# Proteus: A Scalable BFT Consensus Protocol for Blockchains

Mohammad M. Jalalzai[1,2], Costas Busch[1], Golden G. Richard III[1,2]
[1]Computer Science and Engineering Division
[2] Center for Computation and Technology
Louisiana State University Baton Rouge, Louisiana, USA
Email: mjalal7@lsu.edu, busch@csc.lsu.edu, golden@cct.lsu.edu

***Abstract*—Byzantine Fault Tolerant (BFT) consensus exhibits higher throughput in comparison to Proof of Work (PoW) in blockchains. However, BFT-based protocols suffer from scalability problems with respect to the number of replicas in the network due to their inherent quadratic message complexity. Previously, proposed solutions improve BFT performance for normal operation, but will fall back to quadratic message complexity once the protocol observes a certain number of failures. To address this issue we propose *Proteus*, a new BFT-based consensus protocol which elects a subset of nodes $c$ as a root committee. Proteus guarantees stable performance, regardless of the number of failures in the network and it improves on the quadratic message complexity of typical BFT-based protocols to $O(cn)$ messages, where $c << n$, for large number of nodes $n$. We tested our protocol on $200$ Amazon $EC2$ instances, with two different baseline BFT protocols (PBFT and Bchain) for comparison. In these tests, our protocol outperformed the baselines by more than $2\times$ in terms of throughput as well as latency.**

## I. Introduction

A Blockchain is a distributed ledger in which blocks of transactions are stored in sequential order. Participants in blockchain networks use consensus through a peer-to-peer network to agree on each block of transactions. Each block contains an ordered set of transactions and a link (hash) to the previous block in the chain. Traversal of the hashes to previous blocks allows to move through the history of transactions. Immutability for blocks is guaranteed since modifying a block invalidates the hashes of all newer blocks in the chain.

Recently, there has been a lot of work to address the scalability (number of replicas in the network), throughput (in terms number of transactions per second) and latency (time to insert the block of a transaction in the blockchain) issues of blockchains [1], [2], [3], [4], [5]. But increasing the number of replicas (participants) in the network can have negative effects on latency and throughput due to the increase in the number of messages being exchanged and processed within the network.

Protocols based on Proof of Work (PoW) are suffering from high latency and low throughput, but have high scalability. The main bottleneck of performance lies in solving a cryptographic puzzle before proposing a block for inclusion in the chain. Bitcoin is an example of a PoW based protocol that has been shown to accommodate thousands of replicas, but its throughput is 6-10 transactions per second and it takes an average of 10 minutes to generate a new block [6], [7]. In general, PoW-based protocols do not seem suitable for applications that require low latency and high throughput.

Byzantine Fault Tolerant (BFT) protocols [8] are able to achieve block consensus in the presence of Byzantine (malicious) replicas with the exchange of a few rounds of messages. Byzantine replicas may fail in arbitrary ways by sending malicious messages, crashing, or coordinating malicious attacks, in order to prevent the network from reaching consensus. BFT protocols can process thousands of requests per second [9], [3], and are known to have higher throughput than PoW-based protocols. BFT-based protocols typically use a single replica as a *primary* to serialize requests/blocks during each epoch of creating blocks. Once consensus is achieved on the request proposed by the primary, the block will be added to the chain at the end of the epoch. If more than one third of replicas observe malicious behavior by the primary replica, a view change will be triggered. In the view change process the failed/malicious primary will be replaced with a new primary. Another attraction of BFT protocols is their finality property, where once a block is committed it will *never* be revoked. That is, BFT protocols do not develop blockchain forks and the application layer can use the result immediately.

The scalability of BFT-based protocols is a major concern. One of the main factors negatively affecting the scalability and performance of BFT-based protocols is that they require $n \times n$ broadcast for $n$ replicas (quadratic message complexity) [10], [7], [11]. This high commu-

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00048

nication overhead is to guarantee that consensus will always be reached even after under Byzantine failures. Typically, the protocols guarantee that agreement will be reached if the total number of Byzantine nodes is less than a third of the total nodes. However, the BFT protocols usually do not distinguish the cases where there are failures or not, resulting in high message overhead.

