The Stellar Consensus Protocol:
A Federated Model for Internet-level Consensus

David Mazières
Stellar Development Foundation

July 14, 2015 DRAFT

Abstract

This paper introduces federated Byzantine agreement (FBA), a new approach to consensus. FBA's robustness stems from quorum slices, individual trust decisions made by each node that together determine system-level quorums. Slices bind the system together much the way individual networks' peering and transit decisions now unify the Internet.

We also present the Stellar Consensus Protocol (SCP), a construction for FBA. Like all Byzantine agreement protocols, SCP makes no assumptions about the rational behavior of attackers. Unlike prior Byzantine agreement models, which presuppose a unanimously accepted membership list, SCP enjoys open membership that promotes organic network growth. Compared to decentralized proof of-work and proof-of-stake schemes, SCP has modest computing and financial requirements, lowering the barrier to entry and potentially opening up financial systems to new participants.

1 Introduction

Financial infrastructure is currently a mess of closed systems. Gaps between these systems mean that transaction costs are high [41] and money moves slowly across political and geographic boundaries [4, 12]. This friction has curtailed the growth of financial services, leaving billions of people underserved financially [16].

To solve these problems, we need financial infrastructure that supports the kind of organic growth and innovation we've seen from the Internet, yet still ensures the integrity of financial transactions. Historically, we have relied on high barriers to entry to ensure integrity. We trust established financial institutions and do our best to regulate them. But this exclusivity conflicts with the goal of organic growth. Growth demands new, innovative participants, who may possess only modest financial and computing resources.

We need a worldwide financial network open to anyone, so that new organizations can join and extend financial access to unserved communities. The challenge for such a network is ensuring participants record transactions correctly. With a low barrier to entry, users won't trust providers to police themselves. With worldwide reach, providers won't all trust a single entity to operate the network. A compelling alternative is a decentralized system in which participants together ensure integrity by agreeing on the validity of one another's transactions. Such agreement hinges on a mechanism for worldwide consensus.

This paper presents federated Byzantine agreement (FBA), a model suitable for worldwide consensus. In FBA, each participant knows of others it considers important. It waits for the vast majority of those others to agree on any transaction before considering the transaction settled. In turn, those important participants do not agree to the transaction until the participants *they* consider important agree as well, and so on. Eventually, enough of the network accepts a transaction that it cannot be undone. Only then do any participants consider the transaction settled. FBA's consensus can ensure the integrity of a financial network. Its decentralized control can spur organic growth.

This paper further presents the Stellar consensus protocol (SCP), a construction for FBA. We prove that SCP's safety is optimal for an asynchronous protocol, in that it guarantees agreement under any node-failure scenario that admits such a guarantee. We also show that SCP is free from *blocked* states—in which consensus is no longer possible—unless participant failures make it impossible to satisfy trust dependencies. SCP is the first provably safe consensus mechanism to enjoy four key properties simultaneously:

- **Decentralized control.** Anyone is able to participate and no central authority dictates whose approval is required for consensus.

- **Low latency.** In practice, nodes can reach consensus at timescales humans expect for web or payment transactions—i.e., a few seconds at most.

- **Flexible trust.** Users have the freedom to trust any combination of parties they see fit. For example, a small non-profit may play a key role in keeping much larger institutions honest.

- **Asymptotic security.** Safety rests on digital signatures and hash families whose parameters can realistically be tuned to protect against adversaries with unimaginably vast computing power.

SCP has applications beyond financial markets for ensuring organizations perform important functions honestly. A good example is certificate authorities (CAs), who literally hold the keys to the web. Experience shows that CAs sign incorrect certificates that get used in the wild [1, 30]. Several proposals address this problem through certificate transparency [23, 31, 5, 33]. SCP can strengthen certificate transparency: not only would anyone who so pleases be able to audit CA behavior, auditors would agree on the set of all issued certificates, making it difficult to backpedal or override previously issued certificates without detection.

