Metadata of the chapter that will be visualized in SpringerLink

Book Title	Blockchain – ICBC 2019	
Series Title		
Chapter Title	An Experimental Evaluation of BFT Protocols for Blockchains	
Copyright Year	2019	
Copyright HolderName	Springer Nature Switzerland AG	
Corresponding Author	Family Name	**Jalalzai**
	Particle	
	Given Name	**Mohammad M.**
	Prefix	
	Suffix	
	Role	
	Division	Department of Computer Science
	Organization	Louisiana State University
	Address	Baton Rouge, USA
	Division	Center for Computation and Technology
	Organization	Louisiana State University
	Address	Baton Rouge, USA
	Email	mjalal7@lsu.edu
Author	Family Name	**Richard**
	Particle	
	Given Name	**Golden**
	Prefix	
	Suffix	III
	Role	
	Division	Department of Computer Science
	Organization	Louisiana State University
	Address	Baton Rouge, USA
	Division	Center for Computation and Technology
	Organization	Louisiana State University
	Address	Baton Rouge, USA
	Email	goldenrichard1@lsu.edu
Author	Family Name	**Busch**
	Particle	
	Given Name	**Costas**
	Prefix	
	Suffix	
	Role	
	Division	Department of Computer Science
	Organization	Louisiana State University
	Address	Baton Rouge, USA

Email kbusch@lsu.edu

Abstract

Byzantine Fault Tolerant (BFT) protocols have been used in blockchains due to their high performance and fast block acceptance. However, their weakness is a lack of scalability to support a large number of nodes in the network due to message demanding broadcasts. There have been recent improvements to the classic Practical Byzantine Fault Tolerant (PBFT) protocol. Evaluating the performance and reliability of the different BFT based protocols in the context of blockchains will give users a better picture of the behaviour and scalability of these protocols under different circumstances. For this purpose, we implemented and evaluated the performance of different BFT based protocols for blockchains under normal conditions as well as when byzantine failures are encountered in the network. Furthermore, we also calculated the reliability of each protocol under the desired throughput.

Keywords (separated by '-')

Byzantine Fault Tolerant - Blockchains consensus - Experimental evaluation - Reliability

An Experimental Evaluation of BFT Protocols for Blockchains

Mohammad M. Jalalzai[1,2(✉)], Golden Richard III[1,2], and Costas Busch[1]

[1] Department of Computer Science, Louisiana State University, Baton Rouge, USA
{mjalal7,goldenrichard1,kbusch}@lsu.edu
[2] Center for Computation and Technology, Louisiana State University, Baton Rouge, USA

Abstract. Byzantine Fault Tolerant (BFT) protocols have been used in blockchains due to their high performance and fast block acceptance. However, their weakness is a lack of scalability to support a large number of nodes in the network due to message demanding broadcasts. There have been recent improvements to the classic Practical Byzantine Fault Tolerant (PBFT) protocol. Evaluating the performance and reliability of the different BFT based protocols in the context of blockchains will give users a better picture of the behaviour and scalability of these protocols under different circumstances. For this purpose, we implemented and evaluated the performance of different BFT based protocols for blockchains under normal conditions as well as when byzantine failures are encountered in the network. Furthermore, we also calculated the reliability of each protocol under the desired throughput.

Keywords: Byzantine Fault Tolerant · Blockchains consensus · Experimental evaluation · Reliability

1 Introduction

Blockchains are data structures consisting of chains of blocks, with each block referencing the hash of the previous block, creating a linked list of blocks. Blocks are added to the chain after consensus, where a majority (depending on the type of consensus) of participants in the network agree on the block and its contents, which typically consists of a set of transactions. There are several different types of consensus protocols available, such as Proof-of-Work, Proof-of-Stake, and Byzantine Fault Tolerant protocols.

In Proof-of-Work (PoW)-based protocols each miner node (i.e., a block proposer) has to solve a crypto-puzzle before proposing the block. Solving this crypto-puzzle is expensive in terms of processing but the solution can be verified quickly by other nodes. This helps the network to achieve consensus in a reliable manner by avoiding Denial of Service (DoS) and Sybil attacks on the network. PoW-based consensus protocols are highly scalable in terms of the number of

J. Joshi et al. (Eds.): ICBC 2019, LNCS 11521, pp. 1–15, 2019.
https://doi.org/10.1007/978-3-030-23404-1_3

nodes in the network, but exhibit low throughput (number of transactions per second). Bitcoin [13] is one of the most successful examples of a PoW protocol and achieves consensus with an average time of ten minutes and throughput of 3 to 7 transactions per second. Ethereum [18] is another example of PoW which generates around 37 transactions per second but still suffers from PoW limitations [19]. According to Luu *et al.* [12] 95% of the Bitcoin network is controlled by ten mining pools, while 80% of Ethereum network mining power resides in six mining pools. As these pools are controlled by pool operators in a centralized fashion, the Bitcoin and Ethereum networks are both susceptible to 51% attacks. Furthermore the presence of forks in PoW protocols potentially allows double spending attacks. To avoid double spending it is recommended that a user/application has to wait a certain number of blocks to make sure that it is highly unlikely that the current blockchain segment will be replaced with a different one. In Bitcoin this confirmation wait time is about 60 min or six blocks of wait time [13].

In systems that use Proof-of-Stake (PoS) [2,4,16,17] protocols instead of PoW, block proposers (so called "forgers") in the network must have some stake in the network. Usually, the next forger is selected randomly proportional to their stake in the network. The forger validates transactions before proposing those transactions in a new block. Thus, the higher the stake, the higher the probability a particular node will be selected as the next forger.

