Enabling secure and resource-efficient blockchain networks with VOLT

Srinath Setty Soumya Basu* Lidong Zhou Michael L. Roberts Ramarathnam Venkatesan
Microsoft Research *Cornell University

Abstract

This paper describes VOLT, a permissioned blockchain network for a group of autonomous organizations to automate cross-organizational business processes. Specifically, VOLT ensures that a correct member—*without* trusting anyone else—shares a consistent history of transactions with other members in the system. VOLT achieves this strong security property using the concept of a *self-verifying ledger*: an append-only ledger of transactions that can be checked locally by a *verifier* process at each node; the ledger also embeds succinct cryptographic messages encoding views of members to enable each verifier to locally determine globally-committed transactions. To orchestrate this concept, VOLT employs a logically centralized—but untrusted—*service provider*. We instantiate the service provider using a highly-available cloud service built atop a fault-tolerant cloud storage service. An experimental evaluation of our implementation demonstrates that VOLT achieves tens of thousands of transactions per second for a realistic 50-node payment network.

1 Introduction

Blockchain technology promises to dramatically transform business processes that span multiple autonomous organizations in areas such as finance [84], supply chains [38], and the public sector [77]. The fundamental reason is that it enables a new form of trustworthy business processes among multiple, mutually distrusting business entities by offering automation, auditability, and integrity without requiring participants to trust any single entity, or mutual trust among the participating entities. Example applications include: (a) a distributed payment network that enables banks to execute clearing and settlement mechanism [39]; (b) a group of financial institutions maintaining asset registries for securities such as bonds and gold [51].

Motivated by these mission-critical applications, our goal is to build a *permissioned* blockchain network: a system that enables a group of mutually distrusting member organizations to define rules for validating transactions and then maintain a shared, append-only ledger containing an ordered list of valid transactions processed by the network. The primary requirement of such a system is that all participants following the protocol must share a *consistent view* of the shared ledger; i.e., a pair of correct participants must share the same prefix of the ledger. Furthermore, for many applications, the system must maintain high throughput (e.g., tens of thousands of transactions per second) at relatively low latency (e.g., a few seconds).

While Byzantine fault-tolerant (BFT) protocols [22, 31, 36, 55] seem to solve the core consensus problem, subtle, but fundamental, aspects render these protocols insufficient for permissioned blockchain networks, even though the underlying substrate in BFT remains relevant. These BFT protocols were primarily designed to replicate services in a datacenter environment—to guard against crash failures and Byzantine behavior by a subset of the replicas. In the context of permissioned blockchain networks, the protocol has to be carried out across trusted domains with each member representing a participating organization (such as a bank), rather than a replica of a replicated service typically in the same trusted domain. Consequently, the desirable guarantees must be defined from the perspective of each member, rather than for a replicated service as a whole only. The assumptions in these BFT protocols must also be revisited. As an example, the availability properties of each member (representing a well-resourced autonomous business entity) are different from a replica running on an unreliable machine in a traditional replicated system. Furthermore, under the status quo, an organization does not trust the computing infrastructure of other organizations for the security of its cross-organizational business transactions.

These observations lead to the design of a system called VOLT, a permissioned blockchain in which a member—*without* trusting anyone else—constructs an append-only ledger such that it shares a verifiably consistent prefix with all other members. To accomplish this, VOLT introduces the concept of a *self-verifying ledger*. Specifically, VOLT employs a simple *verifier* run by each member node, and each verifier *lazily* verifies an append-only ledger maintained by a highly-available—but untrusted—cloud service, which we refer to as the *blockchain service provider (BSP)*. In other words, VOLT's approach is to delegate the task of maintaining a ledger to the BSP and have each verifier execute simple, local checks on the end-to-end behavior of the service. Naturally, these checks are independent of the internal details of how these services are implemented and deployed. More fundamentally, VOLT's approach decouples the work of a member from the rest of the system, thereby offering desirable flexibility to each member in terms of implementation, operation, and configuration.

Operationally, to ensure all correct members in the system share a consistent view of the ledger maintained

by the BSP, VOLT's *verifiers persist succinct messages on the BSP's ledger itself*. Specifically, each verifier periodically submits a special transaction, called an *approval transaction*, to the BSP to be included in the BSP's ledger. Such an approval transaction cryptographically encodes a member's complete local view of the BSP's ledger. We show that each verifier can process approval transactions (submitted by other members) recorded in the BSP's ledger as part of its local checks to compute an *approved prefix*—a prefix of the BSP's ledger shared by other members in the system. Because computing an approved prefix is a local operation, it offers the flexibility to members to layer business-specific policies to determine when a transaction can be considered committed.

We implement VOLT's BSP as a cloud service atop Azure Cosmos DB [1], a fault-tolerant cloud storage system, and VOLT's verifier and the approval protocol in a client library that interacts with the BSP. To evaluate VOLT's costs, we build a distributed payment network. Our experimental evaluation demonstrates that VOLT can process up to 20K transactions/second with one second latency and scales to 50 members in a permissioned blockchain network.

This paper contributes the following:

- A new foundation—the concept of a self-verifying ledger instantiated with a simple verifier and an untrusted BSP—for building practical distributed ledgers.
- Extensions to the core system that offer flexible, member-controlled policies, decentralized membership management, and scale-out throughput via partitioning.
- An experimental evaluation of our implementation that demonstrates VOLT scales to realistic workloads: for 50-member payment network, our system achieves about 20,000 transactions per second.

2 Requirements and design principles

This section provides an overview of VOLT's target applications as well as its high level architecture.

2.1 Applications and requirements

Target applications. Our principal goal is to enable an emerging class of applications that employ blockchain networks to re-architect mission-critical business processes [13, 38, 39, 51, 77, 84]. In a nutshell, these applications involve a group of autonomous entities processing transactions that span multiple entities (e.g., payment transactions, asset trading, transferring title of a property, etc.), and then maintaining a ledger containing those transactions (for auditability and compliance).

Setup and requirements. Our target system consists of a set of members, and at setup time, they agree on the initial list of members identified by their public keys in a digital signature scheme. They also agree on code (or a state machine) for validating transactions. The system must enable these members to construct an append-only ledger containing an ordered list of valid transactions. Specifically, we wish to build a system that satisfies the requirements listed below.

1. *Consistency.* A correct member (i.e., one that follows its prescribed protocol) shares a consistent view of a ledger containing an ordered list of valid transactions processed by the system.
2. *Progress.* If a participant submits a valid transaction, then it eventually (e.g., in a few seconds) appears in the views of the ledger on all correct participants.
3. *Governance.* The system enables members to evolve membership in the system (e.g., admit or evict participants) and transaction processing rules. Concretely, members share a consistent view of membership in a blockchain network.

2.2 Design goals and principles

As discussed in Section 1, traditional BFT protocols focus on replicating a (deterministic) service across a set of unreliable machines to emulate a single correct service. Permissioned blockchains is fundamentally a different problem, even though the core building blocks and mechanisms in a traditional BFT protocol remain relevant in the design of a permissioned blockchain. We now discuss the key observations and principles that guide the design of our system, VOLT.

Egocentric members. Members participating in permissioned blockchains are *egocentric* with their own independent identities, as they usually represent organizations (e.g., banks) to carry out business-critical transactions. This is in contrast to the largely nameless replicas in traditional replicated services, where the correctness of the replicated service is what matters.

Trust thyself only. Each member defines its own trust domain and engages with other distrusting members across trust boundaries in a permissioned blockchain. Thus, the system must empower each member to enforce end-to-end guarantees through a *local* mechanism within its trust domain. Furthermore, the mechanism must be flexible enough to allow each member to implement the mechanism independent of how the rest of the system is architected. Whereas in a traditional BFT protocol, the entire protocol is architected in service of one thing: emulating a single, correct service out of multiple unreliable servers.

Realistic liveness. A BFT protocol's goal is to tolerate machine failures, thus requiring all nodes in the system to participate in every step of the protocol is explicitly ruled out. Whereas in a permissioned blockchain, each member represents an organization and can be expected to have

high availability as any online business-critical services (e.g., financial transactions). Such a member can be expected to be internally fault-tolerant and any downtime over minutes is considered an incident that triggers immediate attention. Thus, permissioned blockchain protocols can expect all members to participate in the system—at least periodically. However, if a member fails to participate at some point, the system must not sacrifice safety.

