MobBFT: a Client-centric State-based BFT Platform for Decentralized and Resilient Community Web-Apps

Anonymous Author(s)

ABSTRACT

The web is shifting to a client-centric, decentralized model where web clients become the leading execution environment for application logic and data storage. However, current solutions to build decentralized web applications with multiple distrusting parties often involve a separate backend blockchain network. In this paper, we present MobBFT, a browser-based platform for decentralized BFT consensus in client-centric, community driven applications. MobBFT runs entirely in the browser, alleviating the need to use an expensive public blockchain or set up a complex private blockchain. We propose a novel, optimistic, leaderless consensus protocol, tolerating Byzantine replicas, combined with a robust and efficient state-based synchronization protocol. This protocol makes MobBFT well suited for the decentralized client-centric web and its dynamic nature with many network disruptions or node failures. Using a state-based protocol, no transaction log is stored, keeping the overall storage footprint small for client-centric devices. MobBFT uses an optimized implementation of the standard BLS scheme for efficient aggregation and storage of signatures. Our performance evaluation shows that MobBFT can achieve transaction finality within seconds in community-driven networks of mobile web clients, even in the context of network problems, node failures, and Byzantine behavior.

KEYWORDS

Peer-to-peer Systems, Byzantine Fault Tolerance, Web Applications

1 INTRODUCTION

Browsers and client-side web technologies offer increasing capabilities to enable fully client-side web applications that can operate independently and in a stand-alone fashion, in contrast to the server-centric model [8, 33]. Web 3.0 can be defined as the decentralized web where users are in control of their data [17], and that replaces centralized intermediaries with decentralized networks and platforms [31, 89]. Community-driven, decentralized networks can open the road to many use cases for the sharing economy [9, 54, 73] or shared loyalty programs for local communities [10, 32]. Such client-centric collaborations can, for example, enable a small network of merchants in a local shopping street, or at a farmer's market to set up a shared loyalty program between the merchants in an ad-hoc fashion. These small-scale, specialized collaborative networks can empower motivated citizens to bring value to their local community, without involving an incumbent big-tech company that can change the rules unilateral at any moment.

However, current state-of-the-art peer-to-peer data synchronization frameworks for the browser such as Legion [82], Yjs [66], Automerge [44], and *Anonymized* [11] focus on full replication and eventual consistency between trusted clients. Each replica can modify all data, and all modifications are automatically replicated to all replicas. These protocols lack Byzantine Fault Tolerance (BFT). Yet, they are easy to set up and *trusted* parties can quickly use these to synchronize and modify a shared data set between them.

Decentralized interactions between *distrusting* parties can be enabled by using a classical BFT consensus protocol such as PBFT [26], BFT-SMaRt [18], Tendermint [23], Algorand [34], Ouroboros [43], or HotStuff [87]. These classical BFT protocols are very fast and have a high throughput, but typically assume server-to-server communication with low-latency network connections, and assume every node is connected to all other nodes. Nakamoto consensus [64], used in several blockchains such as Bitcoin and Ethereum [25], relaxes this requirement and only requires a loosely coupled network. However, blockchains based on Nakamoto consensus are too slow for many use cases. They need minutes, or even an hour, to confirm a transaction with high probability. Moreover, they consume a large amount of energy and need a lot of processing power. At last, Avalanche consensus [74] tries to solve the scalability problem by using the concept of meta-stability. Only a small subset of replicas need to be sampled to reach consensus, however, you still need a connection to every other replica, as the replicas that you need to sample change continuously.

Ultimately, a decentralized web application should be able to run in a robust and resilient way over a network of online client devices such as smartphones. Such devices have a permanent yet unstable internet connection over a data subscription, and are operational and reactive most of the time. However, the existing BFT consensus protocols are designed for more server-like infrastructure that has lots of processing power, storage space, and a stable, low-latency network connection. The motivated citizens in our envisioned use cases do not have this kind of knowledge, budget, and infrastructure available to set up a private network of servers running a BFT protocol between them. They rather want to use their existing hardware such as a low-end computer, or even a mobile device. They could use a public blockchain network, at the cost of paying a fee for every transaction, which lowers the economic viability of this approach. A private network between the citizens without fees is more suitable. This also has the advantage that not all data is publicly readable by the whole world. Even if you conclude you need a blockchain [86], a more lightweight approach to consensus can be more appropriate [10].

In this paper, we present MobBFT, a novel peer-to-peer data synchronization framework for decentralized web applications between mistrusting parties. MobBFT combines the efficient operation and lightweight setup of a peer-to-peer data synchronization framework with the resilience and fault tolerance of a BFT consensus protocol. The novel BFT protocol, optimized for unstable network conditions, does not require that all replicas are connected to each other. It also does not rely on a leader, removing the need for a costly leader-election procedure when this leader is malicious or loses its network connection temporary. The latter scenario is common in our target environment. Each browser replica only maintains the current authenticated state, and does not need to keep track of an operation log or transaction history, keeping the storage footprint small. To further reduce the storage and bandwidth requirements, we use an aggregate signature scheme called BLS [21]. This also reduces the computational requirements when all replicas are honest, as only a single aggregate signature has to be verified. The authenticated state and consensus votes are replicated over multiple hops using a gossip protocol.

To summarize, MobBFT has the following contributions[1]:

(1) Lightweight, leaderless, client-centric Byzantine fault tolerant synchronization and consensus.
(2) Robust, state-based synchronization of both the data and the votes for the consensus protocol using state-based CRDTs and Merkle-trees.
(3) Optimistic fast path when nobody is acting Byzantine, gracefully degrading to the slow path when under attack.
(4) Efficient computation and compact storage of signatures using the BLS signature scheme.

Our evaluation, using our application use case of a shared loyalty program between small-scale merchants, shows that MobBFT is a practical solution for these kinds of community-driven use cases. MobBFT achieves transaction finality in the order of seconds, even in networks with 100 browser clients, or in unstable network conditions. No complex infrastructure is required, the participating merchants only need a browser and an internet connection.

Section 2 further discusses some motivating use cases and background in more depth. Section 3 presents MobBFT's lightweight BFT consensus protocol and the state-based replication strategy. The detailed web-based middleware architecture of MobBFT is elaborated in Section 4. Our evaluation in Section 5 focuses on many aspects of performance in both the optimistic scenario as well as more realistic and even Byzantine scenarios. Section 6 elaborates on important related work. We conclude in Section 7.

2 MOTIVATION AND BACKGROUND

This section further motivates the need for a lightweight, robust consensus middleware by describing several community-driven use cases. Then we give some background on state-of-the-art approaches using a blockchain and BFT consensus.

2.1 Motivational use cases

We describe two initial use cases that would benefit from the lightweight consensus offered by MobBFT. They both involve business transactions happening in real life and need interactive performance and robustness, rather than high throughput or scalability. We then formulate our vision on decentralized web applications.

Sharing economy. Small communities, such as an apartment building or local neighborhood, can share tools or cars [54] with each other using a peer-to-peer platform to keep track of the current possession and reservation of tools and cars [73]. When a tool is being exchanged, it is checked for potential damage which can be registered in the network.

Loyalty programs. Integrated loyalty programs can be more effective than traditional loyalty programs that are limited to a single company [32]. Think about airlines that award *miles* which can be redeemed with several partners. Such collaborations usually introduce an extra trusted intermediary and add more layers of management and operational logistics. This trusted party can charge high transaction costs to be part of the integrated network. For small merchants on a farmer's market or in a local shopping street, this operational overhead is too much of a burden. A decentralized peer-to-peer network can enable fast and secure creation, redemption, and exchange of loyalty points across different merchants.

Vision. We envision that communities will be able to use MobBFT as a platform to explore new applications and use cases that were previously not feasible. While our initial proof-of-concept implementation is targeting the browser, the techniques explained in this paper can be easily ported towards native mobile and lightweight desktop applications. MobBFT does not need any complex infrastructure, and it currently provides a simple JavaScript-based API, which allows many developers to start developing decentralized applications. Those decentralized applications can be made open source, which allows many people to verify and vouch for them. Local communities who want to set up a decentralized application between the local participants, can use such an open-source application and do not need to concern themselves with a complex infrastructure setup to run the application. Nor do they need to rely on a third party general purpose public blockchain network.

2.2 Background on BFT consensus

Existing blockchains can be roughly split into two categories: public and permissioned blockchains. Public blockchains are open for everyone to participate in. Two examples are Bitcoin [64] and Ethereum [25]. Bitcoin allows everyone to host a replica node and submit transactions. However, Bitcoin is quite slow, as a new block is only created every 10 minutes on average. This means that transactions take on average 10 minutes to be confirmed by the network. But as multiple conflicting chains can occur, one must wait for at least 6 blocks to be sure that a transaction will not be reverted. This increases the total latency to one hour, which is too slow for many of the motivational use cases. Ethereum is another public blockchain with a much faster average block time, and consequently a lower latency. Ethereum allows everyone to write *smart-contracts* to be executed by the Ethereum network. Each invocation of a contract costs a small amount of Ether (called gas). This makes Ethereum infeasible for small business transactions such as loyalty points, as the total cost will become too high.

Permissioned or private blockchains use access control to limit who can see and create transactions on the blockchain. Because they can only be accessed by a limited number of known parties, transaction fees are not required to reward miners and combat spam. An example is Hyperledger Fabric [3]. These private blockchains can use a Byzantine fault tolerant consensus protocol to reach consensus over which transactions to execute and in which order. They have much smaller latency and can process more transactions per second compared to the public blockchains. However, to set up Hyperledger Fabric in a decentralized fashion, there is a large back-end infrastructure required. The actual blockchain network consists of many nodes: peers, orderers, REST-API servers, database servers, and a certificate authority. Setting up and managing these

[1]A preliminary workshop paper [10] already described our initial goal, the use case of loyalty points and an initial idea for a solution.

services requires a lot of infrastructural management for small merchants. They do not have the knowledge nor budget for such a deployment, especially considering the maintenance overhead and resource costs. These small merchants want to quickly set up an integrated loyalty network with minimal back-end setup. However, most of them already own a desktop or a mobile computer such as a laptop or tablet.