Byzantine Fault Tolerant (BFT) [11] protocols are another class of consensus protocols that are used in blockchains to achieve consensus efficiently. BFT based protocols do not require solving crypto-puzzles or proving stake for participants in the network. Instead, BFT protocols generally reach consensus through exchange of multiple rounds of messages. BFT protocols can achieve consensus in the presence of less than a third Byzantine replicas and these potentially malicious nodes can behave in arbitrary ways by failing, sending arbitrary or malicious messages, or performing coordinated attacks. BFT protocols generally use a replica called a primary to validate, sort, aggregate, and propose transactions inside a block to all the replicas in the network. Upon receipt of the block, each replica verifies it based on its own history (chain of blocks) and if the block is valid, each replica communicates with other replicas based on specifics of the underlying BFT Protocol mechanism to achieve consensus. Once more than two thirds of replicas agree on the block, the block will be added to the blockchain and committed. In case the primary behaves maliciously and there is sufficient proof of maliciousness present (for example more than one third complaints against primary), a view change is triggered, which results in replacement of the primary. Two thirds majority agreement gives an important characteristic to the BFT protocols, called finality. Finality means that once a block is committed, it will never be revoked. This is in sharp contrast to PoW and PoS protocols which suffer from forking and the associated risks of transactions within a fork being revoked.

There are a variety of BFT based protocols [1,3,5,9,10,13] that address the consensus problem in different ways. Message complexity, cryptography, and

latency are various parameters that effect the performance of BFT protocols. We argue that it is very important for users to know how different BFT-based protocols perform in different scenarios (e.g., error-free mode, in the presence of Byzantine nodes, etc.), when implemented in the context of blockchains.

To address this concern, we present experimental analysis of our Musch BFT protocol for Blockchains [10] against different flavours of BFT based protocols. Theoretical analysis of the Musch BFT protocol shows promising results as it reduces the message complexity to $O(f'n)$ from $O(n^2)$ (where f' is the actual number of faulty nodes, and n the total number of nodes) without affecting the critical path (number of one way message latencies from a client sending a request to the network and associated response). We have implemented the Musch BFT protocol for blockchains in Go, using more than $2.3k$ lines of code. We also implemented other flavors of BFT including PBFT [3], Bchain-3 [5] and SBFT (a Scalable Decentralized Trust Infrastructure for Blockchains) [9]. Each variant of the BFT protocols selected here uses different mechanisms to improve BFT performance and scalability. We tested each protocol on different network sizes ranging from 40 to 160 EC2 instances on the Amazon cloud.

PBFT is classic BFT and the first practical protocol for this type of consensus. It uses multiple rounds of broadcast to achieve consensus. Bchain-3 is a member of a class of protocols called chain-based BFT protocols, where participants/replicas in the network are arranged in an overlay chain structure. This gives Bchain-3 better message complexity and throughput with increased latency. SBFT and Musch are both improvements of PBFT, where both of them in normal mode avoid broadcast through message aggregation. But upon experiencing failure, SBFT falls back to the PBFT protocol, whereas Musch switches to failure mode, where failure is addressed by communicating to other replicas in the network through an increasing size of window of replicas. Initial window size is 1 (communicating with only one node to recover from failure) and increases exponentially $(1, 2, 4, 8, \ldots, (2n/3) + 1)$ until failure is addressed. In this paper, we implemented the Musch BFT protocol for the first time and analyzed and evaluated performance, along with the PBFT, Bchain-3, and SBFT protocols. Additionally, we also provide reliability measures for each protocol when the network is under attack and desired performance is constant.

Paper Outline: In Sect. 2 we present in detail the four BFT protocols that we implemented. Section 3 shows experimental evaluation of BFT based protocols for blockchains. In Sect. 4 we discuss reliability analysis of BFT protocols when network is under attack. We conclude our paper and discuss our future work in Sect. 5.

2 Byzantine Fault Tolerant Protocols

We present a brief overview of protocols under test. This includes how protocols behave in normal operation to achieve consensus and how consensus is achieved in the presence of failures. It should be noted that all of these protocols operate

in a semi-synchronous environment where communication is bounded by time. At any time, the number of failures or Byzantine replicas in the network cannot exceed $f < n/3$, as a BFT protocol cannot guarantee consensus in asynchronous environment if the number of failures reaches one third [6]. Suppose that $n = 3f+1$. The protocols we consider here use authenticated messages for communication.

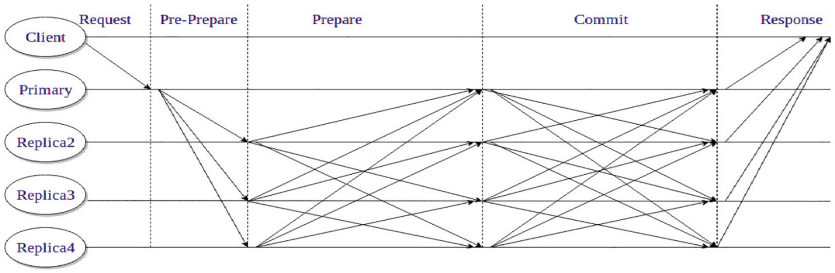

Fig. 1. PBFT communication pattern

2.1 Practical Byzantine Fault Tolerant Protocol

PBFT was the first practical Byzantine fault tolerant protocol to be proposed. It shows that it is possible to implement BFT protocols for use in practical systems, unlike the previous protocols [7,14] that were theoretically feasible but not efficient enough to be used in practice. Normal PBFT operation in the context of blockchains can be described as:

