: rcv_proposal
  :: send_prevote
  :: rcv_prevote
  :: send_precommit
  :: rcv_precommit
  :: tau {= block_commited[id]=true =}
 }
}
```

Figure 6. The misbehaving node

4.6 Remarks and Areas for Improvement

In the current version of the Modest model, the number of nodes participating in the consensus algorithm is hard-coded, which requires manual addition or removal of lines of code in case the node number requires changing.

The concept of rounds of the consensus algorithm is modelled in a simplified way compared to the actual Exonum algorithm. The created model simulates a round during which a consensus must be reached, as opposed to multiple rounds with multiple *proposals* happening in parallel. It is also assumed that all of the transactions included in a *proposal* are valid. Since it is assumed that internal checking of transaction validity takes no time, the part was excluded.

The misbehaving node in the current version of the model is rather generic. It is possible to extend the node with different types of behavior, possibly making its actions more intelligent. Furthermore, a group of malicious nodes can be created that coordinates their actions

5. MEASUREMENTS

5.1 Networks of Legitimate Nodes

The Modest model of the Exonum consensus algorithm allows for simulating a lossy network; the probability of the network losing broadcast messages can be specified when running the model. Figures 8 and 9 show the influence of the message loss probability on the time to commit a block after reaching consensus. In these graphs, the probability of a message loss is referred to as "Packet Error Rate" (PER), and the time it takes for all the network nodes to reach consensus and successfully commit a block is referred to as "Time to Commit". Packet error rate is measured in percentages, and time to commit is measured in the number of messages exchanged between the nodes before successfully committing a block.

The tool from the Modest Toolset that was used to plot the graphs in Figures 8 and 9 is **mcsta**, the exhaustive model checker. After exhaustively exploring the state space, the sequences of non-deterministic choices that lead to the most and the least amount of time to commit a block are known. The two *properties* that were verified with **mcsta** are depicted in Figure 7.

```
property time_to_commit_max =
    Xmax(S, forall_committed(number_of_nodes));

property time_to_commit_min =
    Xmin(S, forall_committed(number_of_nodes));
```

Figure 7. The verified properties

More specifically, the *time_to_commit_max property* represents the maximum amount of time it can take for a block to be committed, according to all of the nodes in the network.

Similarly, the *time_to_commit_min property* retrieves the minimum amount of time the nodes can take to commit a block. These two *properties* are plotted correspondingly as "MAX" and "MIN" in Figures 8 and 9.

