WebLedger: a Byzantine Fault-Tolerant State-Based Ledger for a Decentralized Web without a Blockchain

Anonymous Author(s)
Submission Id: 597

Abstract

One of the visions of Tim Berners-Lee, the founder of the web, is that the web should shift to a client-centric, decentralized model where web clients become the leading execution environment for application logic and data storage. Both Gartner and the Web3 foundation consider client-centric decentralization as one of the key properties of Web 3.0.

However, existing peer-to-peer web middleware only support operation in a fully trusted client network. Other decentralized solutions often use a heavyweight blockchain platform in the backend. Moreover, traditional Byzantine consensus protocols are not well-suited for a decentralized client-centric web with many network or node failures.

In this paper we present WebLedger, a browser-based middleware for decentralized web applications in small, community-driven networks. We propose a novel, optimistic, leaderless consensus protocol, tolerating Byzantine replicas, combined with a robust and efficient state-based synchronization protocol. WebLedger uses an optimized implementation of the standard BLS scheme for efficient aggregation and storage of signatures. No large backend infrastructure is required, as the middleware is purely browser-based. No transaction log or blockchain is stored, keeping the overall storage footprint small for client-centric devices.

Our performance evaluation shows that WebLedger can achieve finality of transactions within seconds in community-driven networks of mobile web clients, even in the context of network problems, node failures and Byzantine behaviour.

1 Introduction

Decentralization is envisioned as one of the key properties of the next-generation Web 3.0 by many. This idea was first introduced by the founder of the web, Tim Berners-Lee [14]. Both Gartner [28, 88] and the Web3 Foundation [89] define Web 3.0 as the decentralized Web, where users are in control of their data, and where centralized intermediated interactions are replaced by decentralized infrastructure and application platforms. Often, blockchain is put forward as the underpinning technology of this decentralized Web 3.0.

Regarding application domains, small-scale, citizen-driven networks can open the road to many application cases, such as use cases in the sharing economy, such as car-sharing in a local neighborhood. Such ad-hoc client-centric collaboration can also enable small merchant networks with use cases such as loyalty cards at a farmer's market or a local shopping street. In such community-driven collaborative distributed systems, web applications could evolve into a decentralized, client-centric architecture in which browsers become the leading execution environment for application logic and data storage. Browsers and client-side web technology also offer more and more capabilities to enable fully client-side web applications that can operate independently and in a standalone fashion, in contrast to the server-centric model [8, 31]. Recently, Progressive Web Apps [79] also focus on client-centric, web application architectures that can operate on mobile devices with problematic network conditions.

However, state-of-the-art peer-to-peer data synchronization systems for the browser like Legion [82], Yjs [64], and Automerge [43] focus on full replication and consistency between fully trusted peers. Each replica can modify all data, and all modifications to the data are automatically replicated to all replicas. Their synchronization protocols lack Byzantine fault-tolerance (BFT). BFT means that it can both tolerate replica and network failures, as well as malicious replicas.

Traditionally, distrust between interacting parties is solved using a centralized trusted party. While this is often beneficial for performance, a lot of power is given to one party, that can decide to manipulate the data and charge high transaction costs. When trust is lacking, one can opt for a more decentralized consensus between several mistrusting parties. Starting with Bitcoin [62], many Proof-of-Work (PoW) blockchains emerged. However, their confirmation time is too slow for many use cases, and they typically lack finality. Bitcoin needs about one hour to confirm a transaction with a high probability. Moreover, PoW needs a lot of processing power and energy which are not available on mobile devices. Blockchains also store an immutable history of all transactions on every replica, leading to large storage overhead. Lightweight clients that use a proxy node to communicate with the blockchain exist, but some party still needs to manage the full node, which clients need to trust. Other types of blockchain use a BFT consensus protocol. Hyperledger Fabric [3] can use BFT-SMART [15] and achieves high throughput and low latency. However, it requires a complex back-end infrastructure, with many different servers, and replicas still need to store the full operation-based transaction history.

Independent of the heavy back-end infrastructure, the consensus protocols such as BFT-SMART are not well-suited for a decentralized client-centric web with many network or node failures. Performance degradation results in confirmation times that are not usable for human transactions where finality must be ensured in a few seconds (e.g a loyalty card).

In this paper we present WebLedger, a web middleware for decentralized, community-driven, web applications between mistrusting clients. It supports a client-centric, browser-based, state-based, permissioned ledger with a low infrastructure and storage footprint. WebLedger offers consistent and robust confirmation times to achieve finality of transactions in the order of seconds, even in failure settings and Byzantine environments.

The state-based ledger does not keep track of an operation log or transaction history in a blockchain. The ledger is fully maintained, synchronized, and agreed on by mobile clients in their web browser. To achieve this, WebLedger combines the following key technical contributions:

- Lightweight, leaderless, client-side Byzantine fault-tolerant synchronization and consensus.
- Optimistic consensus using a fast path when nobody is acting Byzantine, gracefully degrading to the slow path when under attack.
- Efficient computation and compact storage of signatures using an optimized BLS signature scheme.
- Efficient, robust, state-based synchronization and compact storage using state-based CRDTs, instead of storing a chain of transactions.

Our evaluation, using our application use case of integrated loyalty points, shows that applications using the WebLedger middleware can achieve realistic confirmation times and finality for typical business transactions and transaction rates. In our example, safety and liveness can even be guaranteed within communities of 60 merchants and a throughput of one transaction per second. WebLedger achieves a latency of less than 2 seconds in optimal environments, and less than 10 seconds in Byzantine environments.

Section 2 further discusses some motivating use cases and background. Section 3 presents WebLedger's BFT consensus protocol that is both optimistic and state-based. The detailed web-based middleware architecture of WebLedger is elaborated in Section 4. Our evaluation in Section 5 focuses on many aspects of performance in both normal scenario's as well as Byzantine scenarios. Section 6 elaborates on important related work. We conclude in Section 7.[1]

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 three use cases that would benefit from the lightweight consensus offered by WebLedger. They all involve business transactions happening in real life and need interactive performance, rather than high throughput.

Sharing economy. Small communities, such as an apartment building or local neighborhood, can share tools or cars [52] with each other using a P2P platform to keep track of the current possession and reservation of tools and cars [71]. When a tool is being exchanged, it is checked on potential damage which can be registered in the network.

