Formalising the Bitcoin protocol

Making it a bit better

W.J.B. Beukema
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
w.j.b.beukema@student.utwente.nl

ABSTRACT

Bitcoin is a new, popular currency which is based on mathematical and cryptographic principles. The system relies on a decentralized peer-to-peer network of participants in which a majority decides on the validity of transactions. In this paper, the communication protocol used by Bitcoin to communicate between participants is investigated. We formally describe the protocol by specifying it in mCRL2. Using scenario-based verification, we verify that the Bitcoin protocol satisfies a number of requirements under various scenarios.

Keywords

Bitcoin, Cryptocurrency, Formalisation, mCRL2, Scenario-based verification

1. INTRODUCTION

With over 19,000,000 transactions in 2013 and a still-rising daily average of transactions [6], the Bitcoin can be described as an upcoming currency. After 2008, the year in which the initial design paper was published by Nakamoto [14], the Bitcoin has rapidly gained popularity amongst people all over the world.

The design of the Bitcoin as a currency is fundamentally different from traditional currencies. Whereas traditional currencies depend on central authorities, the Bitcoin system relies on a decentralised network of volunteers in order to transfer money. As there is no single point of trust, a majority of the participants decide on the validity of a transaction. Because this is achieved by using mathematical models and cryptographic proof, the entire Bitcoin system is based on mathematical proof instead of trust [14].

As a result of its popularity, cyber criminals have also found their way to Bitcoin. Recently, Bitcoin has been in the news frequently because of failing Bitcoin exchanges, fraud, theft and unstable exchange rates as a result [2]. The question arises whether the design of Bitcoin can be blamed for this current situation.

In this work, it is analysed from a network perspective how the information is communicated through the Bitcoin network. A formal specification of the protocol has been created and verified using a set of requirements. Subsequently, it is analysed to what extent Bitcoin is a usable currency compared to traditional currencies.

1.1 Our contribution

We present an mCRL2 model of the Bitcoin protocol based on the current protocol description [4]. With this model of the real Bitcoin protocol, the Bitcoin network can be simulated and verified. The former can be useful for analysing certain network situations, such as double-spending attacks. The latter can be useful to determine the robustness of the protocol, i.e. if Bitcoin is able to detect and handles protocol deviations correctly.

Based on our findings, we also briefly discuss the Bitcoin protocol and the implications it has for Bitcoin as a currency. Using scenario-based verification, we were able to verify that the Bitcoin protocol satisfies our requirements under various circumstances.

1.2 Outline

Before discussing the model, the relevant aspects of the Bitcoin protocol and the general structure of Bitcoin will be explained in section 2.

Subsequently, the model of the Bitcoin protocol will be introduced in section 3. Then we will specify the requirents in section 4, followed by a verification of these requirements (under various conditions) in section 5.

Based on this, we will then discuss the design of Bitcoin in section 7, followed by a conclusion (section 8). An overview of related literature is discussed in section 6.

2. BITCOIN

We distinguish to different contexts in which the term 'Bitcoin' may be used:

- *Bitcoin* refers to the system as described by Nakamoto [14];
- *bitcoin(s)* or *BTC* refers to the currency unit (like euro and €, respectively);

In order to participate in the Bitcoin network, one needs a Bitcoin wallet and a Bitcoin account. A Bitcoin wallet is a software client that enables one to send and receive bitcoins, such as BitcoinQT [5]. A Bitcoin account is basically a public-private key pair, which as we will see is needed to sign the various messages. An account is identified with its public key. Bitcoin wallets can generate new accounts.

The Bitcoin system consists of several parts in which peer-to-peer communication is involved. Two core elements of the system, transactions and block generation, involve

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

21th Twente Student Conference on IT June 23th, 2014, Enschede, The Netherlands.

Copyright 2014, University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.