Two existing state-of-the-art protocols for BFT consensus, which we will use in our comparative evaluation, are BFT-SMaRt [18] and Tendermint [23, 24]. BFT-SMaRt is a more traditional BFT protocol, similar to PBFT [78], where all replicas are connected to each other, and one leader drives the protocol. If that leader fails, a new one will have to be elected before any progress can be made. BFT-SMaRt can be used in Hyperledger Fabric [79]. Tendermint [24] uses Gossip for communication between the replicas. There is still a leader, however, that leader changes frequently. Tendermint is used in the Cosmos blockchain [48].

3 OPTIMISTIC STATE-BASED BFT

This section explains the state-based consensus protocol used in MobBFT. First, it describes the adversary model and its properties. Then it explains the protocol specification. At last, this section provides safety and liveness proofs.

3.1 Overview and adversary model

The core protocol is an asynchronous, leaderless, Byzantine fault tolerant consensus protocol. In an asynchronous network, messages are eventually delivered, but no timing assumption is made [29]. An adversary might also corrupt up to f replicas of the $n \geq 3f + 1$ total replicas. They can deviate from the protocol in any arbitrary way. Such replicas are called Byzantine, while the replicas that are strictly following the protocol are called honest. We assume attackers are computationally bounded and cannot forge the used asymmetric signatures or find collisions for the used cryptographic hash functions.

The protocol is used to implement an Atomic Register [49] that can hold a single value that can be read and written by multiple replicas. All writes are atomic, meaning that only a single state transition can happen at any time. Extra conditions can be applied to limit who can write to it, and which values are acceptable. MobBFT does not use a leader to coordinate the protocol, removing a common single-point-of-failure compared to many existing BFT protocols. In such leader-based protocols, the failure of a leader leads to a long delay before consensus can be reached. The consensus protocol presented here uses voting, where every replica has exactly one vote. The set of replicas is fixed, and changes to the replica set have to be made outside the protocol. Consensus is reached for each register separately, which means that each register has its own instance of the MobBFT protocol.

Formal properties. Let $\mathfrak{R}$ be a cluster of n replicas with f Byzantine replicas and $n \geq 3f + 1$. MobBFT guarantees the following properties:

- **Non-divergence:** If replicas $R_1, R_2 \in \mathfrak{R}$ are able to construct quorum certificates qc_1 for value val_1 and qc_2 for value val_2 at version v, then $val_1 = val_2$.
- **Termination:** If an honest replica $R \in \mathfrak{R}$ proposes a new value at version v, eventually a replica will be able to construct a quorum certificate qc for *some* value at version v.

The first property is a safety property and guarantees that all state changes are atomic for the whole network. The second property is a liveness property and guarantees that non-conflicting transactions will be eventually executed by all replicas. Notice that the value that is committed in this property is not necessarily the originally proposed value. It is not guaranteed that a value will be committed, as long as other concurrent values are proposed as well.

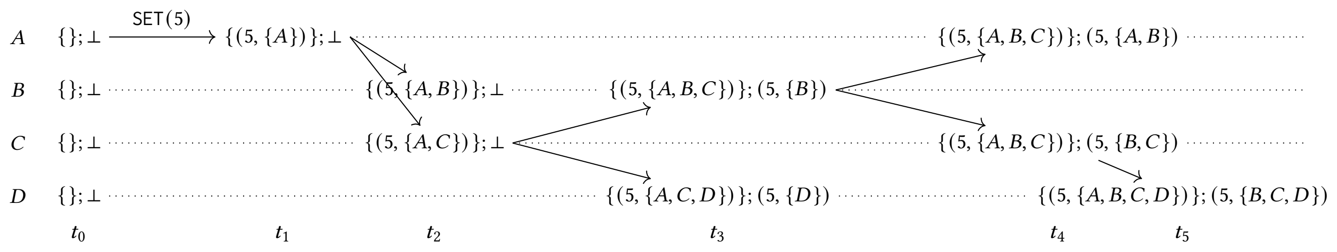
3.2 Protocol specification

The specification of the protocol is shown in Algorithm 1 and 2. Each atomic register has its own state which consists of three parts. The first part is the current value and a quorum certificate. The quorum certificate contains signatures of a supermajority of $n - f$ replicas, and proves the validity of the value. The second part is a map, which maps rounds to a collection of votes for the next value. In each round, there can be multiple proposed values. The third part consists of a new proposed value and a partial quorum certificate for that value. This state is shown at the first 5 lines of Algorithm 2.

Consensus is reached in two steps, first a supermajority needs to be reached in the last round of the *votes*, then a supermajority needs to be reached for the proposed quorum certificate. The first step will establish a resilient quorum, while the second step will guarantee that sufficiently many replicas know that such a quorum has been achieved.

State-based replication protocol. The current value and its quorum certificate, and the votes and proposal when present, are replicated by using a state-based Gossip protocol. This protocol is a peer-to-peer version of *Anonymized* [11], which uses state-based Conflict-free Replicated Data Types (CRDTs) [77] combined with a Merkle-tree [59] to efficiently replicate the updated state. The CRDTs being used are Observed-Removed Maps [11] and Grow-only Sets [77]. There are extra constraints imposed on the CRDTs due to the Byzantine nature. The Merkle tree is used to efficiently replicate the state between any two replicas. If the state of two replicas is the same, only the root hash is sent and compared, which limits the network usage. If the states differ, the protocol descends in the tree looking for mismatching hashes to find out which registers must be synchronized. By using a state-based approach, rather than the operation-based approach of Operational Transformation [30], operation-based CRDTs [77], or blockchains [64], we only need to store the current state together with some metadata. There is no need to store the full log of all operations to later convince replicas that were temporarily offline of the new state. Replicas also do not need to keep track of the state of other replicas, or which messages are already received by which replica [2]. If a new value and quorum certificate with a higher version are received, then the protocol will accept the new state, and the protocol will reset back to line 3 of Algorithm 2 with that newer version.

The replicas execute a gossip protocol to exchange their current state with each other. Each time a new state is received, the local state is merged with the remote state. An example of this replication process is shown in Figure 1. There are four non-Byzantine replicas



A
B
C
D
SET(5)
{}; ⊥
{(5, {A})}; ⊥
{(5, {A, B})}; ⊥
{(5, {A, C})}; ⊥
{(5, {A, B, C})}; (5, {B})
{(5, {A, C, D})}; (5, {D})
{(5, {A, B, C})}; (5, {A, B})
{(5, {A, B, C})}; (5, {B, C})
{(5, {A, B, C, D})}; (5, {B, C, D})
t_0
t_1
t_2
t_3
t_4
t_5

Figure 1: State-based synchronization of an Atomic Register with 4 replicas A, B, C, D. Only the current $votes[0]$ and $proposal$ are shown for brevity. $Version$ and $round$ are not shown as they stay always the same in this example.

with an empty set of votes. Each item lists the value and the set of signatures of the replicas that voted for it. The scenario starts at t_1 with replica A proposing a new value (line 7-8 of Algorithm 2). The state is replicated to the other replicas randomly, and all replicas collect the votes in the set of signatures. For example, at t_2, replica B and C will vote for the current winning value as they did not yet vote (line 10-15 of Algorithm 2).

Note that we do not explicitly show the gossiping in Algorithm 2 to keep the algorithm compact. During all phases in the algorithm, the state is continuously replicated to the other replicas.

Algorithm 1 Utilities (for replica r).

```
1:  function WINNINGVALUE($votesInRound$)
2:      return $argmax_{val}$LEN($\{v \in votesInRound : v.val = val\}$)
3:  function VOTESFORVALUE($votesInRound, val$)
4:      return $\{v \in votesInRound : v.val = val\}$
5:  function HASVOTED($votesInRound$)
6:      return $\exists\, v \in votesInRound : v.r = r$
7:  function VOTE($version, round, val, type$)
8:      $vote \leftarrow$ VOTE($val, r$)
9:      $vote.signature \leftarrow$ SIGN($version, round, val, type, r$)
10:     return $vote$
```

Reading and writing. When reading the value of a register, it will return the currently accepted value. This request is always executed on the local replica and does not involve any network requests. To write a new value, a replica has to propose a new value to the other replicas. This process is the PREPARE phase in Algorithm 2. The proposing replica adds the new value and its vote to round 0 of *votes*. As the protocol is leaderless, any replica can be a proposing replica and multiple replicas can propose a new value simultaneously. Replicas are only allowed to vote once in each round for each version, so if the replica already voted for another value in that round, it will have to wait until consensus is reached for the current set of *votes*, and propose the new value for the version after it.