With these observations, VOLT embraces the following key design choices.

End-to-end verification with a simplified protocol and a local verifier. Instead of adopting the non-trivial BFT protocol, the design of VOLT uses a simplified protocol with a single message type, coupled with a local *verifier* that each member runs locally for end-to-end verification. The local verifier is the only piece of code that a member needs to trust for safety. Due to its simplicity, a member can have its own verifier implementation. The use of a local verifier effectively decouples each member from how the system is architected outside of its trust domain.

Separation of policy and mechanism via a self-verifying ledger abstraction. VOLT separates the policy from its mechanism by implementing a self-verifying ledger abstraction using an untrusted blockchain service provider (BSP). The same mechanism can be used to implement a policy for utmost safety, for example, where a member does not need to trust any other member for safety and can ensure progress as long as all members remain available to participate. Other policies similar to BFT's quorum failure assumptions can also be implemented. A member can even choose a policy in between based on it own business requirements and the policy can even be different for different transactions (e.g., a member might opt for utmost safety for high-value transactions, but wait for only a sufficient quorum for low-value transactions to make faster progress).

3 Design and architecture

This section describes the VOLT's core building blocks to realize a permissioned blockchain network based on a self-verifying ledger.

3.1 Overview of VOLT's self-verifying ledgers

Figure 1 depicts VOLT's high level architecture to implement the concept of a self-verifying ledger. At its core, VOLT consists of a simple verifier process run by each member such as a financial institution that wishes to participate in a blockchain network with other financial institutions. To orchestrate a blockchain network with the properties discussed earlier (§2.1), VOLT relies on an untrusted service provider that we call a blockchain service provider (BSP). The job of the BSP is to maintain an append-only *ledger* of transactions (discussed in the next subsection) submitted by different members in a blockchain network. In practice, the BSP could be offered as a commercial service by a cloud provider such as Amazon AWS and Microsoft Azure [6].

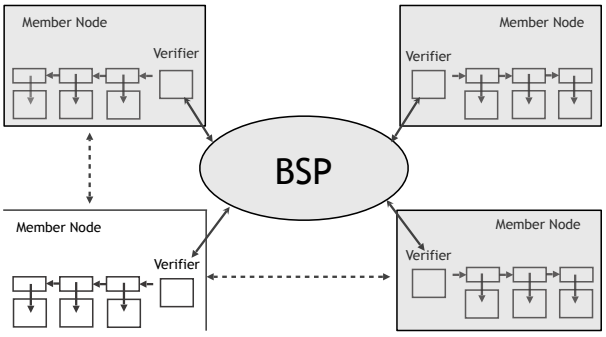

FIGURE 1—High level architecture of VOLT. Members run a simple verifier process that communicates with the BSP, executes local end-to-end checks, and outputs a consistent view of the BSP's ledger. The internal implementation details of the BSP and other member nodes (denoted with shaded color) do not compromise the safety guarantees that VOLT's verifier ensures to a correct member (denoted without a shaded color). Dotted arrows represent optional communication channels.

At setup time, the set of members in a blockchain network initialize a ledger on the BSP. Then, each verifier interacts with the BSP through a set of well-defined APIs and constructs a view of the shared ledger. In the subsections ahead, we discuss how each verifier (running on the infrastructure controlled by a member) can ensure properties we desire (§2.1)—by relying on *end-to-end verification of simple properties* (§3.4).

Threat model. VOLT achieves its goals (§2.1) in the following threat model. We assume participating members and the BSP cannot violate standard cryptographic hardness assumptions. We assume that malicious members can arbitrarily deviate from their prescribed protocol. We assume that the network could be adversarial by dropping, reordering, and duplicating messages. However, we assume that correct entities in the system can eventually communicate with other correct entities. We also assume that the network is eventually synchronous for liveness [44]. Finally, we assume that all members in a blockchain network periodically participate in the system (this assumption can be relaxed at the cost of weakening guarantees; §3.5).

3.2 Ledgers and blockchain network state

A core ingredient in VOLT is an append-only ledger. For concreteness, we define a ledger as follows. A ledger, $\mathcal{L}$, is a chain of blocks $(B_1, B_2, \ldots, B_l)$ where each block, B_i, is a tuple (M_i, D_i). D_i is a *data block*, which contains a list of transactions, and M_i is a *metadata block*. M_i is a triple:

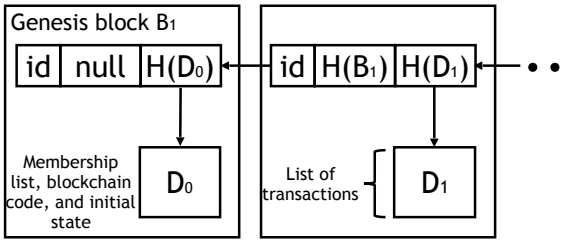

FIGURE 2—Overview of an append-only ledger that VOLT uses to maintain data of a blockchain network. The data structure itself is borrowed from decentralized cryptocurrencies such as Bitcoin: each block contains transactions and embeds a cryptographic hash of its predecessor using a collision-resistant hash function $H(\cdot)$. In VOLT, however, the ledger also acts as a mechanism for disseminating a blockchain network's metadata, for example, the first block in a ledger stores rules that govern the blockchain network it represents.

$(id, prev, h)$ where id is the identity of the ledger (the BSP can host more than one ledger), $prev \leftarrow H(M_{i-1})$ if $i > 1$ and `null` otherwise, $h \leftarrow H(D_i)$, and $H(\cdot)$ is a collision-resistant hash function. Furthermore, the height of a ledger $height(\mathcal{L})$ is the number of blocks in $\mathcal{L}$.

From a ledger to a blockchain network. We now discuss how VOLT uses the ledger data structure discussed earlier to maintain data of a blockchain network. Figure 2 provides an overview of this, but we discuss details below.

Network constitution. A VOLT blockchain network must reliably maintain the list of participating members, which we refer to as the *constitution*. At setup time, the list of members identified by their public keys is recorded on the first block of the ledger, called the *genesis* block, maintained by the network. The constitution is not however a static state of a VOLT blockchain network; VOLT enables its members to evolve the membership list as well as other metadata associated with the network (§3.6).

Network state and code. Besides the constitution of a blockchain network, there are two other elements: (i) blockchain network state and (ii) blockchain code. Each member's state in VOLT is a set of key-value pairs. For example, if a participating member is a financial institution, the list of key-value pairs of a member can represent account balances of its customers, or the ownership information of physical or digital assets [12]. Thus, the state of a VOLT network is a set of individual participating members' state. The blockchain code of a VOLT network defines code that validates transactions and updates the network state. For example, a transaction that transfers money from an account of member bank X must include a valid digital signature by X's private key and there should be enough money in the account. The initial blockchain network state as well as blockchain code are stored in the genesis block of a ledger along with the initial constitution.

3.3 Blockchain service provider (BSP)

As discussed earlier, the BSP's job is to maintain and update an append-only ledger (introduced above), and make it available for members. We abstract the BSP using the following three simple APIs: `init`, `append`, and `read`.

1. `init(Block b)` $\rightarrow$ `LedgerId id`: creates a new ledger with `b` as its first block and returns its identity `id`.
2. `append(LedgerId id, Transaction t)` $\rightarrow$ `Bool b`: appends `t` to the ledger with identity `id` if `t` is a valid transaction according to the associated blockchain code, and returns whether `t` is valid.[1]
3. `read(LedgerId id, Height h)` $\rightarrow$ `LedgerSuffix l`: returns the requested suffix of a ledger containing blocks with height greater than `h`.

VOLT implements these APIs using a cloud-hosted service with high availability by leveraging a fault-tolerant cloud storage service (§5).

3.4 VOLT's verifier and end-to-end verification

Recall that the primary requirement of a blockchain network is ensuring members share a consistent view of an append-only ledger (containing an ordered list of transactions) despite arbitrary failures of various entities in the system. We now discuss the design of a simple verifier run by each member to perform end-to-end verification, which in turn enforces desirable consistency semantics—even if the BSP or any number of members misbehave.

