Modeling and Verification of the Bitcoin Protocol

Kaylash Chaudhary
School of Computing, Information and Mathematical Sciences
University of the South Pacific
Fiji
chaudhary_k@usp.ac.fj

Ansgar Fehnker
School of Computing, Information and Mathematical Sciences
University of the South Pacific
Fiji
fehnker_a@usp.ac.fj

Jaco van de Pol
Formal Methods and Tools
University of Twente
The Netherlands
j.c.vandepol@utwente.nl

Marielle Stoelinga
Formal Methods and Tools
University of Twente
The Netherlands
m.i.a.stoelinga@cs.utwente.nl

Bitcoin is a popular digital currency for online payments, realized as a decentralized peer-to-peer electronic cash system. Bitcoin keeps a ledger of all transactions; the majority of the participants decides on the correct ledger. Since there is no trusted third party to guard against double spending, and inspired by its popularity, we would like to investigate the correctness of the Bitcoin protocol. Double spending is an important threat to electronic payment systems. Double spending would happen if one user could force a majority to believe that a ledger without his previous payment is the correct one. We are interested in the probability of success of such a double spending attack, which is linked to the computational power of the attacker. This paper examines the Bitcoin protocol and provides its formalization as an UPPAAL model. The model will be used to show how double spending can be done if the parties in the Bitcoin protocol behave maliciously, and with what probability double spending occurs.

1 Introduction

Bitcoin [17] is a popular digital currency with over 81,398,896 transactions and 14,544,775 bitcoins in circulation until August, 2015 [2]. Due to its popularity, many merchants have started accepting bitcoins, such as a coffee shop located on the campus of one of the universities in Mexico [3]. Bitcoin eliminates the need for a trusted third party, such as a broker or a bank, to process payments. Every peer in the Bitcoin network keeps the collection of all transactions. This so-called ledger is organized in separate blocks, which are linked to their predecessor, thus forming a chain. Each peer maintains its own chain. This chain will work effectively for honest peers.

A system is not only composed of honest peers but also dishonest ones. If the peers deviate from their expected behaviour, then the chain may fork and double spending may be possible. Recently, there was a fork in the network that created two versions of the block-chain and it continued for six blocks [4]. That created the risk of double spending. Another reason that could increase the risk of double spending is the size of the pools. In June, 2014, Ghash.io, one of the mining pools of the Bitcoin network, came close to obtaining 51% of the Bitcoin network's hash-rate [1]. This could have led to double spending if the pool would have behaved maliciously.

Bitcoin attracted attention of researchers who worked on different aspects of the protocol. Bergstra et. al. identified research questions related to Bitcoin and other information money [11]. Ron et. al. did quantitative analysis of the Bitcoin transaction graph [20]. Herrmann did the implementation, evaluation

R.J. van Glabbeek, J.F. Groote & P. Höfner (Eds.): Workshop on Models for Formal Analysis of Real Systems (MARS 2015)
EPTCS 196, 2015, pp. 46–60, doi:10.4204/EPTCS.196.5

and detection of a double-spending attack [15]. The Bitcoin protocol has been modeled before to analyze different properties. Beukema developed a model in mCRL2 and verified properties such as double spending [12]. This model consisted only of peers that are responsible for mining as well. Block-chain forking was not investigated. Andrychowicz et. al. modeled Bitcoin contracts using timed automata [7]. Bastiaan did stochastic analysis of two-phase proof-of-work in the Bitcoin protocol [10]. This work was based on predicting the rate at which miners generate coins. However, it did not include block-chain forking. Our research focuses on block-chain forking and the probability of double spending, taking into account that the block-chain can be influenced by a malicious pool.

This paper focuses on the formal modeling of the Bitcoin protocol, developed in the UPPAAL model checker. It also presents the preliminary results of our probability analysis for double-spending attacks, based on the number of confirmations. This model will be used in the future to further analyze double-spending attacks, based on the computational power of the pools.

Organization of the paper. The next section introduces the Bitcoin protocol. A discussion on the model is presented in Section 3. Appendix A shows the full description of the Bitcoin protocol in UPPAAL. Our basic analysis of this protocol is discussed in Section 4. The paper concludes with a discussion on future possibilities for analysis, based on the developed model, in Section 5.

2 The Bitcoin Protocol

The Bitcoin protocol proposed by Nakamoto is described as a peer-to-peer electronic cash system [17]. The Bitcoin protocol is a decentralized payment system. There is no trusted third party, such as a bank or a broker, as used in traditional payment systems to guard against double spending. In this online payment system, a payer sends a payment directly to a payee. Bitcoin is a digital currency based on cryptographic principles. Every peer knows about every other peer's transactions; this is unlike a centralized system (Bank) where only the central authority knows about all transactions. Each peer in the Bitcoin network can be a payer (customer) or payee (vendor) or both. To start using the Bitcoin protocol, the user needs a Bitcoin account and a wallet. A prerequisite for a user is to have a private and a public key pair, first of all because the account is identified by the public key. Also, a payer needs to know the public key of the payee to make a payment. The next subsection will discuss the basic terminology.

2.1 Terminology

The following cryptography and payment terminology is used by the Bitcoin protocol [8].