- Clients send transaction requests to the primary replica
- The primary replica is responsible for aggregating (batching), sorting, and proposing a block of transactions
- Once transactions are added to the block, the block is proposed by broadcasting
- Each replica receives the proposed block of transactions (also called the "pre-prepare" message) and verifies the current view (primary), the sequence number (block height), transactions, and primary signature
- Upon successful validation, each replica broadcasts a signed prepare message which includes the block sequence number, view number, hash of the transactions, and its ID
- Upon receipt of $2f$ valid prepare messages the replica broadcasts a commit message that includes view number, sequence number, transactions hash, as well as the replica's own ID
- Upon receipt of $2f + 1$ commit messages, the replica commits the block

The PBFT communication pattern is shown in Fig. 1. It can be seen that the PBFT critical path is constant (4) but it exhibits quadratic message complexity, $O(n^2)$ messages, to guarantee consensus even in the absence of failures.

2.2 Bchain-3

Bchain [5] was developed to address the quadratic message complexity of PBFT. Bchain uses two types of algorithms. Bchain-3 can tolerate $f < n/3$ failures, whereas Bchain-5 can tolerate $f < n/5$ failures. We evaluated Bchain-3, since the number of faults it can tolerate is similar to the other protocols under testing. To achieve $O(n)$ message complexity, Bchain-3 arranges replicas (network participants) in serial/chain order, in such a way that each replica forwards messages to another one positioned next to it in the chain. The replicas in the chain are divided into different categories. The first replica in the chain is called the *head/primary* (P_h). The last replica is the *tail*, and the $(2f + 1)$th replica is called *proxy tail* (P_p). Replicas are also divided into two sections in the chain. A is the first $2f + 1$ replicas in the chain where as B are the last f replicas. Replica organization in a Bchain network is given in Fig. 2.

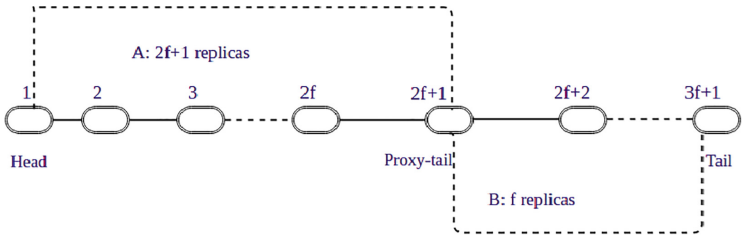

Fig. 2. Replicas are organized serially in Bchain

During normal operation the first $2f + 1$ replicas perform consensus whereas the last f replicas, simply update their chain by appending the block agreed by A replicas. The Bchain protocol transmits two types of messages: $\langle CHAIN \rangle$, that represents request/proposed block to its successor, and $\langle Ack \rangle$ which is the reply/acknowledgement of successful execution of the proposed block to predecessor (in opposite direction of $\langle CHAIN \rangle$). Bchain message pattern is shown in the Fig. 3.

The normal operation for the Bchain protocol can be described as:

- After receipt of a transaction from a client, the head/primary node assigns a sequence number to it
- After aggregating the transaction into a $\langle CHAIN \rangle$ message the primary adds the chain order and forwards the block to the next replica in the chain (its successor)
- Upon receipt of a $\langle CHAIN \rangle$ message by proxy tail, it forwards an *Ack* message to its predecessor toward the head replica and a reply message to the client
- Upon receipt of an *Ack* message, each replica in A commits the block, forwards the *Ack* message to its predecessor along the chain, and forwards its $\langle CHAIN \rangle$ message to replicas in B

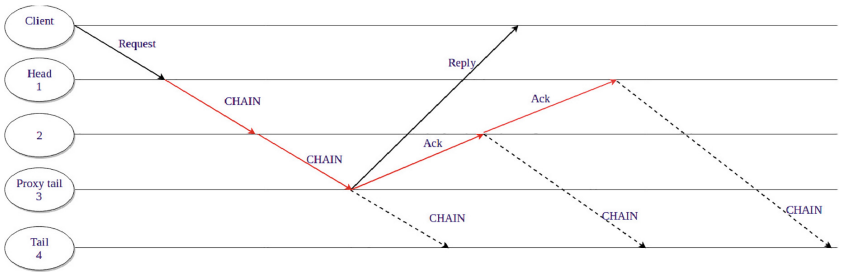

Fig. 3. Bchain communication pattern

Upon encountering failures, Bchain uses a re-chaining mechanism to recover from failure and this mechanism works as follows:

- Every replica (except head and tail) starts a timer after forwarding the $\langle CHAIN \rangle$ message to its successor. If it does not receive an *Ack* message before the timer expires, it will send two $\langle SUSPECT \rangle$ messages to the head (one directly and the other along the chain)
- If the head receives multiple $\langle SUSPECT \rangle$ messages, it will handle the closest one to the tail
- The head will begin the re-chaining mechanism, which involves moving the accused replica to the tail and moving the accuser to the end of A (proxy tail), so that it cannot accuse other replicas. In this way a malicious replica cannot falsely accuse more replicas.

2.3 Scalable Decentralized Trustable Infrastructure (SBFT)

SBFT improved PBFT's performance by using two set of collectors of size c to collect, prepare, and commit messages from replicas, thus avoiding the $n \times n$ broadcast of prepare and commit messages, while achieving $O(cn)$ message complexity during normal operation. As shown in Fig. 4, during failure-free operation, a primary collects transactions, sorts them into a block, adds a sequence number, view number, hash of the block, and broadcasts it to all replicas. Upon receipt of the block proposal/pre-prepare message, replica i verifies the validity of the block and if it is valid, i sends a prepare message containing the block hash, view number, block sequence number, and replica i's signature to a set C of collectors (the sign-share phase). Upon receipt of $3f + 1 + c$ valid signatures, the C collectors generate a signature and broadcast it to all replicas (full-commit-proof).