Approval transactions. A key building block to achieve end-to-end verification (and hence consistency) is a special transaction called an *approval transaction*. An approval transaction succinctly encodes a member's local view of the BSP's ledger. Concretely, given a well-formed ledger $\mathcal{L} = (B_1, \ldots B_l)$, an approval transaction is of the form: $(m, l, h)_{\sigma_m}$, where m is the identity of a member (i.e., its public key), $l = height(\mathcal{L})$, $h = H(B_l)$, and σ_m is a signature using the private key of m.

Observe that an approval transaction encodes the tuple (l, h), so it uniquely certifies a ledger $\mathcal{L}$ (including its contents) up to height l due to the collision-resistance properties of $H(\cdot)$. Also, the transaction is signed using the private key of m, so it cannot be forged by the BSP or any malicious member due to the unforgeability properties of the digital signature scheme.

Constructing a self-verifying ledger by embedding approvals. VOLT embeds approval transactions in the BSP's ledger itself, along with other "normal" transactions submitted as part of the blockchain network. Consequently, members can locally determine a consistent prefix of the

[1] For higher throughput, the BSP batches a configurable number of transactions in a single block.

BSP's ledger—by processing each others' approval transactions embedded in the BSP's ledger, as we explain next.

We first define a *local* validity check on a well-formed ledger that embeds approval transactions as follows. A well-formed ledger $\mathcal{L} = (B_1, B_2, \ldots)$ is *valid* if, for every approval transaction included in it $s = (m, l, h)_{\sigma_m}$, σ_m is a valid signature by m on s, $height(\mathcal{L}) \geq l$, and $H(B_l) = h$.

Approved ledger prefix. The verifier on each member m maintains a local copy of the BSP's ledger, $\mathcal{L}_m$. m periodically executes the following to compute a prefix of the ledger, called an *approved prefix*.

1. Suppose $\mathcal{L}_m = (B_1, B_2, \ldots, B_i)$ where $i \geq 1$.
2. Call the BSP's `read` to get a suffix of the BSP's ledger, $(B_{i+1}, B_{i+2}, \ldots, B_j)$ where $j \geq i+1$. The verifier requires that these ledger entries from BSP are signed with the BSP's private key.
3. Check if the ledger $(B_1, B_2, \ldots, B_i, B_{i+1}, \ldots, B_j)$ is valid (see the definition of a valid ledger above). If so, set $\mathcal{L}_m = (B_1, B_2, \ldots, B_i, B_{i+1}, \ldots, B_j)$. If not, invoke the recovery protocol (§4).
4. Create an approval transaction $(m, j, H(B_j))_{\sigma_m}$. Post the transaction to the BSP using `append`.
5. Compute an approved prefix of the BSP's ledger using the following steps:

5a. Find the "most recent" approval transaction from each member (i.e., an approval transaction that encodes the longest prefix of the ledger) embedded in $\mathcal{L}_m$. Let $\mathcal{A} = [\mathcal{A}_{M_1}, \mathcal{A}_{M_2}, \ldots, \mathcal{A}_{M_k}]$ denote those transactions, where $M_1, M_2, \ldots, M_k$ are the identities of members in the blockchain network.

5b. Compute the approved height $\hat{h} = \min_{\forall a \in \mathcal{A}} a.h$ and the approved ledger is a prefix of $\mathcal{L}_m$ with height $\hat{h}$.

We now prove that this protocol satisfies a desirable consistency property defined below.

Definition 3.1. Two valid ledgers $\mathcal{L}_1$ and $\mathcal{L}_2$ are *consistent* iff, for $0 \leq i \leq min(height(\mathcal{L}_1), height(\mathcal{L}_2))$, block i on $\mathcal{L}_1$ is the same as block i on $\mathcal{L}_2$.

Lemma 3.1. *If two correct members m_1 and m_2 compute approved ledgers $\mathcal{L}_{m_1}$ and $\mathcal{L}_{m_2}$ respectively, then they must be consistent regardless of actions of any other members and the BSP.*

Proof. A correct node always maintains the following invariant: its approved prefix is always a valid ledger. Thus, we just need to prove that approved ledger prefixes computed by a pair of correct nodes are consistent according to the above definition. We use proof by contradiction.

Suppose $\mathcal{L}_{m_1}$ and $\mathcal{L}_{m_2}$ are not consistent. Let i be the first index such that block i on $\mathcal{L}_{m_1}$ is different than block i on $\mathcal{L}_{m_2}$, where block i is in the approved prefix of ledger $\mathcal{L}_{m_1}$ and $\mathcal{L}_{m_2}$. Since blocks in the ledger point to their previous block, the hash of the block at height $j \geq i$ is distinct on each ledger. Because the members are correct, any approval transaction from a member node for height $j \geq i$ must contain a different block hash on the two ledgers.

Now, by the computation of the approved height, we see that there must exist some approval transaction on the full ledger $\mathcal{L}_{m_2}$ of the form $(m_1, j, H(B_j))_{\sigma_{m_1}}$ for $j \geq height(\mathcal{L}_{m_2})$. Since block i is contained in the approved prefix of $\mathcal{L}_{m_2}$, we know that $height(\mathcal{L}_{m_2}) \geq i$. Thus, we see that $j \geq i$.

Since m_2 is correct, we know that $\mathcal{L}_{m_2}$ is a well-formed ledger. So, existence of the approval transaction $(m_1, j, H(B_j))_{\sigma_{m_1}}$ from m_1 means that the block hash at height j on $\mathcal{L}_{m_2}$ is $H(B_j)$. Since m_1 has signed that approval transaction, this means that the block hash at height j on $\mathcal{L}_{m_1}$ must be the same as the hash in the approval transaction (i.e., $H(B_j)$). However, this is not possible since we assumed that block i on both ledgers are distinct and $j \geq i$. Hence the two ledger prefixes must be consistent. □

3.5 Flexible policies on safety and availability

The verifier in the prior subsection ensures a strong safety property to each correct member. The safety property holds even if the BSP is malicious and even if any subset of other members behave maliciously. Consistent with our realistic liveness principle (§2.2), it requires periodic participation from every member in the system. We now discuss how different blockchain deployments can configure policies to trade off safety for availability—without any changes to the core design of the verifier or the BSP. Such a support for flexible policies is a side effect of how VOLT orchestrates self-verifying ledgers: the verifier run by each member lazily validates a ledger created and maintained by the BSP.

Concretely, we describe how the VOLT verifier can relax the availability requirement on members—at the cost of introducing assumptions for ensuring safety properties. At a high level, this refinement adopts quorum intersection ideas from classical fault-tolerance protocols [31, 57]. It guarantees safety and liveness as long as any 2/3rd of members and the BSP participate in the system. Specifically, the verifier on each member runs the same protocol as before except that it computes the approved prefix a bit differently with a *two-round* approval protocol (i.e., step 5 in the protocol discussed in Section 3.4):

a. Find the most recent approval transaction from each member in $\mathcal{L}_m$ as before and sort them based on the length of the ledger prefix encoded in those transactions. Let $\hat{\mathcal{A}} = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k]$ denote the sorted approval transactions, one from each of the k members, embedded in $\mathcal{L}_m$.

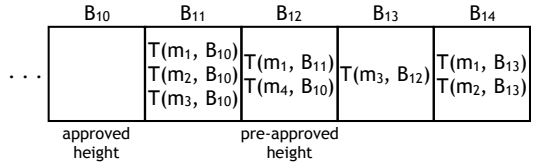

FIGURE 3—State of the BSP's ledger during an example run of the two-round approval protocol (§3.5). We denote B_i as the previous block for B_{i+1} and omit metadata blocks. We use $T(m_i, B_j)$ to denote a approval transaction from a member node m_i for block B_j (§3.4). Suppose there are four members (i.e., $n = 4$), so at most one member can be faulty (i.e., $f=1$). Observe that $2f+1$ members (i.e., m_1, m_2, m_3) have sent approval transactions for block B_{12} or B_{13}, thus all blocks prior to and including B_{12} are `pre-approved`. To find the approved block with the highest height, members run the same algorithm assuming that the ledger ends at the highest pre-approved height. This ensures that at least $2f + 1$ members have witnessed the block at the highest pre-approved height. Thus, all blocks up to and including B_{10} are `approved`.

b. Compute the *pre-approved* height as $\hat{h_p} = \mathcal{A}_{\lceil 2n/3 \rceil}.h$. The pre-approved ledger is the prefix of $\mathcal{L}_m$ with height $\hat{h_p}$.

c. Rerun the above two steps with the pre-approved ledger. Output the resulting ledger prefix as the approved prefix.