- **One-way hash function**: Bitcoin uses the double-SHA256 hashing algorithm to hash transactions and to solve proof-of-work puzzles.
- **E-wallet**: An e-wallet is a database that stores bitcoins and their corresponding transaction hashes. It is usually referred to as the bitcoin wallet which can be software, mobile or web-based.
- **Bitcoin**: Bitcoin is a digital currency exchanged between parties over the Bitcoin network. BTC or ฿ is the currency symbol for bitcoin.
- **Proof-of-work**: A cryptographic puzzle (also known as a challenge) is based on Adam Back's Hashcash [9]. Originally, Hashcash was developed as an anti-spam email tool, where stamps are created and attached to the email. A random number is added to the email header and then a hash is performed on the header. A hash string is considered *valid* if it has a value equal to some set

target. In case of an invalid hash string, another random number is selected, added to the header and the hash is computed again. This process continues until a valid hash string is obtained. The recipient checks for the validity of the date and email address in the hash string and adds the hash string into the database. If the hash string is already present in the database, that hash string is invalid. Bitcoin uses Hashcash for block validity too. Miners perform proof-of-work calculations on recently broadcasted transaction blocks. This process is called *mining* and is performed by mining software. Mining yields bitcoins for the miner as a reward.

- **Block-chain**: The block-chain is a public ledger which stores processed transactions.

If a user wants to make a payment, that user has to create a transaction with all relevant information, i.e., the address of the payee, the amount of bitcoins and a challenge. This transaction is then broadcasted to all nodes in the network. Each node places this transaction into a block and tries to solve the proof-of-work for this block. Once a node finds a solution to the challenge, it broadcasts it to all other nodes. If all transactions are valid and not spent before, then the other nodes accept the block and start working on the next block in the chain. At certain times, the chain may have forks because two blocks were mined and broadcasted at the same time. Nodes always consider the longest block-chain to be the correct one. The following subsections will discuss transactions, blocks and the block-chain in more detail.

2.2 Transactions

There are two types of transactions: coin-base and regular transactions [18]. Coin-base transactions are used for new bitcoins, whereas regular transactions are used for transferring existing bitcoins from one user to another. Each transaction has one or more transaction inputs and outputs. A *transaction input* is a reference to an output of the previous transaction, which proves that the sender possesses the bitcoins it claims to have. A transaction input contains the response corresponding to the challenge of the previous transaction output. A *transaction output* determines the amount to be transferred to a payee account. Transaction outputs consist of an amount in BTC and a challenge, which specifies the conditions under which this output transaction can be claimed.

2.3 Blocks

A block is a set of transactions. Transactions in the same block can be considered to have happened at the same time. Transactions are only confirmed if they appear in some block. Unconfirmed transactions are kept in a transaction pool. Any node can create a block of transactions and broadcast it to the rest of the network. Since there can be many block creators, the network needs to decide which block should be added to the chain next, as the blocks can arrive in a different order to different peers. The protocol uses the proof-of-work solution to induce a unique order on blocks.

This proof-of-work solution is accomplished by the mining process. Mining is done by peers, called miners, or even by pools of miners. Initially, ordinary peers could participate in the mining process. Since the difficulty for finding a solution to the proof-of-work increased over the years, peers started working together in groups to solve the challenges. Such a group is called a pool. Today, there are many pools available with different percentages of participation in the Bitcoin network as shown in Figure 1 (August, 2015)[6]. Pool *BitFury* occupies 15% of the network hash-rate. The network hash-rate (hashes per second) is a measure for the processing power of the Bitcoin network. Related to the network hash-rate, the Bitcoin network has a global block difficulty which is automatically adapted after every 2016 blocks. It is a measure to ensure that it is not too easy to solve the challenges with the current network hash-rate. This difficulty is based on a target value: the miners must find a block solution less than the

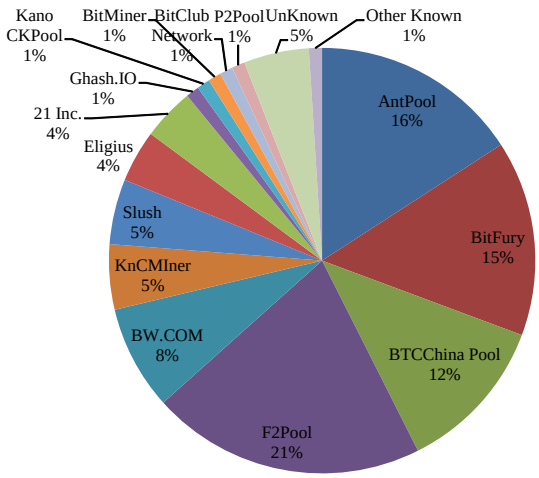

Figure 1: Pools with percentage hash-rate

target value. So the target value is a maximum hash. A decrease in target value means an increase in difficulty. On 1st August, 2015, the network hash rate was 413,204,212.12 GH/s and the difficulty was 52,278,304,845.59 [5]. The average time taken to find a solution by a particular pool is based on the hash-rate of that pool. The time is calculated using Equation 1 [10].

$$Time = \frac{difficulty * 2^{32}}{hash\text{-}rate} \tag{1}$$

For example, the average time taken to find a block solution by *BitFurry* is:

$$\begin{aligned} Time &= \frac{52,278,304,845.59 * 2^{32}}{0.15 * (413,204,212.12)} \\ &= 3622\ s \simeq 60\ mins \end{aligned} \tag{2}$$

To solve a challenge, miners in a pool randomly select a so-called *nonce* for a block. The hash of the block (including this nonce) is calculated and the result is compared with the target value. If the hash of the block is lower than the target value, then the proof-of-work is solved; otherwise the miners randomly select another nonce and calculate the hash again. This process repeats until a nonce is found that results in a hash value lower than the target value. Once a nonce is found, the block becomes valid and is broadcasted. On average, it takes about ten minutes for a pool to find a solution. Peers receiving this block can easily verify the solution by taking the hash of the block. Note, that the new block created by peers should include the hash of the most recent block on the longest path in the block-chain. Recall that every peer has its own version of the block-chain. The first block, called the genesis block, is mined with the maximum target value.