Consensus. Consensus about which value will be accepted for a version is reached in two phases, called PRE-COMMIT and COMMIT in Algorithm 2. Honest replicas will always vote for the value with the most votes in round 0. If a round has reached a supermajority of votes for a single value, then no new round can be started anymore, and the replicas will start creating a new proposed quorum certificate. If a supermajority of the replicas have voted, but not a single

Algorithm 2 Basic protocol (for replica r).

```
1:  $value \leftarrow \bot$
2:  $commitQC \leftarrow \bot$
3:  for $version \leftarrow 1, 2, 3, \ldots$ do
4:      $votes \leftarrow \emptyset$                          ▷ $round \mapsto votesInRound$
5:      $proposal \leftarrow \bot$
    ▷ PREPARE phase
6:      as a proposing replica:
7:          wait for value $val$ from client
8:          $votes[0] \leftarrow \{$VOTE($version, 0, val,$ PRE-COMMIT)$\}$
9:      as a non-proposing replica:
10:         wait for any value in $votes$
11:     for $round \leftarrow 0, 1, 2, 3, \ldots$ do
    ▷ PRE-COMMIT phase
12:         if $\neg$HASVOTED($votes[round]$) then
13:             $val \leftarrow$ WINNINGVALUE($votes[0]$)
14:             $vote \leftarrow$ VOTE($version, round, val,$ PRE-COMMIT)
15:             $votes[round] \leftarrow votes[round] \cup \{vote\}$
16:         wait for $(n - f)$ votes in $votes[round]$
17:         $val \leftarrow$ WINNINGVALUE($votes[round]$)
18:         $valVotes \leftarrow$ VOTESFORVALUE($votes[round], val$)
19:         if LEN($valVotes$) $\geq (n - f)$ then
20:             $proposal \leftarrow$ PROPOSAL($val$)
21:             $proposal.qc \leftarrow \{$VOTE($version, round, val,$ COMMIT)$\}$
22:         else
23:             $val \leftarrow$ WINNINGVALUE($votes[0]$)
24:             $vote \leftarrow$ VOTE($version, round + 1, val,$ PRE-COMMIT)
25:             $votes[round + 1] \leftarrow \{vote\}$
26:             continue
    ▷ COMMIT phase
27:         wait for $(n - f)$ votes in $proposal.qc$:
28:             if LEN($votes$) $- 1 > round$ then
29:                 $proposal \leftarrow \bot$
30:                 continue
31:         $value \leftarrow proposal.val$
32:         $commitQC \leftarrow$ QC($version, round, proposal.qc$)
```

value reaches a supermajority, a new round is started and all replicas can vote again in this new round. The replicas are only allowed to vote on the current winner in round 0 in their view. Because each replica might have different views on the current set of votes in round 0, there can still be multiple values in the next round without any supermajority for a single value. Another factor is Byzantine nodes trying to halt the system by voting not according to the rules. However, the set of possible values to vote on gets smaller with every round, and eventually the view of all the replicas on the votes in round 0 will become the same, and the winning value can be chosen unanimously. The reason for this is that a replica will not only gossip $votes[i]$ in a certain round, but also all votes for the previous rounds. This means that when two replicas disagree with each other in a certain round, once they communicate with each other, they will learn each other's state and in the next round they will both vote for the same value (as their views on $votes[0]$ will be the same). Malicious replicas can try to shift the balance to violate liveness, but with each round they have less possibility to do so, because when they gossip $votes[i]$ they also gossip the previous rounds which should show why they voted on a certain value. If a replica detects that another replica is Byzantine, it will exclude this Byzantine replica permanently, and its votes do not count anymore. A replica can act Byzantine by sending invalid state, invalid signatures, or by voting on a value which can impossibly be the winner in round 0. We prove the correctness of this approach in Section 3.3.

Once a replica observes that a supermajority of the replicas supports a single value, it starts working on a proposed quorum certificate to determine if at least a supermajority of the replicas also knows about this. In the example in Figure 1, at t_3 both replica B and replica D observe a supermajority for value 5, and they start creating a new proposed quorum certificate. At t_5, replica D has a proposed quorum certificate signed by a supermajority of the replicas. This means that the new value 5 can be committed. The proposed quorum certificate becomes the new quorum certificate and the $votes$ are removed. When another replica now receives the state of replica D, that replica will notice that it has a value associated with a valid quorum certificate with a larger version number as his own. Therefore, it will accept this new value and remove all of its own votes and the proposed certificate if any.

Optimistic fast path. For brevity, we did not show the actual verification of signatures in Algorithm 2. However, in the basic protocol, each time a new signature is received, it needs to be verified. This can become quite costly, and therefore MobBFT will use an optimistic approach. MobBFT will delay the verification of any incoming signatures and will just accept and replicate them, until a decision needs to be made, such as starting a new round or starting to create a new proposed quorum certificate. Only then, all signatures will be verified in one batch. If all signatures are valid, the protocol can continue as normal. If there are invalid signatures, then those will be removed and MobBFT will continue to collect more signatures. However, MobBFT will remember this occurrence and from now on verify all signatures once they come in. Once consensus is reached for this version, MobBFT will move back to the optimistic fast path. This hybrid approach enables very fast consensus when all replicas are honest, while gracefully degrading to a slower, more costly protocol that can detect which replicas are actively acting Byzantine.

3.3 Safety and Liveness

This section sketches the proof that the algorithm provides safety and liveness. The protocol described before guarantees both safety and liveness when there are at least $2f + 1$ honest replicas available.

3.3.1 Safety.

The safety property is defined as *non-divergence.*

Lemma 3.1 (Non-divergence). *Let $\mathfrak{R}$ be a cluster of n replicas with f Byzantine nodes and with $n > 3f$. If replicas $R_1, R_2 \in \mathfrak{R}$ are able to construct quorum certificates qc_1 and qc_2 for value val_1 and val_2 respectively with $qc_{1\ version} = qc_{2\ version}$, then $val_1 = val_2$.*

We will first prove this for the simplified case when both quorum certificates belong to the same round, and we will then prove that once a quorum certificate can be constructed, no more rounds can be started.

Lemma 3.2. *If replicas $R_1, R_2 \in \mathfrak{R}$ are able to construct quorum certificates qc_1 and qc_2 for value val_1 and val_2 respectively with $qc_{1\ version} = qc_{2\ version}$ and $qc_{1\ round} = qc_{2\ round}$, then $val_1 = val_2$.*

Proof. Assume two different replicas R_1 and R_2 have constructed a quorum certificate qc_1 and qc_2 for value val_1 and val_2 respectively with $qc_{1\ version} = qc_{2\ version}$ and $qc_{1\ round} = qc_{2\ round}$. They are constructed in the same round, so of the n possible votes, at least $n - f$ replicas have voted on val_1, and at least $n - f$ replicas have voted on val_2. Honest replicas will never vote twice in the same version and round. Therefore, at least $n - 2f$ honest replicas have voted on val_1 and $n - 2f$ *different* honest replicas have voted on val_2. In total, we have $(n - 2f) + (n - 2f) + f \equiv 2n - 3f$ replicas that have voted. We defined $n \geq 3f + 1$ before, which gives $2n - 3f \geq 3f + 2 \geq n + 1$ replicas. This is a contradiction, there need to be at least $n + 1$ replicas to construct two such certificates for different values, however, we only have n replicas. So the two values val_1 and val_2 have to be equal. □

Lemma 3.3. *If replicas $R_1, R_2 \in \mathfrak{R}$ are able to construct quorum certificates qc_1 and qc_2 for value val_1 and val_2 respectively with $qc_{1\ version} = qc_{2\ version}$, then $qc_{1\ round} = qc_{2\ round}$.*

Proof. Assume two different replicas R_1 and R_2 have constructed a quorum certificate qc_1 and qc_2 for value val_1 and val_2 respectively with $qc_{1\ version} = qc_{2\ version}$ and $qc_{1\ round} < qc_{2\ round}$. Since qc_1 is accepted, at least $n - f$ replicas vote on the proposed quorum certificate and at least $n - f$ replicas voted on val_1 in round $qc_{1\ round}$. The fact that $n-f$ replicas voted on the proposed quorum certificate means that at least $n - 2f$ honest replicas observed $n - f$ votes for val_1. Of those votes, at least $n - 2f$ are coming from honest replicas. The only way to now construct a quorum certificate qc_2 for val_2 is to start a new round. To start a new round, a replica needs to not have voted for the proposed quorum certificate qc_1, and observe a different winning value val_2. Yet, at least $n - 2f$ honest replicas observed that at least $n - 2f$ honest replicas think that val_1 is the winning value. This leaves only $2f$ replicas who can prefer another value val_2. By definition we have $n \geq 3f + 1$. This means that in the worst case, $f + 1$ honest replicas observe $f + 1$ honest replicas thinking val_1 is the winning value, together with f Byzantine replicas. Value val_2 has only $2f$ supporting replicas, which is not enough to start a proposed quorum certificate. So, at least one replica currently supporting val_1 needs to switch votes in a future round. However, once a replica has voted for a proposed quorum certificate, it will not change their opinion unless it is convinced that a new valid round is started. So once $n - 2f$ honest replicas are locked on a value, by voting on a proposed quorum certificate, it is impossible that a valid new round can be started. □

3.3.2 Liveness. The liveness property is defined as *termination*. When a new value is proposed, eventually the protocol will end and a valid quorum certificate is created for a new value. This value is not necessarily the first proposed value, and it is not even guaranteed that a specific value ever gets committed as long as other values continue to be proposed. Safety is always chosen over liveness. When there are not enough honest replicas online to reach a supermajority, no consensus can be reached and the protocol will simply block and wait for more votes. However, all those replicas do not need to be online at the same time, since the state is replicated to all available replicas over time, and votes can be verified by all replicas in the end.