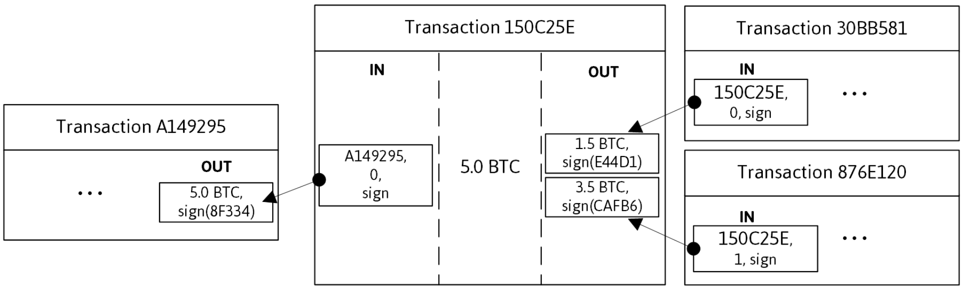

Figure 1. An example of a Bitcoin transaction. In transaction 150C25E, the input transaction is worth 5.0 BTC, and split into output transactions of 1.5 BTC and 3.5 BTC. In new transactions 30BB81 and 876E12, these transactions are refered to as input transactions.

communication of significant importance. The following subsections describe on communication level how these elements work and how they relate to each other.

2.1 Transactions

In order to transfer bitcoins, one has to create a message containing the transaction details. In essence, the message should contain three components: a source, a destination and the amount of bitcoins to transfer. The Bitcoin protocol does not include this information directly into the transmitted messages, but instead uses *inputs* and *outputs*, which we will introduce now.

First of all, the transaction message contains one or more *outputs*, a list that indicates what amounts should be transferred to what accounts. Thus, it is possible to address multiple receivers in only one transaction. An output is basically a tuple with a numeric value of bitcoins and a condition to claim the output.

Because it is hard for other nodes in the network to determine whether or not the sender has the amount of money it claims to have, the sender has to prove that it is in the possession of the bitcoins by referring to previous Bitcoin transactions that were made to him. In order to refer to another transaction, the hash of the textual representation of a transaction (called a *tx message*) is used. Together with the reference, a proof of ownership is included. The references and the proofs together are called the *inputs* of the newly created transaction.

In order to start the transaction, the sender should create the *tx message* and inform its connected peers about it. It does so by advertising the hash of the *tx message*; this message is called an *inv message*. The connected peers will check whether the advertised hash is new to them. If so, they will request the complete *tx message* by sending a *getdata message* containing the hash of the *tx message*. Subsequently, the original sender will send the *tx message*. To summarise, the elements of a *tx message* are the inputs (list of references to previous transactions and ownership proof) and the outputs (list of receivers and the amount of bitcoins they receive).

Once a peer receives a *tx message* from another peer, it will check its validity by analysing its content. For a transaction to be valid, at least all of these criteria must be fulfilled:

- All inputs should refer to valid transactions;
- None of the inputs may have been used in other transactions;
- The sum of the values of the inputs should not be smaller than the total amount of bitcoins in the outputs;

If a peer considers a received transaction to be valid, it will add the transaction to its *pool*. The peer will advertise all transactions in his pool to the connected peers. Hence, the transaction will eventually spread through the network.

Nevertheless, inconsistencies may occur while broadcasting this information through the network. For instance, a malicious node might try to spend the same coins more then once. It might submit two nearly the same transactions into the network: using the same (valid) inputs, but with different outputs. Since there is no guarantee that the transactions will arrive in the same order at every peer, there will be confusion in the network about what should be considered the ”real” transaction. This attack is called a *double-spending attack*. This problem is characteristic for distributed currencies as Bitcoin: in the case of a traditional currency, the central transaction authority would recognise such an attempt in an earlier stage and prevent it from happening.

2.2 Blocks

As mentioned before, synchronising the order in which transactions occur is a fundamental problem for decentralised transaction systems. Bitcoin's solution to create agreement on transaction order amongst all peers in the network is achieved through *blocks*. A block is basically a set of transactions. Every participant is able to create a block: it contains all the transactions the participant has received after the previous block that the participant considers to be valid (thus, all transactions in his pool). If a block is generated, all transactions it contains are removed from the pool. The block is then transmitted into the network; other peers will verify that all transactions are valid indeed. If so, they will remove the transactions from their pool too and safe the block.

The transactions included in the block can be seen as confirmed: now the block forms a base for new transactions. New transactions that conflict with the transactions committed in the block, will be considered as invalid. Thus, in general, the creator of the block decides which transactions will be confirmed and which will not.

This may again cause problems, as it could be possible for a group of block creators to exclude transactions from a certain sender or to a certain receiver. To limit the power of individual block creators, the Bitcoin system requires that every block should include a solution to a proof-of-work: a piece of data that is difficult to produce. Every block message should contain a byte string, called a *nonce*[1]. The block creator should find a nonce such that the hash of the textual representation of the block message is lower than a predefined value. In practice, this means that this hash should start with a number of zero-bits. Since hash functions are irreversible and can thus not be predicted, generating such a hash can be only achieved by repeatedly trying different nonces. If one finds a nonce that satisfies this condition, the block is considered to be valid. If other participants receive this block, they can easily verify the solution by calculating the hash of the block. The predefined hash minimum[2] is set in such a way that every 10 minutes (on average), someone in the Bitcoin network will find a new block.

