Information propagation in the Bitcoin network

Conference Paper · September 2013

DOI: 10.1109/P2P.2013.6688704

2 authors, including:

Christian Decker
Blockstream Inc.
13 PUBLICATIONS **630** CITATIONS

SEE PROFILE

Information Propagation in the Bitcoin Network

Christian Decker,* Roger Wattenhofer†

*ETH Zurich, Switzerland cdecker@tik.ee.ethz.ch

†Microsoft Research wattenhofer@microsoft.com

***Abstract*—Bitcoin is a digital currency that unlike traditional currencies does not rely on a centralized authority. Instead Bitcoin relies on a network of volunteers that collectively implement a replicated ledger and verify transactions. In this paper we analyze how Bitcoin uses a multi-hop broadcast to propagate transactions and blocks through the network to update the ledger replicas. We then use the gathered information to verify the conjecture that the propagation delay in the network is the primary cause for blockchain forks. Blockchain forks should be avoided as they are symptomatic for inconsistencies among the replicas in the network. We then show what can be achieved by pushing the current protocol to its limit with unilateral changes to the client's behavior.**

I. Introduction

Bitcoin is the first truly decentralized global currency system. Like any other currency, its main purpose is to facilitate the exchange of goods and services by offering a commonly accepted good. Unlike traditional currencies however, it is not issued by a state or even a single authority.

Since its inception in late 2008, Bitcoin has enjoyed a rapid growth, both in value and in the number of transactions. Its success is mostly due to innovative use of a peer-to-peer network to implement all aspects of a currencies lifecycle, from creation to its transfer between users. This is the fundamental difference from previous research, which concentrated on building systems that rely on either a centralized issuer [5], [16], [18] or creating inter-user credit [9]. These systems required users to trust the original issuer, which was still used to eventually clear transactions.

Bitcoin has often been compared to cash as transactions are near-instantaneous and non-refundable. However Bitcoin goes beyond the scope of cash, allowing truly global transactions, processed at the same speed as local ones. It offers a public transaction history and it introduces many new and innovative uses such as smart properties, micropayments, contracts and escrow transactions for dispute mediation.[1]

Bitcoin is slowly growing into becoming a possible alternative to the US Dollar or the Euro as more and more businesses start accepting bitcoins for their products and services. The fact that Bitcoin is still around indicates that the underlying principles are sound. Nevertheless, there is some room for improvement.

The main problem Bitcoin sets out to solve is the distributed tracking and validation of transactions. For this, the network needs to reach a consensus about the balances of the accounts it tracks and which transactions are valid. Bitcoin achieves this goal with guarantees which are best described as eventual consistency: the various replicas may be temporarily inconsistent, but will eventually be synchronized to reflect a common transaction history.

As transactions are validated against the replica states, any inconsistency introduces uncertainty about the validity of a given transaction. Furthermore, an inconsistency may jeopardize the security of the consensus itself. This may facilitate an attacker that attempts to rewrite transaction history.

In this work we analyze Bitcoin from a networking perspective, i.e., how information is disseminated or propagated in the Bitcoin network, we identify key weaknesses as well as the resulting problems. In particular, we analyze the synchronization mechanism which fails to synchronize the information stored at the ledger with a non-negligible probability. This problem not only causes a prolonged inconsistency that goes unnoticed by a large number of nodes, but also weakens the system's defenses against attackers. We then propose some changes to the current protocol that, while not a solution to the intrinsic problems of the communication model used by Bitcoin, may mitigate them.

II. Bitcoin

In this section we give a general overview about Bitcoin, adding the details that will be needed later in this paper. Depending on the context, the name Bitcoin may refer to any of the following three parts of the Bitcoin ecosystem:

- Bitcoin, the system: the abstract protocol first introduced by Nakamoto in the original publication [14];
- bitcoins or BTC, the currency unit;
- bitcoind, the reference implementation. Written by Nakamoto as a proof-of-concept implementation, bitcoind still remains the most used Bitcoin client.

In this work we focus on the system and its protocol, in particular on how information is disseminated on the network. In Bitcoin two distinct types of information are disseminated: *transactions* and *blocks*. Transactions are the primitives that allow the transfer of value, whereas blocks are used to synchronize state across all nodes in the network.

Unlike traditional currencies, like the US Dollar or the Euro, Bitcoin does not rely on a centralized authority to control the supply, distribution and verification of the validity of transactions. Bitcoin relies on a network of volunteers, to

[1] See https://en.bitcoin.it/wiki/Contracts

978-1-4799-0521-8/13/$31.00 ©2013 IEEE

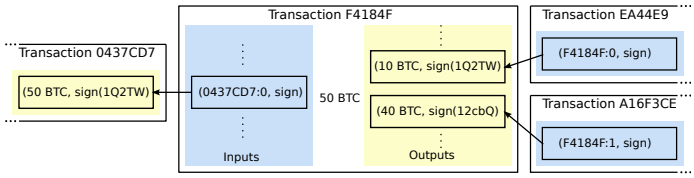

Fig. 1. The first real transaction F4184F. It claims the 50 bitcoins output from transaction 0437CD7 and creates two outputs of 10 and 40 bitcoins respectively. Those outputs are then later claimed by transactions EA44E9 and A16F3CE.

collectively implement a replicated *ledger*. The ledger tracks the balance of all accounts in the system. Each node keeps a complete replica of the ledger. It is crucial for the replicas of the ledger to be in a consistent state across all nodes at all times as the validity of transactions is verified against them.

A. Transactions

At an abstract level a transaction transfers bitcoins from one or more *source accounts* to one of more *destination accounts*. An account is in essence a public-/private-keypair.[2] An address derived from the public key is used to identify the account. To transfer bitcoins to an account a transaction is created with the address of the account as destination. To send bitcoins from an account, the transaction has to be signed with the private key associated with the sending account.[3]

Instead of aggregating the balance of each account, the ledger tracks *outputs* that transferred the bitcoins to the account. An output is a tuple of a numeric value in bitcoins and a condition to claim or spend that output. Hence, the balance of an account is the sum of the numeric values of all unspent outputs for that account.