The next section discusses previous approaches to consensus. Section 3 defines federated Byzantine agreement (FBA) and lays out the notions of safety and liveness such a system hopes to achieve. Section 4 discusses optimal failure resilience in an FBA system, thereby establishing the security goals for SCP. Section 5 develops federated voting, a key building block of the SCP protocol. Section 6 presents SCP itself, proving safety and absence of blocked states for nodes with satisfiable dependencies. Section 7 discusses limitations of SCP. Finally, Section 8 summarizes results. For readers less familiar with mathematical notation, Appendix A defines some symbols used throughout the paper.

2 Related work

Figure 1 summarizes how SCP differs from previous consensus mechanisms. The most famous decentralized consensus mechanism is the proof-of-work scheme advanced by Bitcoin [34]. Bitcoin takes a two-pronged approach to consensus. First, it provides incentives for rational actors to behave well. Second, it settles transactions through a proof-of-work [19] algorithm designed to protect against ill-behaved actors who do not possess the majority of the system's computing power. Bitcoin has overwhelmingly demonstrated the appeal of decentralized consensus [7].

Proof of work has limitations, however. First, it wastes resources: by one estimate from 2014, Bitcoin might consume as much electric power as the entire country of Ireland [36]. Second, secure transaction settlement suffers from expected latencies in the minutes or tens of minutes [22]. Finally, in contrast to traditional cryptographic protocols, proof of work offers no asymptotic security. Given non-rational attackers—or ones with extrinsic incentives to sabotage

mechanism	decentralized control	low latency	flexible trust	asymptotic security
proof of work [34]	✓			
proof of stake [24]	✓	maybe		maybe
Byzantine agreement [11]		✓	✓	✓
Tendermint [25]	✓	✓		✓
SCP (this work)	✓	✓	✓	✓

Figure 1: Properties of different consensus mechanisms

consensus—small computational advantages can invalidate the security assumption, allowing history to be re-written in so-called "51% attacks." Worse, attackers initially controlling less than 50% of computation can game the system to provide disproportionate rewards for those who join them [20], thereby potentially gaining majority control. As the leading digital currency backed by the most computational power, Bitcoin enjoys a measure of protection against 51% attacks. Smaller systems have fallen victim [14, 9], however, posing a problem for any proof-of-work system not built on the Bitcoin block chain.

An alternative to proof of work is proof of stake [24], in which consensus depends on parties that have posted collateral. Like proof of work, rewards encourage rational participants to obey the protocol; some designs additionally penalize bad behavior [10, 15]. Proof of stake opens the possibility of so-called "nothing at stake" attacks, in which parties that previously posted collateral but later cashed it in and spent the money can go back and rewrite history from a point where they still had stake. To mitigate such attacks, systems effectively combine proof of stake with proof of work—scaling down the required work in proportion to stake—or delay refunding collateral long enough for some other (sometimes informal) consensus mechanism to establish an irreversible checkpoint.

Still another approach to consensus is Byzantine agreement [40, 29], the best known variant of which is PBFT [11]. Byzantine agreement ensures consensus despite arbitrary (including non-rational) behavior on the part of some fraction of participants. This approach has two appealing properties. First, consensus can be fast and efficient. Second, trust is entirely decoupled from resource ownership, which makes it possible for a small non-profit to help keep more powerful organizations, such as banks or CAs, honest. Complicating matters, however, all parties must agree on the the exact list of participants. Moreover, attackers must be prevented from joining multiple times and exceeding the system's failure tolerance, a so-called Sybil attack [17]. BFT-CUP [2] accommodates unknown participants, but still presupposes a Sybil-proof centralized admission-control mechanism.

Generally, membership in Byzantine agreement systems is set by a central authority or closed negotiation. Prior attempts to decentralize admission have given up some of the benefits. One approach, taken by Ripple, is to publish a "starter" membership list that participants can edit for themselves, hoping people's edits are either inconsequential or reproduced by an overwhelming fraction of participants. Unfortunately, because divergent lists invalidate safety guarantees [42], users are reluctant to edit the list in practice and a great deal of power ends up concentrated in the maintainer of the starter list. Another approach, taken by Tendermint [25], is to base membership on proof of stake. However, doing so once again ties trust to resource ownership. SCP is the first Byzantine agreement protocol to give each participant maximum freedom in chosing which combinations of other participants to trust.