Microloans. Microloans enable individuals, rather than banks, to issue loans to other individuals or small businesses. This has the advantage that also individuals with a bad credit rating or without enough collateral can receive a loan. This community initiative can prevent loan sharks, especially in developing countries.

Loyalty programs. Integrated loyalty programs can be more effective than traditional loyalty programs that are limited to a single company [30]. Think about airlines who 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 P2P network can enable fast and secure creation, redemption, and exchange of loyalty points across the different merchants.

In the remainder of this paper, we focus on the loyalty use case, as this use case has the largest scale in terms of the transaction throughput and the number of participants.

2.2 Background on blockchains and 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 [62] and Ethereum [85]. 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.

---

[1] A preliminary workshop paper [9] already described our use case of loyalty points in more detail together with an early solution. This paper presents the full technical results and includes a novel consensus algorithm with stronger liveness guarantees and the state-based replication protocol, the use of aggregate signatures, and an extensive evaluation.

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, 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 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 mobile computer such as a laptop or tablet.

Two existing state-of-the-art protocols for BFT consensus are BFT-SMART [15] and Tendermint [20, 21]. BFT-SMART is a more traditional BFT protocol, similar to PBFT [77], 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 [78]. Tendermint [21] uses Gossip for communication between the replicas. There is still a leader, however, that leader changes frequently. Tendermint is used in the Cosmos blockchain [47].

3 Optimistic state-based BFT consensus

This section explains the state-based consensus protocol used in WebLedger. First, it describes the communication and adversary model. Then it explains the detailed consensus protocol, followed by the state-based communication protocol. At last, this section discusses safety and liveness.

3.1 Overview and adversary model

The core protocol is a partially synchronous, leaderless, Byzantine fault-tolerant consensus protocol. Communication is partially synchronous if there is an unknown upper bound Δ on message delivery [26]. An adversary can delay the network for a finite amount of time, however, after at most Δ, some stream of messages can be delivered. This bound on communication is necessary as deterministic Byzantine consensus is not possible with fully asynchronous communication [29]. An adversary might also corrupt up to f replicas of the $3f + 1$ total replicas. They can deviate from the protocol in any arbitrary way. Such replicas are called Byzantine replicas, while the replicas that are strictly following the protocol are called honest replicas. We assume attackers cannot forge the used asymmetric signatures or find collisions for the used cryptographic hash functions.

The protocol is used to implement an Atomic Register [48] 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.

The protocol does not use a leader to coordinate the protocol, removing a common performance bottleneck compared to many existing BFT protocols. The consensus protocol uses voting, where every replica has exactly one vote. One or more replicas propose a new value. Other replicas start voting on those proposals. Once a proposal has reached a supermajority of at least $2f+1$ votes, with f the total number of Byzantine replicas the system should tolerate, the proposal is accepted and becomes the new value. Unlike blockchains, consensus is reached for each register separately, and there is no chain of transactions. Only the current state and proposals for the next state are stored. The next section explains this protocol in more detail.

3.2 Protocol in detail

The detailed specification is depicted in Figure 1. Each register has its own state which consists of the current value, and zero or more proposals for new values. The current value is signed by a supermajority of $2f + 1$ replicas. The `SET` operation creates a new proposal. The proposal has a version one greater than the current accepted value, a round and step equal to zero, and the new value. This proposal is signed by the proposing replica. The current value and proposals are replicated by using a state-based Gossip protocol. The `MERGE` operation is called when the state of another replica is received. This operation gets as input the state of another replica and advances the current local state. The new value will be the value with the highest version number. Since each accepted value is always signed by a supermajority of the replicas, it can be accepted without the need to verify intermediate versions. The new set of proposals is the union between the local and received set of proposals. All proposals for a smaller or equal version than the accepted value can be discarded. If there are proposals left, it means that replicas are trying to reach consensus for a new value. Consensus is reached in two steps (0 and 1). Once a supermajority of the replicas vote on the same value in step 0, the replicas move on to step 1. If now a supermajority of the replicas vote on the same value in step 1, the value is accepted. An honest replica can only vote for a proposal when it has not voted for any other proposal with the same version, round and step.

Split-votes. Since there is no leader driving the protocol, multiple valid values might be proposed concurrently. this can lead to a split vote between the proposals, which all get a portion of the votes. No supermajority is reached, so the

```
1:  types
2:      $Versions \equiv \mathbb{N}$
3:      $Rounds \equiv \mathbb{N}$
4:      $Step \equiv \{0, 1\}$
5:      $Sigs \equiv \mathcal{P}(\mathbb{I} \times \Sigma)$                          ▷ Signatures
6:      $Proposals \equiv \mathcal{P}(Version \times Round \times Step \times value \times Sigs)$
7:  initial state
8:      $V \longleftarrow \bot \in Proposals \cup \{\bot\}$              ▷ current value
9:      $P \longleftarrow \emptyset \subset Proposals$             ▷ set of current proposals
10:     $I$                                                  ▷ replica ID
11:     $ID$                                                 ▷ register ID
12:     $Q$                               ▷ quorum size, $\lfloor \frac{2}{3} \times n + 1 \rfloor$
13: define
14:     $P_0 \equiv \{p \in P : p_{step} = 0 \wedge p_{round} = \text{MAX}(P_{round})\}$
15:     $P_1 \equiv \{p \in P : p_{step} = 1 \wedge p_{round} = \text{MAX}(P_{round})\}$
16: procedure SET($value$)
17:     if $\neg$ HAS_VOTED_IN_LAST_ROUND($P$)
18:         $\sigma_I \longleftarrow \text{SIGN}(ID \oplus (V_{version} + 1) \oplus 0 \oplus 0 \oplus value)$
19:         $P \longleftarrow P \cup \{((V_{version} + 1), 0, 0, value, \{\sigma_I\})\}$
20:     else          ▷ wait until previous consensus is reached
21: procedure MERGE($V', P', I'$)
22:     if $V'_{version} > V_{version}$
23:         if $\text{SIZE}(V'_{sigs}) < Q \vee \neg\text{VERIFY}(V'_{sigs})$
24:             return DISTRUST($I'$)
25:         $V \longleftarrow V'$
26:     if $\exists\, p \in P' : \neg\text{VERIFY}(p_{sigs})$
27:         return DISTRUST($I'$)
28:     $P \longleftarrow \{p \in P \cup P' : p_{version} = V_{version} + 1\}$
29:     if SIZE($P_1$) > 0
30:         if $\neg$ HAS_VOTED_IN_LAST_ROUND($P_1$)
31:             $p \longleftarrow$ PROPOSAL_WITH_MOST_VOTES($P_1$)
32:             $\sigma_I \longleftarrow \text{SIGN}(ID \oplus p_{version} \oplus p_{round} \oplus 1 \oplus p_{value})$
33:             $p_{sigs} \longleftarrow p_{sigs} \cup \{\sigma_I\}$
34:         if $\exists\, p \in P_1 : \text{SIZE}(p_{sigs}) > Q$                ▷ commit
35:             $V \longleftarrow p$
36:             $P \longleftarrow \emptyset$
37:     else if SIZE($P_0$) > 0
38:         if $\neg$ HAS_VOTED_IN_LAST_ROUND($P_0$)
39:             $p \longleftarrow$ PROPOSAL_WITH_MOST_VOTES($P_0$)
40:             $\sigma_I \longleftarrow \text{SIGN}(ID \oplus p_{version} \oplus p_{round} \oplus 0 \oplus p_{value})$
41:             $p_{sigs} \longleftarrow p_{sigs} \cup \{\sigma_I\}$
42:         if $\exists\, p \in P_0 : \text{SIZE}(p_{sigs}) > Q$          ▷ pre-commit
43:             $\sigma_I \longleftarrow \text{SIGN}(ID \oplus p_{version} \oplus p_{round} \oplus 1 \oplus p_{value})$
44:             $P \longleftarrow \{(p_{version}, p_{round}, 1, p_{value}, \{\sigma_I\})\}$
45:         else if NUM_VOTES_IN_LAST_ROUND($P_0$) > $Q$
46:                               ▷ possibly blocked, start new round
47:             $\sigma_I \longleftarrow \text{SIGN}(ID \oplus p_{version} \oplus (r+1) \oplus 0 \oplus p_{value})$
48:             $P \longleftarrow P \cup \{(p_{version}, (r + 1), 0, p_{value}, \{\sigma_I\})\}$
```