## A. Related Work

Jalalzai *et al.* [1] have recently presented the Musch BFT protocol that addresses BFT scalability where the message complexity has been reduced to $O(f'n + n)$, where $f'$ is the actual number of Byzantine failures ($f' < n/3$). The message complexity for small $f'$ is linear in $n$, making the Musch protocol scalable, but if $f'$ is large and close to $n/3$ the message complexity becomes $O(n^2)$. SBFT [4] is another protocol that uses sets of collectors called $c$ and $e$, which are randomly selected and collect signatures for the prepare and commit phases of consensus for the block proposed by the primary. These collectors help to avoid all-to-all broadcast. During optimistic execution, SBFT achieves $O(cn)$ message complexity, but it switches to quadratic communication $O(n^2)$ if the optimistic execution fails (i.e. if the collectors fail). On the other hand, our protocol never switches to quadratic communication.

FastBFT [2] also tries to address the scalability and performance issues. In FastBFT the replicas are arranged in a tree structure where the primary node acts as the root of the tree. The message complexity in FastBFT in optimistic execution is $O(n \log n)$, but can switch to $O(n^2)$ in case of failures. Additionally, during optimistic execution signature aggregation is done over the tree and each tree node is responsible for collecting signatures from its children. The improvement in message complexity and reduction in signature size occurs with cost of latency, as the critical path length grows to $O(\log n)$.

In chain-based BFT protocols [3], [**?**], the nodes in the network are arranged serially in a chain order. Message complexity during normal execution is $O(n)$ but can grow to $O(n^2)$ in the worst case (a view change of primary). These protocols have shown high throughput but the latency is proportional to the length of the critical path, which reaches $O(n)$.

All of the aforementioned solutions switch to $O(n^2)$ message complexity once a certain number of failures are detected in the network (this threshold varies among protocols). Thus, reaching the failure threshold causes the protocol to switch to broadcast mode (fallback mode), resulting in unusable performance for large-scale systems. As a result, these protocols only provide desired performance when failure thresholds have not been met.

## B. Contributions

Our proposed protocol's performance is not bounded by any threshold on the number of failures. In other words, protocol performance is not affected by the number of failures detected in the network and remains constant (guarantees stable performance) during normal execution. This is a very important and strong characteristic that enables the protocol to provide consistent performance guarantees. Additionally, our protocol provides constant latency in terms of critical path length, which is the number of one-way messages from block proposal to completion of consensus.

These improvements are achieved in Proteus by randomly selecting a *root committee* consisting of $c$ replicas picked from a large set of replicas with total size $n$. Typically, $c$ is small compared to $n$, namely, $c << n$. Our algorithm can tolerate up to $f < n/3$ Byzantine failures. In each epoch, the BFT consensus is first executed among the $c$ replicas of the root committee, instead of the overall $n$ replicas. The block proposed by the root committee will then be validated by the remaining $n - c$ correct replicas. Since the main BFT process will be executed within the root committee of size $c$, which is much smaller than $n$, we get a performance advantage. The message complexity in our protocol is $O(c^2 + cn)$ for normal as well as view change mode, and when $n$ is large ($c << n$), it becomes $O(cn)$. Through experimental evaluation we also show that Proteus outperforms both the PBFT [10] and Bchain in terms of throughput and latency.

Our protocol can tolerate up to $c/2 - 1$ Byzantine failures in the root committee. Therefore, the root committee is more resilient than the typical case which tolerates less than $c/3$ nodes failing. The $n$ regular replicas keep an eye on failures of the $c$ root committee replicas that generate a block. In case $c/2$ or more nodes fail, a view change will occur and another root committee is selected. This is another unique aspect of our protocol in that a view change causes the whole root committee to be replaced, whereas other BFT-based protocols only replace the primary node.

**Paper Outline:** In Section II we give the system model and in Section III we present the Proteus protocol. Section IV contains the experimental evaluation of Proteus. We conclude our work in Section V.

## II. System Model

We consider the Byzantine fault model for the replicas. Under this model, servers and clients may deviate from their normal behavior in arbitrary ways, which includes hardware failures, software bugs, and other malicious behavior. Our protocol can tolerate up to $f$ Byzantine replicas where the total number of replicas in the network is $n$ such that $n = 3f + 1$.

This model also assumes that replicas will not be able to break collision resistant hashes, encryption and signatures. We assume that all messages sent by the replicas are signed. To ensure liveness in the asynchronous network we use a timeout to place an upper bound on the block generation time. Moreover, time is divided into epochs and during each epoch a block is generated. For example a regular replica $j$ will end its current epoch upon committing a block and will start its timer for the next epoch while it is expecting another message (block proposal) from the root committee.

