Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin

Martijn Bastiaan
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
martijn@hmbastiaan.nl

ABSTRACT

The security of Bitcoin (a relatively new form of a distributed ledger) is threatened by the formation of large public pools, which form naturally in order to reduce reward variance for individual miners. By introducing a second cryptographic challenge (two phase proof-of-work or 2P-PoW for short), pool operators are forced to either give up their private keys or provide a substantial part of their pool's mining *hashrate* which potentially forces pools to become smaller. This document provides a stochastic analysis of the Bitcoin mining protocol extended with 2P-PoW, modelled using *CTMC*s (continuous-time Markov chains). 2P-PoW indeed holds its promises, according to these models. A plot is provided for different "strengths" of the second cryptographic challenge, which can be used to select proper values for future implementers.

Keywords

Bitcoin, proof-of-work, secondary challenge, mining model, continuous-time Markov chains, 51%-attack

1. INTRODUCTION TO BITCOIN

Bitcoin is the first decentralised computer network maintaining a public ledger (see: appendix A) of transactions, reaching a consensus through a mechanism called proof-of-work. It was first proposed[14] by an anonymous identity known as Satoshi Nakamoto, triggering interest from computer scientists as it was the first to solve the Byzantine Generals' Problem in a practical manner. This, in turn, solved the double-spending problem[5] which allows spending the same token of value twice. This problem has generally been present in digital currency schemes.

An important property of Bitcoin thus is its ability to act as a decentralised payment system. It eliminates the need for a trusted third-party, such as a bank or an e-commerce business such as PayPal to hold and transfer units of monetary value. For various reasons this has led to the interest of the general public, with more than six-thousand (non-website) parties accepting it according to CoinMap[1] at the time of writing (January 2015).

[1] Publicly editable database containing physical locations of merchants accepting cryptocurrencies

22nd Twente Student Conference on IT January 23rd, 2015, Enschede, The Netherlands.

The public ledger consists of separate blocks of transactions, each block chained together by miners to form the *blockchain*. For all other nodes to accept a new *(transaction)block* a token which is hard to produce but easy to verify must be supplied. This is known as proof-of-work; an often proposed method to prevent spam[13]. Although other forms of proof-of-work exist, Bitcoin uses SHA-256 hashes together with a *nonce* to provide such a scheme.

The exact capabilities and inner workings of transactions are beyond the scope of this document. It is sufficient to realise transactions carry bitcoins from one account (public key) to another. Only the holder of the private key corresponding to an account can authorise the movement of funds. An authorised (signed) transaction may be broadcasted to all other nodes on the Bitcoin network.

For all other nodes to accept the transaction as permanent and spendable however, it must have been collected and successfully processed by a miner. Just like any other node, a miner listens to the network for broadcasted transactions. A miner can decide whether to accept or reject a transaction, based on its own set of rules, although it should at least be a valid transaction. That is, forged by a fixed set of rules prescribed by the Bitcoin protocol.

Upon accepting a transaction, a miner appends it to its transaction block and starts (or continues) a process called mining. Mining is comparable to finding a winning ticket among a number of scratch-off cards: one must simply scratch until finding a winning one. This challenge is emulated by trying to produce an SHA-256 hash which lies beneath a certain threshold by cycling through *nonces*. If a miner can find such a hash, it may broadcast its solution and it is rewarded bitcoins in return.

The produced hash not only protects transactions, but also covers the hash of the previously found block. It is thus that the blockchain is formed. Nodes receiving a new block will switch to the "new" longer chain if they deem it valid. For all other miners, it is therefore beneficial to work on top of the new chain. If they would not, nodes would need to pick a chain which leads to one chain being rejected. This, in turn, results in one miner missing out on its reward.

1.1 Difficulty

As multiple blockchains can form this way one must understand how nodes will choose the same chain, thus creating consensus. For Bitcoin this is simply the longest chain, or in case of a tie the one having the highest difficulty. In order for a node to switch branches, the alternative branch must be at least six blocks ahead and must at least be as "difficult".

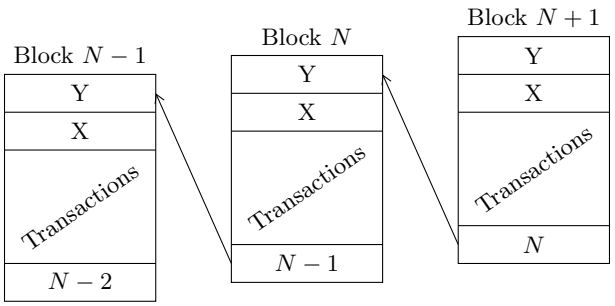

Figure 1. Blockchain with 2P-PoW in illustrated form. Each transaction block includes a pointer to the previous block, forming a "blockchain".

This global difficulty is recalculated every 2016 blocks[2] by estimating the current *hashrate* of the network and aiming for a ten-minute interval of releasing blocks in the future. As a result, difficulty is recalculated roughly every two weeks.

Since Bitcoin has greatly appreciated in value since its inception in 2008, an arms race arose between miners each trying to outpace the others. This has led to the development of specialised hardware and an enormous increase in hashrate. As of writing (December 2014), it is estimated the network calculates roughly 107 PH/s (petahash[3] per second); an increase of around 2100% compared to December 2013.

1.2 Pooling

Due to the high difficulty, solo miners (those not running an array of mining equipment, i.e. not running *a mining farm*) have a hard time solving a block even with the latest, specialised equipment. A miner running a 3 TH/s (terahash[4] per second) node for example, which is priced at €4000 at the time of writing (July 2014), is expected to find a block each half year.