Figure 1. Consensus protocol for the Atomic Register.

protocol cannot progress to the next step. If a replica detects that this happened, it creates a new proposal with the value of the currently winning one, and increases the round number of the proposal. A new round is started, and all replicas have a new chance to vote. Since all honest replicas are voting on the winning proposals, it is likely that in only a few rounds one of the proposals will have reached a supermajority. This concept is known as meta-stability [72, 73].

In practice, however, it is not possible to reliably detect if the consensus is blocked. Up to f replicas can act Byzantine, including not sending anything at all. This means that after receiving $2f + 1$ votes, a replica needs to make a decision, as it is possible that no more votes will arrive. If all those votes are for the same proposal, a supermajority is reached and a new value is selected. Otherwise, the replica assumes that the consensus is blocked and starts a new round. We will prove that this assumption is safe in Section 3.4.

Optimistic BFT consensus. The outlined protocol is resilient against Byzantine actors. However, it includes a costly verification step each time a new state is received (Figure 1, line 26). If none of the replicas are acting Byzantine, this step can be delayed until a supermajority is reached (Figure 1, line 34 and 42). When the verification succeeds at that time, it is safe to accept the proposal as the new value. However, if the verification fails, the proposal cannot be accepted and it is not possible to find out which replicas are Byzantine.

The protocol uses a hybrid approach starting with a fast path for round numbers equal to zero. When verification in the end fails, a new round is created and the verification for all the following rounds is done every time a new state is received. This slow path is used until consensus is reached. The next time a new proposal is submitted for the next version, the round number will again be zero and the fast path will be used. 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 Data synchronization protocol

The previous section described the conceptual consensus protocol. This section explains how the state of an Atomic Register is replicated to other replicas.

The state of an atomic register, consisting of the current value and the set of proposals, is a state-based Conflict-free Replicated Data Type (CRDT) [76]. By using a state-based approach, rather than the operation-based approach of operation-based CRDTs, Operational Transformation [27], or blockchains, we only need to store the current state together with some metadata. This metadata is the version number and the set of current proposals. Replicas do not need to keep track of the state of other replicas, or which messages are already received by which replica. The replicas execute a Gossip protocol to exchange their current state

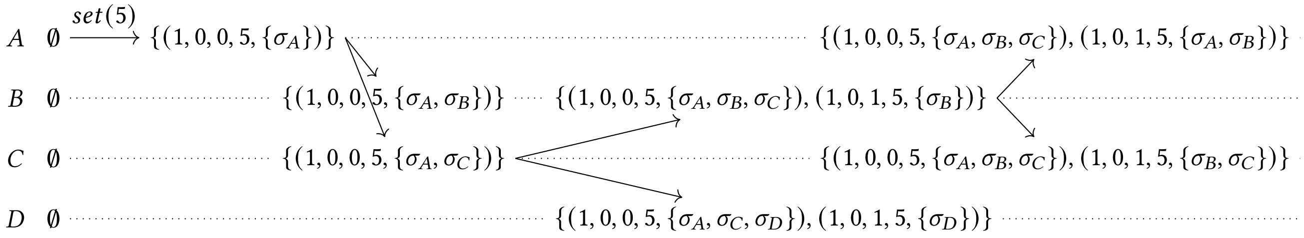


Figure 2. State-based synchronization of an Atomic Register with 4 replicas $A, B, C, D \in \mathbb{I}$. Only the set of current proposals is shown, containing tuples of ($version$, $round$, $step$, $value$, $signatures$).

with each other. Each time a new state is received, the local state is merged with it using the `MERGE` procedure in Figure 1.

An example of this process is shown in Figure 2. There are four non-Byzantine replicas with an empty set of proposals. Each proposal lists the version, the round, the step, the value, and the set of signatures of the replicas that voted for that proposal. The scenario starts with replica A proposing a new value. The state is replicated to the other replicas randomly, and all replicas aggregate the votes in the set of signatures. Once enough votes are aggregated for value 5, the proposal moves to step 1 and replicas can again vote to commit the value. Once a supermajority is reached in step 1, the value will be committed (not shown in Figure 2).

WebLedger uses Merkle-trees [57] to efficiently synchronize only the state of the registers that require an update [24]. Our approach is similar to Merkle Search Trees [10]. 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 the mismatching hashes to find out which registers must be synchronized.

3.4 Correctness

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.

Safety. To provide safety, all honest replicas need to decide on the same value for any version number. Assume two different replicas have committed different values X and Y with the same version. Either they are committed in the same round or in different rounds.