3 Federated Byzantine agreement systems

This section introduces the federated Byzantine agreement (FBA) model. Like non-federated agreement, FBA addresses the problem of updating replicated state, such as a transaction ledger or certificate tree. By agreeing on what updates to apply, nodes avoid contradictory, irreconcilable states. We identify each update by a unique *slot* from which inter-update dependencies can be inferred. For instance, slots may be consecutively numbered positions in a sequentially applied log.

An FBA system runs a *consensus protocol* that ensures nodes agree on slot contents. A node v can safely apply update x in slot i when it has safely applied updates in all slots upon which i depends and, additionally, it believes all correctly functioning nodes will eventually agree on x for slot i. At this point, we say v has *externalized* x for slot i. The outside world may react to externalized values in irreversible ways, so a node cannot later change its mind about them.

A challenge for FBA is that malicious parties can join many times and outnumber honest nodes. Hence, traditional majority-based quorums do not work. Instead, FBA determines quorums in a decentralized way, by each node selecting what we call quorum slices. The following subsection discusses examples. Finally, we define the key properties of safety and liveness that a consensus protocol should hope to achieve.

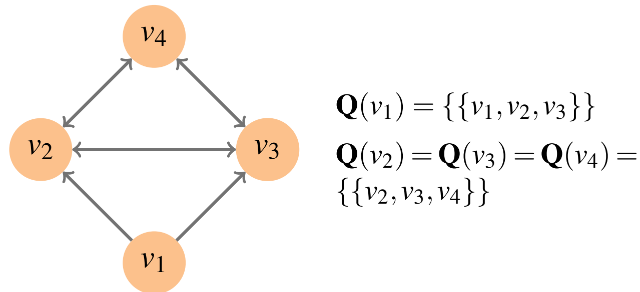


Figure 2: v_1's quorum slice is not a quorum without v_4.

3.1 Quorum slices

In a consensus protocol, nodes exchange messages asserting statements about slots. We assume such assertions cannot be forged, which can be guaranteed if nodes are named by public key and they digitally sign messages. When a node hears a sufficient set of nodes assert a statement, it assumes no functioning node will ever contradict that statement. We call such a sufficient set a *quorum slice*, or, more concisely, just a *slice*. To permit progress in the face of node failures, a node may have multiple slices, any one of which is sufficient to convince it of a statement. At a high level, then, an FBA system consists of a loose confederation of nodes each of which has chosen one or more slices. More formally:

Definition (FBAS)**.** A federated Byzantine agreement system, or *FBAS*, is a pair $\langle \mathbf{V}, \mathbf{Q} \rangle$ comprising a set of nodes $\mathbf{V}$ and a quorum function $\mathbf{Q} : \mathbf{V} \to 2^{2^{\mathbf{V}}} \setminus \{\emptyset\}$ specifying one or more quorum slices for each node, where a node belongs to all of its own quorum slices—i.e., $\forall v \in \mathbf{V}, \forall q \in \mathbf{Q}(v), v \in q$. (Note $2^X = \text{powerset}(X)$)

Definition (quorum)**.** A set of nodes $U \subseteq \mathbf{V}$ in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ is a *quorum* iff $U \neq \emptyset$ and U contains a slice for each member—i.e., $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$.

A quorum is a set of nodes sufficient to reach agreement. A quorum slice is the subset of a quorum convincing one particular node of agreement. A quorum slice may be smaller than a quorum. Consider the four-node system in Figure 2, where each node has a single slice and arrows point to the other members of that slice. Node v_1's slice $\{v_1, v_2, v_3\}$ is sufficient to convince v_1 of a statement. But v_2's and v_3's slices include v_4, meaning neither v_2 nor v_3 can assert a statement without v_4's agreement. Hence, no agreement is possible without v_4's participation, and the only quorum including v_1 is the set of all nodes $\{v_1, v_2, v_3, v_4\}$.