Transactions are identified by the hash of their serialized representation (tx message). A transaction claims some outputs by providing a proof of ownership. The references to the claimed outputs along with the proofs of ownership form what is called an *input* to the transaction. The transaction may then specify one or more new outputs as destination.

Outputs are the fundamental unit of information that is tracked in the ledger and their status has to be consistent across all replicas. For a transaction to be valid the following criteria must be fulfilled by the outputs they claim and create:

- An output may be claimed at most once;
- New outputs are created solely as a result of a transaction;
- The sum of the values of the claimed outputs has to be greater or equal than the sum of the values of the newly allocated outputs.

As transactions are broadcast through the network the state of the ledger replicas changes. When a node receives a new transaction, it is verified and committed to the local replica. Over time the various replicas of the ledger at different nodes may become inconsistent:

- A node might receive a transaction that transfer coins from an account, but it did not yet receive the transaction that made those coins available to the account;
- Two or more transactions might attempt to transfer the same coins multiple time. This is called a *double spending attack*.

Double spending attacks have a direct impact on the consistency of the ledger replicas. During a double spend, whether intentional or by mistake, two or more transactions attempt to simultaneously claim the same output. The real world equivalent of double spending attacks would be a user that submits multiple transactions to her bank, spending the available balance multiple times. While in this case the double spend attempt would be recognized by the bank and would not result in a transfer, in Bitcoin this contradiction is harder to resolve. A node receiving the first transaction will verify it and commit it to its ledger replica. When the node later receives the other transactions, the validation fails as the output has already been spent. As there is no guarantee that all nodes receive the conflicting transactions in the same order, the nodes will disagree about the validity of the conflicting transactions and any transaction that builds on top of them by claiming their outputs.

B. Blocks

In order for the ledger replicas to remain consistent a common order over the transactions has to be agreed among the nodes. Agreeing on a common order of transactions in a distributed system is not trivial. Bitcoin solves this problem by tentatively committing transactions and then synchronizing at regular intervals by broadcasting a *block* created by one of the nodes. A block b contains the set of transactions $\mathcal{T}_b$ that the node which created the block has committed since the previous block. The block is then distributed to all the nodes in the network and each node receiving it will roll back the tentatively committed transactions since the last block and apply the transactions from the current block.

At this point all the nodes have agreed on the validity of all the transactions in the block. Transactions that were committed as part of the block are confirmed and do not have to be reapplied. The transactions that have been rolled back will then be validated again and reapplied on top of the new base state. Transactions that are now invalid because they conflict with transactions committed as part of the block are discarded.

The node that created the block effectively imposes its view of the changes since the previous block. However, the decisions of the block creator are limited. The node cannot forge any transactions as long as the underlying public-/private-key cryptosystem is secure. The block creator may only decide in which order transactions arrived and whether to include transactions in its block.

To determine which node may impose its view the nodes attempt to find a solution to a proof-of-work [7] with a given probability of success. The proof-of-work consists in finding a byte string, called *nonce*, that combined with the block header

[2]Bitcoin currently uses ECDSA for the signatures.

[3]The described method to send bitcoins to an account and claiming them by providing a signature is only one of the possible scenarios. We limit the description to this method as it is the most commonly used method.

results in a hash $\mathcal{H}_b$ with a given number of leading zero-bits, or *target*. As cryptographic hashes are one-way functions, finding such a nonce can only be done by actually calculating the hash of the block for all possible nonces until a valid solution is found. It is therefore difficult to find an input that produces a solution, but straight forward to verify it. The nonce is part of the block so that nodes receiving it can verify that the creator solved the proof-of-work. The hash $\mathcal{H}_b$ is also used as the block's identity. The target is determined via consensus by all nodes in order to achieve an average of one result every 10 minutes in the entire network and is adjusted every 2016 blocks, which should occur once every 14 days in expectation.

Nodes attempting to find a solution to the proof-of-work are often called *miners*. To incentivize miners, the node finding a block receives a reward in the form of newly minted bitcoins, i.e., it may include a transaction that has no inputs but may specify outputs for a predetermined number of coins into the block. This reward transaction is only valid if it appears in the block and is the only exception to the rule that the sum of a transaction's outputs has to be smaller or equal to the sum of the transaction's inputs.

C. Blockchain

Up to this point, blocks do not provide any added synchronization on top of the individual transactions. This changes when the blocks are chained together, creating a chronological order over the blocks and therefore about the transactions contained within them.

The blocks are organized in a directed tree. Each block contains a reference to a previously found block. The block b referenced by a block b' is called its *parent*. The transitive closure of this relation gives its *ancestors*. The root block in the tree is the *genesis block*, which is hardcoded into the clients. This block is an ancestor of all blocks by definition.

The *blockchain* is defined as the longest path from any block to the genesis block. The distance between a block b and the genesis is referred to as its *block height* h_b. The genesis block g therefore has height $h_g = 0$. The block with maximal height, i.e., the block that is furthest away from the genesis block is referred to as *blockchain head*, with height h_{head}. We use the notation $\mathcal{B}_h$ to reference the set of blocks at height h.

Since to include a reference to the parent block, that parent block's identity (its hash) has to be known, the child block must have been found after the parent. The chaining is used to assign a chronological order to the transactions: transactions in lower height blocks have been verified before transactions in higher blocks.

As only blocks appearing in the blockchain will be rewarded with newly minted coins that are accepted by other users, miners will always attempt to find a block that builds on the current blockchain head. Building on an earlier block would require the resulting branch to become longer than the currently longest branch, i.e., the blockchain, to be rewarded.

D. Blockchain forks

From the definition of blockchain directly follows that there can be multiple heads at a time, i.e., when $|\mathcal{B}_h| > 1$ with $h = h_{\text{head}}$. This situation is called a *blockchain fork*. During a blockchain fork the nodes in the network do not agree on which of the blocks in $\mathcal{B}_h$ is the current blockchain head.

Two blocks $b, b' \in \mathcal{B}_h$ are guaranteed to disagree about the current state of the ledger, because they both introduce a reward transaction. Hence, a blockchain fork implies that the system is no longer consistent.

