Byzantine Fault Tolerance: Prime vs the Bitcoin Blockchain

James Charles
Sunayu LLC
james.charles@sunayu.com

ABSTRACT

The rise of global, peer-to-peer cryptocurrency networks inspired by the Bitcoin blockchain has given new relevance to the design and implementation of Byzantine fault tolerant systems. In this paper, we will describe the meaning of "Byzantine" faults, present two very different solutions to the problem—Prime and the Bitcoin blockchain—and explore their comparative strengths and weaknesses.

KEYWORDS

Byzantine, replication, fault tolerance, Bitcoin, blockchain, Prime

1 INTRODUCTION

A "Byzantine" fault is any failure or malicious action of a component within a distributed system that could prevent other components from reaching agreement, wherein such agreement is required for correct operation of the system. Byzantine fault tolerance (BFT) is a characteristic of distributed systems that can defend against these types of failures, allowing them to operate correctly in environments where certain nodes may be untrustworthy or outright malicious.

This problem was originally articulated by Leslie Lamport, Robert Shostak, and Marshall Pease as *The Byzantine Generals Problem* [1]:

> We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide on a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. The generals must have an algorithm to guarantee that:
>
> A. All loyal generals decide upon the same plan of action
>
> [...]
>
> B. A small number of traitors cannot cause the loyal generals to adopt a bad plan [1]

In the simplest case, the generals must vote to decide whether to attack or retreat. If the loyal generals attack together, they will take the city; if they retreat, they will live to fight another day. However, if a traitor can convince some generals to attack and others to retreat, then the halfhearted attack will be wiped out and the forces severely diminished. Thus, it is imperative that all loyal generals agree upon the same information before making a decision.

This problem has direct applications for distributed computer systems that need to operate correctly in environments where certain nodes may be untrustworthy or outright malicious. Even for internal applications, any system perceived as sensitive or high-value is a potential target of advanced persistent threats (APTs) and, consequently, could contain latent, untrustworthy actors at any time. In the face of these threats, critical systems must be able to tolerate Byzantine faults.

2 CONSENSUS PROPERTIES

For many practical BFT systems, the Byzantine Generals problem is generalized as a state machine replication (SMR) problem. Rather than deciding

upon a simple Boolean value ("attack" or "retreat"), nodes in the system agree upon a globally ordered sequence of "updates" (alternatively "transactions", "operations", etc.), which determines the state of the system. If all correct nodes in the system are able to agree upon the updates that are accepted and their ordering, then the state of each replica will be consistent—this is referred to as the "safety" of the system. Additionally, if a correct node proposes an update to the system, it is expected that the update must be executed eventually—this is referred to as the "liveness" of the system.

Most BFT protocols are generally expected to satisfy both of these criteria, even while under attack; i.e.:

> *Safety*: A malicious replica cannot convince a correct node to accept a state that is inconsistent with other correct nodes in the system
>
> *Liveness*: A malicious replica cannot prevent correct updates from being accepted by correct nodes in the system

There are many protocols for solving this problem in BFT systems. Some, like Prime, are optimized to provide strict performance guarantees in addition to safety and liveness. Others, like the Bitcoin blockchain, are designed to operate under specific assumptions/environments, and take weaker approaches to safety and liveness. Sections 2.1 and 2.2 provide high-level overviews of the respective approaches to BFT taken by Prime and Bitcoin. Section 3 presents a brief analysis of their comparative strengths and weaknesses.

3 PRACTICAL BFT SYSTEMS

There are many systems for implementing BFT SMR. Some, like Prime, are optimized to provide strict performance guarantees in addition to safety and liveness. Others, like the Bitcoin blockchain, are designed to operate under specific assumptions/environments, and take weaker approaches to safety and liveness. Sections 3.1 and 3.2 provide high-level overviews of the respective approaches to BFT taken by Prime and Bitcoin. Section 4 presents a brief analysis of their comparative strengths and weaknesses.