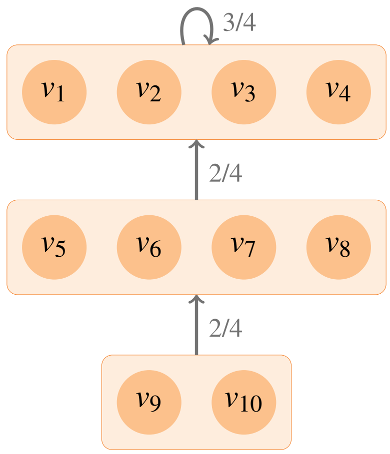


Figure 3: Tiered quorum structure example

Traditional non-federated consensus requires all nodes to accept the same slices, meaning $\forall v_1, v_2, \mathbf{Q}(v_1) = \mathbf{Q}(v_2)$. Because any slice suffices to convince all nodes of an assertion, centralized systems do not distinguish between slices and quorums. The downside is that membership and quorums must somehow be preordained, precluding open membership and decentralized control. A traditional system, such as PBFT [11], typically has $3f+1$ nodes, any $2f+1$ of which constitute a quorum. Here f is the maximum number of Byzantine failures—meaning nodes acting arbitrarily—the system can survive.

FBA, introduced by this paper, generalizes centralized consensus to accommodate a greater range of settings. FBA's key difference is that each node v choses its own quorum slice set $\mathbf{Q}(v)$. System-wide quorums thus arise from individual decisions made by each node. Nodes may select slices based on arbitrary criteria such as reputation or financial arrangements. In some settings, it may be impractical for any single node to track the complete set $\mathbf{V}$ of all nodes in the system, yet consensus should still be possible.

3.2 Examples and discussion

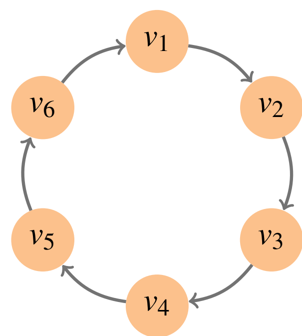
Figure 3 shows an example of a tiered system in which different nodes have very different slice sets, something possible only with FBA. A top tier, comprising $v_1, \ldots, v_4$, is structured like a PBFT system with $f = 1$, meaning it can tolerate one Byzantine failure so long as the other three nodes are reachable and well-behaved. Nodes $v_5, \ldots, v_8$ constitute a middle tier and depend not on each other, but rather on the top tier. Only two top tier nodes are required to form a slice for a middle tier node. (The top tier assumes at most one Byzantine failure, so two top tier nodes cannot both fail unless the whole system has failed.) Nodes v_9 and v_{10} are in a leaf tier for which a slice consists of any two middle tier nodes. Note that v_9 and v_{10} may pick disjoint slices such as $\{v_5, v_6\}$ and $\{v_7, v_8\}$; nonetheless, both will indirectly depend on the top tier.

In practice, the top tier could consist of anywhere from four to dozens of widely known and trusted financial institutions. As the size of the top tier grows, there may not be exact agreement on its membership, but there will be significant overlap between most parties' notions of top tier. Additionally, one can imagine multiple middle tiers, for instance one for each country or geographic region.

This tiered structure resembles inter-domain network routing. The Internet today is held together by individual peering and transit relationships between pairs of networks. No central authority dictates or arbitrates these arrangements. Yet these pairwise relationships have sufficed to create a notion of *de facto* tier one ISPs [35]. Though Internet reachability does suffer from firewalls, *transitive* reachability is nearly complete—e.g., a firewall might block The New York Times, but if it allows Google, and Google can reach The New York Times, then The New York Times is transitively reachable. Transitive reachability may be of limited utility for web sites, but it is crucial for consensus; the equivalent example would be Google accepting statements only if The New York Times does.