Once a replica receives the valid signature(meets threshold of require signatures), it executes the transactions in the block and signs the new state using signatures and sends it to E collectors. E collectors generate signatures (with threshold of $f + 1$ valid signatures) along with an execution certificate and send it to all replicas through broadcast (sign-state). E collectors also send a reply back to the client, verifying execution of the client transaction (execution-proof).

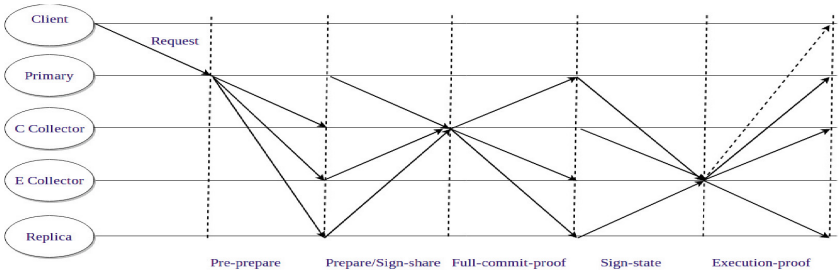

Fig. 4. SBFT communication pattern

Additionally, unlike normal BFT based protocols in which the number of Byzantine replicas are bounded by $f < n/3$, in SBFT the bound is $f < (n-2c)/3$, where $c \geq 0$. If collectors fail to respond before the timeout period, SBFT switches into fallback mode, reverting to $n \times n$ broadcast. Increasing the number of collectors c, will increase message complexity, whereas keeping the number small can cause frequent switching to fallback mode.

2.4 The Musch BFT Protocol for Blockchains

Musch uses an adaptive mechanism of a sliding window or response nodes to address failures/complaints in the network. The main contribution of the Musch BFT protocol is that it improves message complexity without sacrificing latency. For complaints against malicious replicas or primary, the protocol has a set of window nodes that respond to complaints. The window size is adaptive. Initially the window size is 1. If the complaints are not addressed by the window, then the window size doubles. Eventually, when the window size reaches $f' + 1$ then the complaints will be properly addressed, where f' is the actual number of byzantine nodes that have triggered the complaints (and $f < n/3$ is an upper bound on f', $f' \leq f$). The network broadcasts are through the window. Thus, the message complexity stays at $O(f'n + n)$. If there are no failures ($f' = 0$) the message complexity is linear, which improves on the PBFT quadratic complexity.

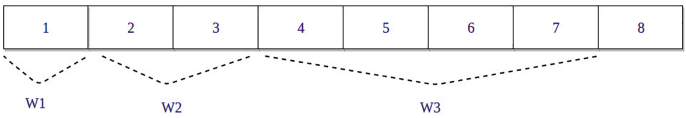

Fig. 5. Exponential increase in window size

Musch BFT uses Echo broadcast [15], to propose the block to replicas in the network. If the proposed block is valid, replicas send back a signed hash of the block to the primary. Upon receipt of $2f + 1$, agreement messages from replicas,

the primary aggregates them into an aggregated signature α_p, and broadcasts it back. Collecting $2f + 1$ agreements for a proposed block means the majority of replicas have agreed on all transactions and the order of their execution. Upon receipt of an aggregated message each replica verifies the validity of the $COMMIT$ message and if valid, they commit the block. If each replica has received the same block, they will agree on the hash and the order of transactions to be executed. In case replicas detect malicious activity by the primary, which might include sending invalid blocks, different blocks of transaction to different replicas, or not sending a block to more than f replicas (which will then complain about it) the primary will be replaced by changing the view.

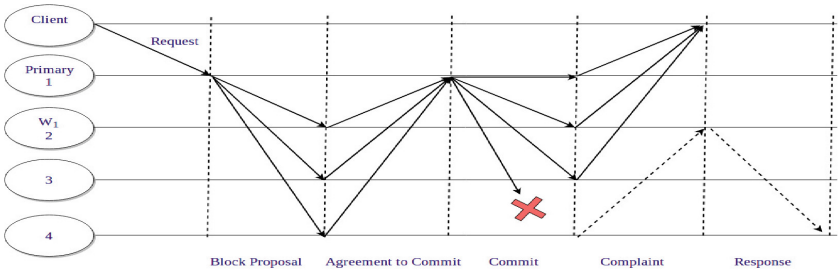

Fig. 6. Musch BFT communication pattern

If failures are encountered replicas switch to recovery mode. In normal BFT protocols if a replica does not receive a pre-prepare/block proposal or receives conflicting or malicious messages from the primary, it broadcasts its complaint ("I hate primary") message. But in case of Musch BFT, a replica will complain to a predetermined subset of replicas called window nodes. Each replica has set of node IDs of other nodes/replicas in the network that are arranged in ascending order. During recovery mode, a replica moves its sliding window of increasing size over the ID list and complains to the nodes in the window. Assuming the complaining replica is i, it will send a $COMPLAINT$ message to the first window (W_1). If it does not receive a $RESPONSE$ before the timeout period ends, the sliding window will double in size and move to the next window W_2 and so on, for sending $COMPLAINT$ messages. If the set $\{1, 2, 3, ..., n\}$ of node IDs is arranged in ascending order, then the sequence of windows over them will be $W_1, W_2, W_3, ..., W_k$ as shown in the Fig. 5. A window W_j will have 2^{j-1} nodes/replicas, where $j \geq 1$. The Musch BFT protocol guarantees that at worst case when the window size reaches $k = \lceil \lg(f' + 1) \rceil$, at least one honest node will be in that window to provide response to replica i. The message exchange pattern of Musch BFT for Blockchains is shown in the Fig. 6.