If they would have been committed in the same round, at least $2f+1$ replicas voted on X, and at least $2f+1$ replicas votes on Y. At most f replicas are Byzantine, so at least $f+1$ honest replicas voted on X, and at least $f+1$ honest replicas voted on Y. Given that no honest replica will vote for two different values in the same version-round-step combination, those $f+1$ replicas are all different. However, this would mean that $(f+1)+(f+1)+f = 3f+2$ replicas voted, which is impossible as there are only $3f+1$ replicas. So, it is not possible that both X and Y are committed in the same round.

Assume that X and Y are committed in different rounds. A new round can only be started when a potential split vote is detected. As discussed earlier, it is not possible to detect a blocked round reliably, as Byzantine replicas might not answer. Either the round was really blocked, or the round commits later, after a new round is already started. If the round would have been really blocked, it will never commit and consensus will be reached in one of the next rounds. Assume that the round (R_0) was not really blocked, and that eventually a supermajority is reached for one value (X). Since a new round (R_1) was started, replicas are allowed to vote on a different value in the new round (Y). In the worst case, a supermajority is reached in R_0 for X, but most replicas do not observe this supermajority and instead observe a split-vote and start a new round. Those observing the supermajority for X, will progress to step 1 and commit after $2f+1$ replicas agree. Replicas will only vote for this lower round R_0 if they haven't observed R_1 yet. So, to commit X, at least $f+1$ honest replicas have not seen R_1. To commit a value in R_1, $2f+1$ replicas need to vote, meaning that at least $f+1$ honest replicas need to vote. However, we just stated that $f+1$ of the $2f+1$ honest replicas have not seen R_1, so the remaining f honest replicas are not enough to reach a supermajority in R_1 and only a single value (X) will be committed. In all cases, only a single value gets committed for each version.

Liveness. To provide liveness, the protocol needs to eventually commit a new value if new values are 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. All those replicas do not need to be online at the same time, since the state is replicated to all available replicas, and votes can be verified by all replicas.

A replica that has not voted yet for the most recent version-round-step combination, will vote for the proposal with the most votes already. So the number of votes will increase over time, and eventually a supermajority of the replicas will have voted for a proposal. If a supermajority of the replicas vote for the same proposal in step 1, the value can be committed. If they do so in step 0, the protocol will progress to step 1. In case consensus was really reached in step 0, it will also be reached in step 1 and the value can be committed. In case the Byzantine replicas tricked an isolated replica into starting a new round with a different value, step 1 might not reach consensus. However, the Byzantine replicas will be identified because they voted twice for the same version-round-step combination in step 0, and consensus can be reached in one of the following rounds, with less Byzantine replicas present.

The other case is that a supermajority has voted, but this ended up with a split vote. In this case, a new round is started and replicas will again vote. They will choose the proposal with the most votes in the previous round, or if both proposals have an equal amount of votes, they will choose the largest value according to their lexicographical order. Different replicas might observe different split-votes, and therefor vote on different values in the next round. A replica will only progress to the next round if $2f + 1$ votes are received and a split-vote is present. Due to the state-based nature of the protocol, all votes for the previous rounds are also present when a new round is replicated. This prevents Byzantine replicas from voting twice on different proposals. Eventually, this will always be detected in one of the following rounds when enough honest replicas have replicated their state to each other, and the Byzantine replica will be excluded from the network. This guarantees liveness with a very high probability due to the concept of meta-stability [72, 73].

4 Architecture and implementation

This section describes the architecture, deployment, and implementation of WebLedger. This middleware architecture is key to support the BFT consensus and synchronization protocol described in the previous section. The middleware 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 proposals. The last subsection lists several performance optimization tactics.

4.1 Overall architecture

The WebLedger middleware architecture consists of five main components (Figure 3): (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. The *Public interface* 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 the value of the register at the given key changes.

Apart from those functions, the middleware also provides a constructor function to initialize the middleware by passing the following configuration as parameters:

- the list of all members of the network, together with their public key,
- the private key of the replica,
- the URL to the signaling server to set up the peer-to-peer connections,
- 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 a single party, and also 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) [40]. 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 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 state-based CRDTs to replicate the local state to other replicas using the *P2P Network* component. The Merkle-tree is constructed using the Blake3 [66] cryptographic hash function.

(iv) Membership. The *Membership* component contains all cryptographic material and is responsible for the signing and verification operations. The *Consensus* component uses

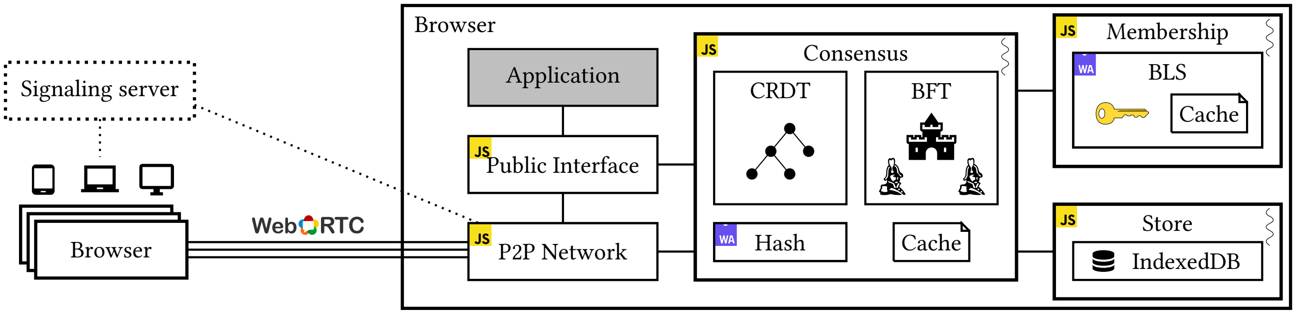


Figure 3. Browser-based architecture of WebLedger.

this for all cryptographic operations. We implemented two different versions of this component, one using ECDSA for signatures using the built-in WebCrypto [84] browser API (not shown in Figure 3), and a second implementation using an aggregate signature scheme called BLS [18]. Section 4.2 provides more details about the BLS implementation.

(v) Store. At last, the *Store* component saves all state to the IndexedDB [1] database. IndexedDB is a key-value datastore built inside the browser. Each atomic register and the Merkle-tree are serialized to bytes and stored here under the respective key. This enables users to close the browser and continue afterward without losing the current state.