Figure 3 depicts an example run. We now prove that this modified protocol preserves consistency and liveness. Our proofs rely on the assumption that at most $\lfloor n/3 \rfloor$ members deviate arbitrarily from their prescribed protocol where n is the total number of members at any instance in the system. Consider $n = 3f + 1$ members where at most f members can deviate from their protocol.

Lemma 3.2. *If two correct members m_1 and m_2 output approved ledgers $\mathcal{L}_{m_1}$ and $\mathcal{L}_{m_2}$ respectively, then the two ledgers are consistent as long as at most f members deviate arbitrarily.*

Proof (sketch). We first prove the following: if a block b is in the pre-approved ledger at height h for one correct member, then no block b' can be in the pre-approved ledger at height h for any other correct member. If a block b is in the pre-approved ledger at height h, then there is at least $2f+1$ members whose approval transactions include block b at height h. Thus, if block b' is in the pre-approved ledger at height h for some other correct member, then there is another set of $2f + 1$ members whose approval transactions include block b at height h. Since there are only $3f + 1$ members, these two quorums intersect in at least $f + 1$ members, of which at least one is correct. Therefore, at least one correct member must have sent an approval transaction for block b and block b' at height h, which is a contradiction. Since the approved ledgers are a prefix of the pre-approved ledgers, our lemma implies that any two approved ledgers are consistent. □

Lemma 3.3. *If the BSP is live, then any transaction submitted to the BSP will eventually appear in the approved prefixes of correct members as long as at most f members deviate arbitrarily.*

Proof (sketch). The liveness property is satisfied since no coalition of f members can stop the protocol from making progress: our two-round approval protocol requires participation of only $2f + 1$ members and the BSP. □

3.6 Network governance: dynamic membership

One of the requirements of permissioned blockchain networks is support for network governance. The simplest example is managing membership in the network—adding and removing members from the network. Other examples include evolving transaction processing state machine, or updating the identity (i.e., public key) of a member. Such reconfiguration is a thorny problem in traditional replication protocols.

Fortunately, it is straightforward to implement network governance tasks with VOLT. Our key insight is that a self-verifying ledger (where members share a consistent view of the ledger) is a powerful abstraction to efficiently implement desirable network governance tasks. Specifically, we can treat the list of members or their identities as state on the BSP's ledger. Thus, we can mutate that state consistently with our transaction processing logic; and share a consistent view of such transactions via VOLT's approval protocol.

Concretely, the ledger's state machine contains a variable for the list of members identified by their public keys. Members can submit transactions to add new members or remove members from that list. Such transactions get sequenced in the BSP's ledger similar to regular transactions. Since members share a consistent view of the BSP's ledger, every pair of correct members share a consistent view of the network membership. To prevent malicious members from misusing such transactions, a simple refinement is to require that such network governance transactions carry digital signatures from a quorum of members (where the quorum size can be configured depending on the application). Finally, we expect such governance transactions to be executed infrequently in practical permissioned blockchains, so network governance tasks create negligible impact on transaction processing throughput.

4 Recovery

The BSP is untrusted by members in a VOLT blockchain network, except for its availability. However, a compromised BSP can deviate from its desired behavior. For example, the BSP can create an ill-formed ledger, which

is easy to detect locally by the verifier on each member. It can also mount a more sophisticated attack that is hard to detect locally (e.g., the BSP can create a *fork* in a ledger that assigns two different blocks to the same height and expose different forks to different members). This by itself does not violate any safety property (from a member's perspective), but members must detect and recover from such behavior. Otherwise, it can turn into a liveness violation.

To address this, VOLT includes a simple mechanism to reinitialize the state of the BSP. Conceptually, the BSP in VOLT is similar to a leader in traditional BFT protocols—although a leader in BFT is an unreliable replica whereas VOLT's BSP is an internally-replicated highly-available service hosted on the cloud. We adapt ideas from the view-change protocol in traditional BFT protocols to recover from failures of the BSP. We now discuss VOLT's recovery protocol for the one-round approval protocol (§3.4). The same mechanism can be adapted to the two-round approval protocol (§3.5) where only 2/3rd of members need to participate in the protocol.

4.1 Initializing the state of the new BSP

When a correct member detects misbehavior of the BSP, it convinces other correct members that the BSP is misbehaving by broadcasting a message to other members. For safety violations, this is done by broadcasting a proof of misbehavior, such as a forked chain signed by the BSP or a malformed ledger signed by the BSP. For liveness violations, however, it is hard to prescribe a solution for every application. For some applications, it may be enough for there to be a programmatic solution like sending a "suspect liveness violation" message to everyone, and then if a threshold number of members sign the same, correct members mark the BSP to be faulty and members move to a new BSP. However, other applications may rely on human operators or other complex business processes to mark the BSP faulty. We abstract this and assume that if one correct member marks the BSP as faulty, then all other correct members will eventually mark the BSP as faulty. Recovering from a failed or faulty BSP involves two steps.

(1) Selecting a new BSP. The new BSP could simply be a different instance of the BSP on the same service provider who hosted the prior BSP that is now marked faulty. Alternatively, the new BSP may be selected ahead of time or via an offline method. We assume that eventually every member agrees on the same new BSP (this only affects liveness, but not safety as we show later in the section).

(2) Initializing the new BSP. The new BSP creates a ledger $\mathcal{L}*$ by using the local states of members, $\mathcal{L}_1, \ldots, \mathcal{L}_n$. The new BSP selects an $\mathcal{L}*$'s identity (§3.2) to be greater than the identity of the ledger that members want to recover. This is analgous to ballot (or view) numbers in traditional BFT protocols [31]. The new BSP simply records the local state of all members on the new ledger.

4.2 The verifier's checks

When the verifier on a correct member marks the BSP as faulty, it stops interacting with the old BSP. Furthermore, when the verifier receives a ledger $\mathcal{L}*$ from the new BSP, it executes the following checks.

1. By reading $\mathcal{L}*$, the verifier locally computes a prefix as follows:

1a. The verifier first drops any $\mathcal{L}_i$ that is sent from a misbehaving member node by checking two invariants: (i) $\mathcal{L}_i$ is a valid ledger and (ii) the height of $\mathcal{L}_i$ is greater than or equal to the height of any approval transaction sent by member m_i that appears on any other ledger. Define $\mathcal{A} = [\mathcal{L}_1, \mathcal{L}_2, \cdots \mathcal{L}_n]$ to be the list of the remaining ledgers.

1b. Let the longest consistent prefix of a set of ledgers $\mathcal{C}$ be the ledger, $lcp(\mathcal{C})$, that is consistent with every ledger in $\mathcal{C}$ and is of maximum height.

1c. The verifier then computes $\mathcal{L}' = lcp(\mathcal{A})$.

2. The verifier then sends an approval transaction that certifies $\mathcal{L}'$ on the new BSP's ledger $\mathcal{L}*$. Once the verifier sees approval transactions from all other members, then it continues to use $\mathcal{L}*$.