The next section describes how blocks are used to form a chain.

2.4 Block-Chain

The block-chain is used to provide an order on the transactions, whereas the transaction chain is used to track the ownership history. Every valid block that is broadcasted joins the longest path of the chain,

appending it to a predecessor in the longest chain. A block can only be added to the chain if it contains a hash of some previous block in the chain. A chain is formed by linking blocks which have a hash of a previous block in the block-chain. This defines an ordering of the blocks, thus preventing double spending.

A chain can have different paths but the longest path is the valid one. Transactions in blocks that are not in the longest chain are added back to the pool of transactions, and they can be used to build new blocks. This process allows miners to select the newest block from the chain to create the next block. Selection of an older block can result in a shorter branch and making this branch grow longer than the current longest branch will require a lot of effort. Whenever a block becomes part of the block-chain, the miners are rewarded. The reward is included in every mined block as a coin-base transaction. This coin-base transaction is the first transaction in the block. This incentive will tempt miners to select the newest block.

Since bitcoins are used over a large network, in which peers and pools can even be temporarily disconnected, there is a possibility that two blocks are created and broadcasted over the network simultaneously. Some peers might receive the first block and other peers the second one. In this situation, peers will continue building the chain on the received block. This will be resolved when a new block will be broadcasted, received by all, since one of them makes the longest chain. This shows that the blocks which are at the end of the chain could be less secure, compared to the blocks at the start of the chain. The transactions in the block at the end of the chain should not be considered confirmed as there is a non-negligible possibility that another chain will become the longest.

Double spending is spending a coin twice. It is possible in Bitcoin if the block-chain can be influenced. There are different scenarios for double spending such as the race attack, Finney's attack, the brute force attack, and the $> 50\%$ attack [19]. The last attack relies on the hash-rate percentage of a pool. We are looking at double spending based on the hash-rate of pools, and how a pool can race in order to make its chain longer. This scenario is described in the next section.

3 Bitcoin Model Discussion

We have used the UPPAAL statistical model checker to model and analyze the Bitcoin protocol [14]. The model consists of four types of automata: pool, peer, malicious peer and malicious pool. A peer creates transactions and places them in the transaction pool. A pool gets transactions from the transaction pool, includes them in a block and proceeds to the mining step. Upon successful mining, the mined block is broadcasted to other peers and pools for inclusion in the block-chain.

A malicious peer and pool collaborate together to cheat the system. The malicious peer creates two transactions of the same input, places one in the transaction pool for other pools to mine and gives the other one to the malicious pool. After the first transaction has been included in the chain, the malicious pool adds the block with the malicious transaction to the chain, creating a fork. The race is between these two chains. If the chain with the malicious transaction exceeds the length of the chain with the first transaction then the former becomes valid and all pools continue building blocks on the wrong chain. The detailed model description is presented in Appendix A. This model was built with the intention to investigate the probability of double spending in relation to the pool size. We have abstracted the size of the pool by the rate at which blocks are generated by each pool. This rate is calculated using Equation 1 in Section 2.

It was difficult to model the block-chain in UPPAAL due to the limited support for unbounded data structures. A fork in the block-chain leads to a side chain. The longest chain is the main chain. A side

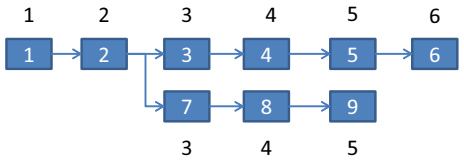

Figure 2: Block-chain fork with identifiers and length indicators

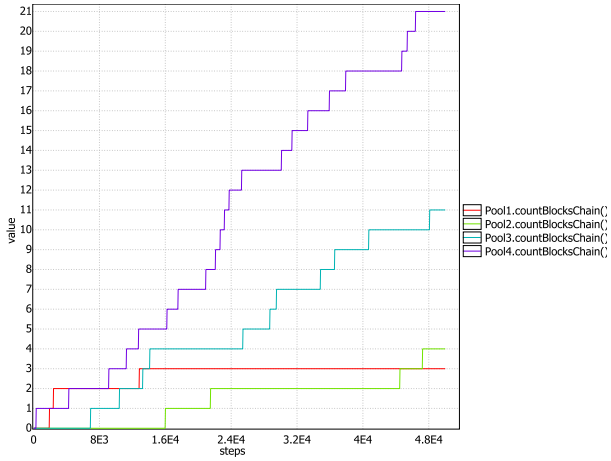

Figure 3: Number of blocks generated by each pool

chain could increase in length as well and possibly become the main chain later on. Therefore, it was important for us to store all chains. We modeled a block-chain by a list of triples, which store the current block number, the previous block number and the length of the chain to that block. For example, the following is the block-chain structure for the block-chain shown in Figure 2.

```
{[1,0,1],[2,1,2],[3,2,3],[4,3,4],[5,4,5],[6,5,6],[7,2,3],[8,7,4],[9,8,5]}
```

In Figure 2, the length for the chain to block number 9 is 5 and the length of the chain to block number 6 is 6. So the former is the side chain and the latter is the main chain. The length stored in the chain was used to determine the longest chain, to insert a block in the longest chain and to check whether a block is in the longest chain.