When a node, whose blockchain head b_h is at height h, receives a block $b_{h'}$ for height $h' > h$ it switches its blockchain head to this block. The new block $b_{h'}$ may either be on the same branch as b_h, i.e., b_h is an ancestor of $b_{h'}$, or on another branch.

Should block b_h be on the same branch as the newly found blockchain head $b_{h'}$ it will retrieve all intermediate blocks on the branch and apply their changes incrementally. On the other hand, should $b_{h'}$ be part of another branch, i.e., b_h is not an ancestor of $b_{h'}$, then they share a common ancestor. Since $b_{h'}$ is on a longer chain than b_h it becomes the new blockchain head, therefore the node has to revert all changes down to the common ancestor and apply the changes in the branch of $b_{h'}$.

A blockchain fork may be prolonged by the partitions of the network finding more blocks $\mathcal{B}_{h+1}, \mathcal{B}_{h+2}, \ldots$ building on their respective blockchain heads. Eventually one branch will be longer than the other branches, and the partitions that did not adopt this branch as theirs will switch over to this branch. At this point the blockchain fork is resolved and the ledger replicas are consistent up to the blockchain head. The blocks discarded by the blockchain resolution are referred to as *orphan blocks*.

Bitcoin never commits a transaction definitively. Every transaction can be invalidated if a longer chain that started below the block including the transaction is created. If a single entity could control a majority of the computational power on the network, and thus be able to find blocks faster than the rest of the network combined, it could revert any transaction. If an attacker attempts to revert a transaction that was included in block b_h it would create a new transaction that conflicts with the original transaction and include it into a block $b_{h'}$ with $h' < h$. The attacker would then proceed to create blocks on top of $b_{h'}$ until this new chain overtakes the original blockchain and thus becomes the new blockchain.

One may argue that the existence of blockchain forks is the very reason that transactions are never definitively committed. The tight coupling between blocks and the validity of a transaction not only slows down the confirmation time of a transaction but also limits the confirmation to be a probabilistic statement about the validity.

III. Information Propagation

The Bitcoin network is a network of homogeneous nodes. There are no coordinating roles and each node keeps a complete replica of all the information needed to verify the validity of incoming transactions. Each node verifies information it receives from other nodes independently and there is only minimal trust between the nodes.

A. Network topology

By construction the nodes in the network form a random graph. When a node joins the network it queries a number of DNS servers. These DNS servers are run by volunteers and return a random set of bootstrap nodes that are currently participating in the network. Once connected, the joining node learns about other nodes by asking their neighbors for known addresses and listening for spontaneous advertisements of new addresses. There is no explicit way to leave the network. The addresses of nodes that left the network linger for several hours before the other nodes purge them from their known addresses set. At the time of writing approximately 16000 unique addresses were advertised, of which approximately 3500 were reachable at a time.

Each node attempts to keep a minimum number of connections p to other nodes open at all times. Should the number of open connections be below p the node will randomly select an address from its set of known addresses and attempt to establish a connection. On the other side, incoming connection are not closed if they result in the number of open connections to be above the pool size p. The total number of open connections is therefore likely to be higher for nodes that also accept incoming connections.

We observed that a node running bitcoind which accepts incoming connections, has an average of 32 open connections. This greatly exceeds the default connection pool size of $p = 8$. On nodes that are not reachable, due to either being behind a network address translation or a firewall, the number of simultaneous open connections never exceeded p.

Partitions in the connection graph are not actively detected, and should they occur the partitions will continue operating independently. While this is certainly desirable from a liveness point of view, the state tracked in the partitions will diverge over time, creating two parallel and possibly incompatible transaction histories. It is therefore of paramount importance that network partitions are detected. Such detection could be done by tracking the observed aggregated computational power in the network. A rapid decrease in the block finding rate might indicate that a partition occurred.

B. Propagation Method

For the purpose of updating and synchronizing the ledger replicas only transaction (*tx*) and block (*block*) messages are relevant. These messages are far more common than any other message sent on the network and may grow to a considerable size. In order to avoid sending transaction and block messages to nodes that already received them from other nodes, they are not forwarded directly. Instead their availability is announced to the neighbors by sending them an *inv* message once the transaction or block has been completely verified. The inv message contains a set of transaction hashes and block hashes that have been received by the sender and are now available to be requested. A node, receiving an inv message for a transaction or block that it does not yet have locally, will issue a *getdata* message to the sender of the inv message containing the hashes of the information it needs. The actual transfer of the block or transaction is done via individual block or tx messages. Figure 2 visualizes the protocol flow for a single hop in the broadcast. Node A receives a block, verifies it and announces it to its neighbors. Node B receives the inv message and, since it does not know about the block, it will issue a getdata message. Upon receiving the getdata message, Node A will deliver the block to Node B.

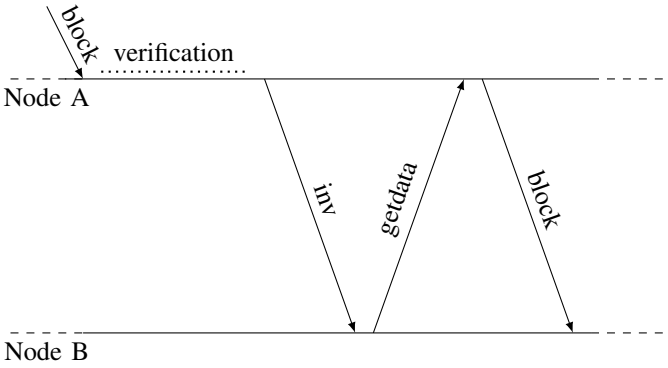

Fig. 2. Messages exchanged in order to forward a block message a single hop from Node A to Node B.

Each block or transaction is introduced to the network at one of the nodes, its *origin*, and is then propagated throughout the network using the above broadcast mechanism.

At each hop in the broadcast the message incurs in a *propagation delay*. The propagation delay is the combination of transmission time and the local verification of the block or transaction. The transmission time includes an announcement in the form of an inv message, a request from the receiving party and a delivery. While the inv and the getdata messages are relatively small in size (61B in most cases, as immediate broadcasts only contain a single block or transaction being announced), the block message may be very large — up to 500kB at the time of writing. Before the block is announced to the neighbors of a node, it is verified. The verification of a block includes the verification of each transaction in the block. Transaction verification in turn requires random access to data stored on discs.