3.1 Prime

Prime is a Byzantine fault tolerant replication engine designed to provide strong performance guarantees and protection against denial-of-service attacks. It was developed at Johns Hopkins University as an improvement upon PBFT (Practical Byzantine Fault Tolerance). Like PBFT, Prime is a *leader-based* protocol, which means that it relies on a select node—i.e. a "leader"—to coordinate certain tasks in order to increase performance by reducing the amount of network communication required between nodes during normal operation. It also provides similar, strong guarantees for safety and liveness (details below). Unlike PBFT, however, Prime provides stronger guarantees for performance by limiting the possible impact of malicious servers. This is achieved by effectively forcing any leader that remains in power to meet a threshold level of performance, where the threshold is a function of the message delays between the correct servers in the system—which cannot be arbitrarily increased by malicious servers [2].

Prime provides strict guarantees of safety and liveness:

> *Safety*: If two correct Prime servers execute the i-th update, then these updates are identical [2]
>
> *Liveness*: If a stable Prime server initiates an update, all stable servers will eventually execute the update [2]

Additionally, it provides a strong performance guarantee:

> *Bounded-delay*: There exists a time after which the update latency for any update initiated by a stable server is upper-bounded

Given a topology of n=3f+2k+1 nodes, Prime guarantees these properties as long as 3 assumptions hold:

1. No more than f replicas are compromised
2. No more than k replicas are crashed/unavailable
3. The network is sufficiently stable and servers are not overloaded

Prime is able to guarantee performance while under attack because it limits the role of the leader when ordering updates. It does this by ordering updates in two phases:

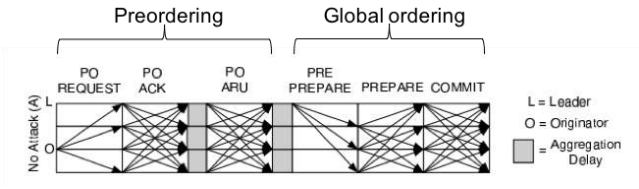


Figure 1: Common case operation of Prime (f = 1), showing the protocol messages for each round when the leader is correct. This figure is from the original Prime whitepaper [2]; modified to show the two major protocol phases

Preordering phase: Each server disseminates its updates to the other servers and coordinates an agreement protocol, which associates a preordering sequence number with each update. Once the update has been acknowledged by enough servers (which is done periodically), it is then considered eligible for global ordering.

Global ordering phase: An elected leader periodically initiates a global ordering protocol, which selects the next block of preordered updates to be executed. The final total order of updates within each block is then deterministically assigned based on their preorder identifiers.

The preordering phase ensures that all correct servers are aware of any pending updates before they are submitted to the leader. This allows non-leader servers to evaluate the leader's performance every time a new block is executed. If the leader is too slow in executing new blocks, or new blocks are missing valid updates that should have been included, the non-leader servers are able to detect these failures and force a new leader to be elected.

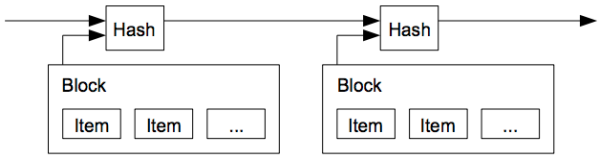
3.2 The Bitcoin blockchain*

The original aim of Bitcoin was to provide:

> […] an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party [3]

To accomplish this, it proposed a solution to the double-spending problem based on "a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions"[3].

This timestamp server works by grouping pending transactions into a block, computing a hash of that block, and widely publishing the result. The hash of each block includes the hash of the block immediately preceding it, forming a "blockchain". Because executing the transactions of a block n require first knowing the hash of block n-1—and, in effect, executing block n-1—it follows that the transactions included in block n must causally follow the transactions in block n-1. Thus, succeeding blocks in the blockchain establish an ordered sequence of executed transactions. Each additional block, in turn, appends to this sequence—otherwise referred to as a "ledger"—advancing the state of the system.



