Solving Byzantine Problems in Synchronized Systems using Bitcoin

Felix Sun, Peitong Duan

December 9, 2014

Abstract

We apply principles from the bitcoin online currency to the Byzantine consensus problem. Previous work used the bitcoin block mining algorithm to solve Byzantine consensus for up to $n/2$ failures with high probability. We fix a fault in the proof of this algorithm. We then generalize this algorithm to replicated state machines. We show that bitcoin principles can be used to build a replicated state machine that tolerates Byzantine failures.

1 Background

Bitcoin is a form of digital currency operating on a peer-to-peer network, which means transactions are sent directly between those involved, without going through a third party. [1] This has several advantages, including increased protection from fraud and lower transaction costs. The way the bitcoin network works is that it keeps track of all the transactions made so far and eventually confirms them. Transactions are confirmed by a process known as mining. During mining, a set of transactions from a given time period is verified, packaged into a block, and a hash of the block is generated. The hash must meet the bitcoin protocol requirements. Every time a process successfully mines a block, the block will then be shared with the entire network and added to the common blockchain containing confirmed blocks. Each process maintains a copy of this common blockchain and attempts to mine blocks on this chain.

Thus, the bitcoin algorithm is a consensus protocol, in that all processes must agree on the set of confirmed transactions. But, bitcoin is fundamentally different from other Byzantine consensus protocols, because up to half of the nodes can be faulty. Conventionally, Byzantine agreement within bounded time has been proven to be impossible if a third or more of the processes fail. There have been many efficient algorithms created that meet this optimal error bound. Feldman and Micali [4] found a Byzantine consensus algorithm that is always correct and runs in probabilistic polynomial time by making use of randomization and private information sharing. Each process randomly generates a bit and commits it to a secret sharing protocol (this protocol tolerates failures), which will generate a common bit from all the committed bits. All processes will then decide on this common bit. Furthermore, Castro and Liskov [2] constructed a Byzantine consensus algorithm that is always correct and runs efficiently in asynchronous systems, which can be used in practice. This algorithm uses replicated state machines. Each process in the network contains a copy of the state machine that executes service operations. A client issues a request to the primary process that broadcasts the request to all its replicas. Each copy executes the request and sends the result back the client who waits to hear back from a given number of processes.

The connection between Bitcoin and Byzantine agreement was previously recognized by LaViola and Miller [3]. They showed that a version of the synchronous bitcoin algorithm can be used to solve Byzantine consensus tolerating a higher error bound – it can decide as long as a minority of the processes fail.

This result, while surprising, does not contradict the $n = 3f + 1$ rule for Byzantine failure because the bitcoin algorithm does not actually guarantee correctness. Rather, there is only a high probability that the correct processes agree. In other words, the bitcoin algorithm is a "Las Vegas" randomized algorithm, which is fast, but not always correct. The Feldman and Micali probabilistic algorithm is a "Monte Carlo" algorithm, which is always correct, but does not always terminate. Monte Carlo algorithms are strictly stronger than Las Vegas algorithms, because any Monte Carlo algorithm can be converted into a Las Vegas one for the same problem (terminate early, return null), but not the other way around. Therefore, bitcoin offers a weaker solution than most of the previous literature on Byzantine consensus.

However, we discovered a flaw in the LaViola and Miller proof. Namely, it does not account for message delays when a new block is announced. The proof assumes that, when a process mines a new block, all other processes instantaneously hear about it, and can immediately mine the next block in the chain. (See section 2 for an explanation of terminology.) In reality, other processes must wait a turn to hear the announcement. This delay significantly changes the dynamics of block mining, invalidating their correctness bound.

We present a corrected version of the bitcoin consensus proof in this paper. We also frame the bitcoin algorithm as a replicated state machine algorithm, in which the processes emulate a centralized state machine, and maintain consistent state in the face of Byzantine opponents. Bitcoin consensus can be seen as a special case of Byzantine state machine replication, in which the state machine is a single decision variable. We prove that bitcoin can perform state machine replication with the same fidelity as it performs consensus.

2 The synchronized bitcoin algorithm

First, we will port the core parts of the bitcoin algorithm into the synchronized distributed model. The resulting procedure will implement replicated state machines in the presence of Byzantine failures. The bitcoin currency uses a specific state machine designed to mint and track money, but our algorithm can replicate any state machine.