Lemma 3.4 (Termination). *If an honest replica $R \in \mathfrak{R}$ creates a proposal p for a new value val, eventually the replica will be able to construct a valid quorum certificate qc.*

Lemma 3.5. *If only a single replica $R \in \mathfrak{R}$ creates a proposal p for a new value val, eventually the replica will be able to construct a valid quorum certificate qc.*

Proof. As there is only a single proposed value, all honest replicas who observe this will cast their vote for that value. Eventually, one replica will observe $n - f$ votes for val and a new proposed quorum certificate will be constructed. Eventually, $n - f$ votes will be cast to this proposed quorum certificate and a valid quorum certificate qc is constructed and val is committed. □

Lemma 3.6. *If x replicas $R_{1..x} \in \mathfrak{R}$ create proposals $p_{1..x}$ for values $val_{1..x}$, and no Byzantine replicas vote twice in the same round, eventually the replica will be able to construct a valid quorum certificate qc.*

Proof. Either a single value reaches a quorum, in which case the previous lemma holds. Or a split vote occurs and a new round will be started after $n - f$ votes are observed. All replicas will base their vote for this new round on the winning value that they observed from round 0. At least $n - f$ votes are known, and only f votes are still unknown. As long as not all votes are known to all voting replicas, the winning value might change. In each new round, either an unknown vote stays unknown, or it becomes known. In the former case, then the currently known votes will all be the same, and a proposed quorum certificate can be started. In the latter case, one extra vote is known, which might again result in the system ending up in a split vote, and a new round will be started. However, this last case can only happen at most f times. After $f + 1$ rounds, all replicas will have voted in round 0, and every replica will observe the same winning value, and a quorum certificate can be created. □

Lemma 3.7. *If x replicas $R_{1..x} \in \mathfrak{R}$ create proposals $p_{1..x}$ for values $val_{1..x}$, eventually the replica will be able to construct a valid quorum certificate qc.*

Proof. If no Byzantine replicas vote twice in the same round, or only a single value is proposed, the previous two lemmas hold. If a split vote occurs, a new round will be started after $n - f$ votes are observed. f of those votes might belong to Byzantine replicas who can vote for multiple values. As a new round is only started after $n - f$ votes, a least $n - 2f$ honest votes are observed. No Byzantine replica can send conflicting votes to any of those $n - 2f$ honest replicas, as otherwise those replicas will detect this conflicting vote and exclude the Byzantine replica. If this happens repeatedly, at most f times, all Byzantine replicas are excluded and the previous lemma holds. Moreover, no Byzantine replica can continue to vote on values that are not the winning value. Each replica is only allowed to vote on the winning value or any other value that has at least support from $f + 1$ replicas in the previous round. All honest replicas converge to a single value, even with Byzantine replicas supporting other values. Because the protocol only looks to the first round to determine the winning value. In the rounds after that, the f Byzantine replicas can support a different value, but if they do, they will be excluded as $f < f + 1$. This means that after at most $2f + 1$ rounds, a proposed quorum certificate can be started, which will be committed. □

4 ARCHITECTURE AND IMPLEMENTATION

This section describes the architecture, deployment, and implementation of MobBFT. This middleware architecture is key to support the BFT consensus and synchronization protocol described in the previous section. MobBFT is fully web-based and can execute in any recent browser without any plugins. This section first describes the overall architecture. Then it explains our use of aggregate signatures using BLS to reduce the size of the data. The last subsection lists several performance optimization tactics.

4.1 Overall architecture

The MobBFT middleware architecture consists of five main components (Figure 2): (i) a *public interface* that offers an API for developers, (ii) a *peer-to-peer network* component to communicate directly with other browsers, (iii) a *consensus* component to handle the consensus protocol described in the previous section, (iv) a *membership* component to handle all cryptographic operations, and (v) a *store* component to save all state to persistent storage.

(i) Public interface. This component provides an API to application developers to use this middleware. It provides four functions to modify the application state:

- `GET(key)` returns the current value of the atomic register at the given key,
- `SET(key, value)` submits a proposal to update the atomic register at the given key,
- `DELETE(key)` deletes the atomic register at the given key. A tombstone is kept for correct replication,
- `LISTEN(key, callback)` supports reactive programming by calling the callback with the new value each time a new value for the register is confirmed by the network.

Apart from those functions, the middleware also provides a constructor function to initialize the middleware by passing the following four configuration parameters: the list of all members of the network together with their public key, the private key of the

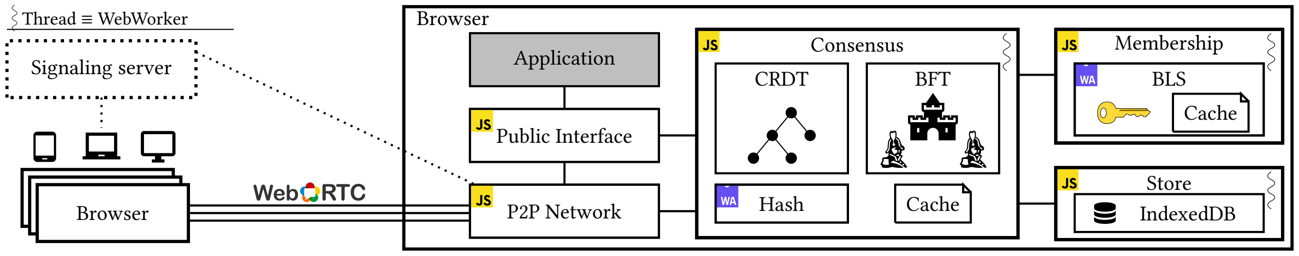


Figure 2: Browser-based architecture of MobBFT.

replica, the URL to the signaling server to set up the peer-to-peer connections, and an access-control callback to verify state changes. This access control callback is called before voting for a new proposed value, with both the old and new values as arguments. It should return a `boolean` whether to allow this change or not. This callback enables the implementation of basic access control policies on the values. One example is to embed the public key of the owner into the value and requiring each new value to be signed by the owner. This value can only be changed by the owner, and supports passing ownership by changing the embedded public key.

(ii) Peer-to-peer network. The *P2P Network* component manages the peer-to-peer network and is responsible for the replication of the state-based CRDTs. Many browser-based replicas are connected to each other using WebRTC (Web Real-Time Communications). WebRTC enables a browser to communicate peer-to-peer. However, to set up those peer-to-peer connections, WebRTC needs a signaling server to exchange several control messages. Once the connection is set up, all communication can happen peer-to-peer, without a central server. Another WebRTC peer-connection can also be used as a signaling layer, so once a replica is connected to another one, it can also connect to all of its peers, without the need of a central signaling server. In our adversary model, this server is assumed to be trusted. If this signaling server would be malicious, the safety of the system is not endangered as no actual data is sent to this central server. However, some peers might not be able to join the network and the required supermajority might not be reached, which violates liveness. The use of multiple independent signaling servers can lower the risk of this happening.

(iii) Consensus. The *Consensus* component handles the consensus protocol described in Section 3. It maintains a Merkle-tree of all atomic registers and uses the state-based CRDT framework *Anonymized* [11] to replicate the local state to other replicas using the *P2P Network* component. The Merkle-tree is constructed using the Blake3 [68] cryptographic hash function.

(iv) Membership. The *Membership* component contains all cryptographic material and is responsible for all cryptographic operations such as signing and verification of signatures. We use an aggregate signature scheme called BLS [21]. Section 4.2 provides more details about the BLS implementation.

(v) Store. At last, the *Store* component saves all state to the IndexedDB database. IndexedDB is a key-value datastore built inside the browser. Each atomic register and the Merkle-tree are serialized to bytes and stored there under the respective key. This enables users to close the browser and continue afterwards without losing the current state.

$\mathbb{G}_0$ and $\mathbb{G}_1$ are two multiplicitive cyclic groups of prime order q. $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}_0$ and $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$ are hash functions viewed as random oracles.

(1) *Parameters Generation:* $\mathsf{PGen}(\kappa)$ sets up a bilinear group $(q, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t, e, g_0, g_1)$ as described by [19]. e is an efficient non-degenerating bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_t$. g_0 and g_1 are generators of the groups $\mathbb{G}_0$ and $\mathbb{G}_1$. It outputs $params \leftarrow (q, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t, e, g_0, g_1)$.
(2) *Key Generation:* $\mathsf{KGen}(params)$ is a probabilistic algorithm that take as input the security $params$, generates $sk \xleftarrow{\$} \mathbb{Z}_q$, computes and sets $pk \leftarrow g_1^{sk}$, and outputs (sk, pk).
(3) *Signing:* $\mathsf{Sign}(sk, m)$ is a deterministic algorithm that takes as input a secret key sk and a message m. It computes $t \leftarrow \mathsf{H}_1(pk)$, and outputs $\sigma \leftarrow \mathsf{H}_0(m)^{sk \cdot t} \in \mathbb{G}_0$.
(4) *Key Aggregation:* $\mathsf{KAgg}(\{(pk_i, r_i)\}_{i=1}^n)$ is a deterministic algorithm that takes as input a set of public key pk and the multiplicity r pairs. It computes $t_i \leftarrow \mathsf{H}_1(pk_i)$, and outputs $apk \leftarrow \prod_{i=1}^n pk_i^{t_i \cdot r_i}$.
(5) *(Multi-)Signature Aggregation:* $\mathsf{Agg}(\sigma_1, ..., \sigma_n)$ is a deterministic algorithm that takes as input n signatures. It outputs $\sigma \leftarrow \prod_{i=1}^n \sigma_i$.
(6) *Verification:* $\mathsf{Ver}(apk, m, \sigma)$ is a deterministic algorithm that takes as input aggregated public keys $apk \in \mathbb{G}_1$, and the related message m and signature $\sigma \in \mathbb{G}_0$. It outputs $e(g_1, \sigma) \stackrel{?}{=} e(apk, \mathsf{H}_0(m))$.

Figure 3: Formal specification of the BLS signature scheme.

4.2 Aggregate signatures using BLS

The consensus protocol in Section 3 is resource-intensive with respect to aggregation and verification of digital signatures. Signatures must be continuously collected and verified. This means, in every intermediate state of a transaction, each party needs to keep track of all incoming signatures and verify them to prevent malicious scenarios. Persistence, management, and transmission of these signatures are costly, especially in a browser-based setting. Therefore, our protocol requires short and compact signatures to reduce storage and network footprint.