4.2 Aggregate signatures using BLS

The consensus protocol in Section 3 is aggregation and verification intensive in terms 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 signatures to reduce storage and network footprint.

Boneh–Lynn–Shacham (BLS) [18] presented a signature scheme based on bilinear pairing on elliptic curves. The size of a single signature produced by BLS is short, since a signature is an element of an elliptic curve group. The aggregation algorithm [17] outputs a single signature as short as the others, unlike other approaches that rely on ECDSA or DSA (e.g. Schnorr [75]). These approaches require the protocol to store all signatures for aggregation and verification.

Other state-of-the-art BFT systems such as SBFT [33] and HotStuff [86] also use aggregate or threshold signatures. However, they use it in a different way. They let the leader compute the aggregate signature. WebLedger uses a different approach, as not all replicas are connected to each other, signatures need to travel accross multiple hops. WebLedger already aggregates the individual signatures immediately. This also means that a signature from one replica can be included multiple times in the aggregate signature.

$\mathbb{G}_0$ and $\mathbb{G}_1$ are two multiplicitive cyclic groups of prime order q. $\mathsf{H}_0 : \{0,1\}^* \rightarrow \mathbb{G}_0$ and $\mathsf{H}_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ 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 [16]. e is an efficient non-degenerating bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_t$. 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 $sk \overset{\$}{\leftarrow} \mathbb{Z}_q$, 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 $e(g_1, \sigma) \overset{?}{=} e(apk, \mathsf{H}_0(m))$.

Figure 4. Formal specification of the BLS signature scheme.

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 [16] 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 (multi-)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 once. Now, any replica can run Agg by only computing $\sigma_1 ... \sigma_n$. The security properties of BLS remain intact [16], and we obtain more efficient aggregations at scale. For the interested reader, we provide the mathematical background and formal specification of our optimized BLS scheme in Figure 4.

Aggregation of overlapping signatures. Replicas are required to aggregate multi-signatures in intermediate states of the consensus protocol. Figure 2 illustrates an example of such a situation. Replica B receives signature σ_A; it computes σ_B; and it aggregates them as $\sigma_{(A,B)}$. Later on, replica B receives $\sigma_{(A,C)}$ from replica C. Aggregation of $\sigma_{(A,B)}$ and $\sigma_{(A,C)}$ naturally includes a duplicate signature σ_A. The situation becomes worse when replica B wants to aggregate $\sigma_{(A,A,C,B)}$ and $\sigma_{(A,C,B)}$, which results in $\sigma_{(A,A,A,C,C,B,B)}$ (beyond Figure 2). Since each (multi-)signature is an element of an elliptic curve group, we are not aware of any technique merely relying on BLS to detect overlapping signatures as well as aggregating signatures resulting in ones with distinct public keys. Therefore, we keep extra metadata describing the multiplicity r of each public key. This information is (de)serialized and sent across the network along with the signatures. We encounter numerous multiplicities at different stages of the consensus protocol and the data synchronization mechanism. This results in many point additions on the curve. To reduce the performance overhead when key aggregation involves many duplicates, we can use this metadata to enable a better ordering of the operations.

4.3 Performance optimization tactics for browsers

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

Polyglot middleware using WebAssembly. WebAssembly [74] 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 [36], however, it is not as fast as native code [39].

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 [53] and compiled to WebAssembly to run inside a browser. Besides the performance improvement of WebAssembly over JavaScript, using Rust also enabled us to make use of well-tested Rust libraries instead of implementing these components ourselves in JavaScript.

Parallellization using Web Workers. Web Workers [37] 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* component run on the main thread together with the application. *Public interface* helps set up the other threads and pass the API-calls to the *Consensus* component. *P2P Network* 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, for example BLS-signature verification, to run in a separate thread without blocking concurrent operations in the other threads.

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 be allocated a spot in the WebAssembly memory buffer. In WebAssembly, these bytes can be decoded to the appropriate Rust 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 so a costly fetch from disk and decoding from bytes can be avoided.

Batching of writes for IndexedDB. The last important optimization concerns IndexedDB [1]. 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, 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 actually executed. On fixed intervals of five seconds, 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.

As not everything is immediately written to disk, failure can happen and lead to data loss. For updates received through the peer-to-peer network, this is no problem as those updates can be synchronized again later since the Merkle-tree will detect the missing updates. Local update operations by the user on this replica, are immediately written to disk and bypass the write-batching to avoid data loss.

5 Evaluation

We validated the WebLedger middleware with the loyalty points use case. The first section presents this validation. Next, we presents three different benchmarks with different scales. The first benchmark shows the performance results in the optimal scenario with no network failure or Byzantine failures. The second benchmark evaluates the performance in case of network failures, and in case of Byzantine failures. The last benchmark measures a detailed performance breakdown of WebLedger to show the bottlenecks in the current architecture and explain the previous results.

5.1 Validation in the loyalty points use case

The deployment 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 server to set up WebRTC peer-to-peer connections between the browsers. The web server is optional. Every merchant can also store those 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 setup multiple signaling servers to reduce potential misbehavior.

If we compare this lightweight setup with the infrastructure requirements for Hyperledger Fabric, we assess that WebLedger needs two central components and one browser per merchant. Hyperledger Fabric needs at least one peer server, one REST server, one certificate authority and two CouchDB servers per merchant. Merchants at small stores or farmers' markets will prefer to use a simple browser-based web application with a minimal back-end infrastructure.

To have a baseline, we compare WebLedger to two other existing state-of-the-art systems for BFT consensus: BFT-SMART [15] and Tendermint [20, 21].

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 [2], which simulates the latency of a 4G network [68]. To make sure the test results are reliable, every test is executed 10 times. We implemented two versions of the middleware with different signature schemes. The first version uses BLS signatures which supports signature aggregation as explained in Section 4.2. The other version uses ECDSA signatures which are aggregated in a set.

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 giving some loyalty points to a customer or a customer redeeming their loyalty points with a merchant. We compare the latency, network bandwidth, and disk usage for both implementations with ECDSA and BLS, with a different number of browsers and transaction throughputs. We show the 99th percentile latency as all users should experience fast confirmation times, and not only the average user [24].