Let $t_{i,j}$ be the time difference between the first announcement by the origin to the network and the time at which node j receives the item i. If node o is the origin of the data item i, i.e., either the finder of the block or the node that created the transaction, then $t_{i,o} = 0$.

The times $t_{i,j}$ at which nodes learn about the existence of a data item follow a double exponential behavior. Similar to randomized rumor spreading [12], the propagation of a data item can be divided into two phases: an initial exponential growth phase in which the most of the nodes receiving inv messages will request the corresponding data item as they do not have it yet, and an exponential shrinking phase in which most of the nodes receiving an announcement already have the corresponding data item.

To measure the propagation delay we implemented the bitcoin network protocol and connected to a large sample of

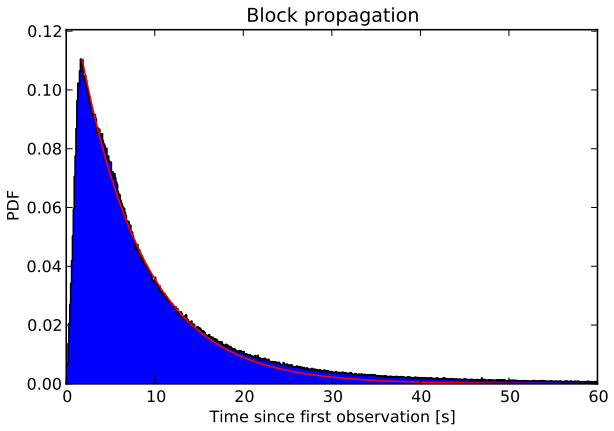

Fig. 3. The normalized histogram of times since the first block announcement with fitted exponential curve.

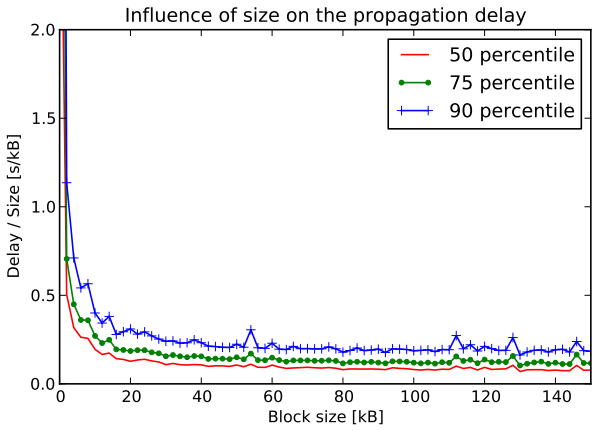

Fig. 4. Delay costs for the 50, 75 and 90 percentile. The plot is focused on the lower y-range to show the constant behavior after 20kB.

nodes in the network. Our implementation behaves exactly like a normal node with one caveat: it does not relay inv messages, transactions or blocks. It tracks how transactions and blocks are propagated through the network by listening for the announcement of their availability in the form of inv messages. Once the measuring node receives an inv message containing the reference to a block we know that the node which sent the announcement has received and verified the block.

The measuring node collected timing information from blockchain height 180'000 for 10'000 blocks. The timing information contains the hash of the block, the announcing nodes IP and a local timestamp when the announcement was received. An estimate for the $t_{i,j}$ is given by subtracting the timestamp of the first announcement of a block from all announcements for that data item.

Figure 3 shows the normalized histogram of $t_{b,j}$ for all blocks b in the measured interval. The normalization allows us to use this as an approximation of the probability density function of the rate at which nodes learn about a block. Notice that we do not differentiate between the blocks' sizes and instead aggregate over all blocks. The median time until a node receives a block is 6.5 seconds whereas the mean is at 12.6 seconds. The long tail of the distribution means that even after 40 seconds there still are 5% of nodes that have not yet received the block.

C. Size Matters

There is a strong correlation between the size of a message and the propagation delay in the network. The *delay cost* is defined as the time delay each kilobyte causes to the dissemination of a transaction or block. Notice that the cost is a combination of both verification and transmission time. Figure 4 plots the costs for the 50, 75 and 90 percentile for various sizes. For sizes larger than 20kB the cost can be said to be constant, whereas for small sizes there is a considerable overhead. The overhead for small sizes is caused by the roundtrip delay, i.e. the fact that even small messages are announced via an inv message and then retrieved via a getdata message. The roundtrip delay is dominant for transactions as 96% of all transactions are smaller than 1kB. For blocks, whose size is larger than 20kB, each kilobyte in size costs an additional 80ms delay until a majority knows about the block. It would therefore be sensible to forward transactions directly, and thus avoiding the overhead of the added roundtrip. However the same cannot be said for blocks where the overhead does not contribute as much to the overall time to disseminate.

D. Information Eclipsing

So far we have discussed how information is propagated in the case that the information is not contradicting. Another important part in the dissemination of information in the Bitcoin network is the visibility of information. When a node receives a new block or transaction, that it deems invalid, possibly because it contradicts information it received earlier, it will ignore it and not forward it.

Let us consider the case of a block being disseminated in the network and how it may lead to a blockchain fork that is only detected by a minority of the nodes.

Let $G = (V, E)$ be the network's underlying connection graph, V being the set of all nodes and E the set of connections between the nodes. Starting from a single partition $P_h \subset V$ containing all nodes whose blockchain head is at height h, i.e., they do not know any block for the next height $h + 1$. Finding a new block b_{h+1} introduces a new partition $P_{h+1,b}$ which contains the nodes that believe this block to be the head, i.e., it is the first block for height $h + 1$ they received. If no other block is found, then nodes adjacent to the cut between P_h and $P_{h+1,b}$ leave P_h and join $P_{h+1,b}$ until P_h is empty and the network as a whole proceeds with the new blockchain height $h + 1$.

On the other hand, should another block b'_{h+1} for height $h + 1$ be found by a node in P_h, it again introduces a new

partition $P_{h+1,b'}$. In this case nodes from P_h will join $P_{h+1,b}$ and $P_{h+1,b'}$ concurrently until P_h is empty, and all nodes are in one of the partitions with height $h+1$.