Figure 8 demonstrates the results for a network consisting of four well-functioning nodes. Figure 9 demonstrates the same for a network of seven well-functioning nodes. In both, the dependency between the specified packet error rates and the MAX/MIN time values is shown.

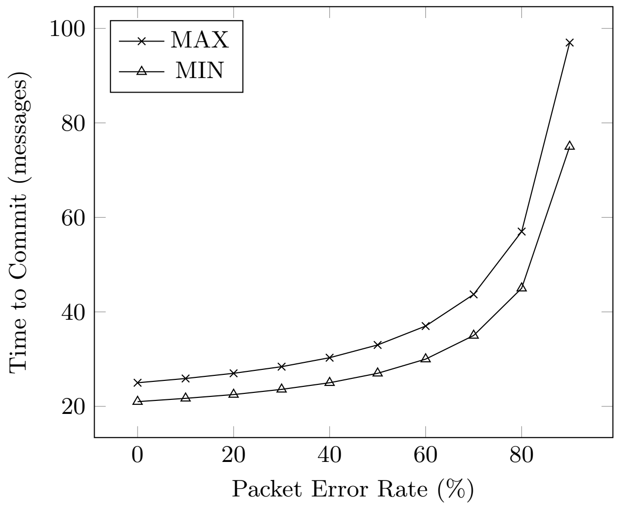

Figure 8. PER vs Time to Commit (4 nodes)

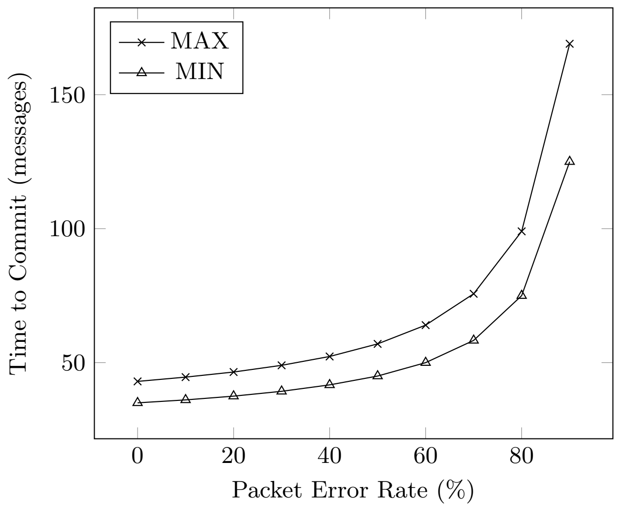

Figure 9. PER vs Time to Commit (7 nodes)

5.2 Networks with Misbehaving Nodes

In the following measurement results, a network of four nodes is used. Three of the nodes are legitimate fully-functioning nodes, and one is a misbehaving node, as described in Section 4.5.

The X and Y axes are identical to the one from Section 5.1, they correspondingly depict the packet error rate measured in percentages and the time to commit a block measured in the messages exchanged between nodes before a block is committed. However, by using the **mcsta** tool it was discovered that in the worst case the misbehaving node is able to flood the network with messages, so that reaching a consensus becomes impossible. This is a limitation of the model: in practice, a node sending a considerate amount of packets above a certain rate would be prevented by different means such as lower level network congestion control. However, such a model behavior makes the MAX time to commit a block go to infinity. Surprisingly, in the MIN case, the time is slightly decreased compared to the MIN case from Section 5.1. This can happen when the misbehaving node accidentally makes the consensus process faster due to the randomness of its actions.

Due to the specified behavior of the misbehaving node, the results obtained by **mcsta** are not particularly useful. Instead, in this section the **modes** tool is used. It is a *statistical* model checker that is used here to obtain the mean time it takes to commit a block. Using a statistical model checker on the existing model gives rise to non-determinism between existing actions. This problem is solved by assigning a uniform distribution to the non-deterministic actions. The graph depicting the results is in Figure 10.

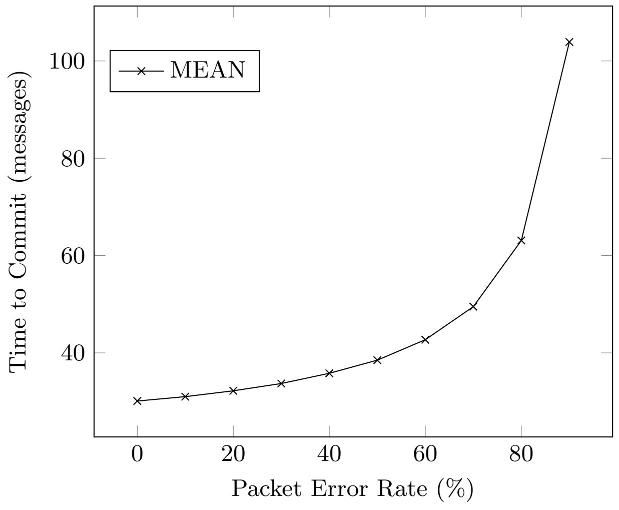

Figure 10. PER vs Time to Commit (4 nodes, network with one misbehaving node)

6. DISCUSSION

As seen in the results of Section 5, varying the packet loss for the network slows down the speed at which a new block gets committed. However, according to pre-existing Bitcoin models [5], this is not the case for Bitcoin. Varying the packet error rate for Bitcoin does however increase the chances of a Bitcoin split occurring, where two branches of the blockchain exist at the same time. As soon as these branches get in touch with the same network member, the longest of these branches wins and the other transactions are rolled back.