## III. Protocol

In Proteus, we have moved the burden of consensus to the root committee with size $c$. The root committee is running a customized $BFT$ based algorithm (Algorithms 1 and 2). Regular replicas that are not members of the root committee simply verify the result of consensus, i.e. the proposed block. If $\lfloor c/2 \rfloor + 1$ root committee members first agree and then also $2f + 1$ nodes (root committee plus regular replicas) agree on the proposed block, then the block is committed and will be added to the blockchain. It should be noted that our protocol can tolerate up $\lceil c/2 \rceil - 1$ Byzantine replicas in the root committee. Upon failure of the root committee to propose a block, the whole root committee is replaced by the view change process.

During normal execution the BFT protocol in the root committee successfully generates a block (collects more than $c/2$ *prepare* as well as *pre-commit* messages for the block) and proposes it to the regular replicas through broadcast. Upon receipt of a block, regular replicas verify the block against their histories and check if the block is signed by at least $\lfloor c/2 \rfloor + 1$ replicas from the root committee. If the block is valid, each regular replica signs the block and sends back the signature to the root committee. Upon receipt of $2f + 1$ signatures, each root committee member commits the block and broadcasts the proof of acceptance ($2f + 1$ signatures) to regular replicas. Upon receipt of proof, each regular replica commits the block, which is permanently added to the local history.

### A. Root Committee Selection and Failure Probability

The root committee is formed by randomly and uniformly picking a set of $c$ nodes out of $n$. On expectation, the root committee will have less than $c/3$ Byzantine nodes. Hence, it is likely the root committee to have less than $c/2$ faulty nodes. However, there is a small chance that we will have the problematic situation where the root committee will have at least $c/2$ faulty nodes. Moreover, the primary of the root committee may be faulty as well. However, as we show below, a view change can rectify the problematic situation. We continue with analyzing the different scenarios for picking the root committee.

Suppose that the root committee has $a$ honest nodes and $b$ faulty nodes ($c = a + b$). The probability $P_f$ of having at least $c/2$ faulty nodes ($b \geq c/2$) is:

$$P_f = \sum_{b=\lceil c/2 \rceil}^{c} \frac{\binom{n-f}{a}\binom{f}{b}}{\binom{n}{c}}. \tag{1}$$

If the root committee fails to generate a valid block, it is replaced by another randomly chosen committee (view change). The probability $P_f$ (Equation 1) for different practical sizes of $c$ and $n$ used in our experiments is low and in the range of $0.0044 - 0.009$.

There are two cases that can ultimately trigger a view change: (i) having the number of faulty nodes $b$ in the root committee to be $b \geq c/2$ (this occurs with probability $P_f$), or (ii) when the number of faulty nodes is $b < c/2$ and the primary is faulty. The primary node is the lowest id node in the root committee, and it can be assumed to be picked uniformly at random among the $c$ nodes of the root committee (since the root committee was randomly picked).

Thus, for the latter case ii, if the number of faulty nodes in the root committee is $b < c/2$ (this occurs with probability $1 - P_f$), then the probability of primary being faulty is $b/c < (c/2)/c = 1/2$; hence, case ii occurs with probability less than $0.5(1 - P_f)$. Therefore, the probability $P_v$ of having view change due to a problematic root committee will be $P_v < 0.5(1 - P_f) + P_f$. Since $P_f$ is approaching 0, the upper bound of probability $P_v$ can be approximated by $0.5$.

Therefore, on the expected case, if the protocol performs two consecutive view changes, one of them will result in a good root committee choice where at least $c/2$ replicas are honest and also the primary is honest. Thereafter, such a good root committee will stably generate blocks for long time (without being replaced). Therefore, the amortized cost of view change will be negligible. Note that at least one honest node is needed in the root committee to guarantee that view change takes place which is true with high probability (at least $1 - 3.8 \cdot 10^{-22}$) for the parameters in our experiments.

### B. Detailed Protocol Operation

The basic operation of Proteus is given in Algorithms 1 and 2 which describe the normal execution in the root committee. Note that the root committee members also run the protocol as regular nodes as well.