Boneh–Lynn–Shacham (BLS) [21] presented a signature scheme based on bilinear pairing on elliptic curves. The size of a signature produced by BLS is compact since a signature is an element of an elliptic curve group. The aggregation algorithm [20] outputs a single aggregate signature as short and compact as the individual signatures, unlike other approaches that rely on ECDSA, DSA or Schnorr [76].

Other state-of-the-art BFT systems such as SBFT [35] and HotStuff [87] also use aggregate or threshold signatures. However, they use it in a different way. They let the leader compute the aggregate signature. MobBFT uses a different approach, once a proposed quorum certificate has reached a supermajority of the votes, any replica can aggregate these into one single aggregated BLS signature.

Efficient aggregation. The protocol described in Section 3 performs a considerable number of signature aggregations. But the standard scheme is vulnerable to rogue public key attacks. The state-of-the-art approach [19] to mitigate such attacks is to compute $(t_1, ..., t_n) \leftarrow \mathsf{H}_1(pk_1, ..., pk_n)$ for each Agg invocation and compute $\sigma \leftarrow \prod_{i=1}^{n} \sigma_i^{t_i}$, where pk_i is the public key of replica i, H_1 is a hash function, and σ_i is a signature produced by replica i. Although the t_i values can be cached, the computation of σ would be costly. Moreover, Agg does not take as input the same set of public keys at different states of a transaction in our consensus protocol. Therefore, we distribute the computations by moving the calculations of the t_i and $\sigma_i^{t_i}$ values to the signing parties, and as a result, these computations are performed only once. Now, any replica can run Agg by only computing $\sigma_1 ... \sigma_n$. The security properties of BLS remain intact [19], and we obtain more efficient aggregations at scale. We provide the mathematical background and formal specification of the optimized BLS scheme in Figure 3.

4.3 Performance optimization for browsers

This section contains four important performance optimizations to host this middleware inside web browsers at scale.

Polyglot middleware. WebAssembly is a binary instruction format that addresses the problem of safe, fast, and portable low-level code on the Web. Higher-level languages such as C, C++, and Rust can be compiled to WebAssembly and can be executed in a modern browser on any platform independent from the underlying hardware. WebAssembly executes significantly faster than JavaScript [39], however, it is not as fast as native code [41]. We used WebAssembly for two key components that are computationally intensive. These components are the hashing component to build the Merkle-tree and the BLS module for aggregate signatures. They are implemented in the Rust programming language [55] and C respectively, and they are compiled to WebAssembly to run inside a browser. Besides the performance improvement of WebAssembly over JavaScript, using Rust and C also enabled us to use well-tested libraries (BLAKE3[2] and blst[3]) instead of implementing these components ourselves.

Parallelization using Web Workers. Web Workers are separate browser threads, which enable us to run computations off the main thread to keep the User Interface responsive. The middleware is distributed over four different threads. The *Public interface* and *P2P Network* components run on the main thread together with the application. The *P2P Network* component is also located on the main thread because WebRTC is not available inside Web Workers. The other three components: *Consensus*, *Membership* and *Store*, are each located in a separate Web Worker. This enables long-running computations, e.g., BLS-signature verification, to run in a separate thread without blocking concurrent operations in the other threads.

[2]https://github.com/BLAKE3-team/BLAKE3/

[3]https://github.com/supranational/blst/

Caching. Caching is used in several places for performance reasons. The most important place is in the *Membership* component in WebAssembly. While WebAssembly itself is fast, the boundary between JavaScript and WebAssembly is not. Function calls between the two environments can only use numbers directly. Any other data structure has to be serialized to bytes and is allocated a spot in the WebAssembly memory buffer. In WebAssembly, these bytes can be decoded to the appropriate Rust or C data structure. For this reason, all cryptographic material such as public keys and the private key are passed to WebAssembly at initialization, avoiding this costly transfer for subsequent operations. In the *Consensus* component, all CRDT and Merkle-tree structures are cached in memory. As such, a costly fetch from disk and decoding from bytes can be avoided.

Batching of writes for IndexedDB.. The last important optimization concerns IndexedDB. IndexedDB is an in-browser database for structured data supporting fast reads and lookups by using indexes. We found that when too many write requests are sent to IndexedDB, the latency significantly starts to increase up to one second or even more. When one atomic register is updated, also all intermediate nodes until the root node of the Merkle-tree are updated. This is due to the tree-shaped structure of the Merkle-tree. So, one write somewhere down the tree, leads to a cascading of writes, and every write causes the root node to be written as well. To reduce the high latency, we batched all writes to IndexedDB in-memory in the *Store* component. If multiple writes for the same key happen in the same batch, only the last one is executed. At fixed intervals, the whole batch is written to IndexedDB. Since many duplicate writes are now avoided, the number of writes is reduced significantly. This solved the problem of high read latency. To avoid data loss, local update operations by the user or consensus votes on this replica are immediately written to disk and bypass the write-batching.

5 EVALUATION

We validated the MobBFT middleware with the loyalty points use case. The first section presents this validation. Next, we present four different benchmarks with different scales. The first benchmark shows the performance results in the optimistic scenario with no network failure or Byzantine failures. The second benchmark evaluates the performance in a more realistic scenario with some network failures. The third benchmark evaluates the performance in the presence of a Byzantine replica. The last benchmark compares two different implementations of MobBFT. The default version uses BLS signatures which supports signature aggregation using WebAssembly as explained in Section 4.2. The other version uses ECDSA signatures using the built-in native WebCrypto [85] APIs from the browser.

5.1 Validation in the loyalty points use case

The deployment of the loyalty points use case consists of three services: a web application running in a browser for each merchant, a web server to serve the static web application files, and a signaling

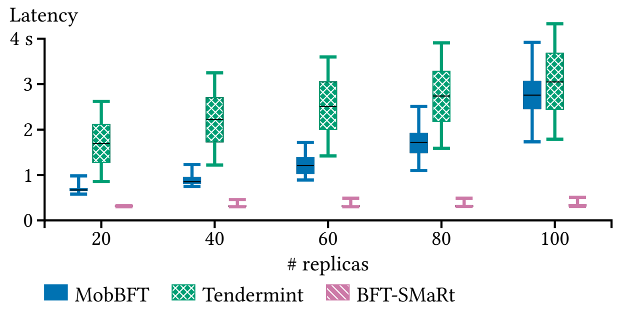


Figure 4: Latency in the optimistic scenario with no failures.

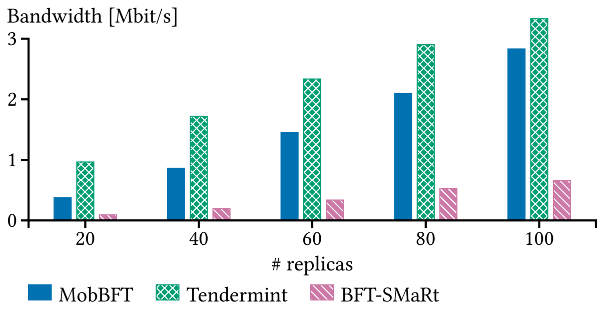


Figure 5: Network usage in the optimistic scenario with no failures.

server to set up WebRTC peer-to-peer connections between the browsers. The web server is optional. Every merchant can also store those application files themselves and load them from their local file system. The signaling server is a trusted component. However, if trust is not present, you can set up multiple signaling servers to reduce potential misbehavior. No actual data is sent to the signaling server. It is only used to discover other peers on the network. To have a baseline, we compare MobBFT to two other existing state-of-the-art systems for BFT consensus: BFT-SMaRt [18, 79] and Tendermint [23, 24]. BFT-SMaRt is a more traditional BFT protocol, similar to PBFT [78], where all replicas are connected to each other, and one leader drives the protocol. If that leader fails, a new one will have to be elected before any progress can be made. Tendermint [24] uses Gossip for communication between the replicas. There is still a leader, however, that leader changes frequently.

Test setup. To test the performance of the middleware, we implemented the use case and deployed it on the Azure public cloud. We used 21 VMs (Azure F8s v2 with 8 vCPUs and 16 GB of RAM) with one VM acting as a central server running the web server and signaling server. The other VMs are running Chrome browsers inside a Docker container. Each of those VMs holds one to five browser instances for different scales of the benchmarks. To simulate a truly mobile environment, the network is delayed to an average latency of 60 milliseconds using the Linux `tc` tool, which simulates the latency of a 4G network [70]. Every test is executed 10 times to ensure the results are reliable.

We are interested in the time it takes to confirm a transaction, experienced by the browser that submitted the transaction. Each transaction is a group of loyalty points being changed from owner. For example, a merchant gives some loyalty points to a customer or a customer redeems their loyalty points with a merchant. In the evaluation, the browser clients will do one transaction per second. This throughput is more than enough for the local community-scale use cases we envision. We compare the latency, network bandwidth, and disk usage with a different number of browsers. We show a boxplot of the latency results instead of only the average, as all users should experience fast confirmation times, and not only the average user [27].

5.2 Optimistic scenario

In the optimistic scenario, every replica is honest and no replicas fail, so the fast path can be used. One single aggregate signature is verified before each decision, avoiding costly signature verifications after every message. As every replica is honest, this aggregate signature is correct and the new value can be accepted by all replicas.

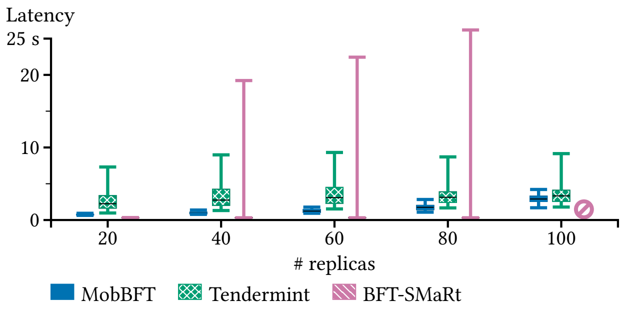


Figure 6: Latency in the realistic scenario with network failures.