To reduce the risk of not finding a block, pools are formed to still allow small miners to contribute to the network's hashrate. The operator of a pool generally sets a target much lower than the expected difficulty of the network. Once a client finds a block which conforms to that particular difficulty, it reports to the operator. This serves both as a proof-of-work, and insurance against theft as the coinbase address of the operator must be included in the hash. Other than that, clients generate blocks like they would normally, although harvested rewards are transferred to the operator. Again, while reward-addresses are embedded in the proof-of-work they cannot be forged and can thus serve as proof of stake for the operator, allowing it to distribute the reward among its clients.

It is easy to see that large pools therefore have a low variance on the number of blocks they find. This reduces risks and mostly increases profits for small miners. Large pools are thus likely to grow even more, leading to "superpools".

1.3 51%-ish attack

Pool operators can perform certain attacks[5] on the network as soon as their pool reaches a substantial percentage of the network. This is often described as the 51%-attack, although it has been shown attacks with less hash power can be performed[6][9]. These attacks all rely on the ability to generate the longest chain *in the long run*.

To understand this, consider two blockchains both having at least one block in common (or, in other words, having a common ancestor) with lengths n and m ($n > m$). If chain n is the honest one, and m the one of the attacker (holding more than 50% of the network's capacity), then both parties can generate a chain of length k ($k > n$), with probability p^{k-l} where l represents the current length of their chain (either n or m) and p the fraction mining capacity held. Therefore, if the attacker chooses k to be large enough, he will more often than not find a longer chain than its honest counterpart.

Recently, a pool (*GHash.IO*) held 54% of the hashrate for a day, which exceeds the theoretical attack threshold of 51%. Although the community swiftly responded by transferring mining capacity to other pools, the incentives to create large pools are still present, luring for another chance to disrupt the network.

1.4 Two phase Proof-of-Work (2P-PoW)

In response to *GHash.IO* holding more than half of the network's hashrate, a method to prevent large pools was proposed[7]. According to the authors, their proposed solution satisfies the side conditions of a good solution (paraphrased):

.. must preserve the existing blockchain, and with it, the existing Bitcoin balances.

.. must also preserve the large investments many miners have made and are planning to make in their equipment.

.. provides a seamless transition from the existing system to the new one, and provide adjustable knobs that can be fine-tuned for a desired trade-off that fits the community's needs.

In addition to the traditional hash of the block header, the authors propose a second proof-of-work, which signs the produced header with the private key controlling the coinbase (*payout*) address. Similar to existing hashing procedures this signature must meet a target set by the network. This second difficulty (Y), as opposed to the traditional difficulty (X), forces pool operators to distribute their private key to their clients if the pool operator wants to retain the same level of decentralisation. However, if an operator would naively share its private key, all clients would be authorised to move funds from the coinbase address to any destination.

Operators unwilling to share their private key therefore need to install mining equipment needed to solve Y in a timely manner. It is estimated that *GHash.IO* owns only a small percentage of the networks hashrate in hardware, as the pool shrank significantly after public outrage. Depending on the difficulty of the second cryptographic puzzle, this would only allow a certain number of untrusted individuals to join. This would, as *GHash.IO* is a public pool, severely limit its size.

The key to a well-balanced system wherein small pools can thrive but large pools cannot is thus a matter of choosing the ratio between the difficulties of both cryptographic puzzles right.

Starting with $X = D_{current}$ and $Y = \infty$ agrees with the current situation, as Y is not picky in the solutions it

[2] Also see: https://blockchain.info/en/charts/difficulty
[3] 10^{15} hashes
[4] 10^{12} hashes
[5] Reverse his own transactions and prevent transactions from gaining confirmations. See appendix E.

accepts. Decreasing Y would lead to more difficult hashing "puzzles" including the private key of the coinbase to be solved. Eventually, this method proposes a smooth path where X and Y both end up to be a certain ratio.

Bitcoin is a rather new concept with a sometimes confusing nomenclature. To prevent some of the confusion, its terms and concepts are summarised in appendix A.

2. PROBLEM STATEMENT

The proposal of Eyal *et al.*[7] does not include a rule for X and Y to follow, in order to allow a smooth transition to *two phase proof of work* (2P-PoW). The rule would need to include the rate at which mining equipment becomes obsolete, among with the estimated rate at which specialised signing equipment can be developed.

This research sets out to find these values and thus propose a smooth path for upgrading to this new approach. Thus it answers the following question:

> How to prevent the encouragement of large pools within Bitcoin mining [through *two phase proof of work*]?

To answer this question, two subquestions need to be answered:

> 1. Which values of X and Y allow small pools to exist, while not stimulating the creation of large pools?
> 2. At which rate does mining equipment become obsolete? I.e., can a rule for the transition to a smaller Y be applied?

As said, implementing 2P-PoW promises to provide a set of desirable properties, which this document will try to (dis)prove using various methods expanded upon in section 4. These properties form the hypotheses of this document.

1. A pool's market share will drop linearly as soon as it is unable to dedicate enough of its hashrate to Y challenges, all other things being equal.
2. The average reward of miners will not suffer in the long term, for different ratios as long as the total capacity stays the same.

If a positive result is found (i.e., the model yields acceptable values for X and Y), the research will focus on finding a smooth path to it by studying existing literature and analysing the rate at which hardware costs declined in the past.

3. PREVIOUS WORK

Bitcoin has been formalised for the first time by Beukema[2], and most recently by Andrychowicz *et al*[1]. Similar to Andrychowicz *et al.*, we are interested in the probabilistic nature of Bitcoin. In contrast to their research however, we do not need timed automata to develop our model, as we are only interested in the predictable (but probabilistic) rate at which miners generate coins.