To ensure the accuracy of the model, we simulated the model several times. We checked whether each functionality included in the model is working correctly and according to the Bitcoin protocol. One of the main functionalities includes validating a transaction before including it in a block. To test this functionality, we created a malicious peer who was responsible for sending negative or zero amounts, and already spent transactions. The model behaved as expected. Simulation was also done to validate the structure of the block-chain at each pool. Since we have abstracted the size of the pool by the hash-rate of the network, we simulated to validate whether the hash-rate corresponds to the number of blocks generated in the main chain. In theory, a pool with a high hash-rate generates a larger number of blocks. Figure 3 shows the result of this simulation. Pool 4 has 50% network hash-rate while Pools 1, 2 and 3 have 10%, 18% and 22%, respectively. Our result validates that Pool 4 has created and successfully mined more blocks than the other pools.

4 Basic Analysis of the Bitcoin Protocol

Double spending is possible in fast payment transactions according to Karame et. al. [16]. Recall that blocks in the main chain are less secure towards the end. Therefore, vendors should wait for a certain number of confirmations, before considering a transaction in a block confirmed. The number of confirmations is the depth of a block from the end of the chain. For a vendor to consider a transaction to be confirmed, it has been mentioned [13] that the vendors need to wait for at least a depth of 6 blocks in the main chain. Based on this, we wanted to examine the relation between the hash-rate of a pool and the double-spending attack, following the scenario described in Section 3.

We did the analysis on the possibility of double spending for depths of 1-4 blocks. The result presented in Table 1 is based on four peers and four pools. The hash-rate for the pools are 18%, 22% ,10% and 50%. The last one is the malicious pool. There are 3 honest peers and 1 malicious peer. The malicious peer collaborates with the malicious pool for the double-spending attack. The upper bound for the transactions and the blocks are set to 200. The query given below was used to get the probability of double spending by simulation:

```
Pr[#<=500000](<> Pool4.checkBlockinChain(Pool4.MBlock))
```

The malicious transaction by the malicious peer is included in the block *MBlock* mined by Pool 4. The function *checkBlockinChain*() checks whether this block is in the longest chain or not. If it is, the double spending has occurred. For each confirmation depth 1-4, this query was executed for five hundred thousand steps.

Table 1: Probability of double spending based on the confirmation depth

No. of Confirmation	Probability of Double Spending	Runs
1	[0.870781, 0.970278]	118
2	[0.855061, 0.954924]	140
3	[0.797987, 0.897941]	211
4	[0.734029, 0.833722]	277

Our results show that double spending is still possible with a depth of 4 blocks. The probability was approximately above 80% for a depth of 4 blocks. The reason could be the hash-rate of the pool. However detailed analysis on double spending with different hash-rates is still needed.

5 Conclusions and Future Research

This paper presented a model of the Bitcoin protocol in the UPPAAL model checker and described how the important characteristics of the protocol have been modeled. The model was developed with the aim of investigating double spending in an environment where some peers are dishonest. This paper showed how a malicious pool could abuse its computational power and race against the network to include a malicious transaction in the longest chain. By statistical model checking, we analyzed how the probability that a malicious transaction is included in the longest chain, depends on the number of confirmations. The results suggest that the double-spending attack is possible for the parameters used in this scenario. However, our analysis is not complete yet, as we want to investigate also double spending with different hash-rates.

Future research is to do rigorous analysis of the Bitcoin protocol using this model. One could investigate the malicious act of a pool under different network hash-rates. Also, it would be interesting to model the block-chain for the Bitcoin protocol with two-phase proof-of-work [10], and compare the probability of double spending with the standard Bitcoin protocol with single proof-of-work.

References

[1] *51% double spending attack*. Available at http://www.coindesk.com/51-attacks-real-threat-bitcoin/.

[2] *Bitcoin charts*. Available at https://blockchain.info/charts.

[3] *Bitcoins at coffee shop*. Available at http://www.coindesk.com/bitcoin-payments-make-debut-at-mexican-university/.

[4] *Double spending risk*. Available at http://www.coindesk.com/double-spending-risk-bitcoin-network-fork/.

[5] *Hashrate and difficulty*. Available at https://blockchain.info/stats.

[6] *Hashrate Chart*. Available at https://blockchain.info/pools.

[7] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski & Lukasz Mazurek (2014): *Modeling Bitcoin Contracts by Timed Automata*. In: *Proceedings of FORMATS'14*, LNCS 8711, Springer, pp. 7–22, doi:10.1007/978-3-319-10512-3_2.

[8] Andreas M. Antonopoulos (2014): *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st edition. O'Reilly Media, Inc.

[9] Adam Back (2002): *Hashcash - A Denial of Service Counter-Measure*. Technical Report. Available at www.hashcash.org/papers/hashcash.pdf.

[10] Martijn Bastiaan (2015): *Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin*. Available at http://referaat.cs.utwente.nl/conference/22/paper/7473/preventing-the-51-attack-a-stochastic-analysis-of-two-phase-proof-of-work-in-bitcoin.pdf.

[11] Jan A. Bergstra & Karl de Leeuw (2013): *Questions related to Bitcoin and other Informational Money*. *CoRR* abs/1305.5956. Available at http://arxiv.org/abs/1305.5956.