We now show that, for any correct member m, the above protocol results in the new BSP containing a ledger $\mathcal{L}'$ that has m's approved prefix as a prefix of $\mathcal{L}'$. This protocol is deterministic given the same $\mathcal{L}*$, so all correct verifiers will compute the same $\mathcal{L}'$. First, note that the approved prefix of each member should be a prefix of *all* local ledgers included in $\mathcal{A}$. Any ledger not satisfying property would have been dropped due to Lemma 3.1.[2]

Finally, to prove correctness of our protocol, we start with the following lemma:

Lemma 4.1. *For a correct member m_1, let $\mathcal{A}$ be the list of the ledgers computed in step 1a of the recovery protocol, then m_1's approved prefix $\mathcal{L}_{m_1}$ is a prefix of any ledger $\mathcal{L}'_{m_2} \in \mathcal{A}$ from a member m_2.*

Proof. Because ledger $\mathcal{L}_{m_1}$ is an approved prefix, due to the invariants in step 1a, there exists some approval transaction from m_2 of the form $(m_2, j, H(B_j))_{\sigma_{m_2}}$, where $j \geq height(\mathcal{L}_{m_1})$ and $H(B_j)$ is the hash of the block at height j for both $\mathcal{L}_{m_1}$ and m_2's ledger

[2]Note that, if a member m is compromised with its local ledger corrupted or lost, VOLT does not guarantee that the previously approved prefix on m remains a prefix of the new ledger after the recovery. But the compromise of m does not affect others; the property holds for any correct member.

$\mathcal{L}'_{m_2}$. Furthermore, since $\mathcal{L}_{m_1}$ is an approved prefix, we know that it must be a prefix of a ledger whose block at height j has a block hash $H(B_j)$. Call this ledger $\mathcal{L}^{(j)}$. Additionally, this ledger is unique as $H(\cdot)$ is collision-resistant.

However, since $\mathcal{L}'_{m_2}$ is valid and contains the above approval transaction, it must have a block at height j whose block hash is $H(B_j)$. Therefore, we see that $\mathcal{L}^{(j)}$ is a prefix of $\mathcal{L}'_{m_2}$. Thus, we see that $\mathcal{L}_{m_1}$ is a prefix of $\mathcal{L}'_{m_2}$, as desired. □

Lemma 4.2. *Suppose a correct member m_1's verifier computes $\mathcal{L}'$ during recovery and m_1 has an approved prefix $\mathcal{L}_{m_1}$ before recovery. Then, $\mathcal{L}_{m_1}$ is a prefix of $\mathcal{L}'$.*

Proof. By Lemma 4.1, we see that $\mathcal{L}_{m_1}$ is a prefix of any ledger in $\mathcal{A}$ computed in step 1a. Since a correct member m_1 computes $\mathcal{L}'$ as the longest common prefix of all ledgers in $\mathcal{A}$, we see that $\mathcal{L}'$ must contain $\mathcal{L}_{m_1}$ as a prefix. □

Note that our recovery protocol's correctness (due to Lemma 4.2) only relies on Lemma 4.1. Thus, we can adapt our recovery protocol slightly to work for any safety policy that satisfies Lemma 4.1, which includes our two-round approval protocol (§3.5).

5 Implementation

We implement VOLT in 6,000 lines of Rust. The verifier implementation—that a member must trust or can implement itself—is only 1,500 lines of code, and the rest is the BSP implementation, which is about 4,500 lines of code. Each member persists its local state using LevelDB.[3] We implement the BSP as a Web service built with the Rocket framework.[4] Each API of the BSP (§3.3) is handled by a separate microservice. All the BSP's microservices operate on state maintained in a fault-tolerant data store (our prototype uses Azure CosmosDB).[5]

Concretely, the microservice that implements the `append` API creates the ledger data structure as an entry in a table on CosmosDB; and the microservice that implements `read` API serves its requests by executing read operations on the same table. Each block of a ledger created with the BSP is keyed by the tuple (lid, height) where lid is the identity of the ledger and height is the height of the block stored at this location. To reduce roundtrips to the database, `read` API issues range queries to retrieve a batch of blocks at once from the table in CosmosDB. Finally, we use `SHA-256` for a collision-resistant hash function, and `Ed25519` for digital signatures. For these cryptographic operations, we use `rust-crypto` and `ring` crates.[6] We have not implemented extensions for recovery (§4), but it does not impact the normal-case performance.

Application: Distributed payment network. To demonstrate the costs and benefits of VOLT, we build a realistic application: a *distributed payment network* for a group of financial institutions, such as banks, to process and settle payment transactions on behalf of their customers. At a high level, a payment network consists of a set of accounts. Each account has a name and a balance. A transaction in such a network consists of a transfer of value from one account to another. In our case, each account is named by a public key in a digital signature scheme (i.e., an `Ed25519` public key), which makes it easy to authenticate transactions. We now discuss how we implement such a payment network with VOLT.

Network constitution. Naturally, financial institutions that wish to transact with one another form the set of members in VOLT's protocol. The member set can grow or shrink over time using VOLT's governance (§3.6). These members maintain a self-verifying ledger (containing all transactions including approval transactions processed by the network) such that all members share a consistent view of the shared ledger. However, we need a way to bootstrap the system with a set of accounts and balances.

Bootstrapping accounts. To bootstrap accounts, our implementation employs a group of *credit providers* who have the ability to inject new currency into the network, which is similar in spirit to many enterprise payment networks [41, 64]. A credit provider is any bank in the member set who can issue coins on the ledger to any account.[7] Note that our payment network application generalizes easily to any digitally-issued assets [84].

Processing transactions. The internal state of the ledger at any point includes: (1) the list of members in the blockchain network, (2) the list of credit providers, and (3) accounts. All these entities are identified by their public keys. The blockchain network processes two types of transactions. A credit provider can create an `issue` transaction, which supplies new coins to an account. The account owner can then spend those coins by creating a `spend` transaction, which transfers coins from one account to another. To prevent an adversary from replaying transactions, we employ a standard technique: transactions include per-account sequence numbers, and they must be monotonically increasing.

[3] https://github.com/google/leveldb

[4] https://github.com/SergioBenitez/Rocket

[5] https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

[6] https://crates.io/crates/ring

[7] The idea is that such a credit provider abides by a legally-enforceable contract that the coins it creates can be converted back to a fiat currency at anytime by transferring those issued coins back to the credit provider's account in the payment network. Another example is a central bank of a country who issues coins on this payment network to a particular financial institution, perhaps after deducting fiat money in that institution's account at the central bank. The latter example is similar to a use case considered in prior works [26, 41].

6 Experimental evaluation

Our goal in this section is to demonstrate that the verifier in VOLT incurs modest resource costs to participate in a blockchain network and that VOLT can achieve performance sufficient for many of our target applications.

Our experimental evaluation answers the following questions. First, what are the resource costs imposed on members in a VOLT blockchain network? Second, what is the end-to-end performance of VOLT in terms of throughput and latency? And how does it compare to other blockchain systems? We answer these questions in the context of an enterprise application: a distributed payment network that we built atop VOLT (§5). Note the same application can be used for other forms of consortium applications for asset tracking and value exchange. Figure 4 summarizes our results.

6.1 Setup, metrics, and microbenchmarks

We deploy the BSP on a cluster of five machines on Microsoft Azure cloud, each with the following configuration: Ubuntu Linux 16.04 running on Standard_F16s_v2 VMs (i.e., an Intel Xeon E5-2673 v3 processor, 16 cores, 32 GB RAM, and 128 GB local SSD storage). In our experiments, we reserve 250 GB for storage and 500,000 request units per second on Azure's CosmosDB.[8] We use a similar cluster to run members, but we pack multiple members on the same machine to allow us to experiment with larger network sizes without deploying excessive hardware. Note that members and the BSP machines run in the same geographic region, so our results do not consider the effect of a geo-distributed deployment. We expect the peak throughput to remain the same, but discuss the effect on end-to-end latency due to a wide-area network below. We run all experiments at least five times and report averages. Standard deviations are less than 10%, so we do not report them.

Our principal evaluation metrics include: resource costs (CPU, network, etc.) imposed on members to participate in a blockchain network; we are also interested in throughput (in transactions/second) and latency (in seconds). In our experiments, we use a dedicated machine to submit transactions to the BSP, and members execute the VOLT approval protocol to locally compute their approval prefixes.

Microbenchmarks. To put our end-to-end performance results in context, we run a set of microbenchmarks to measure the cost of various cryptographic primitives that VOLT uses. Figure 5 depicts our results. As expected, the cryptographic operations that VOLT relies on are lightweight, thus we expect VOLT's approval protocol to add negligible overheads to end-to-end performance.