Obviously, Ittay Eyal *et al.*[7] laid the groundwork for *two phase proof of work*, which properties are explored in this document.

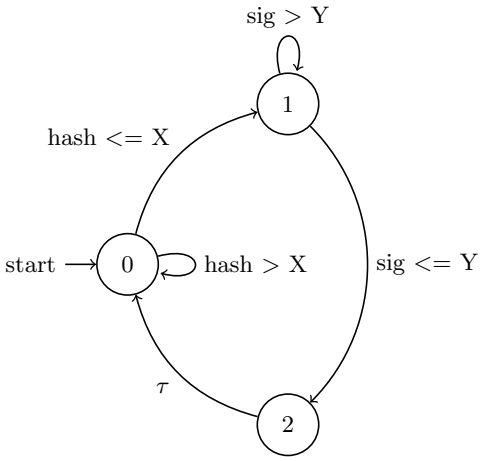

Figure 2.

A lot has been written on Markov Chains, both discrete[18] and continuous[16][17]. In this document these concepts are used within PRISM, a model checking tool. Various case studies have been published for this tool, most notably on generic stochastic model checking[12] and more specific on continuous-time Markov chains[15].

4. APPROACH

This document proposes to use Markov Chains[8], to simulate the average reward for pools given a certain difficulty for X and Y, and a ratio $\frac{X}{Y}$ in a pool's mining equipment. Markov chains are well-defined and researched in literature, which allows us to build on previous work of the scientific community.

To build this model PRISM[10][11] will be used. This software is well-documented, widely used and supports an array of probabilistic models as well as analysis. Furthermore, it allows a wide range of export methods which allows further analysis in other tools such as Matlab. Other tools such as `pymc` have been considered, but deemed too poorly documented in comparison to PRISM. No previous models have been found, so building a new model seemed justified.

4.1 Continuous-time Markov chains

As each of the miner's transitions is probabilistic and can most suitably be expressed as *rates*, a model is well resembled by *Continuous-time Markov chains*[15].

A discrete-time Markov chain is a mathematical model which exists of a certain set of states in which the system can exist. The system can only move from one state to another if a *transition* exists which allows it. The system may transition from one state to multiple others, each with a certain probability. However, its behaviour may not be dependend on previous behaviour of the model, which is called the Markov property.

Continous-time Markov chains allow the system to spent some time in a state. For each transition an expected amount of time to transition is given. A CTMC thus has an exponential distribution. Again, its behaviour does not depend on past behaviour.

Within PRISM a CTMC can easily be constructed by defining some variable, for example `a: [0..2] init 0`. The model has three states in total, for each possible value of the integer `a`. It's trivial to see the real number of possible states is equal to one as no transitions are defined.

Transitions can be defined however, by specifying them using *labels*, *guards*, *rates* and *modifiers*. These are defined as follows:

```
[label] guard -> rate:modifier;
```

For example:

```
[y_found] x = 0 -> 0.2:(x'=3);
```

In natural language, it says 'if `x` equals zero', 'transition with an expected rate of once every five seconds' and then 'modify variable `x` to hold value three'. The rates have a exponential distribution, $e^{\lambda t}$. The label instructs PRISM to only transition if and only if all other modules can transition using the same label.

Each additionally defined module is added to the state space using parallel composition. If we consider three such modules with each 2, 5 and 7 states, respectively, the total number of states is equal to $2{\cdot}5{\cdot}7 = 70$. PRISM can reduce the state space when using action labels. For example:

```
module a
  x: [0..2] init 0;
  [label_a] x = 0 -> 0.2:(x'=1);
  [label_b] true -> (x'=2);
endmodule

module b
  y: [0..2] init 0;
  [label_b] y = 1 -> 0.2:(y'=2);
  [label_a] true -> (y'=1);
endmodule
```

PRISM would consider this model to have 3 states. This, because `x` and `y` can not possibly have different values. In order for `x` to become 1, y needs to become the same value due to the label `label_a`. In a similar fashion, both can only be equal to two at the same time. Omitting a label results in an always-valid transition.

4.2 PRISM queries

PRISM can serve as a simulation engine emitting a random path through a model, but it can also derive formal conclusions using *properties*. Two of the most important properties are P and S. The former is used to derive the probability of an event's occurrence, while the latter is used to reason about its behaviour in the long-run (or *equilibrium*).

In this document only the S property is used, which syntax is `S bound [ prop ]`. Depending on the type of bound, it can return a boolean answer or a quantitative one. The latter one is used in this document. For example:

```
S=? [ x = 0 ]
```

answers the question: *what is the long-run probability of x being equal to zero?* This answer, of course, yields nonsensical results if the model does not converge to a single state.

4.3 Model

To analyse the various properties discussed in the previous sections an accurate computer model is needed. Bitcoin mining is a pretty simple process when only considering one miner or pool. Figure 2 shows a state diagram of a Bitcoin miner in its most simple form. A miner generates a hash (using data + nonce). If it does not match the target set by the network, it tries again with a different nonce. If it does, it moves to state 1, and runs a similar process for its Y challenge. Upon solving an Y challenge, the miner is rewarded by the network. A miner can exist in three states: 0, 1 and 2 associated with "No hashes solved", "X hash solved" and "Y signature solved" respectively. A miner reaching state 2 earns a globally set number of bitcoins.