[12] W.J.B Beukema (2014): *Formalising the Bitcoin protocol*. Available at http://referaat.cs.utwente.nl/conference/21/paper/7450/formalising-the-bitcoin-protocol.pdf.

[13] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll & Edward W. Felten (2015): *Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. IACR Cryptology ePrint Archive* 2015, p. 261, doi:10.1109/SP.2015.14.

[14] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis & Danny Bøgsted Poulsen (2015): *Uppaal SMC tutorial*. *STTT* 17(4), pp. 397–415, doi:10.1007/s10009-014-0361-y.

[15] M. Herrmann (2012): *Implementation, Evaluation and Detection of a Doublespend-attack on Bitcoin*. ETH Zürich, Department of Computer Science.

[16] Ghassan O. Karame, Elli Androulaki & Srdjan Capkun (2012): *Double-spending Fast Payments in Bitcoin*. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, ACM, New York, NY, USA, pp. 906–917, doi:10.1145/2382196.2382292.

[17] Satoshi Nakamoto (2008): *Bitcoin: A peer-to-peer electronic cash system*. Available at http://bitcoin.org/bitcoin.pdf.

[18] Krzysztof Okupski (2014): *Bitcoin Developer Reference*. Available at http://enetium.com/resources/Bitcoin.pdf.

[19] David Perry (2012): *Bitcoin Attacks in Plain English*. Available at http://codinginmysleep.com/bitcoin-attacks-in-plain-english/.

[20] Dorit Ron & Adi Shamir (2013): *Quantitative Analysis of the Full Bitcoin Transaction Graph*. In: *Proceedings of Financial Cryptography and Data Security (FC'13)*, LNCS 7859, Springer, pp. 6–24, doi:10.1007/978-3-642-39884-1_2.

A Description of the Bitcoin Protocol in UPPAAL

This section provides the models and their description for the Bitcoin protocol. Block-chain, blocks and transactions form an integral component of the Bitcoin protocol. The validity of a transaction is dependent on the block-chain. If the block of a particular transaction is included in the main chain, then it can be considered confirmed. Due to its importance, we have modeled block-chains together with blocks and transactions. There are two parties involved: pools and peers. A Pool is a group of miners responsible for solving the proof-of-work problem for individual blocks. Each Pool maintains its own block-chain. A Peer can be a customer or a vendor who is responsible for transferring bitcoins from one peer to another by creating transactions. Peers do not take part in the mining process. This model will show the interaction between these two parties.

The next subsection will discuss the data structure used to store all necessary information (transactions and blocks) in this protocol.

A.1 Global Declarations

Declaration 1 shows the structure of a transaction. A transaction is made up of an input and output transaction. Input transactions, *TXIN*, contain the previous transaction. This means that this transaction is based on the output of the previous transaction (positive BTC). Output transactions, *TXOUT*, contain the *Address* of the peer and the amount *value*. Note that in the Bitcoin protocol a transaction does not have a sequence number. Instead, it has hash. We have abstracted this by a number in our model. To maintain the uniqueness of transactions, we have the transaction number *Tx_num* stored and incremented globally. *Tx_num* is initialized to *PEERS*. This is because every peer will have one transaction to start off. Variable *status* is used to indicate the status of a transaction. It can be in three states; *UNCONFIRMED*, *CONFIRMED* and *INVALID*. Initially, a transaction is *UNCONFIRMED*. Once it is included in a block it is *CONFIRMED*. If it can not be included in a block, its status will be *INVALID*. *INVALID* is used to indicate to other pools that this transaction should not be included in a block. An invalid transaction has a negative or zero value in the output transaction, or it has already been spent.

Transactions are placed in a block. Declaration 2 provides the structure of a block. Every block has a unique block number, a previous block number that forms a link to the main chain, and the transactions. A block also contains a hash. We have represented it as a number *Block_num* which is maintained globally. *Block_num* is initialized to 2. The first new block created by a pool will be numbered 2. Block number 1 is reserved for the first (genesis) block.

We have also defined a structure *Wallet* shown in Declaration 3. Every peer will have a wallet to maintain unspent output from every transaction.

Other global declarations include channels and variables. Channel *blockSolution* is used to broadcast blocks to other pools and peers. This channel is used when a pool has found a block solution. Channel *request* and *send* are used to request and send predecessors of orphan blocks. There are two meta variables used in this model. One is *Blocknum* used for storing block numbers when synchronizing on channel *request* and the second is *chain* used to store block details when broadcasting a block solution.

In the Bitcoin protocol, when a peer creates a transaction it broadcasts it over the network. A pool collects those transaction and places it in its pool. Instead of having a transaction pool for each pool, we

Declaration 1 Transaction Datatype in UPPAAL

```
const int MAX_BTC_AMOUNT = 12;
const int MAX_TRANSACTION = 200;
const int PEERS = 4;
typedef int[0,PEERS] PEER;
typedef int[0,MAX_BTC_AMOUNT] AMOUNT;
typedef int[0,MAX_TRANSACTION]TXNUMBER;
typedef int[0,2] STATUS;

//output transaction
typedef struct{
    PEER Address;
    AMOUNT value;
}TXOUT;

//input transaction
typedef struct{
    TXNUMBER id;
}TXIN;

//transaction
typedef struct {
    TXNUMBER Id;
    TXIN TxInput;
    TXOUT TxOutput[2];
    STATUS status;
}Tx;

TXNUMBER Tx_num = PEERS;
```

Declaration 2 Block Datatype

```
typedef int[0,HASHSIZE] BLOCKHASH;

typedef struct {
    BLOCKHASH num;
    BLOCKHASH prevBlocknum;
    Tx tx;
}Block;

BLOCKHASH Block_num = 2;
```

Declaration 3 Wallet Datatype

```
typedef struct{
    TXNUMBER Id;
    AMOUNT value;
}Wallet;
```

have created a global transaction pool; *Tx TxPool[MAX_TRANSACTION]*.