[8] https://docs.microsoft.com/en-us/azure/cosmos-db/request-units

6.2 Resource costs of VOLT's verifier

The principal cost incurred by a member in VOLT is participating in the approval protocol: members must validate the BSP's ledger, send approval transactions, and compute approval prefixes by using approval transactions of other members. To understand these costs, we run a series of experiments in which the verifier interacts with the BSP and executes the approval protocol (§3.4).

We measure resource costs in terms of CPU, storage, and network incurred by the verifier in these experiments. Figure 6 summarizes our results. As we expect from our microbenchmarks, the verifier incurs little overheads to participate in VOLT's approval protocol.

6.3 End-to-end performance of VOLT

To understand VOLT's performance, we deploy the BSP and members on our cluster (§6.1), configured to run our payment network (§5). For comparison, we consider the following baselines: (1) a private Ethereum network using Azure Blockchain as a Service [6]; (2) Chain [2], a system closest to VOLT except that it does not provide the same level of security and recovery guarantees as VOLT; (3) VOLT–, A version of VOLT that does not include its approval protocol (hence trusts the BSP for security).

To put our results in perspective, we also compare VOLT's performance to recent BFT-based permissioned blockchains [17, 47]. Asymptotically, VOLT requires the same number of messages to reach consensus as prior BFT protocols, although it achieves a natural batching effect due to its approval transactions (§3.4). Thus, our goal here is not to demonstrate that VOLT performs better or worse than BFT-based baselines, but rather assess whether VOLT's performance is in the same ballpark—while achieving its design goals (§2.2), which are relevant for practical deployments of permissioned blockchains. For comparison, we rely on results reported in their papers (on comparable hardware); in the future, we plan to run those systems on the same hardware platform.

For other baselines, we implement the payment network application on Ethereum using Solidity [10], and deploy a private Ethereum network using Azure Blockchain as a Service [6]. The other baseline, Chain [2], natively supports such an application. We deploy both of them on the same cluster and measure their transaction processing throughput. By default, Chain generates a block with ≈ 1 second latency, so we configure VOLT so members send their approval transactions every second. For VOLT and VOLT–, we batch 1,000 transactions in a block and use our one-round approval protocol (§3.4).

Throughput. To measure the throughput of processing transactions, we use a client process that submits transactions to a given blockchain network: the client process creates multiple threads where each thread submits trans-

The verifier in a VOLT blockchain network incurs small resource costs	§6.2
VOLT scales to realistic workloads: 20,000 transactions/second and sub-second block latencies, with 50 members	§6.3
The approval protocol adds negligible overheads on the transaction processing throughput	§6.3

FIGURE 4—Summary of evaluation results

Size of signature material	
key pair	85 bytes
public key	32 bytes
signature	64 bytes
CPU cost of signature operations	
generate a key pair	85.7μs
sign a message	43.3μs
verify a signature	0.14 ms
Ledger parameters	
transaction size	$\approx$ 188 bytes
block size (1000 transactions)	$\approx$ 220 KB

FIGURE 5—Microbenchmarks for VOLT's underlying cryptographic primitives and ledger building blocks.

CPU costs	
generate approval transaction	<100μs
verify approval transaction	0.15 ms
validate ledger (100 blocks, batch=1000)	0.40 s
validate ledger (200 blocks, batch=1000)	0.85 s
validate ledger (400 blocks, batch=1000)	1.22 s
Storage costs	
block (batch=1000)	$\approx$ 220 KB
Network costs	
approval transaction	256 bytes

FIGURE 6—Resource costs of members in a VOLT.

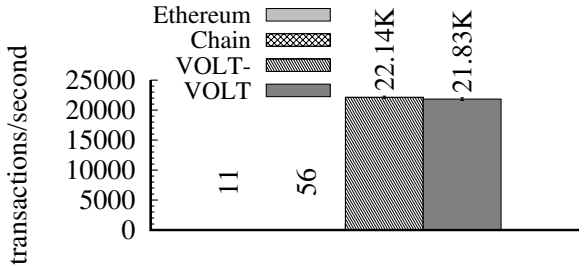

FIGURE 7—Transaction processing throughput of VOLT and its baselines (§6.1) for our payment network application (§5). VOLT does not rely on mining, so its throughput is orders of magnitude higher than Ethereum. We find Chain to be unoptimized for throughput, but we include it here for rough comparison. Both VOLT and VOLT- achieve excellent throughput due to their powerful BSP. See text for additional details of the experiment.

actions in a closed loop. We then measure the number of transactions processed per second by the blockchain network and latency for each transaction from the perspective of the client process. To determine peak throughput, we progressively increase the number of threads in the client process. Figure 7 depicts our results.

As expected, VOLT's throughput is orders of magnitude better than a private Ethereum network since Ethereum uses expensive proof-of-work, which is unnecessary for permissioned blockchain networks. Perhaps surprisingly, Chain's throughput is not significantly better than Ethereum (though it does not use proof-of-work to construct a ledger): we find that its codebase is not optimized for high throughput, possibly because we use the developer edition.[9] Furthermore, the results we describe for Chain is optimistic for Chain: we depict results with a single member (whereas for VOLT and VOLT–, we use 10 members in the network) as our measurements show much worse performance for Chain when we vary the number of members from 1 to 8. This is indeed not fundamental as we demonstrate with VOLT–, which is similar to VOLT except that it does not use the approval protocol (so VOLT– is closer to Chain except for the microservices architecture with a highly-available cloud storage service).

The key takeaway is that VOLT and VOLT– both achieve similar throughput i.e., VOLT's approval protocol adds negligible overheads. The fundamental reason is that members send their approval transactions every second i.e., every $\approx$20 blocks. Since there are only 10 approval transactions (i.e., one from each member) per second submitted to the BSP (vs. 0 in VOLT–), the overheads on end-to-end throughput are negligible: the BSP processes $\approx$20K regular transactions/second and approval transactions constitute less than 0.1% of regular transactions (for this configuration we experiment with).

Finally, compared to BFT-based systems, VOLT achieves competitive performance: VOLT has higher throughput (by a small factors), but it also incurs higher latency compared to those BFT-based systems (not depicted). For example, Hyperledger Fabric [17] achieves about 4,000 transactions/second with several hundred milliseconds of latency, and SBFT [47] achieves about 10,000 transactions/second with 100 ms of latency,
whereas VOLT's latency is about 2 seconds with one-round approval protocol. VOLT can also lower its latency to hundreds of milliseconds since the verifier is fast enough to validate a ledger and submit its approval transactions (Figure 6)—while not significantly overloading the BSP with their approval transactions.

[9] https://github.com/chain/chain

Larger network sizes. Many permissioned blockchain applications require the system to scale to tens of members. We now assess whether VOLT can support such network sizes. we run many experiments with number of members varied in increments of 10. We keep the same parameters for the approval protocol as before.

Figure 8 depicts our results. We find that VOLT with its experimental infrastructure backed by a powerful cloud storage can sustain high throughput (>20K transactions/second) with tens of members in a network. However, our implementation does not scale indefinitely, and the BSP appears to the bottleneck in our current implementation: for larger network sizes with 60 or 70 members, the BSP's throughput drops to several thousand transactions per second. We hypothesize that VOLT can sustain its peak throughput if we introduce a sophisticated caching layer at the BSP between its microservices and CosmosDB and shard the data stored on CosmosDB so that it can support collectively support larger request rates. Experimenting with these refinements is work in progress.

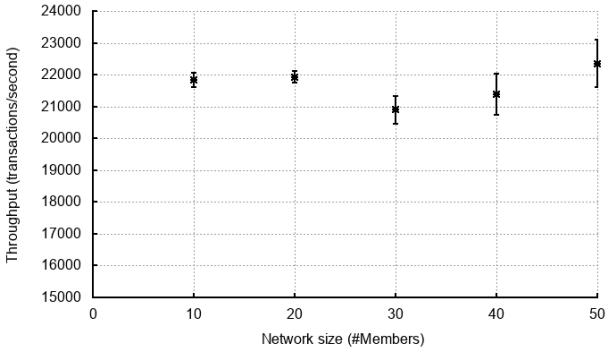

FIGURE 8—Throughput of VOLT with increasing network sizes. Our experimental results demonstrate that VOLT can sustain realistic throughput for modest-sized networks (see text for details of the experiment).