Only nodes adjacent to the cut between $P_{h+1,b}$ and $P_{h+1,b'}$ will know both b and b' and therefore able to detect the resulting blockchain fork. Nodes that are in the partition $P_{h+1,b}$, and not adjacent to $P_{h+1,b'}$, will only know b and be completely oblivious to the existence of a conflicting block. A partition $P_{h+1,b}$ could potentially contain only a single node, in the case that the node's neighbors already know a conflicting block and immediately stop the propagation of b.

The above also applies for transactions that are being propagated. If two transactions that attempt to spend the same output are propagated in the network only the first transaction a node receives will be deemed valid, the second transaction will be invalid according to that node's state and will therefore not be announced to its neighbors.

This behavior has the advantage that a malicious node may not flood the network by issuing hundreds of contradicting transactions with no additional cost, in the form of fees, to the malicious node. On the downside this very behavior makes double spend attacks that are invisible to the merchant [11] possible.

In the case of transactions, stopping the propagation is a reasonable trade off, that protects the network from transaction spam, at the expense of individual users. However, in the case of blocks, stopping the propagation is not reasonable. The blockchain forks, that are hidden from a majority of the nodes by doing so, are an important indicator of an ongoing unresolved inconsistency. As valid, but potentially conflicting blocks, cannot be created at an arbitrary rate like transactions, forwarding them would not create the possibility of an attack.

IV. Blockchain Forks

In this section we focus on the block propagation and the blockchain forks that occur in the network. We show that blockchain forks are caused by the long propagation time by presenting a model that matches the observed blockchain fork rate.

A. Observing Blockchain forks

Some blockchain forks may be observed by participating in the network and receiving the two conflicting blocks. Observing all blockchain forks however is difficult. If a node detects that an incoming block conflicts with the block it believes to be the chain's head, then it will not propagate the block any further.

Recall that the partitions in a blockchain fork may have size 1. As a direct result, faithfully reporting all blockchain fork would require being connected to every node in the network. Due to some nodes not being reachable, either because they are behind a firewall or network address translation, only an approximation of the actual number of blockchain forks can be given.

Using the implementation from Section III we collected the blocks that have been propagated in the network between height 180'000 and height 190'000. We are confident that due to our large sample, which includes all reachable nodes, nearly all the found blocks have been propagated to us, allowing us to identify close to all blockchain forks that occurred in the measurement interval.

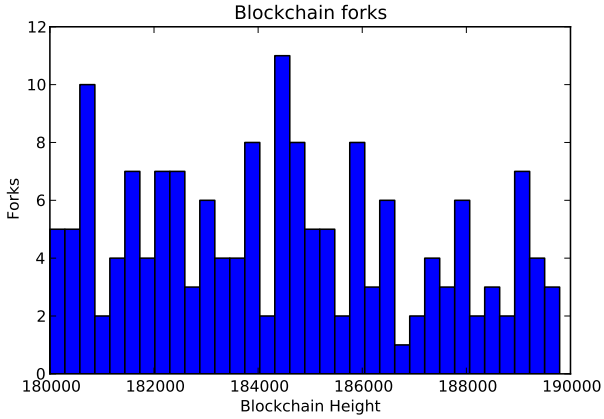

Fig. 5. Histogram of blockchain forks for 10'000 blocks starting at height 180'000, observed while participating in the network.

Figure 5 shows the histogram of blockchain forks in the collected blocks. There were 169 blockchain forks in the observed 10'000 block interval, resulting in an observed blockchain fork rate $r = 1.69\%$.

B. Model

The proof-of-work causes valid blocks to be found independently at random. Since blocks are found independently at random by the participants in the network, a block might be found while a conflicting block is being propagated in the network. We claim that blockchain forks are caused by the block propagation delay in the network.

1) Probability of finding a block: The bitcoin protocol adjusts the difficulty of the proof-of-work required to find a block so that in expectation one block is found every 10 minutes.

If X_b is the random variable of the time difference in seconds between a block being found and its predecessor being found, then the probability of a block being found by the network as a whole in any given second is

$$P_b = Pr[X_b < t+1 | X_b \geq t] \approx 1/600 \quad (1)$$

2) Part of the network that could find a conflicting block: A blockchain fork occurs if, during the propagation of a block b, a conflicting block b' is found. Such a block b' may only be found by the part of the network that does not yet know about b.

Let t_j be the time in seconds at which node j learns about the existence of b since it has been found. Let the $I_j(t)$ be the indicator function whether node j knows about b at time t. Let $I(t)$ be the indicator function that counts the number of

informed nodes, i.e., the nodes that have received and verified block b, at time t.

$$I_j(t) = \begin{cases} 0 & t_j > t \\ 1 & t_j \leq t \end{cases}$$
$$I(t) = \sum_{j \in V} I_j(t)$$

Then the ratio of informed nodes is

$$f(t) = \mathbb{E}[I(t)] \cdot n^{-1}$$

Notice that $f(t)$ is equivalent to the cumulative distribution function (CDF) of the rate at which peers are informed. We may therefore use the PDF of the rate at which peers are informed from Figure 3 as an estimate during the measurements.

Only the uninformed nodes may produce a conflicting block. Combining the probability of finding a block and the ratio of nodes that is uninformed we derive the probability of a blockchain fork. Let F be a discrete random variable that counts the number of conflicting blocks being found while another block is being propagated, then the probability of a blockchain fork is:

$$Pr[F \geq 1] = 1 - (1 - P_b)^{\int_0^\infty (1 - f(t))dt} \quad (2)$$

Notice that this last step requires the simplifying assumption that the probability of node finding a block is distributed uniformly at random among all nodes.

Hence, knowing the probability of the entire network to find a block P_b and the distribution of how the nodes learn about the existence of the block allows to derive the probability of a blockchain fork. P_b and the distribution of the I_j depends on the computational power in the current network and the topology and size of the network.

C. Measurements

To validate the model we compare the resulting probability of a blockchain fork with the observed rate of blockchain forks.