The following subsections will discuss the two parties involved in the Bitcoin protocol.

A.2 Pool

Figure 4 shows the timed automaton *Pool*. This automaton has four locations; *Initial*, *Wait*, *Verify* and *Mine*. There are some local declarations, for the block-chain, the index of the longest chain and the length of the longest chain. We have implemented a block-chain as an array of integers, *chain[HASHSIZE][3]*. This array records the block number, the previous block number, the length of the chain to that block, and the ID of this pool (in case this block was created by this pool). The reason behind storing only block numbers and the previous block number is to achieve simplicity of the chain description. The third field in the array is used for the length of the chain. The fourth field is used for verification purposes only, to determine the number of blocks created by a particular pool. The local variables

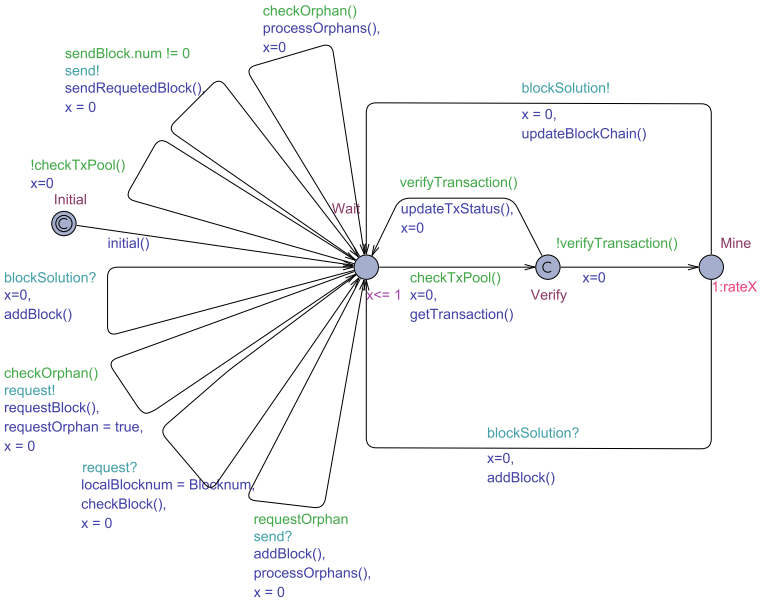

Figure 4: Pool automaton

indexLongestChain and *lengthLongestChain* are used to store the index and the length of the longest chain, respectively. These variables are used during block generation. The block-chain contains only block numbers in our model. Other details included in a block, such as transactions, are stored in an array *confirmedBlock[MAX_BLOCK]*. Variable *confirmedBlock* stores all blocks mined by any pool. There can be orphan blocks in the protocol. Orphan blocks are those whose predecessor is not in any chain. Therefore, an array is defined for orphan blocks, *orphanBlock[MAX_BLOCK]*. Variables *localTx* and *localBlocknum* are used to store intermediate results when verifying transactions and processing orphans.

This automaton models the three tasks of the pool. The first task is verifying and placing a transaction in a block. The second is mining the block to solve the proof-of-work challenge. Upon successful mining, the block is broadcasted to other pools and peers. The final task is maintaining the block-chain by adding blocks received from other pools, and requesting the predecessors of orphans.

The automaton starts from location *Initial*, taking a transition to *Wait*. During the transition, the automata initializes the block chain *chain* and sets the first block in the chain to 1. So when the first block is created by this pool, it should refer to block 1 as the previous block. At the location *Wait*, there are 5 outgoing edges and the rest are incoming edges, synchronized on channels with other automata.

If there are unconfirmed transactions in the transaction pool then the transition with the guard *checkTxPool()* will be enabled. The automaton will transit to location *Verify* if this transition is taken. The function *getTransaction()* copies the transaction into *localTx* and sets this transaction as confirmed in the transaction pool. The *verifyTransaction()* function checks the transaction for negative or zero values in the output section and it also compares the input of this transaction and the total value of the output transactions with the transactions in the main chain. The latter implements the double-spending check. If the input transaction number and the total value match with another transaction, then the input of this transaction has already been spent. Thus, the automaton transits to *Wait*, setting the transaction status in the pool as *INVALID*. If the check succeeds, then the automaton transits to the *Mine* location. The

location *Mine* has the rate of exponential set to *1:rateX*. This *rateX* is calculated from Equation 1 in Section 2.

At this location, if a pool solves the proof-of-work challenge then it takes a transition to location *Wait* by broadcasting the block to other pools and peers on channel *blockSolution*. The function *updateBlockChain()* adds the block with the block number stored in *Block_num* and the transaction stored in the *localTx* to *confirmedBlock*. This function also updates the global variable *chain*, increments *Block_num*, resets *localTx* and adds the block number and the previous block number to its block-chain *chain*. The block found by the pool will always be added to the main chain and it will refer to the last block in the main chain. The index of the last block in the main chain is stored in variable *indexLongestChain*.