5.2 Optimal scenario

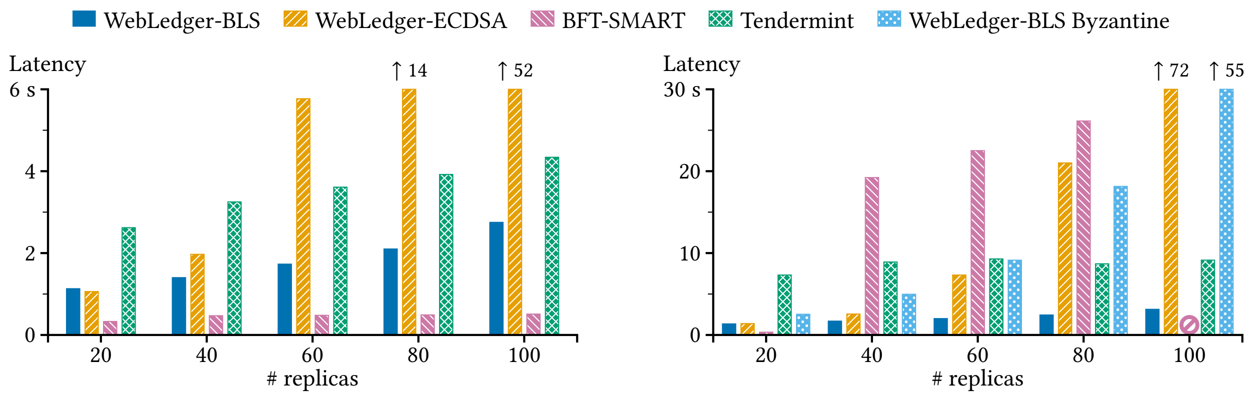
In the optimal scenario, every replica is honest and no replicas fail, meaning that the optimistic fast path can be used. The aggregate signature is verified only at the end, avoiding costly verifications after every message. As every replica is honest, this aggregate signature is correct and the new value can be accepted by all replicas.

Figure 5a shows the 99th percentile latency for different number of browsers and 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. The BLS implementation can confirm transactions within 3 seconds, even with a network of hundred browsers. The ECDSA implementation performs well for small networks, but needs too much time in the larger networks with 80 and 100 replicas. BLS only needs a single aggregate signature, while ECDSA needs to keep a set with 100 signatures in the largest network we tested.

BFT-SMART can confirm transactions within half a second. This is because all replicas communicate directly with each other, whereas both WebLedger and Tendermint use Gossip and need multiple hops before all replicas are reached. Furthermore, BFT-SMART uses HMAC to sign requests, which are an order of magnitude faster than the assymmetric signatures used in WebLedger and Tendermint.

Figure 6 shows the bandwidth requirements for all four technologies. BLS uses always less bandwidth compared to the ECDSA implementation of WebLedger. In the large scale scenario with 100 browsers, WebLedger-BLS uses about 1.2 Mbit/s, which is acceptable for a typical mobile network. Tendermint and WebLedger-ECDSA have a higher network usage as they need to store the individual signatures, instead of one aggregate signature using BLS.

Figure 7 shows the disk usage. BLS improves the disk usage 8 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 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



(a) Normal scenario

(b) Failure scenario

Figure 5. 99th percentile latency in seconds for different number of replicas for the different scenarios and technologies.

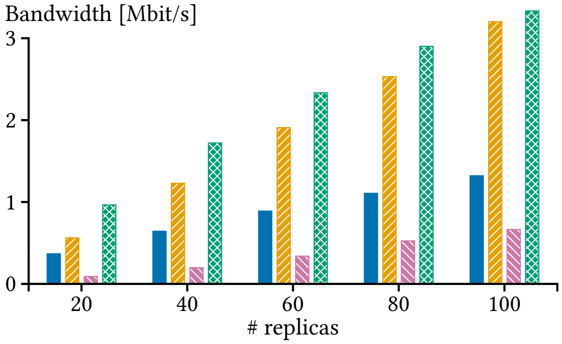


Figure 6. Network usage for different number of replicas.

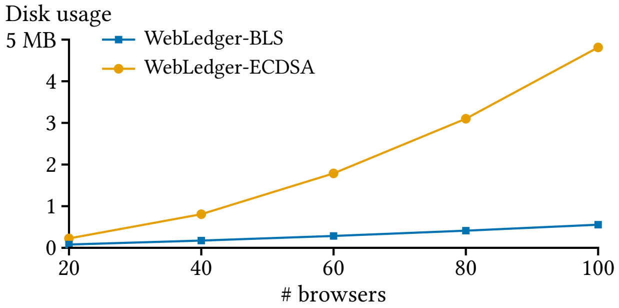


Figure 7. Average disk usage for WebLedger.

do not have hundreds of gigabytes storage capacity to store a full blockchain.

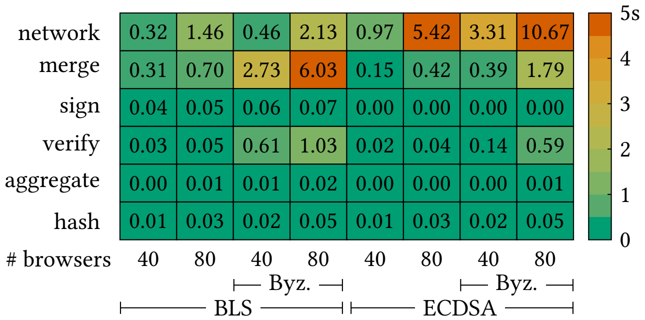
5.3 Failure and Byzantine scenario

The same benchmark is repeated with 25% of the replicas failing during the benchmark. As all systems are Byzantine fault-tolerant, they should be able to tolerate up to 33% of the replicas failing or acting Byzantine.

Figure 5b shows the latency in this scenario. WebLedger-BLS is not impacted much by the failing replicas, and can still confirm transactions within 4 seconds. The impact on Tendermint is also small, but latency is doubled to about 10 seconds. BFT-SMART can handle the small scale test with 20 replicas well. But with larger network configurations the latency grows to more than 20 seconds. It cannot handle the case with 100 replicas. The latency is increased this much because a new leader needs to be elected when the old one fails or disconnects. This process takes some time, during which no transaction can be committed. WebLedger and Tendermint do not suffer from this problem. WebLedger does not have a leader, so the failure of random replica has little impact. 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.

For WebLedger-BLS, we performed an extra benchmark with Byzantine replicas. In every optimistic round, the Byzantine replicas make the aggregate signature invalid. As the signature is 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. The work done in the first round is therefore always lost in this scenario. For the other rounds, the signatures are verified for every message, so malicious replicas can be detected and excluded from the network. In these rounds, 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 also shown in Figure 5b. WebLedger can handle Byzantine replicas very well for smaller networks, however for networks of size 80 and 100 replicas, latency becomes respectively 19 and 55 seconds. Which is too much for our interactive use cases.