The key cryptographic primitive used by bitcoin is reversal of a hash function. Guessing the value that hashes to a particular b-bit target has only a $1/2^b$ success rate. In reality, the success rate increases with the number of failed attempts because failed values can be eliminated, but we assume that the number of possible values is much larger than the number of attempts, so we can approximate non-replacement. To adjust the difficulty of hash reversal, the bitcoin protocol requires processes to guess a value whose hash contains b leading zeros. (The probability of success is still $1/2^b$ per attempt.)

In the synchronized model, we assume that a process can execute a fixed k hashes per turn. Therefore, we define a function `reverseHash(valuePrefix, nZeros)`, which finds a value that starts with valuePrefix, and that hashes into a target that starts with nZeros zeros. In any turn, a process can only run one instance of this function, and this function has a $p = k/2^{nZeros}$ chance of returning valid value.

Using the hardness of the `reverseHash` function, bitcoin creates a voting procedure, in which processes vote with their hashing power. In short, processes in the bitcoin algorithm build chains of hashes, where each link ("block") in the chain consists of the target of the previous hash, appended with some data about transactions that happened at the time the link was made. Processes consider the longest chain the correct one; they only put new invocations into the longest known chain, and they only try to `reverseHash` on the longest known chain. The result is a feed-forward effect on chain length: the longest chain attracts processes that run `reverseHash`, which creates additional blocks, which makes the chain even longer.

We assume full connectivity - this is unrealistic for actual bitcoin operation, but the theoretical consensus problem allows us to assume full connectivity.

2.1 External interface

A user of this replicated state machine can submit an invocation to any node. The invocation can be any state transition on the machine. Sometime after, it will receive an acknowledgement that the transition has either occurred or been denied. A user can also query any node to get the latest state of the machine.

Our algorithm can simulate any state machine, but for practical use in adversarial environments, the state machine should at least support cryptographic signatures on transactions. Users should sign each transaction request with their private key, and the state machine should verify the signatures before processing a transaction. Otherwise, the Byzantine nodes could create fake transactions at will. Note that a state machine with signature verification is simply a specific kind of state machine, so our algorithm is agnostic to this process.

2.2 Messages

This algorithm has several kinds of messages: Transaction messages announce that a process wants to do an invocation on the state machine. ("Alice is giving Bob bitcoin number 321" in the currency state machine, for example.) Block discovery messages announce that a new block has been discovered.

2.3 Process state

Each process p maintains a list of all blockchains that it has heard about. A blockhain is a sequence of blocks $b_1, b_2, b_3, ...$ such that $hash(b_i)$ (a) has the required number of leading zeros, and (b) is the prefix of b_{i+1}.

b_i is made of several items concatenated together. Item 1 is by necessity $hash(b_{i-1})$. Item 2 is the hash of all the transactions associated with this block.

Item 3 a variable padding value (called the "nonce" in bitcoin literature) that allows $hash(b_i)$ to have the correct number of leading zeros.

In addition, each block has associated with it a set of transactions. These transactions are hashed to make item 2 in the block. This hashing serves as a signature for the transactions, preventing transactions from being inserted after the block is made. Unlike the block hash, which is designed to be reversed, we make sure that this hash is completely irreversable, by making item 2 a large number of bits. (We could alternately put the set of transactions directly in the block, but the bitcoin protocol uses hashing to limit block size.) In an optional optimization, we could append to item 2 a hash of the current machine state, and include the current machine state in the block. This makes it more compuationally efficient to read off the lastest state, but does not affect the correctness of the algorithm.

Finally, each process maintains a queue of pending transactions that it has heard about, but is not in any block.

Block chains are allowed to branch. For example, blocks b_2 and b_3 may both be children of b_1, in the sense that b_2 and b_3 have the correct item 1. The length of a block chain is measured from leaf to root.

2.4 Taking a turn

In a turn, each node first processes any incoming messages. For each transaction message, it checks to see if the invocation can be applied to the present state of the machine. The present state is the state at the tip of the longest block chain, modified by any pending transactions at the process. If an incoming transaction is inconsistent with respect to the current state, it is discarded. Consistent transactions are added to the queue, for future hashing.

The process then deals with any incoming block discovery messages. It checks that the block discovery message is consistent. This means that the new block (1) contains the hash of the previous block in the blockchain, (2) contains transactions that are valid, with respect to previous blocks in its blockchain, and (3) has a hash with the correct number of leading zeros. Consistent new blocks are appended to the correct blockchain. To add an extra layer of security, consistent blocks are also broadcast to the network.