Through this mechanism, every participant should have an equal chance to create a block, as finding a nonce that results in a block whose hash is small enough is decided by chance. However, in reality this means that participants with higher computational power can find blocks easier than those with less computational power.

Nodes that create blocks and try to find a solution to the proof-of-work are called *miners*. In order to stimulate participants to create blocks, the participant that found a valid block gets an incentive for its effort: newly created bitcoins. The miner is allowed to include a special transaction in his block, containing no input transaction and just one output transaction. This is the only way new bitcoins can be introduced into the Bitcoin system.

Concluding, a block can be found valid only if it satisfies at least the following criteria:

- All transactions should be valid;
- No transaction should have been included in previous blocks;
- A block should contain exactly one mining reward;
- The hash of the block message (including the nonce) should be lower than the predefined value;

2.3 Blockchain

Finally, in order to create a chronological transaction order, the blocks are chained together. Every block references to a previous block, called the *parent*. Blocks that refer to a parent, are called *ancestors*. The first block ever generated is called the *genesis block*. The longest path from any block to the genesis block is called the *blockchain*. The blockchain determines the chronological order of Bitcoin transactions.

Because mining rewards are only valid if the related block is part of the blockchain, miners will try to find blocks that builds on the newest block in the blockchain. Building on an older block would be inefficient, because it would create a new, shorter branch. Since rewards are only valid if the blocks are part of the blockchain, the shorter branch has to become longer than the currently longest branch, which requires more effort.

It is also possible that two different participants find a block at the same time, having the same parent. This may again cause confusion in the network about what block to build on afterwards. In this case, participants generally try to generate blocks that build on the block they received first. This phenomena, *blockchain forking*, is not covered in depth in this paper.

3. MODEL

The Bitcoin protocol, as described in the previous section, is specified in mCRL2 [12]. The mCRL2 language is a formal definition language based on process algebra. The associated toolset contains tools for simulation, visualisation and verification of modelled protocols. The created mCRL2 specification of the Bitcoin protocol is an abstracted version of the actual Bitcoin system, and will be used to analyse the behaviour of the protocol under several conditions.

3.1 mCRL2

micro Common Representation Language 2 [12] is based on the process algebra ACP. Fundamentally, the mCRL2 language focusses on processes extended with abstract data types, which enables the specification of both data and process behaviour.

Processes contain actions representing events that can be performed. The most important operators that are supported in mCRL2 are sequential composition (`.`), nondeterministic choice (`+`), communication (`|`) and parallelism (`||`). When processes are put parallel in mCRL2, they communicate through synchronised actions. Besides this, mCRL2 supports conditional steps (`(Expression) -> Action`) and logical quantors (such as `forall V . ( Boolean ) -> Action`) to create more advanced processes. Data types in mCRL2 include inter alia integers, booleans, maps, lists and sets. Processes can use these data types for their actions and conditions.

3.2 Bitcoin model

The created model [3] is an abstract view of the Bitcoin Protocol. Within the model, the behaviour of participants in the Bitcoin network is specified. This section discusses several aspects of the model.

The model focuses on making transactions, generating blocks and exchanging information about generated blocks and transactions. Blockchaining is implemented only to a limited extent: blockchain forking is not implemented in the model. Finally, in order to keep the model as compact as possible, only strictly necessary information fields are included. For instance, checksum and message size fields are omitted.

3.2.1 Actions

The `Peer` process is the most important process in the created model, as it is specified to behave like a participating node in the Bitcoin network. It has the following external actions on network level:

`inv (n1:NodeID, h:Hash, n2:NodeID)`
: Represents an actual *inv message* from node `n1` to node `n2` with inner message hash `h`. In other words, this is the trigger action to start information exchange.

`getdata (n1:NodeID, h:Hash, n2:NodeID)`
: Indicates that node `n1` is interested in hash `h`. The action request the full transaction or block with hash `h` from node `n2`.

`tx (n1:NodeID, h:Hash, n2:NodeID, t:Transaction)`
: This action is a result of the `getdata` action, as it transfers transaction `t` corresponding with hash `h` from node `n1` to node `n2`.