* The blockchain community is fast-moving and has many applications/variations beyond Bitcoin. This overview is based on the blockchain consensus protocol as described in the original Bitcoin paper released by Satoshi Nakamoto (often referred to as "Nakamoto consensus").

Figure 2: Simplified view of blocks in the Bitcoin blockchain. This figure is from the original Bitcoin whitepaper [3].

In order for this to work in a distributed peer-to-peer (P2P) network, however, the protocol must include some mechanism for determining consensus. Rather than relying on cryptographic identities (like Prime, and many other distributed consensus protocols), however, Bitcoin proposed a system for ascribing value to each block using a proof-of-work (PoW).

Generally speaking, a PoW is a token which is the result of a cost-function. According to the *Hashcash* whitepaper [4], a cost-function is a mathematical function that is "efficiently verifiable, but parameterisably expensive to compute". In order for a peer to generate (or "mint") a PoW token, they must execute the cost-function—thusly binding the "cost" of the PoW token to the physical cost of executing this function.

Bitcoin uses a CPU cost-function based on *Hashcash*, which is "a non-interactive, publicly-auditable, trapdoor-free cost function with unbounded probabilistic cost"[4]. Per the Bitcoin whitepaper:

> For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it. [3]

By requiring that the hash value of each block function as a PoW token, this associates a cost of physical CPU resources to the production of each block. Additionally, because the average cost of generating a block is fixed, a value can be associated with any view of the blockchain based simply on the number of blocks in the chain: e.g. if two nodes have conflicting views of the blockchain, the one with the longest chain has higher value, since a greater number of resources must have been spent in order to generate that chain. Thus, nodes in the Bitcoin network are able to determine consensus using this property: i.e. "the majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it"[3]. This is commonly referred to as "one-CPU-one-vote".

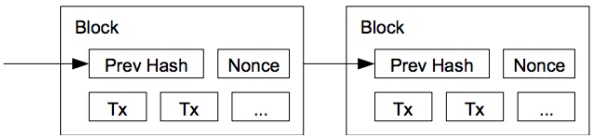


Figure 3: View of blocks in the Bitcoin blockchain with nonce value used to solve PoW. This figure is from the original Bitcoin whitepaper [3].

This process can be thought of a never-ending race to calculate (or "mine") the next block in the chain. In order for a block to ultimately be executed, it has to be included in the longest chain that eventually becomes accepted by a majority of the network. Once a node solves the PoW for a block, it must broadcast the result to other nodes in the system with the expectation that they will accept the block and begin calculating their next PoW on top of it. Because the longest chain is always selected, these nodes can only make progress if they have the most up-to-date view of the blockchain and, consequently, are incentivized to accept newly mined blocks as soon as they are broadcast. If a malicious actor wished to override or modify any blocks in the chain, they would have to re-solve not only the PoW for that block, but also the PoWs for all subsequent blocks until becoming the longest chain in the network. Thus, as long as a majority of CPU resources in the network continue to make progress together, they would most likely outpace the minority branch—preventing the malicious change from ever being accepted by the honest nodes in the network.

4 COMPARISON

4.1 Safety & Liveness

Prime and Bitcoin take very different approaches to safety and liveness. Prime (again) provides very strict guarantees:

> *Prime-Safety*: If two correct Prime servers execute the i-th update, then these updates are identical
>
> *Prime-Liveness*: If a stable Prime server initiates an update, all stable servers will eventually execute the update

The Bitcoin blockchain provides an "eventual consistency" guarantee for safety; i.e.:

> Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one that they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one. [3]

Bitcoin can also be said to provide a "best effort" guarantee for liveness; i.e.:

> New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. [3]