Each transition has rates associated with it which depend on the difficulty set by the network and the pool's hashrate. The chances follow a similar logic for both X and Y, while both are essentially the same operation. The target for difficulty $D \in [target_1, \infty)$ is equal to $\frac{\texttt{0xffff}\cdot 2^{208}}{D}$. That is, `0xffff` bitshifted to the left 208 times (see: definition *target 1*). Therefore the expected amount of hashes is $E(H) = \frac{2^{256}}{\frac{\texttt{0xffff}\cdot 2^{208}}{D}} = \frac{2^{256}D}{\texttt{0xffff}\cdot 2^{208}}$. The expected number of time steps thus is defined as $\frac{E(H)}{hashrate}$.

A single instance of a miner as shown in figure 2, is defined as a *module* in PRISM. Local variables are used to store the state of the pool, in agreement with figure 2. Each state transition has a rate associated with it.

4.4 Composition

One problem which immediately arises is the need for pools to communicate their status with other pools. That is, once a block is found other miners should work upon this new block instead of the one before it. All miners which have already found an X hash should drop their work, to prevent block racing. Block racing (two miners finding a block at the same time) in itself however can be safely ignored by the model, as this only happens once every ten days[4].

The model used in this document mitigates block racing by requiring each pool to synchronise on actionlabels, an example of which is given in figure 3. It shows a state diagram of Bitcoin mining pool (`ghash`), in context of `eligius` and `antpool`. It defines an *active* actionlabel `p_ghash_y` and *passive* actionlabels `p_antpool_y` and `p_eligius_y`. As each pool synchronises on these labels, it is not possible to simultaneously end up in state 2. Note that no label is defined upon finding an X solution, as pools do not need to synchronise on *that* label.

Like said, every pool defines its own label `p_poolname_y` ($1 \rightarrow 2$) and applies a rate to it, thereby creating an *active* label. For every other pool a *passive* label is added for transition $1 \rightarrow 3$. A passive label does not include a rate, but functions as a synchronisation method.

Only one pool has to be defined explicitly; all others can be built using a process called renaming. PRISM's renaming engine is a simple text based search-and-replace algorithm, which thus allows for renaming everything including action labels.

An example of such a model is given in subsection 4.5. As writing these models is quite a tedious task for a large amount of pools, these models are generated by Python code (see section 8).

4.5 Example

To further clarify the manner in which a model is built, this subsection implements a *CTMC* based upon real-world parameters. For the sake of brevity it will only cover the traditional X thresholds and consider the Y threshold

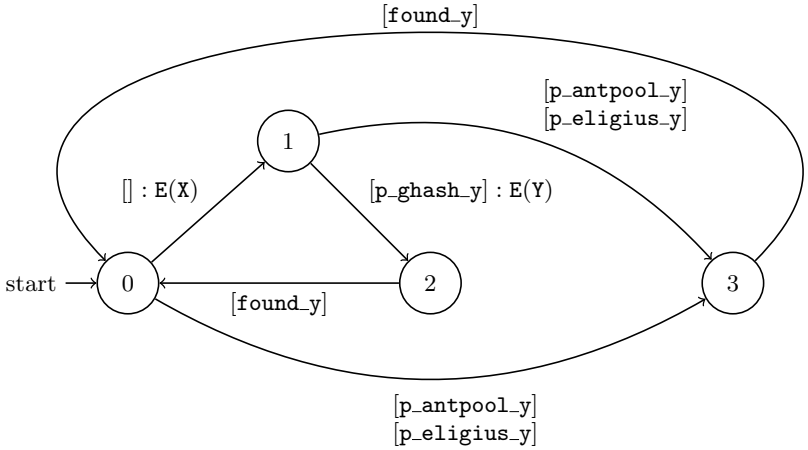

Figure 3.

Scope	Property	Value
Network	Hashrate	290.562.134 GHz s^{-1}
Network	Difficulty	35.985.640.265
Pool A	Hashrate	$0.25 \cdot 290.562.134$ GHz s^{-1}
Pool B	Hashrate	$0.18 \cdot 290.562.134$ GHz s^{-1}

Table 1. Values are based upon a snapshot of the Bitcoin network in December 2014.

to be infinite.

For each pool the expected amount of time it takes to find a block needs to determined, based upon the difficulty of the network and hashrate of the pool. For that, the expected number of hashes is needed. This works out to be:

$$E(H) = \frac{2^{256} \cdot 35.985.640.265}{\texttt{0xffff} \cdot 2^{208}} = 1.55 \cdot 10^{20}$$

The expected time to find a block for a specific pool is equal to the expected number of hashes, divided by the hashrate of the pool. For pool A this is equal to:

$$E(H_A) = \frac{1.55 \cdot 10^{20}}{0.25 \cdot 290.562.134 \cdot 10^9} = 2228\,\text{s}$$

These calculations suggest pool A finds a block approximately every three quarters of an hour. These results are expected as pool A has a hashrate equal to one fourth of the global hashrate. One might notice a slight disparity in hashrate and difficulty, which is caused by difficulty being recalculated every two weeks while the global hashrate is based on a rolling window.