7 Related work

VOLT relates to prior work in distributed systems and cryptocurrencies. We now discuss their relationship with VOLT. (Bonneau et al. [24] and Cachin et al. [30] provide an excellent survey of many of these works.)

BFT and permissioned blockchains. Byzantine fault-tolerant (BFT) replication protocols enable building distributed systems that can tolerate the compromise of a fraction of machines in the system. Following PBFT [31], there is a long line of work to improve performance [49, 55, 56, 63, 75], robustness [32, 37, 60], confidentiality [85, 86], and fault models [15] of BFT protocols. These protocols and systems were developed for replicating a service for fault-tolerance. However, one could apply them to build a permissioned blockchain network [5, 8, 26, 78]. There are also recent proposals [17, 47, 64, 67, 74] to improve prior BFT protocols in the context of permissioned blockchain networks. VOLT departs from these works by embracing a different set of design principles that are more suitable in the context of a permissioned blockchain (§2.2), which leads to a significantly different design (§3)—while being competitive on performance with the state-of-the-art BFT systems (§6).

Public cryptocurrencies. Cryptocurrencies such as Bitcoin [68] and Ethereum [83] create a trustworthy ledger abstraction using a combination of resource-intensive puzzles and incentives [45, 69, 73]. However, enterprises cannot directly use them to build our target business applications (§2.1). If we look at them from a traditional systems perspective, they scale poorly [27, 40]. There are proposals to address the performance limitations of these systems [14, 42, 46, 53, 66, 75]. However, they all operate in a permissionless model (so they must defend against Sybil attacks). Thus, they inherently require incentives (provided in the form of a cryptocurrency by the protocol) in addition to proof-of-work [14, 42, 53] (or alternate approaches such as proof-of-stake [20, 21, 52, 88]). Thus, the guarantees of these systems are rooted in consuming resources to prevent Sybil attacks. Finally, these cryptocurrencies are monetarily expensive. For example, it costs over millions of USD per GB of storage on Ethereum.[10] Compared to these works, VOLT targets enterprise applications where entities have well-known identities sidestepping the need for expensive permissionless consensus protocols.

Untrusted storage. As in VOLT and other blockchain systems, prior untrusted storage systems [29, 43, 59, 61, 62, 76, 79] use hash chains to prevent untrusted participants in the system from tampering with operation histories. For example, they use hash chains to implement various storage interfaces such as file systems [59], key-value stores [43, 61, 76, 79]. All these systems target low latency and high availability (e.g., they strive to avoid client-to-client communication), so they only provide weaker consistency guarantees such as variants of fork consistency [65]. Since VOLT targets mission-critical applications with financial implications (§2), VOLT prefers stronger safety and higher throughput over low latency. While these systems implement various storage interfaces, VOLT is more general since it exposes the abstraction of a ledger, which it uses to store a general-purpose state machine and transactions. Cachin proposes a similar abstraction atop a ledger, but retains the weaker fork-linearizability [28].

Trusted hardware. A2M [34] and Trinc [58] explore small trusted primitives (at each server) for building trust-

[10] It costs 20,000 gas/256 bits [83]; the gas price is $\approx 10^{-8}$ ETH [3, 4].

worthy distributed systems. Relatedly, there are other proposals for permissioned blockchains by relying on trusted hardware such as Intel SGX enclaves (e.g., Coco [7] and Sawtooth Lake [9]). REM [90] leverages enclaves in conjunction with proof-of-useful-work to operate in an open, permissionless model. VOLT does not require a trusted hardware. We leave it to future work to explore extensions to our current design to offload more work to the BSP.

Other substrates. Several works [11, 89] propose using secure hardware for trusted data injection into blockchain systems. VOLT can naturally incorporate these ideas. Virtual chain [71] proposes a layer atop blockchains (like VOLT's protocol), but only provides fork* consistency. Many works [16, 23, 35, 82] propose techniques to reduce (in a security proof sense) equivocation by entities that generate hash chains to the security of a public blockchain. VOLT can leverage them to limit BSP's misbehavior when members are offline. Finally, CoSi [80] enables a group of entities to create succinct signatures on a given statement. It can be used in VOLT to compress approval transactions. However, it requires increased coordination between member nodes and the BSP to generate a signature. Furthermore, the BSP must know how many signatures it has to collect and what policies member nodes follow to commit transactions. All of these are oblivious to the BSP in our current design. Thus the use of CoSi increases the complexity of the BSP.

8 Discussion

Verifier acceleration and additional outsourcing. In VOLT, members re-execute transactions on the BSP's ledger to verify their validity. As a simple extension, the verifiers can save CPU resources by batch verifying signatures. In the future, we wish to employ recent progress on verifying concurrent applications and services [81] to make the BSP process transactions concurrently and to further accelerate the verifier. Relatedly, we wish to augment the BSP to employ cryptographic verifiable computation protocols [19, 25, 72] to produce a succinct proof that the ledger contains only valid transactions. Members can simply verify such succinct proofs before sending their approval transactions.

Confidentiality. VOLT does not aim to provide *confidentiality* properties, for example, hiding data and/or metadata about transactions from entities not involved in a transaction. There are broadly two existing approaches to confidentiality: (1) using trusted execution environments such as Intel SGX to process transactions [7, 9]; and (2) applying cryptographic proof systems [33, 54, 70]. Both approaches compose well with VOLT; we leave it to future work to integrate them with VOLT.

Fairness. A key concern in blockchain systems is providing a form of *fairness* to transactions. Broadly speaking, fairness typically means that transactions will be confirmed in a timely manner. Permissionless blockchains use transaction fees (i.e., economic incentives) to achieve fairness. An potential approach is to encrypt the transactions and hide useful metadata through the use of hardware enclaves and/or other cryptographic methods [18, 54]. However, there is no perfect solution: while we can try to hide the transaction data and some metadata, malicious entities can always filter transactions based on any metatdata that can be inferred via collusion with other malicious members, a point made by Herlihy and Moir [50] who also propose a scheme to improve fairness in blockchains via accountability [48, 87]. VOLT can be extended to incorporate these ideas depending on the application.

9 Summary

This paper studies the problem of designing a permissioned blockchain for an emerging class of mission-critical cross-organizational business applications among a group of mutually distrusting members. Our goal was to build a system that is (1) member-centric (ii) achieves practical performance, (iii) ensures strong safety guarantees to each correct member in the system. These goals led us to depart from prior work on BFT (§2.2,§7) and embrace the concept of a self-verifying ledger orchestrated with a simple verifier on each member and an untrusted blockchain service provider (BSP). Our system, VOLT, implements this concept and achieves all of the above goals; it is also simple to implement and deploy in practice.

References

[1] Azure Cosmos DB. https://docs.microsoft.com/en-us/azure/cosmos-db/introduction.

[2] Chain Core. https://chain.com/technology/.

[3] Ether historical prices (USD). https://etherscan.io/chart/etherprice.

[4] Ethereum average gasprice chart. https://etherscan.io/chart/gasprice.

[5] Hyperledger – Blockchain for Business. https://www.hyperledger.org/.

[6] Microsoft Azure Blockchain as a Service (BaaS). https://azure.microsoft.com/en-us/solutions/blockchain/.

[7] Microsoft Coco framework. https://github.com/Azure/coco-framework/blob/master/docs/Coco%20Framework%20whitepaper.pdf.

[8] Quorum. https://github.com/jpmorganchase/quorum.

[9] Sawtooth Lake. https://intelledger.github.io/.

[10] Solidity. https://solidity.readthedocs.io/en/develop/.

[11] The Cryptlet Fabric. https://github.com/Azure/azure-blockchain-

projects/blob/master/bletchley/CryptletsDeepDive.md.

[12] TradeWind markets. http://www.tradewindmarkets.com/technology-solutions/.