1) Probability of finding a block: Each block includes a timestamp of the time the block was found. As nodes do not synchronize clocks, but rather sample the current time of their neighbors, there is a non-trivial clock skew in the timestamps.

In some cases the clock skew is quite pronounced, producing even impossible constellations. For example block at height 209873 in the blockchain has a timestamp of 22:10:13 whereas the following block at height 209874 has a timestamp for 22:08:44. Since the latter includes the hash of the former, the blocks were found in the correct order. Thus the conflicting timestamps may only be caused by the non-synchronized clocks.

As alternative, because we are participating in the network and have a large sample of the nodes, we may also use the time we first saw the block being announced to the network as the time the block was found. While this does not suffer from clock skew, it may have a small delay between the block being found and the first announcement reaching the measurement node and we only have the timestamps the blocks that were found while actively measuring the network.

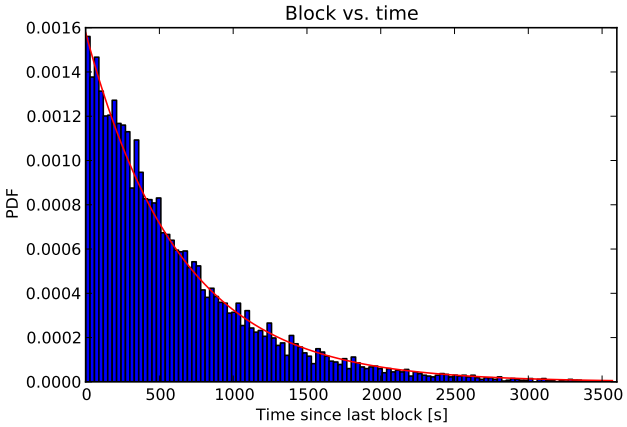

Fig. 6. Shifted time difference distribution for blocks being found between height 180'000 and 190'000.

The proof-of-work is a Poisson process, therefore the time difference follows an exponential distribution. The convolution of the random clock skew and the time between blocks being found in the timestamp of the block causes a right shift of the maximum. This can be corrected by a left shift until the maximum is at $t = 0$. The announcement time observed while measuring does not suffer from the clock skew and directly produces the correct histogram.

$$g(t) = \lambda e^{-\lambda \cdot x}$$

By extracting the timestamps from the blocks at height 180'000 through 190'000 we get the distribution shown in Figure 6. By fitting the observed distribution to the exponential distribution we find a value for $\lambda = 0.001578$ and therefore an expected time between two blocks of $1/\lambda = 633.68$ seconds. By fitting the probability density of the time between first announcements from the measurements we find a value $\lambda = 0.001576$ resulting in an expected time between two blocks of $1/\lambda = 634.17$. The two approximations of λ are consistent, but both are slightly above the targeted value of 600 seconds. The difference is most likely due to a decrease in computational power in the network.[4]

2) Block propagation in the network: Due to the normalization the histogram in Figure 3 also represents the probability density function (PDF) of the random variables $t_{b,j}$ for all blocks b in the measurement interval. Hence, the ratio of informed nodes $f(t)$ is the area under the histogram up to time t.

Combining the above probability of finding a block and the function for the informed ratio results in the following

[4]See http://bitcoin.sipa.be/ for June 2012

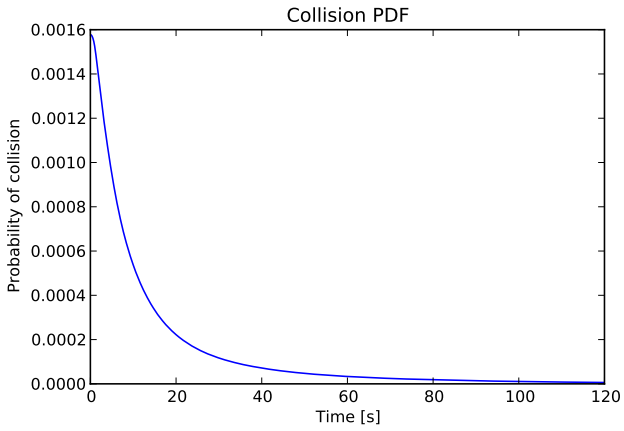

Fig. 7. Probability density function of a conflicting block being found while another block is being broadcast.

probability for a blockchain fork:

$$\begin{aligned} Pr[F \geq 1] &= 1 - (1 - P_b)^{\int_0^\infty (1-f(t))dt} \\ &= 1 - \left(1 - \frac{1}{633.68}\right)^{11.37} \\ &\approx 1.78\% \,. \end{aligned} \tag{3}$$

According to our model the probability of a blockchain fork is therefore 1.78%. Comparing this result to the observed blockchain fork rate of 1.69% in Section IV-A we observe that we overestimate the observed fork rate by only 5%. The slightly higher predicted probability is possibly due to the assumption that the computational power is uniformly distributed over all nodes in the network. However, the good quality of the fit suggests that the model is a good match for the reality.

Because the number of transactions and the size of the network is likely to grow as the adoption of Bitcoin as a payment method picks up the rate of blockchain forks is bound to increase. A larger network, with the random topology and the fixed connection pool size increases the diameter and the average distance between the nodes and the origin of a block. An increase in the number of transactions causes a growth in the block size which in turn increases the verification delay and the transmission delay at each hop in the propagation.

An alternative interpretation of the result in eq. (3) is that each time a block is found, the equivalent of 11.37 seconds worth of computational power of the entire network is wasted. Work, i.e., attempts to find a proof-of-work solution, that goes into building alternative blockchain heads does not contribute to extend the blockchain, making it potentially easier for an attacker to overtake the current blockchain head with an alternative chain of its own. Nakamoto [14] already anticipated that an attacker with more than 50% of the computational power in the network would be able to find proof-of-work solutions faster than the rest of the network. The attacker would therefore be able to eventually replace the transaction history from an arbitrary point in time. While certainly sufficient, the condition is not strict, as our result shows. In reality the efficiency of the network as a whole, including a propagation delay, is not optimal. This inefficiency may give a prospective attacker that can reduce the delay a considerable advantage.