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 currently elected leader happens to be Byzantine, it can delay the consensus until some timers end and the replicas elect a new leader [7].



	BLS				ECDSA			
			Byz.				Byz.	
# browsers	40	80	40	80	40	80	40	80
network	0.32	1.46	0.46	2.13	0.97	5.42	3.31	10.67
merge	0.31	0.70	2.73	6.03	0.15	0.42	0.39	1.79
sign	0.04	0.05	0.06	0.07	0.00	0.00	0.00	0.00
verify	0.03	0.05	0.61	1.03	0.02	0.04	0.14	0.59
aggregate	0.00	0.01	0.01	0.02	0.00	0.00	0.00	0.01
hash	0.01	0.03	0.02	0.05	0.01	0.03	0.02	0.05

Figure 8. Total wall-clock time spend on one replica on average in seconds, including I/O, for 1 transaction. Network excludes network latency. Merge includes sign, verify, aggregate and hash.

5.4 Breakdown of performance results

To explain the results obtained in the previous two benchmarks for WebLedger, we performed another benchmark doing only one update and measuring the time a replica spends on average on each operation. Figure 8 shows this performance breakdown over the 6 most important operations. The *network* row contains the overhead of sending a message through WebRTC to a different browser and receiving this message. This time does not include the latency of the connection. Most of this time is spent inside the internals of the browser itself, rather than in the code of the middleware. The *merge* row contains the time spent merging the state of a remote replica with the local state, it includes maintaining the Merkle-tree, the `merge` operation from Figure 1, as well as the cryptographic operations: sign, verify, aggregate and hash. The *sign*, *verify*, *aggregate* and *hash* row contain exactly what their names say. The *aggregate* row for the ECDSA implementation only takes the union of two sets with signatures, there is no actual cryptography involved. The numbers do not add up to the results shown previously as some operations are executed in parallel in a different WebWorker thread. The times shown are wall-clock times, so also the time spent waiting on another thread is included.

We can see that the performance characteristics of the two implementations are different. The classical implementation using ECDSA is severely limited by the overhead of WebRTC and processing those messages, rather than the core cryptography. The BLS implementation on the other hand is limited by the computational overhead of BLS. The network overhead takes some time, but as the messages are only a fraction of the size of those from ECDSA, this overhead is a lot less. For example with 80 different replicas, an aggregate signature in BLS only takes up the size of one single signature and some metadata of a few hundred bytes. An aggregate signature in ECDSA consists of 80 different signatures, so it takes up as much size as 80 signatures. The aggregation step in BLS is quite fast. However, the verification step takes more time. This is partly because BLS in general is slower than ECDSA, but also because the WebAssembly implementation is slower than a real native environment. The ECDSA implementation uses the built-in WebCrypto [84] libraries which use the native functions provided by Chrome.

5.5 Conclusion

We have shown that WebLedger can be used for our loyalty points use case with up to 60 different merchants, even when some of them are acting maliciously. WebLedger is especially robust against network and node failures, which are typical in a mobile setting. WebLedger can confirm transactions fast, in the order of seconds, without needing a complex back-end setup or wasting a lot of energy. WebLedger has a small storage footprint due to its state-based nature. The current limitation of WebLedger (with BLS) is the verification phase that needs to be performed for every new aggregated signature that is received. When the default operation assumes that there are no malicious replicas being present, WebLedger can scale to even more replicas, since the fast path without intermediate verifications can be used.

6 Related work

Several client-side frameworks for data synchronization between web applications exist: Legion [82], Yjs [64, 65], and Automerge [43]. They make use of various kinds of Conflict-free Replicated Data Types (CRDT) [76] 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 small peer-to-peer discovery service. However, they assume trusted operation as the default setting. None of them can tolerate malicious parties.

Open or permissionless blockchains such as Bitcoin [62] and Ethereum [22, 85] allow everyone to participate and use Proof-of-Work (PoW) to reach agreement over the ledger [35]. However, PoW has several flaws [13]. PoW uses a lot of processing power and energy [67] 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. The Bitcoin blockchain for example has a total size of 304 GB in 2020. Simplified Payment Verification (SPV) mode [62] 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 [45] uses PoW for a separate identity chain to guard against Sybil attacks but uses a BFT protocol to actually 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 [75]. However, CoSi needs multiple communication round-trips through the peer-to-peer network to generate the multi-signature and assumes a synchronous network.

Tendermint [20, 21], used in the Cosmos blockchain [47], 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 cryptocurrency to work together to control the network.

Instead of reaching consensus between all the replicas of the network, Stellar Consensus Protocol [51, 54] 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 [61].

Other protocols use a randomized approach. Ouroboros [42], HoneyBadger [60] and BEAT [25] use distributed coin flipping for the consensus. HoneyBadger [60] also uses threshold signatures [77] for censorship resilience. Algorand [32] uses Verifiable Random Functions [58] to select a random committee to participate in the next consensus round.

Avalanche [72, 73] uses meta-stability to reach consensus by sampling other replicas without any leader. While Avalanche is also a lightweight and scalable, leaderless system, it need to be able to sample all other validators. The number of connections one can open in a browser is limited.

Permissioned blockchains such as Hyperledger Fabric [3] have closed membership and often use a BFT consensus protocol to order transactions. The first known BFT protocol is Practical Byzantine Fault-Tolerance (PBFT) [23]. Other protocols bring improvements to the original PBFT protocol. Zyzzyva [46] 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 deployed on a local area network. They generally work in two phases: the first phase guarantees proposal uniqueness, and the second phase guarantees that a new leader can convince replicas to vote for a safe proposal. HotStuff [86] proposed a three-phase protocol to reduce complexity and simplify leader replacement. This makes HotStuff much more scalable. All of these algorithms use a leader to drive the protocol. When the leader is malicious, performance can degrade quickly [7]. GeoBFT [34] is a topology-aware and decentralized consensus protocol, designed for scalability in a geo-distributed setting.

Another approach is to use a trusted hardware component [11, 41, 50, 83, 87]. These approaches are faster and less computationally intensive but require specialized hardware to be present. Moreover, trusted execution environments have been broken in the past [44, 49, 81].