Every PRISM model starts with a model type definition, in this case a `ctmc`. Then, as hashrates are constants for this work's purposes, expected times to find a block are defined.

```
ctmc // Define model as CTMC

const double a_x = 1.0/2228;
const double a_y = 1.0;
const double b_x = 1.0/2955;
const double b_y = 1.0;
```

Expected times are defined as *rates*, that is "expected transitions per second". Each pool is defined as a separate module, which are composited in a parallel fashion by PRISM. Firstly, each pool has four states, according to figure 3.

```
module a
  a_state : [0..3] init 0;
```

Pool A finds an X solution with rate `a_x` as defined above. That is, pool A moves from state zero to one with rate `a_x`.

```
  [] a_state = 0 -> a_x:(a_state'=1);
```

From state one it can either move to state three (some other pool has found a Y solution), or move to state two (it has found a block itself). Of course, if another pool found a block it should move to state three regardless of its own state.

```
  [a_found_y] a_state = 1 -> a_y:(a_state'=2);
  [b_found_y] true -> (a_state'=3);
```

As long as other pools synchronise on `a_found_y` and move to state three, it is not possible for multiple pools to end up in state two simultaneously. Only transitioning to state zero remains, which is added in the following way:

```
  [found_y] a_state = 3 -> (a_state'=0);
  [found_y] a_state = 2 -> (a_state'=0);
```

Pool B can be defined in a similar manner, which would manually writing this model a tedious job. Renaming, as mentioned in section 4.4, makes this process less cumbersome.

```
module b =
  a[
    a_state = b_state,
```

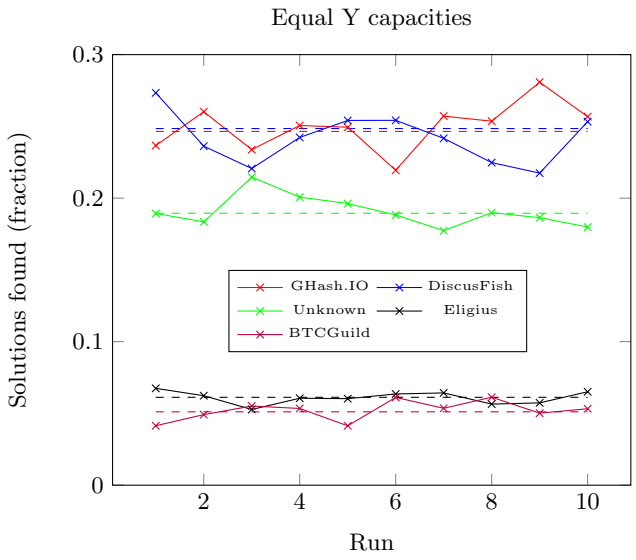

Figure 4.

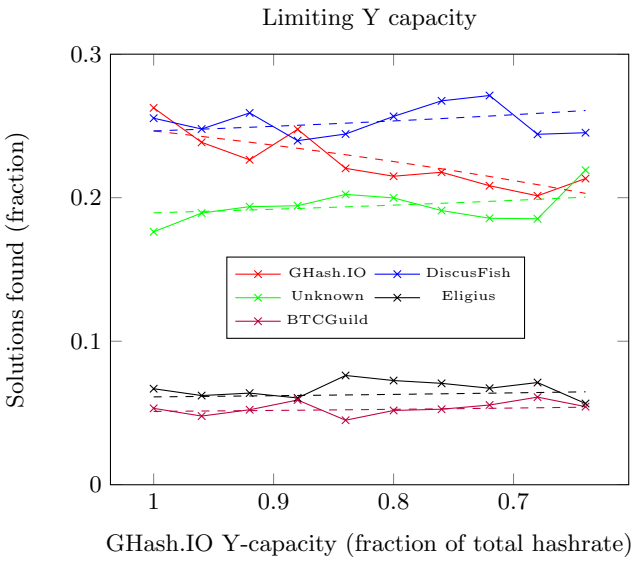

Figure 6.

```
    a_x = b_x,
    a_y = b_y,
    a_found_y = b_found_y,
    b_found_y = a_found_y
  ]
endmodule
```

This extends naturally to more than two pools, as only more actionlabels need to be defined in the first model. The renaming scheme stays the same for any number of pools, however.

The complete model is attached as appendix D.

5. RESULTS

This section covers the results for each tested hypothesis. It will first establish a basic analyses of the model provided in section 4.3, in which it is simulated using PRISM after which it is formally analysed.

All runs are based on real-world values (see appendix B). That is, they were taken from blockchain.info in Juli 2014. This allows a "gut-feeling" for results. Found results are applicable for every other state of the Bitcoin network as the global difficulty is self-regulating and aiming for the same situation.

5.1 Basic analysis

The first basic analysis consists of a simulation (run) of the model described in section 4.3. Each run consists of three-thousand "steps", which corresponds to a runtime of around three months in real time. Figure 4 shows ten runs with all pools having the same Y-solving capacity. A slight variance can be noticed, but overall the values seem to match the values in appendix B. This variance (or *jitter*) is also visible in real-world data as shown in figure 5.

This figure shows a history of pool sizes ranging from December 2013 through 2014. Jitter caused by the variance in rewards is clearly visible. Each point represents the end of an interval of 432 blocks (72 hours). The data was generated by software developed for this paper, see appendix C

With these simulated values in mind, we expect to find similar values when formally derived. In order to do so, the model needs to be changed slightly by removing the transition `found_y` or lines 20 and 23 in appendix D. The ability to simulate has now been lost, but the model has gained terminal states which can be used to efficiently analyse it. The chances of a pool ending up in state 2 for one run, is exactly equal to the chance of solving an Y-challenge while simulating. These finite states can also be interpreted as its *long-run* or *equilibrium* states.

For these cases PRISM has defined an S operator, which can be used in queries made against the model. The query used for each pool is:

```
S=?  [ p_POOL_state=2 ]
```

These values are displayed in figure 4, as a straight dotted line. (The theoretical value for GHash.IO has been slightly decreased for representation purposes.)