Similarly, in the case of a misbehaving node, the influence on the network is different compared to misbehaving nodes in the Bitcoin Blockchain. In a Bitcoin blockchain, a misbehaving node would either need access to a considerable amount of computing power, or work together with other nodes to actually influence the network at all [6].

7. CONCLUSIONS

The constructed model can be used to analyze the BFT Consensus algorithm, and determine the performance impact of variations in factors such as packet error rate and number of misbehaving nodes. The constructed model can also be further extended to account for more edge-cases and reduce the number of assumptions and simplifications. It is however, not possible to answer the initial research question "Does the Exonum BFT consensus algorithm provide the same level of performance as the Bitcoin consensus algorithm?". This is due to the fact that the definition of measurable network performance is considerably different for both projects, causing the assumptions to counter these differences to have more impact on the outcome of the comparison than the actual analysis would have, thus making it possible to end up with any outcome one may want to achieve. While proper comparison between different consensus algorithm types is impossible, it still might be possible to compare equal types of consensus algorithms. This, however, requires further research.

8. FURTHER RESEARCH

Using the presented models and results, it is now possible to conduct further research. For instance, the following questions could be further explored and answered:

- Is it possible to compare Exonum BFT consensus with other BFT consensus algorithms such as the Hyperleger Fabric consensus [2]?
- Is it possible to automate (partial) verification of blockchain software using the results of this research?
- In which ways can probabilistic models be used to increase the performance of blockchain based networks?
- Do custom blockchain frameworks actually provide added value (either through performance, security, or in another way) compared to other blockchain networks, such as Ethereum [26]?

Acknowledgements

I would like to thank my supervisors, Dr. A. Hartmanns and Dr. A. Fehnker of the Formals Methods and Tools Group at the University of Twente, for supporting me while writing this paper and for always being available to answer my questions. In addition, I would like to thank Victor Ermolaev of the Bitfury Group for assisting me with all technical difficulties. Lastly, I would like to thank all of the students in the Software Technology and Formal Methods Track for providing me with regular feedback and insights into this research.

9. REFERENCES

[1] I. Abraham and D. Malkhi, "The Blockchain Consensus Layer and BFT," *Bulletin of EATCS*, vol. 3, no. 123, pp. 1–22, 2017.

[2] E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, A. Barger, S. W. Cocco, J. Yellick, V. Bortnikov, C. Cachin, K.-n. Christidis, A. De Caro, D. Enyeart, C. Ferris, and G.-n. Laventman, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *EuroSys*, vol. 18, 2018, pp. 1–15. DOI: `10.1145/3190508.3190538`.

[3] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in *Proceedings - IEEE Symposium on Security and Privacy*, May 2017, pp. 375–392. DOI: `10.1109/SP.2017.29`.

[4] S. Bag, S. Ruj, and K. Sakurai, "Bitcoin Block Withholding Attack: Analysis and Mitigation," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1967–1978, Aug. 2017. DOI: `10.1109/TIFS.2016.2623588`.

[5] M. Bastiaan, "Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin," *Proceedings of the 22nd Twente Student Conference on IT*, 2015.

[6] J. Becker, D. Breuker, T. Heide, J. Holler, H. P. Rauer, and R. Böhme, "Can we afford integrity by proof-of-work? scenarios inspired by the bitcoin currency," in *The Economics of Information Security and Privacy*, Feb. 2013, pp. 135–156. DOI: `10.1007/978-3-642-39498-0_7`.

[7] BitFury Group and J. Garzik, "Public versus Private Blockchains. Part 1: Permissioned Blockchains," Tech. Rep., 2015, pp. 1–23.

[8] H. Bohnenkamp, P. R. D'Argenio, H. Hermanns, and J. P. Katoen, "MODEST: A compositional modeling formalism for hard and softly timed systems," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 812–829, Oct. 2006. DOI: `10.1109/TSE.2006.104`.