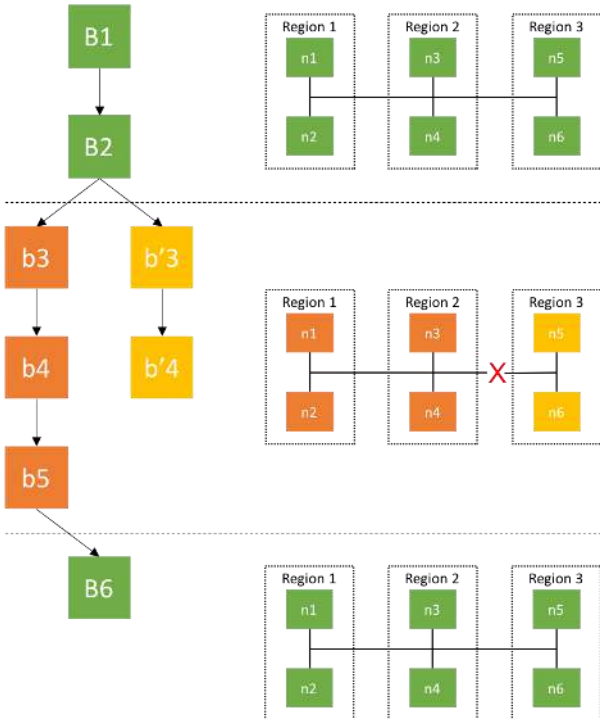


Figure 4: Example of a partition among nodes in three distinct regions for a blockchain-based application. The partition in the network causes the state of the application to fork while all nodes continue to make (potentially unsafe) progress. Eventually, global state is resolved once the partition is healed.

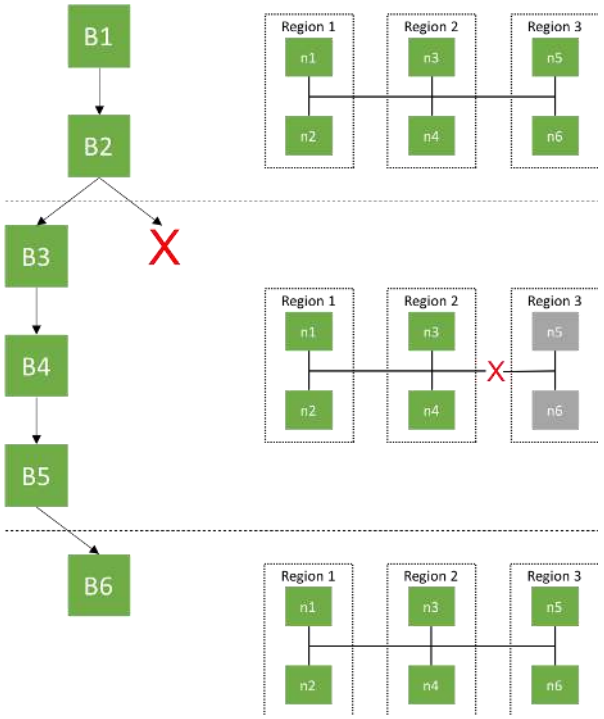


Figure 5: Example of a partition among nodes in three distinct regions for a Prime-based application. The partition in the network causes Region 3 to wait until a quorum of nodes are available. Regions 1 & 2 continue to operate safely without interruption. Once the partition is healed, Region 3 recovers any necessary updates from other nodes and resumes normal operation.

With regards to safety, the differences in these approaches can be seen in figures 4 and 5.

Figure 4 shows how a blockchain-based system responds to a partition among nodes in three distinct regions (for all intents and purposes, these could be datacenters). Before the partition, all nodes are connected and maintain consistent state. Once the partition occurs, Region 3 continues to make progress independently from Regions 1 and 2, causing the state of the state of the blockchain to fork. Once the partition heals, nodes are able to communicate once again. Because the branch of the chain in Region 3 is shorter, all blocks that were calculated are no longer considered valid and must be discarded.