[13] Trust, confidence and verifiable data audit. https://deepmind.com/blog/trust-confidence-verifiable-data-audit/.

[14] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless Byzantine consensus. *CoRR*, abs/1612.02916, 2016.

[15] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 45–58, 2005.

[16] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: a global naming and storage system secured by blockchains. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 181–194, 2016.

[17] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2018.

[18] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[19] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 90–108, Aug. 2013.

[20] I. Bentov, A. Gabizon, and A. Mizrahi. Cryptocurrencies without proof of work. In *Proceedings of the International Financial Cryptography and Data Security Conference*, pages 142–157, 2016.

[21] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016.

[22] A. Bessani, J. a. Sousa, and E. E. P. Alchieri. State machine replication for the masses with BFT-SMART. In *International Conference on Dependable Systems and Networks (DSN)*, pages 355–362, 2014.

[23] J. Bonneau. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *Proceedings of the International Financial Cryptography and Data Security Conference*, pages 95–105, 2016.

[24] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 104–121, 2015.

[25] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 341–357, 2013.

[26] E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. Master's thesis, The University of Guelph, 2016.

[27] V. Buterin. Notes on scalable blockchain protocols. https://github.com/vbuterin/scalability_paper/blob/master/scalability.pdf, 2015.

[28] C. Cachin. Integrity and consistency for untrusted services. In *SOFSEM 2011: Theory and Practice of Computer Science*, pages 1–14, 2011.

[29] C. Cachin, I. Keidar, and A. Shraer. Fail-aware untrusted storage. *SIAM Journal on Computing*, 40(2):493–533, Apr. 2011.

[30] C. Cachin and M. Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017.

[31] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, Nov. 2002.

[32] M. Castro, R. Rodrigues, and B. Liskov. Base: using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, pages 236–269, 2003.

[33] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi. Solidus: Confidential distributed ledger transactions via PVORM. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[34] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 189–204, 2007.

[35] J. Clark and A. Essex. Commitcoin: Carbon dating commitments with bitcoin. In *Proceedings of the International Financial Cryptography and Data Security Conference*, pages 390–398, 2012.

[36] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. UpRight cluster services. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 277–290, 2009.

[37] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 153–168, 2009.

[38] Cognizant. Blockchain's smart contracts: Driving the next wave of innovation across manufacturing value chains. https://www.cognizant.com/whitepapers/blockchains-smart-contracts-driving-the-next-wave-of-innovation-across-manufacturing-value-chains-codex2113.pdf, June 2016.

[39] D. Creer, R. Crook, M. Hornsby, N. G. Avalis, M. Simpson, N. Weisfeld, B. Wyeth, and I. Zielinski. Proving Ethereum for the clearing use case. Technical report, Royal Bank of Scotland, 2016.

[40] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels,

A. M. Ahmed Kosba, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains (a position paper). In *Proceedings of the Workshop on Bitcoin Research (BITCOIN)*, 2016.

[41] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[42] I. Eyal, A. E. Gencer, E. Gün Sirer, and R. van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[43] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. SPORC: Group collaboration using untrusted cloud resources. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 337–350, 2010.

[44] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Symposium on Principles of Database Systems*, pages 1–7, 1983.

[45] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2015.

[46] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[47] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu. Sbft: a scalable decentralized trust infrastructure for blockchains. arxiv:1804/01626v1, Apr. 2018. `https://arxiv.org/abs/1804.01626`.

[48] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: practical accountability for distributed systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 175–188, 2007.

[49] J. Hendricks, S. Sinnamohideen, G. R. Ganger, and M. K. Reiter. Zzyzx: Scalable fault tolerance through byzantine locking. In *International Conference on Dependable Systems and Networks (DSN)*, pages 363–372, 2010.

[50] M. Herlihy and M. Moir. Enhancing accountability and trust in distributed ledgers. *CoRR*, abs/1606.07490, 2016.

[51] J.P. Morgan and Oliver Wyman. Unlocking economic advantage with blockchain: A guide for asset managers. `http://www.oliverwyman.com/our-expertise/insights/2016/jul/unlocking-economic-advantage-with-blockchain.html`, July 2016.

[52] S. King and S. Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake. `http://peerco.in/assets/paper/peercoin-paper.pdf`, 2012.

[53] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *Proceedings of the USENIX Security Symposium*, 2016.

[54] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2016.

[55] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 45–58, 2007.

[56] R. Kotla and M. Dahlin. High throughput Byzantine fault tolerance. In *International Conference on Dependable Systems and Networks (DSN)*, pages 575–584, 2004.

[57] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[58] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–14, 2009.

[59] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[60] J. Li and D. Maziéres. Beyond one-third faulty replicas in Byzantine fault tolerant systems. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[61] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 307–322, 2010.

[62] P. Maniatis. *Historic Integrity in Distributed Systems*. PhD thesis, Stanford, CA, USA, 2003. AAI3104277.

[63] J.-P. Martin, L. Alvisi, and M. Dahlin. Small Byzantine quorum systems. In *International Conference on Dependable Systems and Networks (DSN)*, pages 374–383, 2002.

[64] D. Mazières. The Stellar consensus protocol: A federated model for Internet-level consensus. Technical report, Stellar Devlopment Foundation, Apr. 2015.

[65] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *Proceedings of the ACM Conference on Principles of Distributed Computing (PODC)*, pages 108–117, 2002.

[66] S. Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[67] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The Honey Badger of BFT Protocols. Cryptology ePrint Archive, Report 2016/199, 2016.

[68] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Oct. 2008.

[69] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[70] N. Narula, W. Vasquez, and M. Virza. zkLedger: Privacy-preserving auditing for distributed ledgers. In

Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2018.

[71] J. Nelson, M. Ali, R. Shea, and M. J. Freedman. Extending existing blockchains with virtualchain. In *Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL)*, July 2016.

[72] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 238–252, May 2013.

[73] R. Pass, L. Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016.

[74] R. Pass and E. Shi. The sleepy model of consensus. Cryptology ePrint Archive, Report 2016/918, 2016.

[75] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. Cryptology ePrint Archive, Report 2017/913, Sept. 2017. `https://eprint.iacr.org/2017/913.pdf`.

[76] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling security in cloud storage SLAs with CloudProof. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2011.

[77] J. Schneider, A. Blostein, B. Lee, S. Kent, I. Groer, and E. Beardsley. Blockchain: Putting theory to practice. In *Goldman Sachs' profiles of innovation*, May 2016.

[78] D. Schwartz, N. Youngs, and A. Britto. The Ripple protocol consensus algorithm. Technical report, Ripple Labs Inc, 2014.

[79] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket. Venus: Verification for untrusted cloud storage. In *Proceedings of the Cloud Computing Security Workshop (CCSW)*, pages 19–30, 2010.

[80] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2016.

[81] C. Tan, L. Yu, J. B. Leners, and M. Walfish. The efficient server audit problem, deduplicated re-execution, and the Web. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 546–564, 2017.

[82] A. Tomescu and S. Devadas. Catena: Efficient non-equivocation via Bitcoin. Cryptology ePrint Archive, Report 2016/1062, 2016.

[83] G. Wood. Ethereum: A secure decentralised generalised transaction ledger homestead revision. `http://gavwood.com/paper.pdf`.

[84] World Economic Forum. The future of financial infrastructure: An ambitious look at how blockchain can reshape financial services. `https://www.weforum.org/reports/the-future-of-financial-infrastructure-an-ambitious-look-at-how-blockchain-can-reshape-financial-services`, Aug. 2016.

[85] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Byzantine fault-tolerant confidentiality. In *Proceedings of the International Workshop on Future Directions in Distributed Computing*, pages 12–15, 2002.

[86] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 253–267, 2003.

[87] A. R. Yumerefendi and J. S. Chase. Strong accountability for network storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[88] V. Zamfir. Introducing Casper "The Friendly Ghost". `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost`, 2015.

[89] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town Crier: an authenticated data feed for smart contracts. Cryptology ePrint Archive, Report 2016/168, 2016.

[90] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse. REM: resource-efficient mining for blockchains. In *Proceedings of the USENIX Security Symposium*, 2017.