Because the model is defined as a CTMC, each transition is defined using rates. That is, an expected time it takes for it to be used. Rates entail an exponential statistical distribution. In turn, some "jitter" is expected in the simulated data due to randomness of transitions. This is indeed what figure 4 shows.

This "jitter" should not only be present in the simulated data, but it should also be present in real data. Figure 5 does indeed show this behaviour for intervals of 72 hours.

5.2 Limited Y capacity

Limiting a pool's Y-capacity should surely result in a decline of its market share, as it diminishes the chance of a pool's `y_found` transition. This is shown in 6, with once again dashed lines representing theoretical values, as computed by PRISM.

5.3 Parameter sweep

The previous sections (4.3, 5.1, 5.2) have argued for the validity of both the model with and without its terminal states. With these results, a parameter sweep - as briefly mentioned in section 4 - would prove a useful tool for anyone willing to implement 2P-PoW. It would answer the most prominent question, which is: how big will the

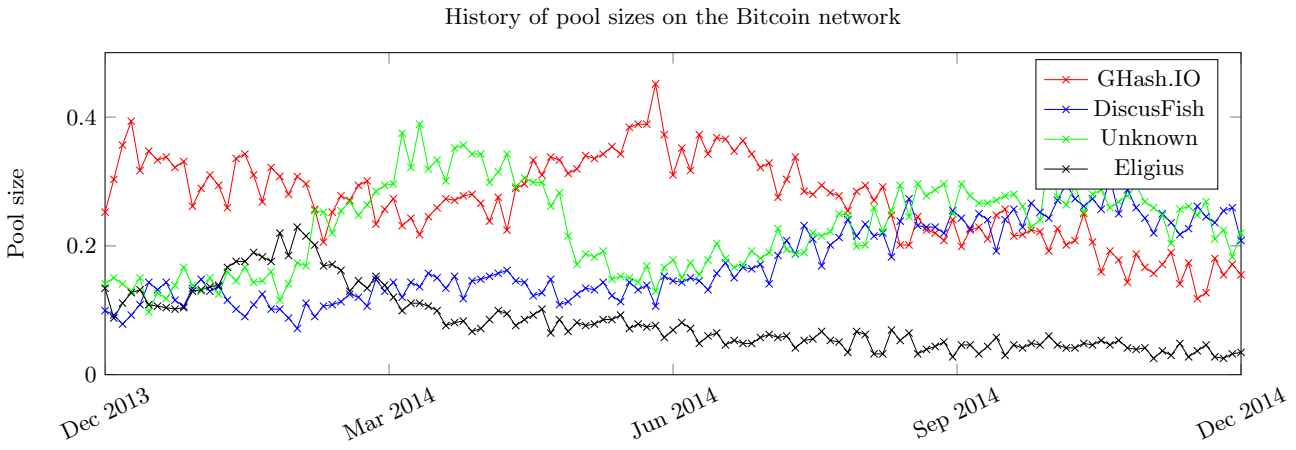

Figure 5.

biggest pool become upon choosing ratio A, while assuming large pools can only dedicate B% of their hardware to Y challenges?

It is generally believed[7] *GHash.IO* owns only 25% of its mining capacity, which puts it at a mere 6% of the network's hashrate, according to the model values used so far. A good case can be made for smaller pools owning more of the hardware themselves. Firstly, the investment needed is a lot smaller and small pools might be more tightly knit, allowing more trust. Furthermore, after deploying 2P-PoW, relative large miners might detach from formerly large pools - staring their own.

The parameter sweep therefore assumes small pools can (in the long run) dedicate 80% of their capacity to an Y solution. Because the internal capacity of large pools is unknown, the parameter sweep will consider both the X/Y difficulty and the internal capacity of large pools.

For large pools a variable considered in the parameter sweep is its Y-ratio. An Y-ratio of 1.2 indicates a relative difficulty with respect to the X-difficulty of 1.2 times. That is, in terms of hashes (or signatures) per second, it takes an expected 1.2 times more hashes for a solved Y-challenge than for an X-challenge.

The results are shown in figure 7. The x-axis (Y-ratio) depicts the difficulty of the Y-challange, relative to the difficulty of the X-challange. A ratio of 1.2 and an X-difficulty of 100 would thus mean a Y-difficulty of 120. The y-axis contains the fraction of hardware owned, and therefore dedicatable to Y-challanges, by large pools. On the z-axis the fraction of the network's hashrate *for the largest pool* is shown. Detailed model parameters are enumerated in appendix B.

It clearly shows a sharp decline in the size of the largest pool upon assuming a lower values for owned hardware. Choosing higher values for the difficulty of Y also yields a reduction in the size of the largest pool.

6. CONCLUSION

In section 2 the outstanding issues and hypotheses concerning 2P-PoW were articulated. Through the use of continuous-time Markov chains a model was built in subsection 4.3 and 4.4, and further clarified using real-world numbers in subsection 4.5.

To gain confidence in the model built section 5.2 and 5.3 provided some basic analysis of the model and compared

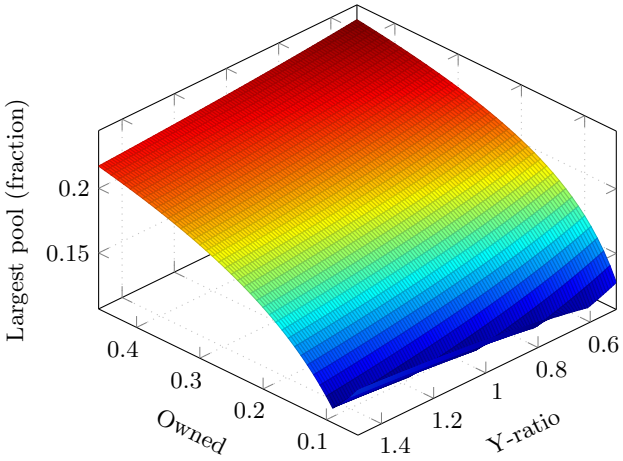

Figure 7.

it to real-world data and expectations. This indeed confirmed our trivial hypothesis that pools shrink upon limiting their Y-capacity.