`block (n1:NodeID, h:Hash, n2:NodeID, bl:Block)`
: In a similar fashion, this action communicates block

[1] In this context, a nonce is a unique, random byte string.
[2] How this hash value exactly is determined and communicated, is not covered in this paper.

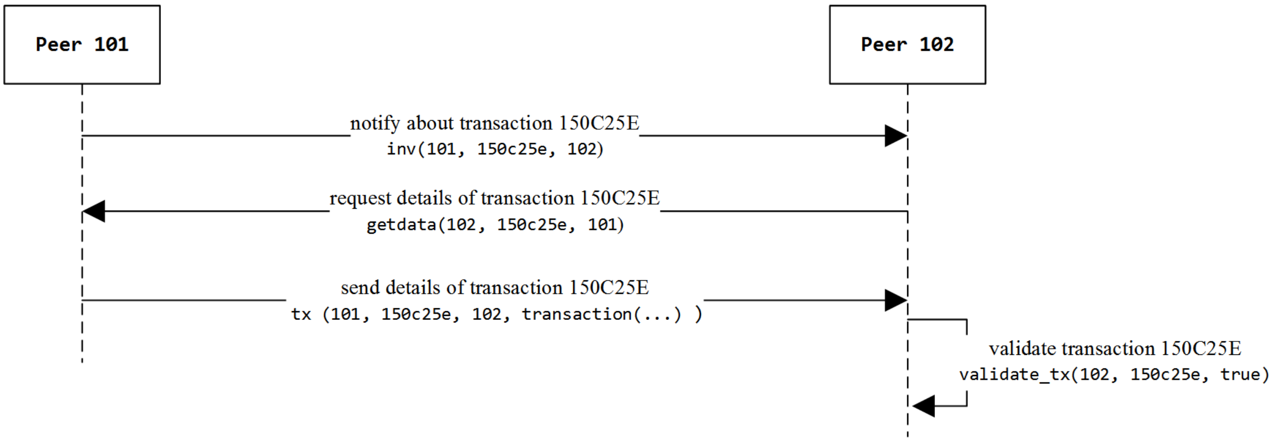
Peer 101
Peer 102
notify about transaction 150C25E
inv(101, 150c25e, 102)
request details of transaction 150C25E
getdata(102, 150c25e, 101)
send details of transaction 150C25E
tx (101, 150c25e, 102, transaction(...))
validate transaction 150C25E
validate_tx(102, 150c25e, true)

Figure 2. A typical use-case scenario of the Bitcoin protocol: a peer informs a connected peer about a transaction. This diagram also shows the corresponding protocol message types, which are used in the mCRL2 model as well.

bl corresponding with hash h from node n1 to node n2 as a response to a getdata request.

The process also contains a number of external actions that are not visible on network level, but only locally:

`generate_block (n1:NodeID, tx_list:List(TX))`
: This action represents the event that node n1 solved a block, containing all unconfirmed *tx messages* it gathered. As a result of this action, the block is added to its own blockchain.

`transfer_btc (n1:NodeID, a:BTC, n2:NodeID)`
: This action represents the event that node n1 transfers an amount of a bitcoins to node n2. Internally, a new transaction will be generated, using previous transactions as inputs and a new transaction to n2 as output. Subsequently, the transaction is added to the user's pool.

`validate_tx (n1:NodeID, h:Hash, b:Bool)`
: Represents the verification of a transaction with hash h by node n1. If boolean b is true, the corresponding transaction will be added to its own pool; if b is false, the corresponding transaction will be disregarded.

`validate_block (n1:NodeID, h:Hash, b:Bool)`
: In the same way, node n1 verifies whether a block is valid or not. If boolean b is true, the corresponding block will be added to its own blockchain, while false means the corresponding block will be disregarded.

3.2.2 Data types

The specification contains a small number of special data types that are necessary to store and transfer information in a proper way.

`Transaction`
: A transaction contains a list of input transactions (List(RefTransaction)) and output transactions (List(OutTransaction)).

`RefTransaction`
: A reference transaction (or *input*) consists of a Hash and an Offset.

`OutTransaction`
: Either contains a sending NodeID, an amount of bitcoins, and a receiving NodeID, or (in case of a mining transaction) an amount of bitcoins and a receiving NodeID.

`TX & TXList`
: A TX contains a Hash and the corresponding Transaction. A TXList is a list of TXs.