Note that new blocks are allowed to be inconsistant with pending transactions. For example, suppose the longest block at process i says that Alice owns 1 bitcoin. Process i has a pending request for Alice to give Bob 1 bitcoin. Then, process i learns about a new block which says that Alice already gave that bitcoin to Carol. As long as the new block is not part of the longest block chain, process i will keep the pending transaction, and continue attempting to hash it into a block.

Now, the process attempts to use `reverseHash` to make a new block. It concatenates the hash of the last block in the longest chain with a hash of all transactions in the queue, and sets this as the valuePrefix for `reverseHash`. If it suceeds in finding a suffix that results in a hash with the correct number of leading zeros, it has made a new block. (The block is simply the valuePrefix concatenated with the suffix, along with all the transactions that were hashed into the block.) It sends a block discovery message to all of its neighbors.

If the user requests any new transactions, it first announces them. Then, it adds checks them for consistency, and adds them to the queue if consistent.

2.5 Claims

Users of bitcoin make a number of claims about this bitcoin algorithm, both explicitly and implicitly by storing their money with it. All of these claims are made in the Byzantine context, in which attackers have infiltrated the network, but control less than half of the processing power. In the synchronized model, this means less than half of the nodes are defective.

The most important claim is a time bound on transactions propagating and getting verified by the system. In the real world, when a vendor receives a bitcoin, he waits for around an hour, until the block containing his transaction has been hashed, and it is obvious that this block is part of the longest block chain. Then, he can be sure that the customer did not double-spend the bitcoin. In the synchronized model, this claim translates to: Suppose a transaction is submitted at node i. Then, if that transaction is in the longest block chain at node i after $x \cdot 2^{nZeros}/k$ turns, then every node knows about and recognizes the transaction with high probability exponential in x.

We will prove this claim first for a consensus state machine, then for general state machines.

3 Solving Byzantine consensus using bitcoin

We now present a simple modification to the bitcoin algorithm that allows it to solve the consensus problem. The entire system consists of a single bitcoin. At the beginning of the algorithm, each process proposes that the ownership of that bitcoin be transferred to itself. Run the algorithm until a clear longest block chain emerges, and read off the winner.

More formally, in this instance of the bitcoin algorithm, there is one user per possible decision value. There is also a single bitcoin in the entire system. At the beginning of the algorithm, each process proposes that it owns the bitcoin. Then, in each round, each process executes a **compute** procedure, where it votes for its **preferred** (longest) blockchain by attempting to mine a block. It does so by accessing a **coinflip** instruction that will randomly generate a solution. If the solution is correct, the process has mined a block and casted a vote for this chain (each block counts as a vote). If the process is nonfaulty, it will then broadcast a block discovery message to all other processes.

Processes can also receive messages. When a process receives a block discovery message for a chain with more votes than its current preferred chain, it will set its preferred chain to this new chain.

Finally, processes must decide by round $\hat{r}$. The pseudocode for process p_i is as follows:

```
initially do
    Votes ← $\emptyset$
        Prefer ← $proposed_i$

on receive (Votes', Prefer') do
        if verify ($v$,Prefer') for all $v \in$ Votes'
                and |Votes'| > |Votes|
                Votes ← Votes'
                Prefer ← Prefer'
```

```
on compute ($r$) do
        nonce ← coinflip
        if verify (nonce, Prefer)
                broadcast (Votes∪{nonce}, Prefer)

        if $r \geq \hat{r}$
                decide Prefer
```

Termination is guranteed by the fact that all processes decide by round $\hat{r}$. Next we will show that this algorithm solves Weak Byzantine Consensus with a certain probability.

We will first make the claim that if all correct processes have mined at least one block on a total of x turns, every process must have a chain with at least x blocks. We assume there are no failures.

This claim can be proven by induction on r, the number of rounds.

Base Case: When $r = 0$, no blocks have been mined. All chains must have positive length.

Inductive Step: Assume at the start of round r, x turns have passed in which a correct process mined a block, and every process has a chain with length at least x. During round r one of the following must occur:

- No blocks are mined by any processes. In this case, only x distinct blocks have been mined. By the inductive hypothesis, the claim holds in round $r + 1$.
- At least one process mines a block. In this case, $x + 1$ mining turns have passed. Block discovery messages will be sent out in this round, so by round $r + 1$, the newly discovered block will have been added to the longest chains of all the processes. By the inductive hypothesis, each process must now have a chain of length at least $x + 1$.