Figure 5 shows how a Prime-based application responds to a partition among nodes in three distinct regions. Like figure 4, the nodes start connected and in consistent state. However, once the partition occurs in the Prime-based network, Region 3 must wait until a quorum is available in order to continue. Any pending updates in Region 3 must be buffered; whereas Regions 1 and 2 continue to safely make progress. Once the partition heals, Region 3 recovers any missing state from the other nodes, and may begin processing any updates received during the outage.

Bitcoin is able to tolerate weak safety in exchange for network flexibility because, in a highly connected and geographically diverse network, any partitions are likely to be short-lived and affect a smaller percentage of nodes. However, this example demonstrates the limitations of this approach for internal or business-to-business (B2B) applications where resources tend to be more concentrated.

4.2 Other Points of Comparison

3.2.1 Topology. One of the most significant differences between Prime and the Bitcoin blockchain are the assumptions made about the topologies of their respective networks.

Prime assumes a closed network wherein all nodes form a clique: i.e. each server knows about every other server in the system and they are able to communicate directly with one another across long term, authenticated channels. This is a reasonable requirement for the majority of internal and B2B applications, but is clearly limiting when compared to a public, P2P network like Bitcoin. Additionally, Prime relies on cryptographic techniques to protect against spoofing, impersonation, replays, and corrupted messages—which requires that the network be provisioned with public key infrastructure (PKI).

Bitcoin, on the other hand, operates across an unbounded, open network. Because consensus is based on PoW, instead of cryptographic identities, nodes across the network can effectively work together for mutual self-interest without any knowledge or trust of one another. This has allowed the Bitcoin network to grow to a global scale without significant coordination between parties or centralized authorities.

3.2.2 Attribution. Attribution refers to the ability of the system to associate actions as being caused by a specific person/entity. Prime and Bitcoin treat attribution very differently and, depending on context, it can be seen as either an asset or a liability.

Within Prime, all updates within the system are associated with the cryptographic identity of the originating node. This provides strong attribution, since a specific node can generally be associated with the responsible person/entity. This is typically a desired feature for most internal and B2B applications with may have specific auditing, compliance, or regulatory requirements.

Bitcoin, by comparison, offers no intrinsic mechanism for associating a block with the original miner. While it is possible in some cases to determine attribution for blocks or even transactions, these methods are not part of the protocol itself and generally require some case-specific, external information. In the context of a public blockchain, however, it is generally considered an asset of the protocol, since it makes the system largely intractable for regulation by centralized authorities.

5 CONCLUSION

Prime and the Bitcoin blockchain represent two fundamentally different solutions to the Byzantine generals problem. By taking a weaker approach to safety and liveness, Bitcoin is able to support an unbounded, unauthenticated, global peer-to-peer network--enabling a new generation of applications based on the public blockchain. This new paradigm of "public" applications is entirely made possible by the topological flexibility of the blockchain.

However, while this has been a major advancement in the design and construction of distributed systems, there are many use cases for non-public applications wherein Byzantine fault tolerance is still a necessary requirement. These include federated applications wherein information/state is hosted and shared only among specified set of distinct stakeholders, or even internal applications that are extremely high-value and/or security critical. For many of these applications, it is advantageous to have a fixed network wherein all nodes are known and authenticated. Additionally, lack of strong attribution and heavy resource (CPU) requirements make most blockchain protocols preclusive for these use cases. Thus, the strong guarantees for safety and liveness provided by Prime, along with its performance gains, make it highly applicable for these types of applications.

REFERENCES

[1] L. Lamport, R. Shostak, M. Pease (1982). "The Byzantine Generals Problem" (PDF). ACM Transactions on Programming Languages and Systems. 4 (3): 382–401. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.9525&rep=rep1&type=pdf

[2] Y. Amir, B. Coan, J. Kirsch, and J. Lane, “Byzantine replication under attack,” in Dependable Systems and Networks with FTCS and DCC. DSN 2008. IEEE International Conference on. IEEE, 2008, pp. 197–206. http://www.cnds.jhu.edu/pub/papers/cnds-2014-1.pdf

[3] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system, 2009,” 2012. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[4] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002..