`Block & BlockList`
: A Block contains its own Hash, a reference to the previous block (represented by the previous block's Hash) and a TXList. A BlockList is a list of Blocks.

3.2.3 Data expressions

To be able to analyse the different data objects used in the processes, a number of supporting data expressions are defined in the model. These expressions behave like functions: for a number of parameters, the expression will return an object of a certain data type. For example, the data expression *contains_hash(h:Hash, txl:TXList)* will return a boolean that indicates whether or not the given TXList txl contains a transaction that corresponds with hash *h*. This data expression is used in Peer after an *inv message* is received, to check whether the receiver peer already has the advertised transaction in its pool. If the data expression returns true, the process will allow to do an getdata action, wheras if the result is false, the process will ignore the advertisement.

3.2.4 Process

The state of the Peer process is determined by five parameters: an id (NodeID), a pool (TXList), a blockchain (BlockList), the current balance (Btc) and a list of transactions of which this peer is the receiver that have not been used as an input yet (List(RefTransactions)). These parameters are necessary for the process to determine which actions are allowed. For instance, the action transfer_btc should only be allowed if the current balance of the peer is greater than zero. While the id of a Peer never changes, the other four parameters might change if one or more of the previously mentioned actions are executed.

3.2.5 Intialising the model

Finally, the model is initialised by creating a number of peers in parallel composition. By doing this, the peers will be able to communicate with each other and thus represent a Bitcoin network.

4. REQUIREMENTS

Because cryptocurrencies are a replacement for traditional currencies, cryptocurrencies inherit some properties of them. Even more, one can make additional requirements to make it a stronger type of currency. Together, these form five powerful requirements a cryptocurrency should satisfy:

1. Money is always in the possession of one party and one party only;
2. Money can only be spend by the owning party;
3. Once money has been spent, it cannot be spent again;
4. One should be able to spend its money at all times;
5. Money can be generated by mining and mining only;

We will further investigate whether the Bitcoin protocol satisfies these requirements.

5. VERIFICATION

5.1 Approach

One of the main obstacles in verifying the Bitcoin protocol specification in mCRL2 is that the number of possible states becomes too large. Because every new transaction creates a unique, new state, and the number of possible transactions is infinite, the model also has an infinite number of possible states. This prevents us from verifying whether a condition, as defined in the previous section, will not occur under any circumstance.

In order to still be able to analyse the Bitcoin system, scenario-based verification will be used [9]. With scenario-based verification, we define a couple of (realistic) scenarios in which we will verify the requirements. By this means, we will be able to analyse the behaviour of the Bitcoin protocol in a certain scenario. To achieve this, we make use of parallel composition and `error` actions.

Parallel composition can be applied by expressing characteristics of the scenario as a separate process in mCRL2, defining what actions can occur in what sequence(s). Subsequently, the scenario process and the actual model are put in parallel composition. Finally, by using communication, the two processes are synchronised. In our model, we use the process `Force` for this.

In addition, we introduce new actions into the processes: `error` actions. We define conditions in the processes that should be true at all times. If the condition somehow is not satisfied, the process will allow an `error` action to occur. As we will see shortly, the mCRL2 toolset can detect the occurrence of such actions when linearising a model, and report the shortest trace to the occurrence. Thus, if the model of a scenario can be linearised without `error` actions to occur, the model satisfies all conditions specified in that model. All requirements from section 4 have been translated into `error` actions that will occur if a requirement is not satisfied.

An important advantage of scenario-based verification is that the created scenario models have significantly less states compared to the full model. Thus, the specified scenario can be analysed for anomalies, whereas that cannot be achieved with the complete model. However, as mentioned before, a scenario-based approach does not allow us to draw conclusions on the Bitcoin protocol as a whole, since a scenario is only a subset of all possible situations.

As described before, the mCRL2 toolkit contains a number of powerful tools that are useful in the verification process. The tool `mcrl22lps` checks the mCRL2 syntax and transforms it into a linearised process specification (LPS) [11]. With this LPS, we can manually simulate the model using the tool `lpssim`. Even more, we can transform it into a labelled transition system (LTS) using the `lps2lts` tool. By generating an LTS, the state space of the input model can be determined. Besides that, the tool can detect the occurrence of one or more actions, as well as the shortest trace to the occurrence. Therefore, we can use this to detect `error` actions. If we can generate an LTS without warnings about the occurrence of `error` actions, we can conclude that the model satisfies the requirements as defined in the model.