If some other pool solves the proof-of-work challenge, then this pool will take the transition to the *Wait* location by receiving the block on channel *blockSolution*. The function *addBlock()* checks if the received block's parent is in the chain and if it is, then the block is added to *confirmedBlock*. The block number and the previous block number are added to the block-chain *chain*. It also sets the transaction that it included in the block being mined to *UNCONFIRMED*. If the parent is not in the *chain*, then this block is an orphan and it is added to *orphanBlock*. In either case, the transition is from the *Mine* to the *Wait* location. Likewise, if a pool is at the *Wait* location, it can also receive a block on channel *blockSolution*.

If there is no transaction in the transaction pool, the transition with the guard *!checkTxPool* will be taken. If a pool has orphan blocks in the *orphanBlock*, then the transition with the guard *checkOrphan()* will be enabled. This will let the pool request for the parent of the orphan block on channel *request*. The function *requestBlock()* updates the global variable *blocknum* with the block number of the parent. The automaton synchronizing on channel *request* will copy the value of *Blocknum* to *localBlocknum*. It will then check whether a block with block number *localBlocknum* is present in the *confirmedBlock*. If it is, then the block information is saved in the local variable *sendBlock*. When *sendBlock* has information in it, the transition with the condition *sendBlock.num != 0* is enabled. The transition will be taken on channel *send*, updating *sendBlock* values to zero. The synchronizing automaton adds the block in the *chain* and in *confirmedBlock* if it does not have that block and will process orphan blocks from *orphanBlock* by adding to the chain. That block will then be removed from *orphanBlock*.

The next subsection describes the Peer automaton.

A.3 Peer

Figure 5 shows the *Peer* automaton. Just like pools, peers also keep track of the block-chain using variables *chain* and *confirmedBlock*. The block-chain is needed for a peer to check and update the wallet if there is any transaction which has the output address as the peer's ID and it is included in the main chain. The main functionality of the peer is to create the transaction and to update the wallet. The Peer automaton has two locations: *Initial* and *Wait*.

The first transition from *Initial* to *Wait* is taken to initialize the chain and the wallet. The chain is initialized as described in A.2. All peers will initially be given ten bitcoins with an initial transaction. This will be updated in the wallet. With this, peers can send bitcoins to another peer by creating transactions. The transition on creating transactions will be enabled if there are bitcoins in the wallet and if the current transaction number *Tx_num* is less then the maximum transaction *MAX_TRANSACTION*. The select statement *add:int[0,PEERS-1]* allows peers to send bitcoins to any peer. The selected peer address *add* will be passed as parameter into the function *create_Transaction(add)*. This function gets the index of the wallet where a positive bitcoin value is available, searches for an empty index in *TxPool*

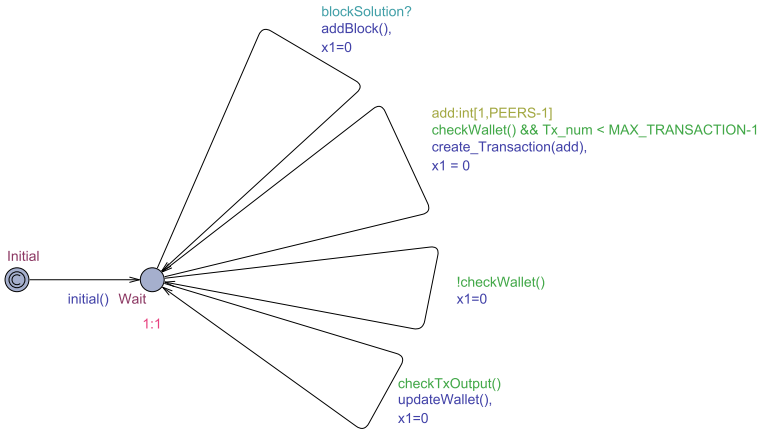

Figure 5: Peer automaton

and updates *TxPool* with the current transaction number *Tx_num*, the input transaction extracted from the wallet, the address of the peer (receiver), and the amount (always 1). If there is any amount left then the peer needs to send it back to itself. The function will also set the value of the transaction in the wallet to 0. This means that this peer cannot use this transaction as an input transaction in another one.

The next transition checks if there are no bitcoins in the wallet; it will be taken if there is none. To update the wallet, the condition *checkTxOutput()* needs to be satisfied. It checks the main chain to see if there is any transaction that has an output with this peer's ID that has not been recorded in the wallet previously. If that is the case, then this transition is taken, and the wallet is updated by function *updateWallet*.

The last transition is on adding blocks to the block-chain using function *addBlock()* as described in subsection A.2.

In this model, all peers and pools are connected to each other. So there will not be any forks in the chain unless there is a malicious peer and pool. The following subsections will discuss the malicious pool and peer.

A.4 Malicious Pool

The purpose of the malicious pool in our model is to race against the network, in order to increase the length of its chain. We wanted to see whether it is possible for a malicious pool to race even when it is some blocks behind. This scenario is further illustrated in Figure 2. Suppose a malicious peer has spent bitcoins to some other peer and the transaction is included in block 3. This malicious peer contacts the malicious pool so that the spent bitcoins can be transferred back to this peer. The malicious pool creates a fork in the chain by adding and linking block number 7 to block number 2. Note that the malicious pool has to assign the predecessor of block 7 as block 2 since the first transaction is included in block 3. Therefore, the malicious pool has to race to make its chain longer than the chain of the honest pools. If it succeeds, then a successful double spending has occurred.