3 Experimental Evaluation

We implemented the PBFT, Bchain-3, SBFT and Musch BFT consensus protocols for Blockchains in Go, in order to evaluate them. We tested all of the protocols on the Amazon Cloud using EC2 instances of type $t2large$. Each $t2large$ instance contains two virtual CPU cores and 8 GB of memory. We evaluated these protocols on different network sizes ranging from $40, 70, 100, 130,$ to 160 nodes and different block sizes ($5,000$, $10,000$, $15,000$ and $20,000$ transactions). Figure 7 shows latency comparisons and Fig. 8 shows throughput comparisons among the implemented protocols.

Each transaction is a simple blockchain transaction generated randomly that transfers funds from one account to another. Each replica processes messages it receives while also maintaining a local copy of the blockchain. Processing messages includes checking historical transactions, verifying validity of each transaction in the block, checking the block format, and verifying hashes and signatures. We also evaluated the latency and throughput of each protocol when they encounter failures. In this case we assume that the primary is not malicious, thus a view change is not required, but other Byzantine replicas can behave maliciously to delay the system in reaching consensus.

Our results in Figs. 7 and 8 (for window-based BFT) show that network performance is affected by the block size as well as the network size. Larger block sizes ($15k$ and $20k$) normally provide better performance when network size is small (40 and 70). Whereas with smaller block sizes ($5k$ and $10k$) the network capacity (for small network sizes) is under-utilized. Smaller block sizes provide better performance in larger networks (see Figs. 7(a), (b) and Figs. 8 (a), (b)). But in larger networks (130 and 160) larger block sizes also cause a performance bottleneck. Additionally it can be seen that the Musch protocol outperforms PBFT and Bchain for all network sizes. Furthermore, during failures in the network as shown in Figs. 9 and 10, the performance of Musch is negligibly effected. This is because even if malicious replicas try to download more blocks from window nodes, they cannot cause a bottleneck as a window node will only send a requested block once and we have implemented the window node functionality as a separate Go-routine (thread), to leverage concurrency. Thus, overall protocol performance of Musch is not affected by the Window go-routine (addressing complaints from other replicas). Since PBFT is using multiple rounds of broadcast, its performance remains stable as f number of failures cannot further degrade its performance. Bchain-3 on other hand is affected by introduction of failures in the chain. Its performance reduction during failure mainly depends on timeout values. We set the performance threshold timer Δ_1 as $\Delta_1 = 1.10\delta_1$ (as in [5]), where δ_1 is the average time taken by the network to reach consensus. During fault free operation SBFT performance matches that of Musch. But due to