5.2 Setup

We have a few assumptions about the scenarios we analyse. First of all, we assume that all peers have the *genesis block* in their blockchain. Secondly, the first peer in the model is the receiver of the mining reward of the genesis block. Finally, we assume that all peers know about each others existence, i.e. have set up connections with each other and are able to communicate with each other. Using this as a basis, we have analysed situations in which peers could transfer bitcoins and generate blocks in random order.

5.2.1 Normal circumstances

The first step is to investigate whether the Bitcoin system satisfies the requirements in a perfect situation, i.e. a situation in which peers exactly behave as they are supposed to behave. We configured this scenario with a small number of peers (2, 3 and 4). In this scenario, a maximum number of allowed transactions is set. The process `Force` limits the number of `transfer_btc` actions all peers together perform to the set maximum.

As can be seen from Table 1, a higher number of peers and a higher number of allowed transactions result in a rapid increase of the state space.

Table 1. Generated number of states, level-depth and number of traces after linearising the scenario of normal circumstances.

No. of peers	No. of allowed transactions	No. of states	Depth	No. of traces
2	1	68	14	98
2	2	857	26	1223
2	3	8611	38	12154
2	4	77992	50	109091
3	1	1839	22	2828
3	2	280567	40	411394
4	1	35419	28	55248

5.2.2 Corrupt peers

Having analysed situations in which every peer behaves exactly as originally designed, the question arises what happens if peers in the Bitcoin network deviate from this pattern. Since the the complete Bitcoin system is based on a peer-to-peer network, it is naive to suppose that every node will adhere to these specifications. Messages transmitted over a network might get lost or arrive delayed. Even more, since bitcoins have real value in the real world, people will try to tamper with the communication in order to steal money from others, or spend money they own multiple times. Besides that, it is even conceivable that people will try to hinder the flow of transactions by confusing the network with conflicting messages.

Therefore, it is interesting to analyse scenarios in which corrupt peers are part of the network. We have analysed the following situations:

1. A peer produces incorrect messages
   (a) Invalid amount of money
   (b) Invalid output transactions
   (c) Invalid input transactions
2. Contradicting messages
   (a) *tx messages* with the same inputs, but different outputs
3. A peer only sends messages of a particular type
   (a) *inv messages*
   (b) *tx messages*
   (c) *block messages*
4. A peer ignores/refuses messages
   (a) From a certain peer
   (b) *tx messages* with a certain receiver
   (c) *tx messages* with a certain sender
   (d) Blocks containing a certain transaction

5.3 Results

Scenarios for these situations have been created, based on the complete model, and have been verified against the requirements using `error` action detection on linearising.

Regarding the ideal scenario, we were able to determine that with up to four peers and up to four transactions, all conditions are satisfied. When it comes to the other scenarios, we were able to determine that for all these scenarios, using three to five peers, all requirements are satisfied. Generally speaking, we can conclude the following: as long as a simple majority (thus, more than 50%) of the peers behaves according to the protocol, the network will recover from the scenarios we analysed in a correct way because the majority will create a correct blockchain. If more than half of the peers deviate from the protocol definitions, it is likely that the corrupt messages will dominate the network. This conclusion is consistent with the work of Nakamoto [14], in which it is stated that the majority makes decisions within a Bitcoin network.

The Bitcoin protocol was updated a couple of times in its relatively short existence in order to limit the effects of the scenarios as given above [13]. For instance, scenario 2a is better known as a *double-spending attack*, in which a person tries to spent the money it has multiple times by creating multiple different, but valid *tx messages* while sending them to different peers. It might take a long time before someone in the network receives two conflicting transactions and will refuse one of them. As we have seen now, the Bitcoin protocol will eventually acknowledge only one of the transactions to be valid as long as a majority adheres to the Bitcoin protocol.

6. RELATED WORK

As a result of the increasing popularity of Bitcoin, several researchers have published articles about the system. Clark and Essex [8], Barber et al. [1] and Karame et al. [13] published about the cryptographic aspect of the Bitcoin, including suggestions for improvement. For instance, G.O. Karame et al. present a study on the security side of the double-spending problem with Bitcoin, in which they show that double-spending attacks are possible at low cost and high success probability. Double-spending attacks and their probabilities are, compared to our work, analysed in more detail by G.O. Karame et al.