Figure 4 shows the latency for the different technologies. For the use case of loyalty points, transactions must be confirmed fast, as people are waiting at checkout to receive or redeem loyalty points. MobBFT can confirm transactions within 4 seconds, even with a network of one hundred browsers. BFT-SMaRt can confirm transactions within half a second. This is because all replicas communicate directly with each other. However, having all replicas directly connected to each other is not realistic in a mobile peer-to-peer network. In contrast, MobBFT and Tendermint use Gossip and need multiple hops before all replicas are reached. This also causes the increased latency. Furthermore, BFT-SMaRt uses HMAC to sign requests, which are an order of magnitude faster than the asymmetric signatures used in MobBFT and Tendermint. We can see a similar pattern in the bandwidth requirements shown in Figure 5. In the large-scale scenario with 100 browsers, MobBFT uses less than 3 Mbit/s, which is acceptable for a typical mobile network.

5.3 Realistic scenario

The same benchmark is now repeated with 25% of the replicas failing during the benchmark. A failure is simulated by dropping all network packets to and from that replica. Replicas fail one by one, with a 5-second delay between each failure. As all systems are Byzantine fault tolerant, they should be able to tolerate up to 33% of the replicas failing or acting Byzantine.

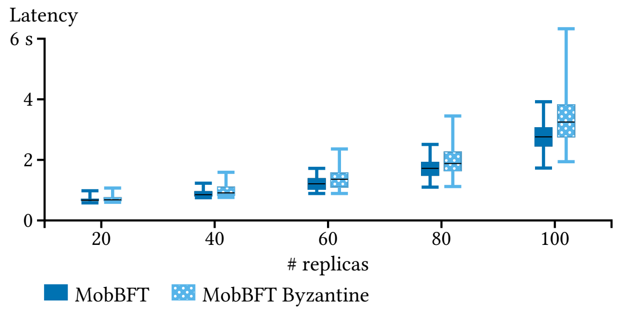


Figure 7: Comparison of the latency in the normal scenario with one where a Byzantine replica tries to halt the network.

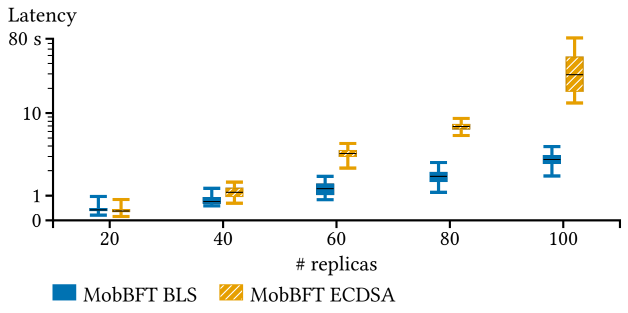


Figure 8: Comparison of the latency in the normal scenario between the use of BLS signatures in WebAssembly and the ECDSA signatures the browser provides.

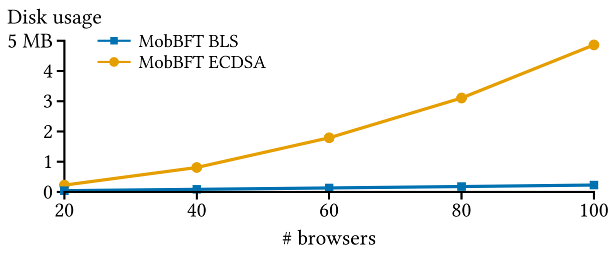


Figure 9: Average disk usage for MobBFT.

Figure 6 shows the latency in this scenario. MobBFT is not impacted much by the failing replicas and can still confirm transactions within 5 seconds. The impact on Tendermint is also small, but the latency is doubled to about 10 seconds. BFT-SMaRt however needs to use a costly leader election protocol when the current leader fails. This process takes some time, during which no transaction can be committed. Once a leader is chosen, the same fast performance can be achieved again. This behavior is clearly visible in Figure 6. The median latency of BFT-SMaRt is not affected by the failures, however, the tail latency increases to 27 seconds for the scenario with 80 replicas. It cannot handle the case with 100 replicas. BFT-SMaRt is unable to handle large network sizes when the latency between the nodes is higher than usual, e.g., in geo-distributed systems or on mobile networks. This has been shown in the literature before [22]. Tendermint does have a leader, but it is rotated round-robin all the time. This makes the failure of a leader less severe, as a new one will quickly be elected anyway.

5.4 Byzantine scenario

For MobBFT, we performed an extra benchmark with Byzantine replicas. As long as the honest replicas are still using the optimistic fast path, the Byzantine replicas will send extra invalid signatures. As the signatures are only verified when a supermajority is reached, the honest replicas only realize this at the end, and they cannot find out which replicas are Byzantine. Once the optimistic fast path is disabled, the signatures are verified for every message, so malicious replicas can be detected and excluded from the network. In this case, the Byzantine replicas keep the signature intact to avoid being detected. However, they will try to slow down the consensus by not voting themselves.

The latency in this Byzantine scenario is shown in Figure 7. MobBFT can handle Byzantine replicas very well for smaller networks, however, for networks of size 100 replicas, the tail latency becomes 7 seconds. Which might already be quite high for the use case of loyalty points. We did not test the effect of Byzantine replicas for BFT-SMaRt or Tendermint. As they do not use a fast path when everyone is honest, the impact is less. However, if the current elected leader happens to be Byzantine, it can delay the consensus until some timers end and a new leader is elected [7].

5.5 Benefits of BLS vs ECDSA

MobBFT uses BLS signatures to limit both the overhead of signature verification and storage. With BLS, only one aggregate signature of the q replicas needs to be verified, compared to q separate signature verifications for ECDSA. Figure 8 (notice the log-scale) compares the latency of the default implementation using BLS signatures with an alternative implementation using ECDSA signatures. The ECDSA implementation performs well for small networks but needs too much time in the larger networks with 80 and 100 replicas.

The BLS signature verifications are performed using WebAssembly. While WebAssembly can be much faster than JavaScript, the resulting WebAssembly code is still an order of magnitude slower compared to a native variant in *x64* assembly (which cannot be executed on the web). For the ECDSA signatures on the other hand, we used the built-in Web Cryptography API. These are implemented in the browser using native functions. But even with the more optimized implementation of ECDSA, the version of MobBFT using BLS and WebAssembly is much faster for larger networks. In the future, browsers could add the BLS signature scheme natively, which would result in even better performance for MobBFT.

Figure 9 shows the storage usage for both implementations of MobBFT. BLS improves disk usage about 20 times for the scenario with 100 browsers. Both implementations need less than 5 MB to store 1000 tokens. This disk usage does not increase over time, as only the current value and proposals are stored. We do not store a chain of all transactions that have happened so far. This is a big difference with blockchains that grow in size with every transaction that is executed and stored in the blockchain. This makes our approach feasible for resource-constrained devices that do not have hundreds of gigabytes of storage capacity to store a full blockchain. At the cost of losing auditability.

5.6 Discussion and conclusions

We have shown that MobBFT can be used for the loyalty points use case with up to 100 different merchants, even when some of them are acting maliciously. MobBFT can achieve similar latencies as other Gossip-based BFT protocols, such as Tendermint. Our evaluation also shows the trade-offs that MobBFT makes. In an optimal scenario where there is a good connection available between all replicas and no network disruptions or crashes happen, then a classical leader-based protocol such as BFT-SMaRt will outperform MobBFT. However, as we mention in the introduction, we envision a more ad-hoc network between low-end devices on a residential or even a mobile network, where short-term disruptions are common. Our evaluation shows that MobBFT is very robust against this kind of setting and achieves similar performance as in the optimal scenario. A leader-based protocol such as BFT-SMaRt is not well suited, the temporary failure of a leader leads to long commit times, and even total failure for larger network sizes. This leader also needs more resources and a direct connection to every other replica. Keeping 100 WebRTC connections open in a browser, while theoretically possible, drastically reduces performance. MobBFT does not impose this, consensus can be reached gradually over time, as the full state of the proposals and votes propagates through the network. MobBFT can confirm transactions fast, in the order of seconds, without needing a complex back-end setup or wasting a lot of energy. MobBFT has a small storage footprint due to its state-based nature.

6 RELATED WORK

Several client-side frameworks for data synchronization between web applications exist: Legion [82], Yjs [66, 67], Automerge [44], and *Anonymized* [11]. They make use of various kinds of Conflict-free Replicated Data Types (CRDTs) [77] to deal with concurrent conflicting operations, and can synchronize data peer-to-peer. They are easy to set up and only require a browser and a peer-to-peer discovery service. However, they assume trusted operation as the default setting. Some work has been done in a semi-trusted setting [12, 83]. None of them can tolerate Byzantine parties.

WebBFT [16] shares a similar vision of client-centric, decentralized web applications. However, instead of running the BFT protocol directly between browsers, they only interface to a back-end BFT-SMaRt cluster.

Open or permissionless blockchains such as Bitcoin [64] and Ethereum [25] allow everyone to participate and use Proof-of-Work (PoW) to reach agreement over the ledger [38]. However, PoW has several flaws [15]. PoW uses a lot of processing power and energy [69] and performs poorly in terms of latency. It assumes a synchronous network to guarantee safety. When this assumption is violated, temporary forks can happen in the blockchain as liveness is chosen over safety. Therefore, PoW blockchains do not offer consensus finality, instead one needs to wait for several consecutive blocks to be probabilistically certain that a transaction cannot be reverted. Blockchains require a lot of storage space, as the full blockchain typically needs to be stored on every node. Simplified Payment Verification (SPV) mode [64] for clients can reduce the resource usage at the cost of decentralization. PoW gains its security from the fact that one needs a lot of CPU power to control the network, which is too costly for an attacker compared to the revenue for a successful attack. Other variants of resource consumption exist, such as Proof-of-Space [4] or Proof-of-Storage [5].