**Algorithm 1:** Customized BFT algorithm for root committee member $i$

```
    // primary is a designated node in root
        committee
1   if i is primary in root committee then
2       Collect transactions and form block
3       Propose block to root committee by broadcasting a
          pre-prepare message with the block
4   end
5   upon receipt of valid pre-prepare message do
6       Broadcast prepare message with the proposed block to
          root committee
7   end
8   upon receipt of $\lfloor c/2 \rfloor + 1$ valid prepare messages do
9       Broadcast pre-commit message to root committee
10  end
11  upon receipt of $\lfloor c/2 \rfloor + 1$ valid pre-commit messages for
      proposed block do
12      Return successfull and continue to Algorithm 2
13  end
14  if received timeout complaint message from regular node $j$ for
      block number $B_t$ then
15      if $j$ has not previously received from $i$ the most recent
          missing blocks since $B_t$ then
16          Update $j$ up to latest block
17      end
18  end
19  if not returned success by block timeout or if received signed
      block hash from regular replicas while not have received the
      block form the primary then
20      Broadcast timeout complaint to regular nodes
21      Return unsuccessful and continue to Algorithm 2
22  end
```

**Algorithm 2:** Root committee member $i$

```
1   upon $i$ returns successful from Algorithm 1 do
2       Broadcast the block to regular nodes and the members of
          the root committee that have not signed it
3   end
4   upon $i$ returns unsucessful from Algorithm 1 do
5       Accept messages from regular nodes to synchronize local
          history
6   end
7   upon receipt of $2f + 1$ valid signatures from regular nodes for
      proposed block hash do
8       Commit block
9       Broadcast aggregated $2f + 1$ valid signatures
10  end
11  upon detecting proof of maliciousness: two different signed
      blocks or receipt of $f + 1$ timeout complaints do
12      Broadcast maliciousness proof
        // Transition to new view, based on
            common random number generation seed
13      Randomly select $c$ members of root committee from set of
          $n$ nodes
14      if node $i$ is not member of new root committee then
15          Execute view change as regular replica
16      else
17          Execute view change as root committee member (and
              concurrently as a regular replica)
18      end
19  end
```

### a) Normal Mode

A designated primary node in the root committee proposes a block by broadcasting a pre-prepare message to the other root committee members (Algorithm 1, lines 1-4). A pre-prepare message $B = (\langle \text{“}Pre\text{-}prepare\text{”}, v, s, h, d\rangle_p, m)$ contains the view number $v$, block sequence number $s$, transaction list $m$, its hash $h$ and the previous blockhash $d$. A replica $i$ in the root committee begins a customized BFT algorithm after receipt of a pre-prepare message. Then, replica $i$ broadcasts a prepare message $\langle \text{“}Prepare\text{”}, v, s, h, i\rangle_i$ if it finds the pre-prepare message to be valid (Algorithm 1, lines 5-7). Upon receipt of $c/2{+}1$ prepare messages from other root committee members, the replica $i$ broadcasts a pre-commit message $\langle \text{“}Pre\text{-}commit\text{”}, v, s, h, i\rangle_i$ to the root committee (Algorithm 1, lines 8-10). If replica $i$ receives $\lfloor c/2 \rfloor + 1$ pre-commit messages from other root committee members (Algorithm 1, lines 11-13), then it will return success and continues to Algorithm 2.

As we discussed in Section III-A, the root committee might be problematic when it has $c/2$ or more faulty nodes. In such a case, the faulty root committee members might attempt to not update (send pre-prepare and other messages) $\zeta \leq f$ regular replicas without triggering view change. These $\zeta$ replicas may not be participating in the consensus process as the malicious root committee members may not send them messages, and they will need to sync their history (download recently committed blocks) with other replicas.

Suppose that $j$ is one of those $\zeta$ regular replicas that was not synchronized. A root committee member $i$ might receive a timeout complaint for block $B_t$ from regular replica $j$. If replica $i$ has not received a timeout message from the same replica $j$ for the same block or any block with larger sequence number, then $i$ will forward missing messages (prepare and/or commit) for missing the block (Algorithm 1, lines 14-17).

If root committee member $i$ does timeout itself for not receiving valid messages during the consensus process (Algorithm 1, lines 18-22), it will broadcast its own complaint to regular replicas and will return non-success error and then continue to Algorithm 2. Therefore, if a replica $i$ in root committee does not receive the block from the malicious primary it can recover the block by asking regular replicas. In return, the same replica $i$ can send the missing message to any regular replica that requests it. In this way, members of root committee and regular replicas download missing blocks from each other (without broadcasting) while keeping message complexity in check ($O(cn)$).