This proves the claim.

Using this claim, it immediately follows that weak consensus is guaranteed as long as fewer than $2x$ blocks have been mined in total. If there exists a chain c of length x, but fewer than $2x$ blocks in all, then c is the only possible chain of length x. We know that every correct process has a chain of length x; this implies that every correct process knows c.

Now, consider a bitcoin system with n processes and $f < n/2$ Byzantine failures. We need to decide how to model the block-mining dynamics among the correct and faulty processes. In one turn, any process, correct or faulty, has a p chance of mining a block. In reality, it is possible for a single process to mine multiple blocks per turn. The process could discover a block at the beginning of the turn, and the next block in the chain later. But, to simplify the math, we will ignore this possiblity.

This means all correct processes together will mine a block with probability $p_c = 1 - (1 - p)^{n-f}$. On the other hand, all the incorrect processes together can mine multiple blocks - each process independently has a p chance of contributing a block. After $\hat{r}$ turns, the correct processes have mined blocks according to a binomial distribution: $Block_c = Binom(p_c, \hat{r})$. The incorrect processes have mined blocks according to a different binomial distrubution: $Block_f = Binom(p, \hat{r} \cdot f)$. Agreement is guaranteed as long as $Block_c > Block_f$.

We would like to approximate the likelihood of agreement. To do this, we apply a normal approximation to both binomial distributions.

$$Block_c \approx N(p_c\hat{r}, \sqrt{\hat{r}p_c(1-p_c)})$$

$$Block_r \approx N(p\hat{r}f, \sqrt{f\hat{r}p(1-p)})$$

Their difference (which we want to be greater 0 with high probability) is also normal:

$$Block_c - Block_r \approx N(\hat{r}(p_c - pf), \sqrt{\hat{r}p_c(1-p_c) + f\hat{r}p(1-p)})$$

We would like the mean to be several standard deviations above 0, which means:

$$\frac{\hat{r}(p_c - pf)}{\sqrt{\hat{r}p_c(1-p_c) + f\hat{r}p(1-p)}} > k$$

for a small integer k. ($k = 3$ gives us around 99% success.)

We can simplify this expression in the special "worst case" where $n = 2f+1$. In addition, we assume that $p << (n-f)^{-1}$, so that $p_c \approx (n-f)p = (f+1)p$. Intuitively, this means mining a block is hard enough such that it is not likely to happen on any given turn. In practice, this is a good idea - mining blocks should be slower than propagation times.

With these assumptions, our bound simplifies to

$$\frac{\hat{r}p}{\sqrt{\hat{r}(f+1)p(1-p_c) + \hat{r}fp(1-p)}} > k$$

$$\frac{\hat{r}p}{\sqrt{\hat{r}fp(2-fp-2p)}} > k$$

$$\frac{\hat{r}p}{f(2-fp-2p)} > k^2$$

$$\hat{r} > O(k^2 f/p)$$

In order to satisfy our $p << (n-f)^{-1}$ assumption, we need $p = O(f^{-1})$, so our round asymptotic growth is $\hat{r} = O(k^2f^2)$. In other words, the bitcoin consensus algorithm give e^{-k^2} likelihood of failure if it is run for $O(k^2f^2)$ rounds when there are f failures out of $2f+1$ correct processes. In general, waiting for $O(k^2n/p)$ rounds is sufficient, as long as $p = O(1/n)$.

4 Replicated state machines using bitcoin

Section 2 introduced the bitcoin algorithm in the context of replicated state machines. Users submit operations to the system, which are called "transactions". Transactions are announced to all nodes, and nodes try to hash them into new blocks. A user gets an acknowledgement if his transaction appears in the longest block chain at the user's submission node $\hat{r}$ turns after the traasaction is submitted.

Using ideas from Section 3, we can prove that, if a transaction is acknowledged, all correct processes have the transaction in their longest block chain, with high probability. More specifically, we will consider cases when process i has transaction T in its longest block chain immediately after verifying and adding all new block announcements on turn $\hat{r}$. Call this point the "decision point" of a turn. We will prove that all other correct processes also have the T in their longest block chains at their decision points on turn $\hat{r}$.