[9] D. Dolev, "The Byzantine generals strike again," *Journal of Algorithms*, vol. 3, no. 1, pp. 14–30, Mar. 1982. DOI: `10.1016/0196-6774(82)90004-9`.

[10] Exonum, *Exonum A framework for blockchain solutions*, 2018. [Online]. Available: `https://exonum.com/` (visited on 05/05/2019).

[11] I. Giechaskiel, C. Cremers, and K. B. Rasmussen, "When the crypto in cryptocurrencies breaks: Bitcoin security under broken primitives," *IEEE Security and Privacy*, vol. 16, no. 4, pp. 46–56, Jul. 2018. DOI: `10.1109/MSP.2018.3111253`.

[12] E. M. Hahn, A. Hartmanns, and H. Hermanns, "Reachability and Reward Checking for Stochastic Timed Automata," in *Electronic Communications of the EASST*, vol. 70, Nov. 2014. DOI: `10.14279/tuj.eceasst.70.968`.

[13] A. Hartmanns, "On the Analysis of Stochastic Timed Systems," PhD thesis, 2015. DOI: `10.22028/D291-26597`.

[14] A. Hartmanns and H. Hermanns, “A Modest approach to checking probabilistic timed automata,” in *QEST - 6th International Conference on the Quantitative Evaluation of Systems*, Sep. 2009, pp. 187–196. DOI: 10.1109/QEST.2009.41.

[15] A. Hartmanns and H. Hermanns, “The Modest Toolset: An integrated environment for quantitative modelling and verification,” *Lecture Notes in Computer Science*, vol. 8413 LNCS, pp. 593–598, 2014. DOI: 10.1007/978-3-642-54862-8_51.

[16] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 2002. DOI: 10.1145/357172.357176.

[17] J. K. Muppala, G. Ciardo, and K. S. Trivedi, “Stochastic Reward Nets for Reliability Prediction,” *Communications in Reliability, Maintainability and Serviceability*, pp. 9–20, 1994.

[18] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.,” *Journal for General Philosophy of Science*, no. 1, 2008. DOI: 10.1007/s10838-008-9062-0.

[19] G. T. Nguyen and K. Kim, “A survey about consensus algorithms used in Blockchain,” *Journal of Information Processing Systems*, vol. 14, no. 1, pp. 101–128, 2018. DOI: 10.3745/JIPS.01.0024.

[20] C. Pinzón and C. Rocha, “Double-spend Attack Models with Time Advantange for Bitcoin,” *Electronic Notes in Theoretical Computer Science*, vol. 329, pp. 79–103, Dec. 2016. DOI: 10.1016/j.entcs.2016.12.006.

[21] M. Rosenfeld, “Analysis of hashrate-based double-spending,” pp. 1–14, 2012.

[22] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, “Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric),” in *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, vol. 2017-Septe, 2017, pp. 253–255. DOI: 10.1109/SRDS.2017.36.

[23] P. Szalachowski, “(Short paper) towards more reliable bitcoin timestamps,” in *Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018*, Jun. 2018, pp. 101–104. DOI: 10.1109/CVCBT.2018.00018.

[24] M. Vasek, M. Thornton, and T. Moore, “Empirical analysis of denial-of-service attacks in the bitcoin ecosystem,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8438, pp. 57–71, 2014. DOI: 10.1007/978-3-662-44774-1_5.

[25] S. Verbücheln, “How Perfect Offline Wallets Can Still Leak Bitcoin Private Keys,” *MCIS 2015 Proceedings*, Jan. 2015.

[26] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” in *Ethereum Project Yellow Paper*, 2014, pp. 1–32. DOI: 10.1017/CBO9781107415324.004.

[27] Y. Yanovich, I. Ivashchenko, A. Ostrovsky, A. Shevchenko, and A. Sidorov, “Exonum: Byzantine fault tolerant protocol for blockchains,” Tech. Rep., 2018, pp. 1–36.

[28] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” in *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, Jun. 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85.