If the root committee members successfully generated a proposal block, with $\lfloor c/2 \rfloor + 1$ signatures for pre-commit as proof, then they will broadcast it to the regular committee members (Algorithm 2, lines 1-3). Upon receipt of a block from the root committee, the regular replicas check if it is signed by at least $\lfloor c/2 \rfloor + 1$ root committee members and if verified, a regular replica $j$ sends back a signed approval message $\langle \text{“}Approval\text{”}, v, s, h, j \rangle_j$ to the root committee. Each member of the root committee aggregates $2f + 1$ signatures $\sigma(h)$ (including the pre-commit signatures from root committee as well as approval message signatures from regular replicas) and then commits the block locally and broadcasts a confirm message $\langle \text{“}Confirm\text{”}, v, s, h, i, \sigma(h) \rangle_i$ to regular replicas. Upon receipt of $2f+1$ valid signatures from the root committee the regular members also commit the block.

### b) View Change

A view change in our protocol will replace the root committee with $c$ new members. A view change occurs if the root committee fails to generate a new valid block.

During each epoch, a regular replica waits to receive a proposed block or *Approval* message from the root committee. If a regular replica $i$ does not receive the block after a timeout then it considers that the root committee has failed and reports this to the root committee. If $f+1$ nodes report a timeout, then this triggers the view change process in the root committee and then the root committee members forward $f + 1$ timeout complaints to regular replicas triggering view change in the regular replicas as well.

In the view change each node selects a new set of $c$ replicas for the new root committee, using a pre-specified common seed for pseudo-random number generation, which guarantees that every replica selects the same root committee. Each replica $i$ broadcasts the last state of its local history $V_i = \langle \text{“}ViewChange\text{”}, v + 1, s', H, i, (\sigma(H)) \rangle_i$ to the new root committee members, which includes the latest block sequence number $s'$, block hash $H$, incremented view number $v + 1$, and signature evidence of at least $2f + 1$ $(\sigma(H))$ replicas that have approved the block. The new root committee members wait to receive $2f+1$ local histories $V_i$ from all replicas and aggregate them into $Q$. Once they receive $V_i$ from replica $i$, its local history $H_i$ (with latest committed block) is extracted from $V_i$. Out of the $2f+1$ histories in $Q$, it is guaranteed, that at least $f+1$ are honest replicas (with at least one having most recent history). Thus, the most recent history in at least $f+1$ honest replicas (that have signed the most recent block) will be the starting point for the next block to be generated.

The new root committee members will broadcast $Q$ to all replicas. Upon receipt of $Q$, replica $i$ makes sure that its history matches with the latest history of $Q$ . If its history matches, replica $i$ sends back a ready message $(R_i)$ to the new root committee. Root committee member $j$ will aggregate $2f + 1$ ready responses into a single $P$ message and broadcast it to all replicas. Upon receipt of $P$, replica $i$ is now ready to take part in new view. If replica $i$'s history does not match that of most recent history in the $Q$, it will need to synchronize its history by communicating with the new root committee.

To guarantee all transactions are executed properly, clients can resend their transactions to the new primary of the new root committee, after they realize that they have not received any response for their transaction(timeout).