ByzCoin [46] uses PoW for a separate identity chain to guard against Sybil attacks but uses a BFT protocol to order transactions. ByzCoin makes use of collective signatures (CoSi) [80] and a balanced tree for the communication flow. CoSi makes use of aggregate signatures by constructing a Schnorr multisignature [76]. However, CoSi needs multiple communication round-trips to generate the multi-signature and assumes a synchronous network. Similar to ByzCoin, Solida [1] only uses PoW to define who is a committee member, and uses a BFT protocol between the current committee members for the actual consensus.

Tendermint [23, 24], used in Cosmos [48], uses Proof-of-Stake (PoS), where voting power is based on the amount of cryptocurrency owned by each replica. Because block times are short, in the order of seconds, there is a limited number of validators Tendermint can have because finality needs to be reached for each block. It is also not resistant to cartel forming, which allows those with a lot of cryptocurrencies to work together to control the network.

Instead of reaching consensus between all the replicas of the network, Stellar [53, 56] uses quorum slices to reach federated Byzantine agreement in an open network. Replicas should choose adequate quorum slices for safety. However, today's Stellar network is highly centralized and many replicas use the same few validators. Two failing validators can make the entire system fail [63].

Other protocols use a randomized approach. Ouroboros [43], HoneyBadger [62], Dumbo [36] and BEAT [28] use distributed coin flipping for consensus. HoneyBadger [62] also uses threshold signatures [78] for censorship resilience. Algorand [34] uses Verifiable Random Functions [60] to select a random committee for the next round. Avalanche [74, 75] uses meta-stability to reach consensus by sampling other replicas without any leader. While Avalanche is lightweight and scalable, it needs to be able to sample all other validators directly. The number of connections one can open in a browser without performance loss is limited. MobBFT supports propagation of votes over multiple hops.

Permissioned blockchains such as Hyperledger Fabric [3] have closed membership and often use a BFT consensus protocol to order transactions. For example BFT-SMART in HyperLedger Fabric [18, 79]. The first known BFT protocol is Practical Byzantine Fault Tolerance (PBFT) [26]. Other protocols bring improvements to the original PBFT protocol. Zyzzyva [47] uses speculative execution which improves latency and throughput if there are no Byzantine replicas. However, its performance drops significantly if this premise does not hold. 700BFT [6] provides an abstraction for these BFT algorithms. These protocols are targeting a small number of replicas in a local network. They generally work in two phases: the first guarantees proposal uniqueness, and the second guarantees that a new leader can convince replicas to vote for a safe proposal. HotStuff [87] proposed a three-phase protocol to reduce complexity and simplify leader replacement. This makes HotStuff more scalable. All these algorithms use a leader to drive the protocol. When the leader is malicious, the performance can degrade quickly [7]. GeoBFT [37] is a topology-aware, decentralized consensus protocol, designed for geo-distributed scalability. MobBFT does not use a leader and replicas communicate only to a subset of the other replicas using a gossip-like protocol. There is not a fixed number of communication rounds to reach consensus in MobBFT. Instead, the PRE-COMMIT phase can consist of multiple rounds in which the replicas communicate with each other until one unique proposal is chosen.

Another approach is to use a trusted hardware component [13, 42, 52, 84, 88]. These are faster and less computationally intensive but require specialized hardware to be present. Moreover, trusted execution environments have been broken in the past [45, 51, 81].

There are several proposals to improve the performance and response time of Hyperledger Fabric. StreamChain [40] reaches consensus over a stream of transactions instead of blocks. Fabric-CRDT [65] uses CRDTs to support concurrent transactions to occur in the same block, using the built-in conflict resolution of CRDTs to resolve the conflict automatically. Other approaches also borrow from CRDTs: PnyxDB [22] supports commuting transactions to be applied out-of-order. A novel design for gossip in Fabric [14] improves the block propagation latency and bandwidth. While these improvements make Hyperledger Fabric faster, none of them try to reduce the infrastructure requirements to be able to easily set up an untrusted peer-to-peer network.

The Bitcoin Lightning Network [72] or state channels for Bitcoin [50] or Ethereum [58, 61, 71] are *off-chain* protocols that run on top of a blockchain. A new state channel between known participants is created by interacting with the blockchain. After its creation, participants can use this channel to execute state transitions by collectively signing the new state. These transactions do not involve the blockchain and have fast confirmation times and no transaction costs. However, state channels assume all participants to be always online and honest. If this is violated, the underlying blockchain needs to be used to resolve the conflict, or a trusted third party can be used [57]. MobBFT uses a similar state-transitioning protocol where only the latest collectively agreed state needs to be stored. However, MobBFT can tolerate both failing and malicious replicas, without resorting to a blockchain or a trusted third party.

7 CONCLUSION

In this paper, we presented MobBFT. A browser-based middleware for decentralized, community-driven web applications. MobBFT uses an optimistic, leaderless BFT consensus protocol, combined with a robust and efficient state-based synchronization protocol. MobBFT uses an optimized BLS scheme for efficient computation and storage of signatures. It supports a client-centric, browser-based, state-based, permissioned ledger with a low infrastructure and storage footprint for small-scale, citizen-driven networks. MobBFT offers consistent and robust confirmation times to achieve finality of transactions in the order of seconds, even in failure settings and Byzantine environments. In contrast to traditional blockchains, MobBFT does not store a transaction log or blockchain, keeping the overall storage footprint small.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2018. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In *21st International Conference on Principles of Distributed Systems (OPODIS 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 95)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 25:1–25:19. https://doi.org/10.4230/LIPIcs.OPODIS.2017.25

[2] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2018. Delta state replicated data types. *J. Parallel and Distrib. Comput.* 111 (2018), 162 – 173. https://doi.org/10.1016/j.jpdc.2017.08.003

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. ACM, NY, USA, Article 30. https://doi.org/10.1145/3190508.3190538

[4] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. 2014. Proofs of Space: When Space Is of the Essence. In *Security and Cryptography for Networks*. Springer, Cham, 538–557. https://doi.org/10.1007/978-3-319-10879-7_31

[5] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '07)*. ACM, NY, USA, 598–609. https://doi.org/10.1145/1315245.1315318

[6] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.* 32, 4, Article 12 (Jan. 2015). https://doi.org/10.1145/2658994

[7] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. Rbft: Redundant byzantine fault tolerance. In *IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, USA, 297–306. https://doi.org/10.1109/ICDCS.2013.53

[8] Anonymous Author(s). [n. d.]. Anonymized for double-blind reviewing.

[9] Anonymous Author(s). [n. d.]. Anonymized for double-blind reviewing.

[10] Anonymous Author(s). [n. d.]. Anonymized for double-blind reviewing.

[11] Anonymous Author(s). [n. d.]. Anonymized for double-blind reviewing. ([n. d.]).

[12] Manuel Barbosa, Bernardo Ferreira, João Marques, Bernardo Portela, and Nuno Preguiça. 2021. Secure Conflict-Free Replicated Data Types. In *International Conference on Distributed Computing and Networking 2021* (Nara, Japan) *(ICDCN '21)*. ACM, NY, USA, 6–15. https://doi.org/10.1145/3427796.3427831

[13] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on Steroids: SGX-Based High Performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems* (Belgrade, Serbia) *(EuroSys '17)*. ACM, NY, USA, 222–237. https://doi.org/10.1145/3064176.3064213

[14] Nicolae Berendea, Hugues Mercier, Emanuel Onica, and Etienne Riviere. 2020. Fair and Efficient Gossip in Hyperledger Fabric. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, USA. https://doi.org/10.1109/ICDCS47774.2020.00027

[15] Christian Berger and Hans P. Reiser. 2018. Scaling Byzantine Consensus: A Broad Analysis. In *Proceedings of the 2Nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers* (Rennes, France) *(SERIAL'18)*. ACM, NY, USA, 13–18. https://doi.org/10.1145/3284764.3284767

[16] Christian Berger and Hans P. Reiser. 2018. WebBFT: Byzantine Fault Tolerance for Resilient Interactive Web Applications. In *Distributed Applications and Interoperable Systems*. Springer, Cham, 1–17. https://doi.org/10.1007/978-3-319-93767-0_1

[17] Tim Berners-Lee. 2017. *Three challenges for the Web, according to its inventor*. World Wide Web Foundation. https://webfoundation.org/2017/03/web-turns-28-letter/

[18] Alysson Bessani, Joao Sousa, and Eduardo E. P. Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*. IEEE, USA, 355–362. https://doi.org/10.1109/DSN.2014.43

[19] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Springer, Cham, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[20] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 416–432. https://doi.org/10.1007/3-540-39200-9_26

[21] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[22] Loïck Bonniot, Christoph Neumann, and François Taïani. 2020. PnyxDB: a Lightweight Leaderless Democratic Byzantine Fault Tolerant Replicated Datastore. In *The 39th IEEE International Symposium on Reliable Distributed Systems (SRDS '20) (The 39th IEEE International Symposium on Reliable Distributed Systems)*. IEEE, Shanghai, China. https://doi.org/10.1109/SRDS51746.2020.00023

[23] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains.* Ph. D. Dissertation. University of Guelph.

[24] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. arXiv:1807.04938

[25] Vitalik Buterin et al. 2013. *A next-generation smart contract and decentralized application platform.* White paper. ethereum.org.