The effective computational power in the current network is $1 - 11.37/633.68 = 98.20\%$. Therefore, a 49.1%, share of the computational power in the network is enough for an attacker to eventually revert any transaction under current conditions. While even this is hard to achieve, increasing propagation delay may further weaken the network as a whole.

V. Speeding up the propagation

In the previous section we have shown that the way information is propagated in the network causes blockchain forks. In this section we explore what the limits of the existing protocol are and whether unilateral changes in the nodes behavior can change the blockchain fork rate. There are several ways to improve the propagation of information in the network:

- Minimize verification
- Pipelining block propagation
- Connectivity increase

Limiting the changes to the ones that can be enacted in an unilateral way allows us to assess their effectivity without major changes to the protocol, which would have to be vetted and accepted by the Bitcoin community.

A. Minimize verification

A major contributor to the propagation delay is the time it takes a node to verify a block before announcing it to the network. There is a strong correlation between the size of a block and the time to verify it. As each hop in the propagation has to verify the block before relaying it to its neighbors the delay is multiplied by the length of the propagation paths.

Currently there is a block size limit of 500kB per block enforced by bitcoind, but this is likely to be relaxed more and more as the average block size grows, so that it may include more transactions.

The first insight is that the verification can be divided into two phases:

- An initial difficulty check;
- A transaction validation.

The difficulty check consists of validating the proof-of-work by hashing the received block and comparing the hash against the current target difficulty. Additionally, it checks that the block is not a duplicate of a recent block and that it references a recent block as its predecessor to verify that the block is not a resubmission of an old block. The bulk of the validation is done in the transaction validation which checks the validity of each transaction in the block. The block can be relayed to the neighbors, as soon as the difficulty has been checked and before the transactions have to be verified.

Therefore the behavior of the node could be changed to send an inv message as soon as the difficulty check is done, instead of waiting for the considerably longer transaction validation to be finished.

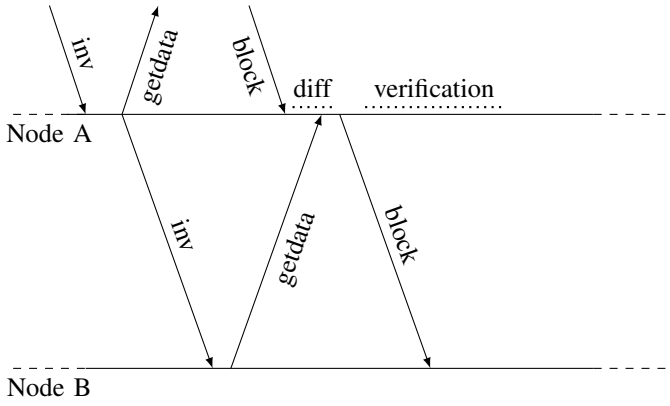

Fig. 8. Message exchange after the behaviour described modifications. Compared to Figure 2 there is a notable difference. The *inv* message is forwarded immediately and the verification is split into two, speeding up the propagation.

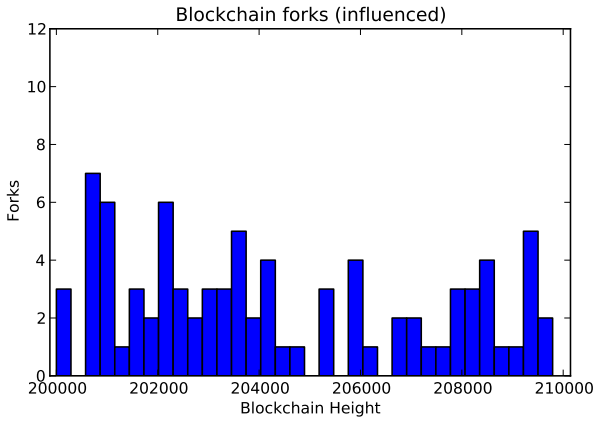

Fig. 9. The histogram of detected blockchain forks while influencing the propagation in the network from blockchain height 200'000 to 210'000.

Any change to the behavior of nodes in the network has to be vetted against the potential for being misused by an attacker to harm the network. In particular relaying information that has not been validated might allow an attacker to send arbitrary amounts of data that is then relayed, overwhelming some nodes in the network and resulting in a distributed denial of service attack.

This change does not increase the risk for a denial of service attack as producing an invalid block that passes the difficulty check is just as hard as producing a valid block with the same difficulty. On the downside this change is unlikely to have a large impact on the overall propagation delay if it is implemented only by a single node that is not well connected. It speeds up a single hop on the path from the origin to the nodes.

B. Pipelining block propagation

A further improvement can be achieved by immediately forwarding incoming *inv* messages to neighbors. The goal of this is to amortize the round-trip times between the node and its neighbors by preemptively announcing the availability of a block earlier than it actually is. The incoming *getdata* messages for the block are then queued until the block has been received and the above difficulty check has been performed, then the block is sent to the neighbors requesting it. Unlike the first change, this may cause some additional messages being sent as from the hash of the block no validation can be done. An attacker may announce an arbitrary number of blocks without being able to provide them when asked for it. Nodes receiving these spam announces will relay them to their neighbors. Should a node detect that one of its neighbors is announcing blocks that it cannot provide it can switch back to the original behavior of first verifying blocks before announcing them.

Even though nodes can be tricked into forwarding inv messages that it cannot provide the block for, the impact is likely to be small as the inv messages have a small constant size of 61B. Note that the same attack is already possible by creating an arbitrary amount of transactions and announcing them to the network. As the attacking node can provide the matching transaction, it will not be recognized as an attack.

Figure 2 and Figure 8 show the changes in the behavior. Node A is the node whose behavior has been altered. Notice the verification being divided into two phases (*diff* and *verification*) and the inv message being sent much earlier.

Again this speeds up a single hop and is unlikely to result in a large improvement if implemented only by single node in the network.

C. Connectivity increase

The most influential problem is the sheer distance between the origin of a transaction or a block and the nodes. To minimize the distance between any two nodes we attempted to connect to every node in the network creating a star sub-graph that is used as a central communication hub, speeding up the propagation of inv messages, blocks and transactions.