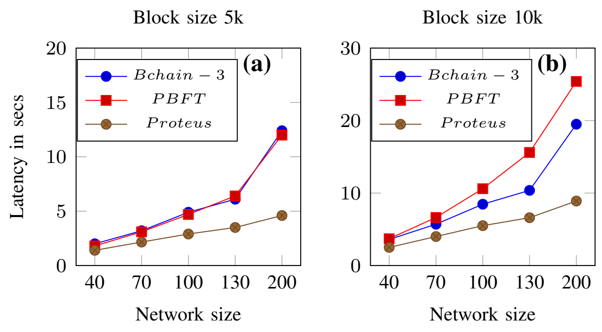

**Fig. 1:** Protocol latencies with block sizes 5000 and 10000

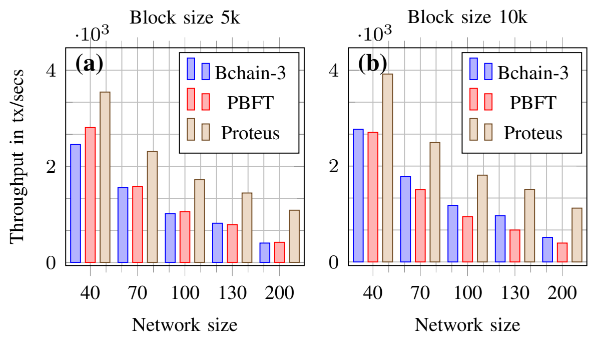

**Fig. 2:** Protocol throughput with Block size 5000 and 10000

## IV. Experiments and Evaluations

We have implemented the Proteus protocol in about 2.5k lines of Golang code. We also implemented PBFT [10] to be used as a baseline. We selected PBFT because while other protocols have improved on PBFT performance, they usually switch to $n \times n$ broadcast (legacy PBFT) as a fallback measure if a certain threshold number of failures have occurred. Additionally we also implemented Bchain-3 [12] which belongs to the family of chain-based BFT protocols, where instead of a broadcast the protocols propagate messages along a predetermined chain order. Bchain-3 has the potential to decrease the number of messages that are sent in the network, however, latency may increase due to a possibly long chain of message relays. Both PBFT and Bchain-3 are also implemented in Golang for the experiments. We think comparing our protocol with different flavors of BFT-based protocols (broadcast and chain based BFT) provide good insight into the performance of Proteus.

We conducted our experiments in the Amazon Web Services (AWS) cloud. For each replica in the network we used instances of type *t2.large* in AWS. Each *t2.large* instance has 2 virtual CPU cores with 8GB of memory. The experiments were performed with network sizes of 40, 70, 100 and 130 and 200 replicas. We used Equation 1 to get suitable value for the root committee size $c$ (for different $n$) we selected the various root committee sizes to be 18 (40), 27 (70), 30 (100), 33 (130), and 36 (200). We also used different block sizes with $5k$ and $10k$ transactions. We used simple transactions that are randomly generated and transfer funds from one account to another account. Replicas maintain a complete history of blocks (complete blockchain), and also perform regular operations on the blocks they receive which include checking the history, hashes, and signatures to verify the transactions in the block.

We measure latency in an epoch as the time between the primary root committee proposing a block until the time that the block is inserted in all the local histories. We average the latency over several epochs. Figure 1 provides a comparison of latency measurements among Bchain-3, PBFT, and Proteus. The difference in latency is small for smaller network sizes, but it increases as the network size grows. Additionally, we observe that the latency is affected by the block size, so that if the block size increases then the latency increases as well. For example in Figure 1 (a), the latency difference among protocols is smaller, but as the block size increases (Figure 1 (b)) we can see that the latency of PBFT increases much faster (due to its high message complexity) followed by Bchain-3 (due to longer length of the critical chain path).

We also measure the throughput, which is the number of committed transactions per second that are appended to the blockchain (Figure 2). Similarly to latency, by observing Figure 2 we can see that Proteus outperforms PBFT and Bchain-3 for all the test cases.

## V. Conclusion and Future Work

In this paper we presented Proteus, a BFT-based consensus protocol for blockchains that provides better latency and throughput than the state of the art BFT protocols (Bchain and PBFT). The main idea is to use a randomly selected root committee of small size to coordinate the consensus process with all the nodes. The experimental analysis shows that Proteus provides consistent performance regardless of the number of failures encountered in the network, whereas other BFT protocols suffer from fall back performance degradation as the number of failures in the network reach a threshold. Our future work will mainly focus on how to further improve BFT scalability (through sharding).

## References

[1] M. Jalalzai and C. Busch, "Window based BFT blockchain consensus," in *2018 IEEE International Conference on Blockchain (Blockchain 2018)*, Halifax, Canada, Jul. 2018.

[2] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *CoRR*, vol. abs/1612.04997, 2016. [Online]. Available: http://arxiv.org/abs/1612.04997

[3] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 363–376. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755950

[4] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: a scalable decentralized trust infrastructure for blockchains," *CoRR*, vol. abs/1804.01626, 2018.

[5] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 45–59.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf."

[7] M. Vukolic, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, 2015, pp. 112–125.

[8] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems." *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 2, pp. 254–280, Apr. 1984. [Online]. Available: http://doi.acm.org/10.1145/2993.2994

[9] R. Kotla, A. Clement, E. Wong, L. Alvisi, and M. Dahlin, "Zyzzyva: Speculative byzantine fault tolerance," *Commun. ACM*, vol. 51, no. 11, pp. 86–95, Nov. 2008. [Online]. Available: http://doi.acm.org/10.1145/1400214.1400236

[10] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296824

[11] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 17–30. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978389

[12] S. Duan, H. Meling, S. Peisert, and H. Zhang, "Bchain: Byzantine replication with high throughput and embedded reconfiguration," in *Principles of Distributed Systems*, M. K. Aguilera, L. Querzoni, and M. Shapiro, Eds. Cham: Springer International Publishing, 2014, pp. 91–106.