Lemma 1: if there is a turn such that no faulty processes announce new blocks to correct processes, then all correct processes will have the same blocks at the decision point of the next turn. Suppose some process x knows about a block that process y does not. What happened when x first learned about this block? It rebroadcasted the block to all other processes. So, process y must have heard about it then. Even if process x learned about this block last turn, it would have announced it last turn, and process y would know about it by the decision point of this turn. ■

Suppose that on turn $\hat{r}$, some process j does not have transaction T in its longest block chain, but process i, the originator of the transaction, does have T in its longest block chain. Then, by lemma 1, some Byzantine process must have sent a new block to at least one of i or j on turn $\hat{r} - 1$. Without loss of generality, say blocks $\{b\} = b_1, b_2, ...$ were sent to process i. Furthermore, $\{b\}$ must make the longest block chain at process i; otherwise nothing would change. The Byzantine processes, working together, must have mined $\{b\}$ at some point. But when?

The key is, there is a limited window of time when the faulty processes could have mined $\{b\}$. Call b_0 the block before b_1 (the parent of the first block in the set of blocks sent to i). The faulty processes cannot start mining $\{b\}$ until b_0 is known to the functional processes. (It is not possible to mine a block until the parent block has been mined.) Suppose b_0 is the $len(b_0)$-th block in a block chain. When b_0 is first revealed to the functional processes - either through mining or announcement by a Byzantine process - every functional process knows about a chain of length at least $len(b_0)$, namely the chain ending with b_0.

Now, in order for $\{b\}$ to become the longest block chain at i, all other block chains at i must be of length at most $len(b_0) + len(\{b\})$. We now want to reason about longest block chains among the functional processes. Let M_t be the minimum over all correct processes of the length of the longest known block chain at time t (following the notation of the Miller and LaViola paper). Call the time when b_0 was discovered $t(0)$. $M_{t(0)} \geq len(b_0)$, because every process knows about b_0. Suppose that at time $t(blockN)$ after $t(0)$, the correct processes mine $blockN$ additonal blocks. (As in the previous section, the correct processes are only allowed to add one block per turn to their total.) Lemma 2 gives us a lower bound on the longest block chain at every process.

Lemma 2: $M_{t(blockN)+1} \geq len(b_0) + blockN$. We can prove lemma 2 using induction over $blockN$. Base case: $blockN = 0$ is obviously true. Now, suppose $M_{t(blockN-1)+1} \geq len(block_0) + blockN - 1$. Now, starting at the beginning of turn $t(blockN - 1) + 1$, all correct processes will be trying to mine a block whose length in the block chain is at least $len(block_0) + blockN$, since they all know of a chain of length at least $len(block_0) + blockN - 1$ by the inductive hypothesis. Suppose one of them succeeds at time $t(blockN)$. Then, by time $t(blockN) + 1$, all processes will have heard about the new block, making their longest block chain at least $len(block_0) + blockN$ blocks long. This proves the inductive hypothesis. ■

Using lemma 2, we see that, if $\{b\}$ mined by the Byzantine processes is to become the longest chain at process i, then the correct processes had to have mined at most $len(\{b\})$ blocks in the time it took the Byzantine processes to mine $len(\{b\})$ blocks. This is the crucial constraint that makes disagreement unlikely - if the Byzantine processes are outnumbered, it is unlikely that they can keep up with the correct processes for a long period of time.

In other words, consistency is guaranteed if the correct processes mine more blocks than the faulty processes over the $\hat{r}$ turns after T is submitted. This is exactly the same correctness condition as for the consensus algorithm in the previous section. The results copy over: the probability of failure is $O(e^{-k^2})$ if we wait for $\hat{r} = O(k^2 n/p)$ turns, assuming $p << f^{-1}$.

References

[1] Nakamoto, S. (n.d.). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf

[2] Castro, M., & Liskov, B. (1999). Practical Byzantine Fault Tolerance. E Proceedings of the Third Symposium on Operating Systems Design and Implementation, 1-14. Retrieved December 9, 2014, from http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

[3] Miller, A., & LaViola, J. (n.d.). Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin. Retrieved from Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin

[4] Feldman, P., & Micali, S. (1997). An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. SIAM Journal on Computing, 26(4), 873-933. Retrieved December 9, 2014, from http://people.csail.mit.edu/silvio/Selected Scientific Papers/Distributed Computation/An Optimal Probabilistic Algorithm for Byzantine Agreement.pdf