We instructed our implementation to keep a connection pool of 4000 connections open. This caused it to connect to every single advertised address, as fewer than 4000 nodes were reachable at any time.

The result is that the distance between any two nodes the hub connected to is close to 2.

D. Measurements

The above changes were implemented in our client and tested from blockchain height 200'000 to 210'000. During this time the client was connected to an average of 3048 nodes in the network and uploaded 20.5 million block messages. For each block the node received an average of 2048 requests.

Figure 9 shows the histogram of blockchain forks while participating in the network with the modified client. Comparing it to the unmodified case shown in Figure 5 a clear improvement is visible. The overall effect of the changes was that the blockchain fork rate dropped from 1.69% to 0.78%, i.e., a 53.41% improvement over the unmodified case.

As mentioned before, the pipelining and the verification minimization only have a small effect, which is multiplied by the last change. The last change however has high bandwidth requirements as blocks caused bandwidth spikes up to around 100 MB/s and resulted in a total upload, during the measurements, of 2.31TB raw block data.

VI. Related work

Although relatively young as a system, Bitcoin has sparked a lot of interest in many research areas. The topics that are being researched include the legal [1], the economic [4] and the technical aspects of Bitcoin.

The problem of double spending has been addressed in the original paper by Nakamoto, but only theoretically. Karame et al. [11] have an in depth analysis of the probability of a double spending attack to succeed in several scenarios. While they do mention the possibility of a double spend that cannot be detected simply by a longer detection period, we introduce the concept of information eclipsing, which causes this problem. Bamert et al. [3] propose some mitigations to the double-spending problem in fast payment scenarios.

Babaioff et al. [2] analyzed the incentives for nodes to forward information at all in the network and found that they are insufficient. A dominating strategy in the current system is for a miner to hold on to transactions that include fees, and claim them by eventually creating a block that includes the transaction.

Bitcoin mining often requires specialized equipment and consumes wast amounts of energy. Becker et al. [4] analyze the ecological impact of Bitcoin as a currency system compared to traditional currencies. Their conclusion is that while the fees to send a Bitcoin transaction are small, actually maintaining and securing the network against takeover is expensive. As we have shown the amount of computational power in the network is likely to be underestimated.

Another highly debated topic is the anonymity of Bitcoin transactions. The fact that all transactions are tracked in a replicated ledger and that the details of the transactions are therefore accessible by any participant in the network would suggest that privacy is not possible. However, Nakamoto claims that, since the identities of the owner of an account and the identity of the account are kept separate, the privacy can be said to be pseudonymous. Reid et al. [15] analyze this claim and point out that by colluding the information of multiple accounts that participated in a transaction details about the owner can in fact be recovered. Shamir et al [17] analyzed the transaction graph, deriving some global statistics, including an estimate that 78% of the issued bitcoins are not circulating, and an in depth analysis of a highly active region in the transaction graph. Elias [8] discussed some legal, and moral, aspects of the anonymity, or lack thereof, in Bitcoin.

The anonymity problem in Bitcoin was later addressed by ZeroCoin [13] which allows the implementation of a Zero-Knowledge based decentralized coin mixing service. Earlier Hanke et al. [10] presented a Pay-to-Contract Protocol that is built on top of Bitcoin and secures transactions between merchants and their clients. CommitCoin [6] is another system that builds on the blockchain to carbon date commitments.

VII. Conclusion

In this paper we analyzed how information in the Bitcoin network is disseminated in order to synchronize the ledger replicas. The reliance on blocks not only delays the clearing of transactions, but it also poses a threat to the network itself. Large blocks are propagated slowly in the network, giving an attacker an advantage.

We introduce a model that explains the existence of blockchain forks, and corroborate the model by matching it to our observations. As blockchain forks are symptomatic for an inconsistency in the ledger replicas, it is important that the nodes in the network are aware about them. However, due to information eclipsing, most nodes are unable to detect them.

Finally, we implemented and measured some changes to the Bitcoin protocol that reduce the risk of a blockchain fork. Our measurements show that a single node implementing these changes reduces the number of blockchain forks in the network by over 50%. The root cause of the problem however is intrinsic to the way information is propagated in the network. The changes may mitigate the problem in the short term, until a scalable long term solution is found.

References

[1] Bitcoin virtual currency: Unique features present distinct challenges for deterring illicit activity. Technical report, Federal Bureau of Investigation, 2012.

[2] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On bitcoin and red balloons. In *Proc. of Electronic Commerce*, 2012.

[3] Tobias Bamert, Christian Decker, Lennart Elsen, Samuel Welten, and Roger Wattenhofer. Have a snack, pay with bitcoin. In *IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, 2013.

[4] Jörg Becker, Dominic Breuker, Tobias Heide, Justus Holler, Hans Peter Rauer, and Rainer Böhme. Geld stinkt, bitcoin auch — eine Ökobilanz der bitcoin block chain. In *BTC 2012: Workshop Bitcoin*.

[5] David Chaum. Blind signatures for untraceable payments. In *Crypto*, volume 82, page 199203, 1982.

[6] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin. In *Financial Cryptography and Data Security*. 2012.

[7] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology*.

[8] Matthew Elias. Bitcoin: Tempering the digital ring of gyges or implausible pecuniary privacy. *Available at SSRN 1937769*, 2011.

[9] Ryan Fugger. Money as ious in social trust networks & a proposal for a decentralized currency network protocol. 2004.

[10] Ilja Gerhardt and Timo Hanke. Homomorphic payment addresses and the pay-to-contract protocol. *CoRR*, abs/1212.3257, 2012.

[11] G.O. Karame, E. Androulaki, and S. Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. In *Proc. of Conference on Computer and Communication Security*, 2012.

[12] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proc. of Foundations of Computer Science*, 2000.

[13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. 2013.

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

[15] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Proc. of the Conference on Social Computing (socialcom)*, 2011.

[16] Ronald L Rivest. Electronic lottery tickets as micropayments. In *Financial Cryptography*, pages 307–314. Springer, 1997.

[17] Dorit. Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph.

[18] Beverly Yang and Hector Garcia-Molina. Ppay: micropayments for peer-to-peer systems. In *Proc. of Computer and communications security*.