Figure 6 shows the malicious pool automaton. This automaton is the same as automaton *Pool*, except that there are two additional transitions. The first transition checks if there is any transaction from the malicious peer and whether that transaction has been included in the chain. It also checks for the number of confirmations to wait for, so that the peer to which this transaction was made, could believe that the

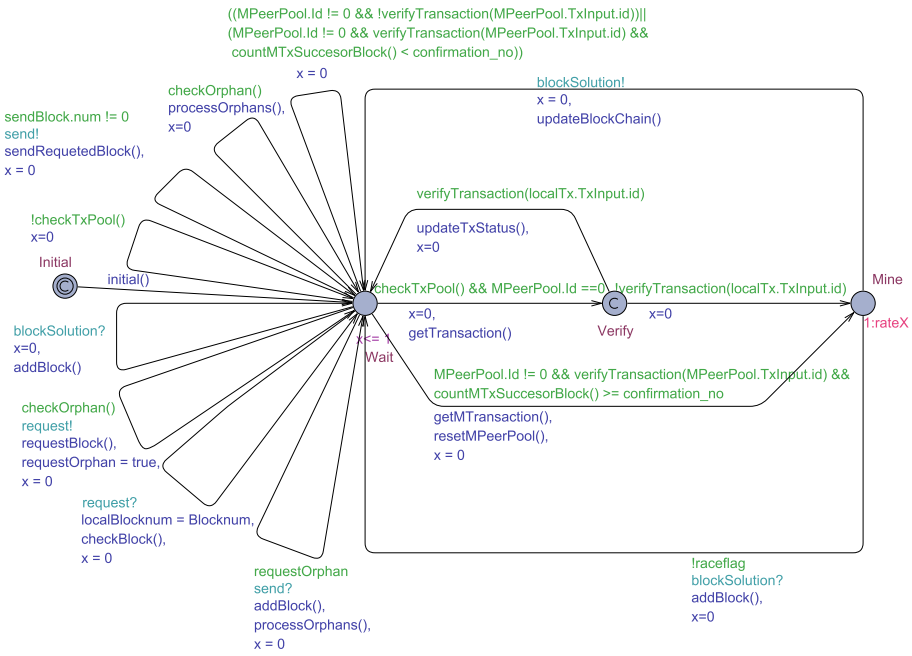

Figure 6: Malicious pool automaton

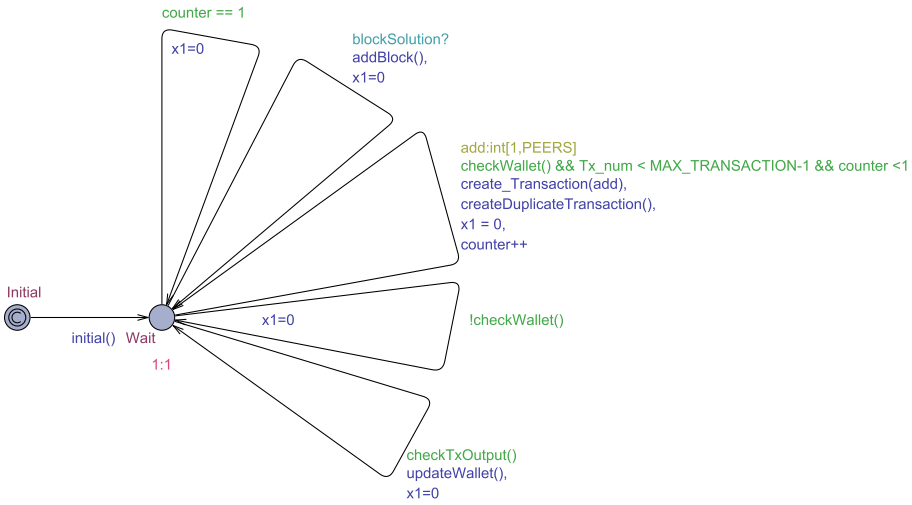

Figure 7: Malicious peer automaton

transaction has been confirmed. The transition will be taken if the condition is true.

The second transition is from location *Wait* to *Mine*. It will be enabled if there is a transaction with the same transaction input as the first transaction created by the malicious peer (malicious transaction) *and* the first transaction has been included in the chain with some blocks of confirmation. If both conditions are satisfied, then the malicious transaction will be copied into the *localTx* and the malicious transaction pool will be reset. At the location *Mine*, the pool will mine a block with this transaction and will broadcast this block while transiting from location *Mine* to *Wait*. The race flag *raceflag* will be set when there is

a malicious transaction and this flag will be set to false when this malicious transaction is included in a block by this pool. This flag is used when receiving blocks at location *Mine*. It will not allow this pool to receive any block until this transaction is included in a block. The block with the malicious transaction will be added to the chain, thus creating a fork. This fork will lead to two chains: the main chain and a side chain. Honest pools will continue building blocks on the main chain while the malicious pool will continue on the side chain. A possible result is shown in Figure 2. The malicious pool will also keep track of the chain built by the others in the network and will monitor when its side chain gets longer than the main chain. When that happens, the main chain for all pools will be same. Other transitions are the same as for normal pools, as described in Subsection A.2.

A.5 Malicious Peer

The malicious peer will create two transactions, which will have the same transaction input. The first transaction will be placed in the transaction pool *TxPool*, so other pools can include it in a block and proceed to the mining step. The second transaction will be given to the malicious pool. Figure 7 shows the malicious peer automaton. One new function is introduced, *createDuplicateTransaction()*. The purpose of this function is to create a malicious transaction and place it in a malicious transaction pool. The malicious peer automaton creates only two transactions.