There are several proposals to improve the performance and response time of Hyperledger Fabric. StreamChain [38] reaches consensus over a stream of transactions instead of blocks. FabricCRDT [63] 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 [19] supports commuting transactions to be applied out-of-order. A novel design for gossip in Fabric [12] 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 [70] or state channels for Ethereum [56, 59, 69] 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 assumption is violated, the underlying blockchain needs to be used to resolve the conflict, or a trusted third party can be used [55]. WebLedger uses a similar state-transitioning protocol where only the latest collectively agreed state needs to be stored. However, WebLedger can tolerate both failing and malicious replicas, without resorting to a blockchain or a trusted third party.

7 Conclusion

In this paper, we presented WebLedger. A browser-based middleware for decentralized, community-driven, web applications. WebLedger uses an optimistic, leaderless BFT consensus protocol, combined with a robust and efficient state-based synchronization protocol based on state-based CRDTs and Merkle-trees. WebLedger 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. WebLedger 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 with traditional blockchains, WebLedger does not store a transaction log or blockchain, keeping the overall storage footprint small.

References

[1] Ali Alabbas and Joshua Bell. 2018. *Indexed Database API 2.0.* Candidate Recommendation. W3C. https://www.w3.org/TR/2018/REC-IndexedDB-2-20180130/

[2] Werner Almesberger. 1999. Linux network traffic control – implementation overview.

[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, New York, NY, USA, Article 30, 15 pages. 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 International Publishing, 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)*. Association for Computing Machinery, New York, 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), 45 pages. 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, IEEE Computer Society, USA, 297–306. https://doi.org/10.1109/ICDCS.2013.53

[8] Anonymous Author(s). 2019. Anonymized to maintain double-blind.

[9] Anonymous Author(s). 2019. Anonymized to maintain double-blind.

[10] Alex Auvolat and François Taïani. 2019. Merkle Search Trees: Efficient State-Based CRDTs in Open Networks. In *SRDS 2019 - 38th IEEE International Symposium on Reliable Distributed Systems.* IEEE, Lyon, France, 1–10. https://doi.org/10.1109/SRDS.2019.00032

[11] 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)*. Association for Computing Machinery, New York, NY, USA, 222–237. https://doi.org/10.1145/3064176.3064213

[12] 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.

[13] 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, New York, NY, USA, 13–18. https://doi.org/10.1145/3284764.3284767

[14] 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/

[15] 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 Computer Society, USA, 355–362. https://doi.org/10.1109/DSN.2014.43

[16] 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 International Publishing, Cham, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[17] 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

[18] 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

[19] 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, 28. https://doi.org/10.1109/SRDS51746.2020.00023

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

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

[22] Vitalik Buterin et al. 2014. *A next-generation smart contract and decentralized application platform.* White paper. ethereum.org.

[23] 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

[24] 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, New York, NY, USA, 205–220. https://doi.org/10.1145/1294261.1294281

[25] 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)*. Association for Computing Machinery, New York, NY, USA, 2028–2041. https://doi.org/10.1145/3243734.3243812

[26] 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

[27] 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

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

[29] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (April 1985), 374–382. https://doi.org/10.1145/3149.214121

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

[31] 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

[32] 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, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[33] 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

[34] 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

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

[36] 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

[37] Ian Hickson. 2015. *Web Workers*. Working Draft. W3C. http://www.w3.org/TR/2015/WD-workers-20150924/

[38] 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)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3284764.3284765

[39] 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.

[40] Cullen Jennings, Henrik Boström, Jan-Ivar Bruaroey, Adam Bergkvist, Daniel Burnett, Anant Narayanan, Bernard Aboba, and Taylor Brandstetter. 2019. *WebRTC 1.0: Real-time Communication Between Browsers*. Candidate Recommendation. W3C. https://www.w3.org/TR/2019/CR-webrtc-20191213/

[41] 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)*. Association for Computing Machinery, New York, NY, USA, 295–308. https://doi.org/10.1145/2168836.2168866

[42] 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 International Publishing, Cham, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

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

[44] 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

[45] 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

[46] 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)*. Association for Computing Machinery, New York, NY, USA, 45–58. https://doi.org/10.1145/1294261.1294267

[47] Jae Kwon and Ethan Buchman. 2019. *Cosmos Whitepaper: A Network of Distributed Ledgers.* White paper. cosmos.network. https://cosmos.network/cosmos-whitepaper.pdf

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

[49] 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.

[50] 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

[51] 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)*. Association for Computing Machinery, New York, NY, USA, 80–96. https://doi.org/10.1145/3341301.3359636

[52] 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

[53] 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)*. Association for Computing Machinery, New York, NY, USA, 103–104. https://doi.org/10.1145/2663171.2663188

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

[55] 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)*. Association for Computing Machinery, New York, NY, USA, 16–30. https://doi.org/10.1145/3318041.3355461

[56] 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 International Publishing, Cham, 35–49. https://doi.org/10.1007/978-3-030-43725-1_4

[57] 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.

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

[59] 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 International Publishing, Cham, 508–526. https://doi.org/10.1007/978-3-030-32101-7_30

[60] 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, New York, NY, USA, 31–42. https://doi.org/10.1145/2976749.2978399

[61] 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.

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

[63] 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)*. Association for Computing Machinery, New York, NY, USA, 110–122. https://doi.org/10.1145/3361525.3361540

[64] 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 International Publishing, Cham, 675–678.

[65] 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, New York, NY, USA, 39–49. https://doi.org/10.1145/2957276.2957310

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

[67] 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 Computer Society, USA, 280–285. https://doi.org/10.1049/cp.2014.0699

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

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

[70] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf

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

[72] Team Rocket. 2018. *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies*. Technical Report. AVA Labs. https://avalanchelabs.org/avalanche.pdf

[73] 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

[74] Andreas Rossberg. 2019. *WebAssembly Core Specification*. Recommendation. W3C. https://www.w3.org/TR/2019/REC-wasm-core-1-20191205/

[75] 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

[76] 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.

[77] 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.

[78] 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

[79] Thomas Steiner. 2018. What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is Not a Web Browser. In *Companion Proceedings of the The Web Conference 2018* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 789–796. https://doi.org/10.1145/3184558.3188742

[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, 19.

[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] 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

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

[85] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[86] 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*. Association for Computing Machinery, New York, NY, USA, 347–356. https://doi.org/10.1145/3293611.3331591

[87] 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

[88] 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/

[89] Web3 Foundation 2020. *About*. Web3 Foundation. https://web3.foundation/about/