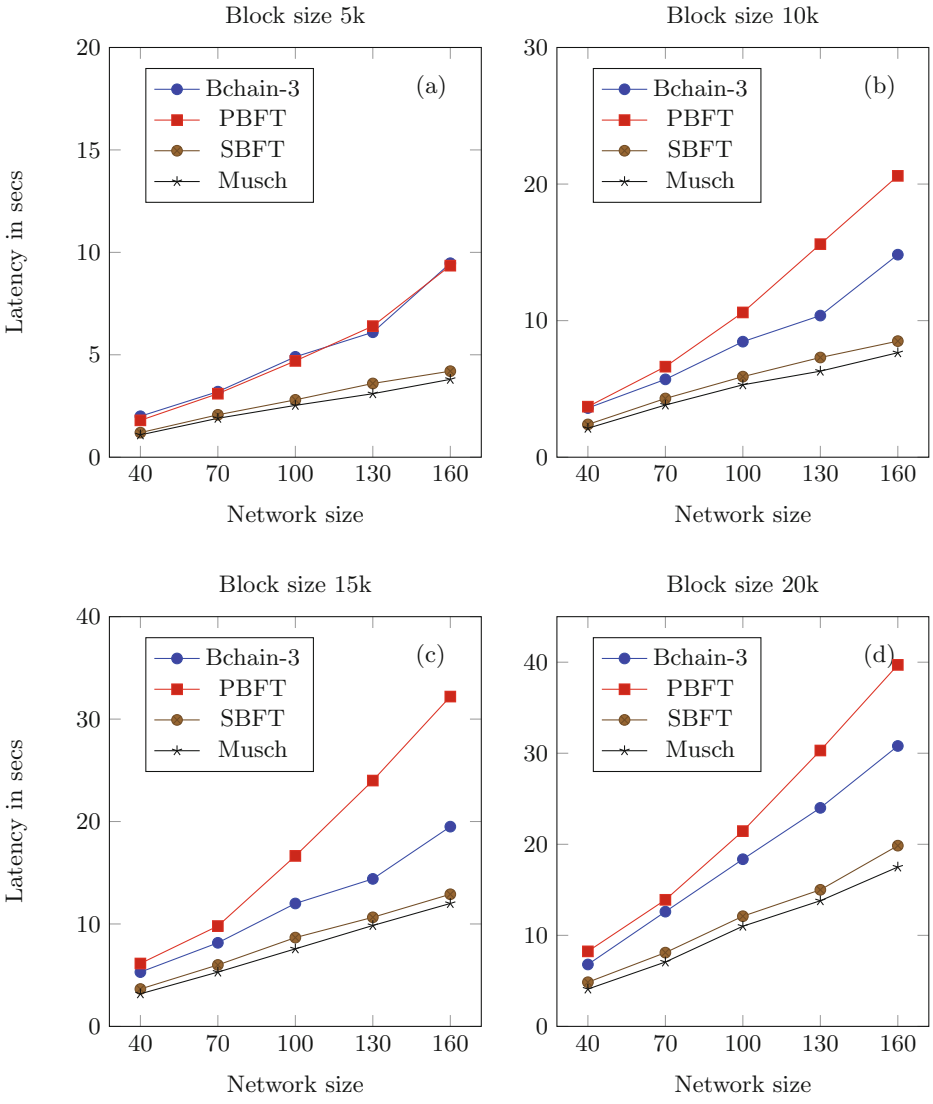

Fig. 7. Latency with block sizes 5000, 10000, 15000 and 20000

falling back to PBFT during failure, SBFT's performance degrades. SBFT has an optimized PBFT fallback protocol but timeout for failure detection (to switch from normal to fallback mode) adds additional latency to it when collectors fail.