To conclude this report the model was run for a range of values, the so-called parameter sweep. This produced a 3D plot which can be used by implementers to determine which values for X/Y they should consider.

The results provide an adequate answer to the first subquestion stated in section 2. The second question remains, which has not been answered due to time constraints. Furthermore, the results do not indicate any validation of the model, which is based on real-world data. Of course, it is mentioned that at a glance it *seems* likely the "jitter" seen in both datasets has an equal source (and thus an exponential distribution), but no concluding proof is provided.

6.1 Hypotheses

Section 2 posed two hypotheses, which can both be confirmed based on the models built in section 4.

The first hypothesis predicted a linear fall in a pool's market share upon toughening the Y challenge. This can be seen in figure 7 along the x-axis, where the Y challenge toughens relative to the X challenge. Of course, this can easily be deduced by reason alone. Consider a single miner, with λ and α defined as rates for its X and

Y challenge, respectively. The challenges need to be executed serially, that is, they are dependent on one another. Thus, the total rate is equal to $\lambda \cdot \alpha$. Now, upon reducing α linearly, $\lambda \cdot \alpha$ also drops in the same way and therefore its market share.

The second hypothesis allows similar reasoning. Consider again the rates α, λ and $\lambda \cdot \alpha$. It is trivial to see a pool's market share does not drop if the capacity for X and Y are swapped, as multiplication has a symmetrical property.

6.2 Mining hardware

After the parameter sweep was completed a literature study was planned, to roughly indicate the rate of miner equipment development. This was planned to allow implementers estimate the time-frame in which the Y challenge could slowly increase in difficulty. Due to time shortage, however, this part has been postponed for further study.

7. FUTURE WORK

This document builds on the excellent contributions of Eyal and Sirer[7], concerning two phase proof of work in Bitcoin. Incorporating it in the current protocol is obviously future work, but apart from this three issues remain.

Firstly, a case study on the development of mining hardware is needed in order to figure out how fast the Y-target should decrease while allowing miners to catch up. This corresponds to the second subquestion posed in section 2, which fell out of scope of this document due to time shortage.

Secondly, the X/Y ratio should ideally be chosen by the network in a similar fashion as the target is now. The network might respond to large pools by automatically increasing Y difficulty.

Thirdly, one could further validate the model by proving the real-world data does indeed follow an exponential distribution.

8. SOFTWARE

All software used in this study is open-source or developed specifically for this study. Once this document becomes publicly available, all software is published to GitHub[6] and shared under an MIT licence.

[6] `https://github.com/martijnbastiaan/twophase`

9. REFERENCES

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Modeling bitcoin contracts by timed automata. *CoRR*, abs/1405.1861, 2014.

[2] W. Beukema. Formalising the bitcoin protocol: making it a bit better. 2014.

[3] BitcoinWiki. Weaknesses.

[4] blockchain.info. Number of orphaned blocks.

[5] N. T. Courtois. Bitcoin: A peer-to-peer electronic cash system. May 2014.

[6] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. 2013.

[7] I. Eyal and E. G. Sirer. How to disincentivize large bitcoin mining pools, 2014.

[8] C. J. Geyer. *Introduction to Markov Chain Monte Carlo*. ?

[9] E. Heilman. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner. *IACR Cryptology ePrint Archive*, 2014:7, 2014.

[10] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. pages 441–444, 2006.

[11] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *Computer performance evaluation: modelling techniques and tools*, pages 200–204. Springer, 2002.

[12] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.

[13] D. Liu and L. J. Camp. Proof of work can work. 2006.

[14] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[15] G. Norman, D. Parker, M. Kwiatkowska, S. K. Shukla, and R. K. Gupta. Formal analysis and validation of continuous-time markov chain based system level power management strategies. In *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International*, pages 45–50. IEEE, 2002.

[16] J. R. Norris. Continuous-time markov chains i. In *Markov Chains*, pages 60–107. Cambridge University Press, 1997. Cambridge Books Online.

[17] J. R. Norris. Continuous-time markov chains ii. In *Markov Chains*, pages 108–127. Cambridge University Press, 1997. Cambridge Books Online.

[18] J. R. Norris. Discrete-time markov chains. In *Markov Chains*, pages 1–59. Cambridge University Press, 1997. Cambridge Books Online.

Appendices

A. DEFINITIONS

Bitcoin is a rather new concept with a sometimes confusing nomenclature. To prevent some of the confusion, its terms and concepts are summarised here.

MH, GH, TH Mega-, giga- and terahash respectively. It is often used in combination with "per second". For example, 10 TH/s means 10^{12} hashes per second.

Public ledger A public set of (economic) transactions, forming a overview of all accounts' balances.

Transaction A signed section of data containing input transactions and output "account numbers". Input transactions need at least one output belonging to the signature's private key. The sum of values of the input transactions need to be greater than or equal to the sum of values of outputs.

Transactions may also hold random data, or validation scripts implemented in a non turning-complete language called SCRYPT.