Decker and Wattenhofer [10] studied the network performance of the Bitcoin and explain how blockchain forks are created. In contrast to our work, this research focuses on the blockchain forking aspect of the Bitcoin system.

Cederquist and Dashti [7] have formally specified a general payment protocol and, under some assumptions, verified properties which prove that the protocol specified is 'fair'. It was also proven that even with a Dolev-Yao intruder, the properties are held. In this research, μ-CRL, the predecessor of *mCRL2* was used. Our work verifies the Bitcoin protocol in a similar way.

7. DISCUSSION

Based on what we have found, we can conclude that in the scenarios investigated, the Bitcoin protocol behaves as expected. Although the scenarios investigated can be considered as frequently occurring scenarios, the results cannot be regarded as a proof that the Bitcoin protocol is a perfect mechanism. After all, Bitcoin is a peer-to-peer network whose behaviour cannot be predicted. There are many scenarios that might occur in the real Bitcoin environment that we have not verified.

Therefore, the verification of the Bitcoin protocol is only to a limited extent proven by this research. Nevertheless, by creating the model, we were able to draw conclusions about a number of realistic scenarios that prove that the Bitcoin protocol satisfy a number of strong requirements.

In addition, the model focuses on information exchange regarding transaction and block data. Other aspects, such as blockchaining, are not investigated thoroughly. Besides that, the model has limitations that are introduced to bound the number of states generated when linearising, as mentioned in section 5.2. Thus, the model as presented is not complete and could be improved in further works.

8. CONCLUSION

We have presented a model for the Bitcoin protocol, which can be used for analysis of the Bitcoin network. Additionally, we have verified the model against a list of requirements which the Bitcoin should satisfy to be a reliable currency. By using scenario-based analysis, we were able to determine that in a number of common scenarios the Bitcoin protocol behaves as described in our requirements: both under a ideal scenario as in more realist scenarios, in which not all peers adhere to the protocol.

These findings contribute to the position of Bitcoin as a (crypto)currency, as we have to some extent proven that Bitcoin satisfies properties it should at least have in order to be safe to be used as currency. Even more, we have proven the robustness of Bitcoin, i.e. that Bitcoin handles scenarios with malicious peers well.

9. ACKNOWLEDGMENTS

The author would like to thank Jaco van de Pol for the helpful comments and discussions throughout the research process.

10. REFERENCES

[1] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make Bitcoin a better currency.

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7397 LNCS: 399–414, 2012. doi: 10.1007/978-3-642-32946-3.

[2] BBC. BBC: Bitcoin news January 2013-March 2014, Accessed 28 March 2014. URL `http://www.bbc.co.uk/search/news/?q=bitcoin+mtgox&video=on&audio=on&text=on&sort=date&start_day=01&start_year=2013&end_day=28&end_month=04&end_year=2014`.

[3] W. J. B. Beukema. mCRL2 model of the Bitcoin Protocol, May 2014. URL `https://github.com/wietze/bitcoin-protocol`.

[4] Bitcoin Foundation, The. Official Bitcoin development repository, March 2014. URL `https://github.com/bitcoin/bitcoin/`.

[5] Bitcoin Foundation, The. Download page of Bitcoin QT, May 2014. URL `https://bitcoin.org/en/download`.

[6] Blockchain.info. Number of Bitcoin transactions per day, March 2014. URL `https://blockchain.info/charts/n-transactions?timespan=2year&showDataPoints=false&daysAverageString=1&scale=0&address=`.

[7] J. Cederquist and M. T. Dashti. Formal analysis of a fair payment protocol. In *Formal Aspects in Security and Trust*, pages 41–54, 2004.

[8] J. Clark and A. Essex. Commitcoin: Carbon dating commitments with Bitcoin. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7397 LNCS:390–398, 2012. doi: 10.1007/978-3-642-32946-3.

[9] P. Dechering and I. V. Langevelde. On the verification of coordination. In *Proc. of COORDINATION, LNCS 1906*, pages 335–340. Springer, 2000.

[10] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013 - Proceedings*, September 2013. doi: 10.1109/P2P.2013.6688704.

[11] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In R. Alur and T. A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1996. ISBN 3-540-61474-5. doi: 10.1007/3-540-61474-5_82.

[12] J. F. Groote et al. mCRL2, February 2014. URL `http://www.mcrl2.org/release/user_manual/index.html`.

[13] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 906–917, New York, NY, USA, 2012. ACM. doi: 10.1145/2382196.2382292.

[14] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009. URL