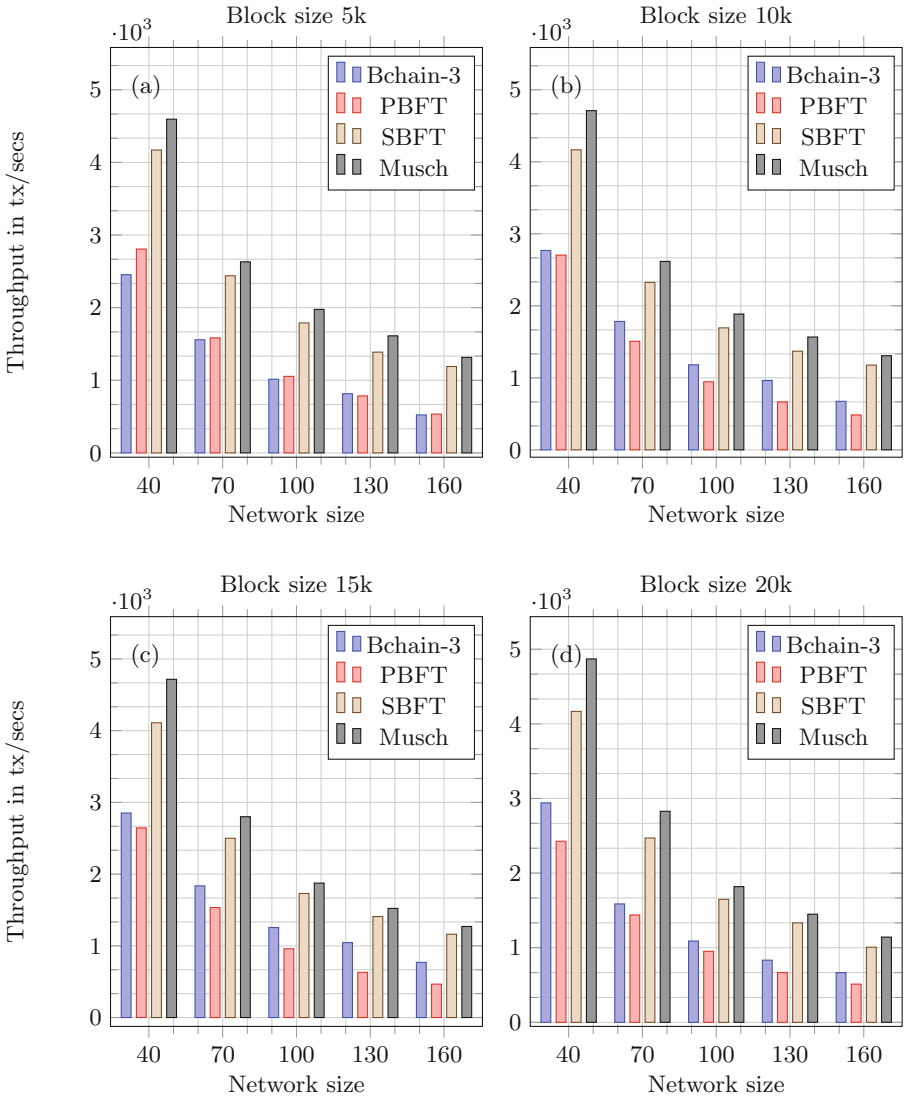

Fig. 8. Throughput with block sizes 5000, 10000, 15000 and 20000

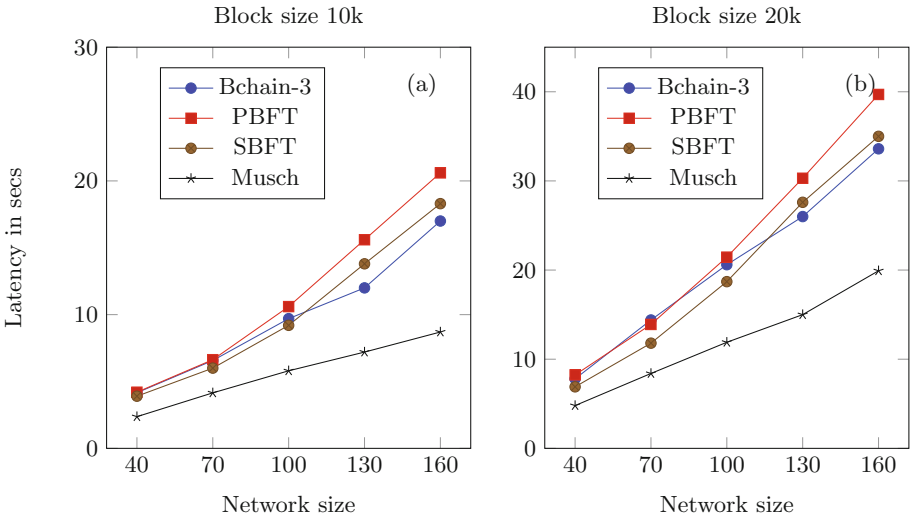

Fig. 9. Latency with failures for block sizes 10000 and 20000

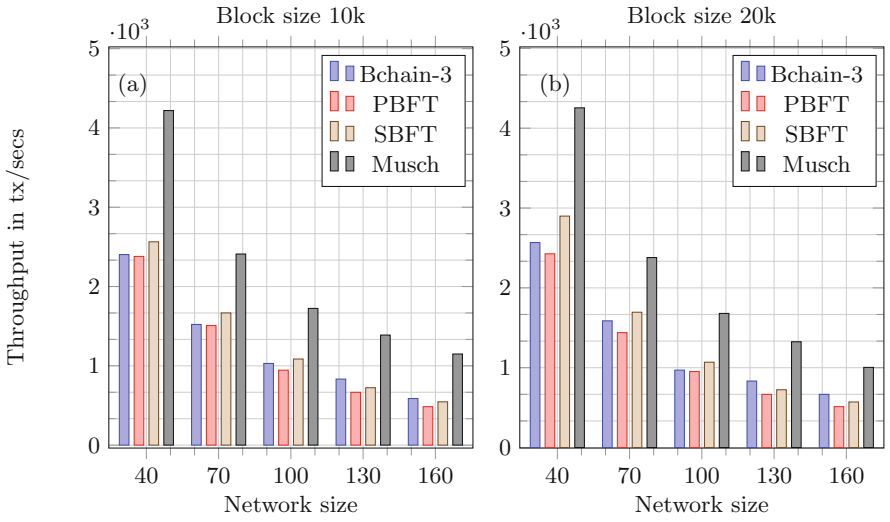

Fig. 10. Throughput with failures for block sizes 10000 and 20000

4 Reliability Analysis Under Attacks

Scalability of a blockchain protocol allows additional nodes to be accommodated and improves decentralization. Seen another way, the ability to add more replicas to the network provides a chance for more parties to be part of the consensus process. Gartner [8], has cautioned against using BFT-based protocols in contexts where Byzantine failures are equated with resilience against malicious attackers.

He argues that Byzantine failure in security is a measurement of reliability [8]. From the FLP impossibility result [6], we know that consensus is impossible if at least one third of replicas in the network are Byzantine ($f \geq n/3$). In practice, we cannot be certain that this assumption will hold. Therefore, Gartner, provides a methodology to estimate the reliability of a Byzantine fault tolerant network (failure assumptions hold in practice), while the network might be under attack by attackers of different strengths. In such a case, the reliability of a Byzantine fault tolerant system can be quantified as:

$$R_C(f,n,t) = \sum_{i=n-f}^{n-C(t)} \binom{n-C(t)}{i} R_{(t)}^{i} (1-R_{(t)})^{n-C(t)-i} \tag{1}$$

where n is total number of replicas in the BFT network, $f < n/3$, and t is a measure of time. Moreover, $C(t)$ defines attacker classes which can be parameterized based on some value p, which is the amount of time for an attacker to take full control of one replica in the network.

For example, a linear class attacker with $p > 0$ is an attacker that compromises each server separately one after another. Such a model assumes replicas uses diverse versions of operating systems and are controlled by different entities. A linear class of attacker can be written as: $C(t) = \min(\lfloor t \cdot p \rfloor, n)$. Logarithmic class attackers can compromise the first replica and then compromising other replicas gets increasingly difficult and follows a logarithmic curve: $C(t) = \min(\lfloor \log_p t \rfloor, n)$. In similar fashion, other classes of attackers like the constant time attacker can be defined. $R_{(t)}$ is the reliability of a single system during a given time t. It can also be called the probability of the system behaving according to expectation at a given time and can be defined as: $R_{(t)} = e^{\lambda \cdot t}$, where failure rate λ is assumed constant for all $t \geq 0$.

By keeping other variables constant (except n and f) in Eq. 1, we can see that the reliability of a BFT network while under attack by an adversary of power p shows more resilience with increase in the number of faults it can tolerate (larger f). Due to the FLP impossibility result [6], $f < n/3$, f can only be increased by increasing n. Thus, greater n will gives higher probability that $f < n/3$ will hold in practice, in the presence of malicious attackers.

To compare reliability R_C in the presence of an attacker $C(t)$, where p is attacker's power and $p = 0.2$, failure rate $\lambda = 10^{-3}$, we considered $1.7k$ tx/sec as desired throughput (D_t). We chose approximate maximum sizes for n_B (number of replicas in Bchain-3 network), n_P (number of replicas in PBFT network), n_S (number of replicas in SBFT network) and n_E (number of replicas in Musch network) that can achieve our desired throughput (with a block size of $10k$) from results presented in Fig. 10(a). Thus, for D_t to be equal or more than $1.7k$, $n_B = 40$, $n_P = 40$, $n_S = 70$ and $n_E = 100$. Similarly, $f_B = 13$, $f_P = 13$, $f_S = 22$ and $f_W = 32$ (maximum number of failures for each protocol) was chosen for each protocol. As shown in Fig. 11, due to its high scalability, the reliability of the Musch protocol shows more resistance to powerful attackers. Then, it is followed by SBFT. On the other hand, both PBFT and Bchain-3 show identical weaker reliability, and by time $t = 70$ they are both compromised and have lost their redundancy.