Transaction block or *block* for short. A collection of transactions with an accompanying proof-of-work hash and nonce, produces by miners. Each block includes a *reward*, which comes in the form of bitcoins. *It is widely accepted to write transaction block as a singular word.*

Bitcoin A network of peers with a formed consensus of an append-only public ledger. Each node holds a complete copy of the ledger. It is simultaneously the name of the first application of the network: a digital currency.

BTC / bitcoin A token (of monetary value) exchanged within the network. This unit has the symbols ₿, ฿, and Ƀ associated with it. In current implementations, this unit can be divided up to a hundredth of a millionth BTC.

Proof-of-work A cryptographic challenge proving an investment of CPU cycles (see: mining hardware). In Bitcoin this is implemented as an SHA256 hash needing to be *below* a certain threshold. Keep in mind this entails a *decreased* threshold actually *increases* the difficulty of the challenge.

Target A number between 1 and $2^{256} - 1$ indicating the maximum value of a valid hash value. A decreased target entails an increased difficulty.

Target 1 An arbitrary value used to calculate the *difficulty* (see: difficulty). It is defined to be 0x00000000 followed by 56 times F. In its truncated form it is defined to be 0xFFFF $\ll 52$, where $\ll$ is a bitshift operator. The exact form is often used in mining hardware, while the approximation is stored as a floating point number and used within Bitcoin clients.

Difficulty A global value which is recalculated every 2016 blocks, aiming for an average block release time of ten minutes. It is defined to be $\frac{target\_1}{current\_target}$.

Node A node in the Bitcoin network relaying transactions, and tracking transaction blocks. A node can, but does not have to be a *miner*.

Miner A special node performing proof-of-work calculations on freshly formed transaction blocks. A hash satisfying the difficulty allows a miner to transmit a block to the network,

Coinbase address A Bitcoin address which receive reaped *rewards*, mined by miners.

Coinbase transaction A special transaction without input-transactions, used to collect a miner reward.

Mining hardware At the start of Bitcoin, mining was purely done by using CPUs. In a later phase mining was executed on GPUs, while at the moment only application specific integrated circuits (ASIC) are profitable.

X / Y Both are targets the final hash needs to meet (see: target). They represent the target of the primary and secondary cryptographic challenge respectively. X is the traditional SHA256 challenge, whereas Y is the new cryptographic challenge which requires the private key of the coinbase address.

Hashrate The number of hashes a miner can calculate per time slot. This is generally expressed as hashes per second.

B. PARAMETERS

X values used in every run, based on real-world values. GH stands for 10^9 hashes per second.

Pool	X (%)	X (GH/s)
Ghash.IO	0.25	72640534
DiscusFish	0.25	72640534
Unknown	0.19	55206805
Eligius	0.06	17433728
BTCGuild	0.05	14528107
1BX5YoL	0.04	11622485
1AcAj9p	0.04	11622485
Polmine	0.03	8716864
Slush	0.02	5811243
KnCMiner	0.02	5811243
P2Pool	0.02	5811243
BitMinter	0.01	2905621
CloudHashing	0.01	2905621
Antpool	0.01	2905621

C. POOLS

To compare the model with real-world values, section 5 used the size of pools over a certain time-frame with a fixed-size window. At the time of writing, no public data could be found about the historical pool sizes. Of course, this information resides in the blockchain itself so there is ample room to analyse it.

Blocks themselves do not contain any information about its origins, however. It is up to the pools themselves to reveal their identity. Luckily, a lot of pools either identify themselves by:

1. Putting their pool name into the `txin`-field of the coinbase transaction. As this is a transaction which creates bitcoins "out of thin air", it is allowed to put random data in it.
2. Having a stable coinbase address. Some addresses have already been deanonymised by the Bitcoin community.

Both methods have been used to identify pools. The two largest pools could be recognised by their coinbase addresses:

```
1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY: Discus Fish
1CjPR7Z5ZSyWk6WtXvSFgkptmpoi4UM9BC: Ghash.IO
```

The other pools which were recognised by looking for their names in `txin` are: Antpool, Stratum, BTCChina, Eligius, KnCMiner, BTCGuild, Bitminter and Slush.

D. PRISM EXAMPLE

```
ctmc

const double a_x = 1.0/2228;
const double a_y = 1.0;

const double b_x = 1.0/2955;
const double b_y = 1.0;

module a
  a_state : [0..3] init 0;

  // Find X solution
  [] a_state = 0 -> a_x:(a_state'=1);

  // Find Y solution
  [a_found_y] a_state = 1 -> a_y:(a_state'=2);
  [b_found_y] true -> (a_state'=3);

  // Y solution found, revert to state 0.
  [found_y] a_state = 3 -> (a_state'=0);

  // We found a Y solution, revert to state 0.
  [found_y] a_state = 2 -> (a_state'=0);
endmodule


module b =
  a[
    a_state = b_state,
    a_x = b_x,
    a_y = b_y,
    a_found_y = b_found_y,
    b_found_y = a_found_y
  ]
endmodule
```

E. 51% ATTACKS

It is often unclear which attacks an attacker can perform once it gets a hold of more than 50% of the network's capacity. This appendix lays out these various attacks, which are cited[3]. An attacker *can*:

- Reverse transactions that he sends
- Prevent transactions from gaining any confirmations
- Prevent other miners from mining any valid blocks

An attacker *cannot*:

- Reverse other people's transactions
- Prevent transactions from being sent at all
- Change the number of coins generated per block
- Create coins out of thin air
- Send coins that never belonged to him

These attacks are online valid while the attacker is in control (i.e., holds 51%). Averted transactions can be included just fine after the attacker loses its majority.