[26] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99).* USENIX Association, USA, 173–186. https://doi.org/10.5555/296806.296824

[27] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07, Vol. 41(6)).* ACM, NY, USA, 205–220. https://doi.org/10.1145/1294261.1294281

[28] Sisi Duan, Michael K. Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18).* ACM, NY, USA, 2028–2041. https://doi.org/10.1145/3243734.3243812

[29] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (April 1988), 288–323. https://doi.org/10.1145/42282.42283

[30] Clarence A. Ellis and Simon John Gibbs. 1989. Concurrency Control in Groupware Systems. *SIGMOD Rec.* 18, 2 (June 1989), 399–407. https://doi.org/10.1145/66926.66963

[31] Homan Farahmand. 2019. *Guidance for Assessing Blockchain Platforms.* Technical Report. Gartner.

[32] Steve Fromhart and Lincy Therattil. 2016. *Making blockchain real for customer loyalty rewards programs.* Technical Report. Deloitte.

[33] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. 2015. Edge-Centric Computing: Vision and Challenges. *SIGCOMM Comput. Commun. Rev.* 45, 5 (Sept. 2015), 37–42. https://doi.org/10.1145/2831347.2831354

[34] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17).* ACM, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[35] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN).* IEEE, USA, 568–580. https://doi.org/10.1109/DSN.2019.00063

[36] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20).* ACM, NY, USA, 803–818. https://doi.org/10.1145/3372297.3417262

[37] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proc. VLDB Endow.* 13, 6 (Feb. 2020), 868–883. https://doi.org/10.14778/3380750.3380757

[38] Suyash Gupta and Mohammad Sadoghi. 2018. *Blockchain Transaction Processing.* Springer, Cham, 1–11. https://doi.org/10.1007/978-3-319-63962-8_333-1

[39] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the Web up to Speed with WebAssembly. *SIGPLAN Not.* 52, 6 (June 2017), 185–200. https://doi.org/10.1145/3140587.3062363

[40] Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2018. StreamChain: Do Blockchains Need Blocks?. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers* (Rennes, France) *(SERIAL'18).* ACM, NY, USA, 1–6. https://doi.org/10.1145/3284764.3284765

[41] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. 2019. Not so Fast: Analyzing the Performance of Webassembly vs. Native Code. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19).* USENIX Association, USA, 107–120.

[42] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: Resource-Efficient Byzantine Fault Tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12).* ACM, NY, USA, 295–308. https://doi.org/10.1145/2168836.2168866

[43] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology – CRYPTO 2017.* Springer, Cham, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[44] Martin Kleppman and Alastair R Beresford. 2018. Automerge: Real-time data sync between edge devices. http://martin.kleppmann.com/papers/automerge-mobiuk18.pdf

[45] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19).* IEEE, USA, 1–19. https://doi.org/10.1109/SP.2019.00002

[46] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16).* USENIX Association, USA, 279–296. https://doi.org/10.5555/3241094.3241117

[47] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA) *(SOSP '07).* ACM, NY, USA, 45–58. https://doi.org/10.1145/1294261.1294267

[48] Jae Kwon and Ethan Buchman. 2019. *Cosmos Whitepaper: A Network of Distributed Ledgers.* White paper. cosmos.network.

[49] Leslie Lamport. 1986. On interprocess communication. *Distributed Computing* 1, 2 (1986), 86–101. https://doi.org/10.1007/BF01786228

[50] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: A Secure Payment Network with Asynchronous Blockchain Access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19).* ACM, NY, USA, 63–79. https://doi.org/10.1145/3341301.3359627

[51] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18).* USENIX Association, Baltimore, MD, 973–990.

[52] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. 2018. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Trans. Comput.* 68, 1 (2018), 139–151. https://doi.org/10.1109/TC.2018.2860009

[53] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. 2019. Fast and Secure Global Payments with Stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19).* ACM, NY, USA, 80–96. https://doi.org/10.1145/3341301.3359636

[54] Akash Madhusudan, Iraklis Symeonidis, Mustafa A. Mustafa, Ren Zhang, and Bart Preneel. 2019. SC2Share: Smart Contract for Secure Car Sharing. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP.* INSTICC, SciTePress, Portugal, 163–171. https://doi.org/10.5220/0007703601630171

[55] Nicholas D. Matsakis and Felix S. Klock. 2014. The Rust Language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology* (Portland, Oregon, USA) *(HILT '14).* ACM, NY, USA, 103–104. https://doi.org/10.1145/2663171.2663188

[56] David Mazieres. 2015. *The stellar consensus protocol: A federated model for internet-level consensus.* Technical Report. Stellar Development Foundation.

[57] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration Outsourcing for State Channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (Zurich, Switzerland) *(AFT '19).* ACM, NY, USA, 16–30. https://doi.org/10.1145/3318041.3355461

[58] Patrick McCorry, Chris Buckland, Surya Bakshi, Karl Wüst, and Andrew Miller. 2020. You Sank My Battleship! A Case Study to Evaluate State Channels as a Scaling Solution for Cryptocurrencies. In *Financial Cryptography and Data Security.* Springer, Cham, 35–49. https://doi.org/10.1007/978-3-030-43725-1_4

[59] Ralf Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87.* Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[60] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (FOCS '99).* IEEE, IEEE, USA, 120–130. https://doi.org/10.1109/SFFCS.1999.814584

[61] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and State Channels: Payment Networks that Go Faster Than Lightning. In *Financial Cryptography and Data Security.* Springer, Cham, 508–526. https://doi.org/10.1007/978-3-030-32101-7_30

[62] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16).* ACM, NY, USA, 31–42. https://doi.org/10.1145/2976749.2978399

[63] Kim Minjeong, Kwon Yujin, and Kim Yongdae. 2019. Is Stellar As Secure As You Think?. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW).* IEEE, USA, 377–385.

[64] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[65] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. 2019. FabricCRDT: A Conflict-Free Replicated Datatypes Approach to Permissioned Blockchains. In *Proceedings of the 20th International Middleware Conference* (Davis, CA, USA) *(Middleware '19).* ACM, NY, USA, 110–122. https://doi.org/10.1145/3361525.

3361540

[66] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. 2015. Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types. In *Engineering the Web in the Big Data Era (ICWE 2015)*. Springer, Cham, 675–678.

[67] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. 2016. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In *Proceedings of the 19th International Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '16)*. ACM, NY, USA, 39–49. https://doi.org/10.1145/2957276.2957310

[68] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. 2020. BLAKE3: one function, fast everywhere. https://blake3.io/

[69] Karl J O'Dwyer and David Malone. 2014. Bitcoin mining and its energy footprint. In *Proceedings of the 2014 IET Irish Signals and Systems Conference (ISSC 2014/CIICT 2014)*. IEEE, USA, 280–285. https://doi.org/10.1049/cp.2014.0699

[70] OpenSignal. 2019. Mobile Network Experience Report. https://www.opensignal.com/reports/2019/01/usa/mobile-network-experience.

[71] Joseph Poon and Vitalik Buterin. 2017. Plasma: Scalable Autonomous Smart Contracts. https://plasma.io/plasma-deprecated.pdf

[72] Joseph Poon and Thaddeus Dryja. 2016. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. White paper. lightning.network.

[73] PwC. 2015. *The Sharing Economy*. Technical Report. Consumer Intelligence Series.

[74] Team Rocket. 2018. *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies*. White paper. avalabs.org.

[75] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and Probabilistic Leaderless BFT Consensus through Metastability. arXiv:1906.08936

[76] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of Cryptology* 4, 3 (01 Jan 1991), 161–174. https://doi.org/10.1007/BF00196725

[77] Marc Shapiro, Nuno Perguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *SSS 2011 - 13th International Symposium Stabilization, Safety, and Security of Distributed Systems (Lecture Notes in Computer Science, Vol. 6976)*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 386–400.

[78] Victor Shoup. 2000. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt 2000)*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 207–220.

[79] Joao Sousa, Alysson Bessani, and Marko Vukolic. 2018. A byzantine fault-tolerant ordering service for the Hyperledger Fabric blockchain platform. In *48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, IEEE, USA, 51–58. https://doi.org/10.1109/DSN.2018.00018

[80] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy (SP) (SP '16)*. IEEE, USA, 526–545. https://doi.org/10.1109/SP.2016.38

[81] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *41th IEEE Symposium on Security and Privacy (S&P'20)*. IEEE, USA.

[82] Albert van der Linde, Pedro Fouto, João Leitão, Nuno Preguiça, Santiago Castiñeira, and Annette Bieniusa. 2017. Legion: Enriching Internet Services with Peer-to-Peer Interactions. In *Proceedings of the 26th International Conference on World Wide Web* (Perth, Australia) *(WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 283–292. https://doi.org/10.1145/3038912.3052673

[83] Albert van der Linde, João Leitão, and Nuno Preguiça. 2020. Practical Client-Side Replication: Weak Consistency Semantics for Insecure Settings. *Proc. VLDB Endow.* 13, 12 (July 2020), 2590–2605. https://doi.org/10.14778/3407790.3407847

[84] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. 2013. Efficient byzantine fault-tolerance. *IEEE Trans. Comput.* 62, 1 (2013), 16–30. https://doi.org/10.1109/TC.2011.221

[85] Mark Watson. 2017. *Web Cryptography API*. Recommendation. W3C. https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/

[86] Karl Wüst and Arthur Gervais. 2018. Do you Need a Blockchain?. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 45–54. https://doi.org/10.1109/CVCBT.2018.00011

[87] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, NY, USA, 347–356. https://doi.org/10.1145/3293611.3331591

[88] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-Efficient Mining for Blockchains. In *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC'17)*. USENIX Association, USA, 1427–1444. https://doi.org/10.5555/3241189.3241300

[89] Gartner 2019. *Blockchain's Big Bang: Web 3.0*. Gartner. https://blogs.gartner.com/avivah-litan/2019/08/08/blockchains-big-bang-web-3-0/