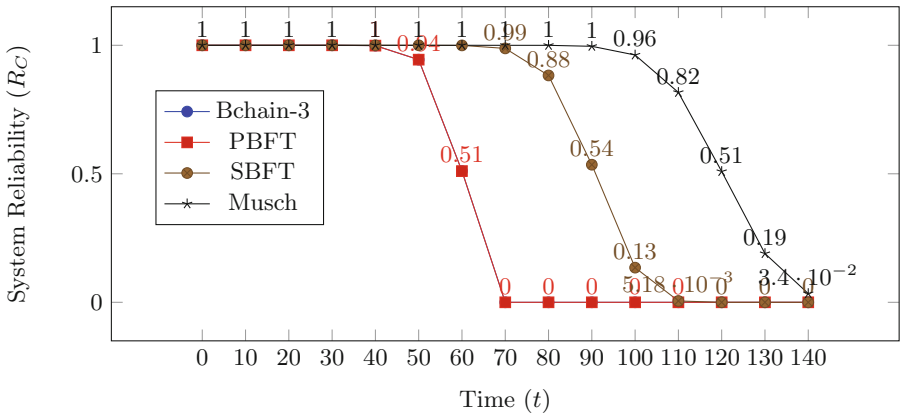

Fig. 11. System reliability in the presence of attacker $C(t)$

5 Conclusion and Future Work

In this paper we presented performance and reliability evaluations for several different BFT protocols in the context of blockchains. We also showed how performance of these protocols is affected when failures are introduced in the network. Additionally, we also calculated the reliability of each protocol, when the network is under attack and showed that a scalable protocol can tolerate more faults and hence offer more resilience to the attacks. Our future work will move focus on developing efficient asynchronous BFT protocols for blockchains and evaluating their performance.

References

1. Buchman, E.: Tendermint: byzantine fault tolerance in the age of blockchains (2016). http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf. Accessed 06 Feb 2017
2. Buterin, V., Griffith, V.: Casper the friendly finality gadget (2017). arXiv:1710.09437. Accessed 06 Nov 2017
3. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI 1999, Berkeley, USA, pp. 173–186. USENIX Association (1999). http://dl.acm.org/citation.cfm?id=296806.296824
4. Pike, D., Nosker, P., Boehm, D., Grishm, D., Woods, S., Marston, J.: Proof-of-stake-time (2015). https://www.vericoin.info/downloads/VeriCoinPoSTWhitePaper10May2015.pdf. Accessed 12 Mar 2019
5. Duan, S., Meling, H., Peisert, S., Zhang, H.: BChain: byzantine replication with high throughput and embedded reconfiguration. In: Aguilera, M.K., Querzoni, L., Shapiro, M. (eds.) OPODIS 2014. LNCS, vol. 8878, pp. 91–106. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14472-6_7

6. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985). https://doi.org/10.1145/3149.214121
7. Garay, J.A., Moses, Y.: Fully polynomial byzantine agreement for n>3t processors in $t+1$ rounds. SIAM J. Comput. **27**(1), 247–290 (1998). https://doi.org/10.1137/S0097539794265232
8. Gärtner, F.C.: Byzantine failures and security: arbitrary is not (always) random. In: INFORMATIK 2003 - Mit Sicherheit Informatik, Schwerpunkt "Sicherheit - Schutz und Zuverlässigkeit", 29 September–2 Oktober 2003 in Frankfurt am Main, pp. 127–138 (2003). http://subs.emis.de/LNI/Proceedings/Proceedings36/article1040.html
9. Golan-Gueta, G., et al.: SBFT: a scalable decentralized trust infrastructure for blockchains. CoRR abs/1804.01626 (2018)
10. Jalalzai, M., Busch, C.: Window based BFT blockchain consensus. In: 2018 IEEE International Conference on Blockchain (Blockchain 2018), Halifax, Canada (2018)
11. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
12. Luu, L., Velner, Y., Teutsch, J., Saxena, P.: Smartpool: practical decentralized pooled mining. In: 26th USENIX Security Symposium (USENIX Security 2017), Vancouver, pp. 1409–1426. USENIX Association (2017). https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/luu
13. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf
14. Canneti, R., Rabin, T.: Optimal synchronous byzantine agreement. Technical report (1992)
15. Reiter, M.K.: Secure agreement protocols: reliable and atomic group multicast in rampart. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security, CCS 1994, New York, USA, pp. 68–80. ACM (1994)
16. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-of-stake (2012). https://peercoin.net/whitepapers/peercoin-paper.pdf. Accessed 12 Mar 2019
17. Vasin, P.: Blackcoin's proof-of-stake protocol v2. https://blackcoin.org/blackcoin-pos-protocol-v2-whitepaper.pdf. Accessed 12 Mar 2019
18. Wood, D.G.: Ethereum: a secure decentralised generalised transaction ledger (2017). https://ethereum.github.io/yellowpaper/paper.pdf
19. Wüst, K.: Security of blockchain technologies (2016). http://e-collection.library.ethz.ch/eserv/eth:49632/eth-49632-01.pdf. Accessed 08 Feb 2019

Author Queries

Chapter 3

Query Refs.	Details Required	Author's response
AQ1	This is to inform you that corresponding author has been identified as per the information available in the Copyright form.	
AQ2	As Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city name "Baton Rouge" in affiliations 1 and 2. Please check and confirm if the inserted city name is correct. If not, please provide us with the correct city name.	