If we think of quorum slices as analogous to network reachability and quorums as analogous to transitive reachability, then the Internet's near complete transitive reachability suggests we can likewise ensure worldwide consensus with FBA. In many ways, consensus is an easier problem than inter-domain routing. While transit consumes resources and costs money, slice inclusion merely requires checking digital signatures. Hence, FBA nodes can err on the side of inclusiveness, constructing conservative slices with greater interdependence and redundancy than typically seen in peering and transit arrangements.

Another example not possible with centralized consensus is cyclic dependency structures, such as the one depicted in Figure 4. Such a cycle is unlikely to arise intentionally, but when individual nodes choose their own slices, it is possible for the overall system to end up embedding dependency cycles. The bigger point is that, compared to traditional Byzantine agreement, an FBA protocol must cope with a far wider variety of quorum structures.



$$\mathbf{Q}(v_i) = \{\{v_i, v_{(i \bmod 6)+1}\}\}$$

Figure 4: Cyclic quorum structure example

3.3 Safety and liveness

We categorize nodes as either *well-behaved* or *ill-behaved*. A well-behaved node chooses sensible quorum slices (discussed further in Section 4.1) and obeys the protocol, including eventually responding to all requests. An ill-behaved node does not. Ill-behaved nodes suffer Byzantine failure, meaning they behave arbitrarily. For instance, an ill-behaved node may be compromised, its owner may have maliciously modified the software, or it may have crashed.

The goal of Byzantine agreement is to ensure that well-behaved nodes externalize the same values despite the presence of such ill-behaved nodes. There are two parts to this goal. First, we would like to prevent nodes from diverging and externalizing different values for the same slot. Second, we would like to ensure nodes can actually externalize values, as opposed to getting blocked in some dead-end state from which consensus is no longer possible. We introduce the following two terms for these properties:

Definition (safety)**.** A set of nodes in an FBAS enjoy *safety* if no two of them ever externalize different values for the same slot.

Definition (liveness)**.** A node in an FBAS enjoys *liveness* if, given appropriate message delivery and timing, it can externalize new values.

We call well-behaved nodes that enjoy both safety and liveness *correct*. Nodes that are not correct have *failed*. All ill-behaved nodes have failed, but a well-behaved node can fail, too, by waiting indefinitely for messages from ill-behaved nodes, or, worse, by having its state poisoned by incorrect messages from ill-behaved nodes.

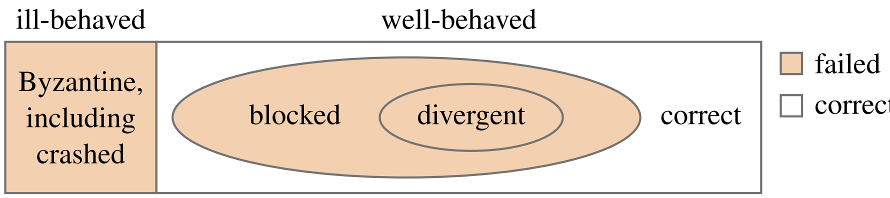


Figure 5: Venn diagram of node failures

Figure 5 illustrates the possible kinds of node failure. To the left are Byzantine failures, the ill-behaved nodes. To the right are two kinds of well-behaved but failed nodes. Nodes that lack liveness are termed *blocked*, while those that lack safety are termed *divergent*. An attack violating safety is strictly more powerful than one violating only liveness, so we classify divergent nodes as a subset of blocked ones.

Our definition of *liveness* is weak in that it admits *perpetual preemption*, a state in which consensus remains forever possible, yet the network continually thwarts it by delaying or reordering critical messages in just the wrong way. Perpetual preemption is inevitable in a purely asynchronous, deterministic system that survives node failure [21]. Fortunately, preemption is transient. It does not indicate node failure, because the system can recover at any time. Protocols can mitigate the problem through randomness (if nodes keep choosing random candidate values until enough happen to pick the same one [6, 8]) or through realistic assumptions about message latency [18]. The latter is more practical when one would like to limit execution time. Of course, only termination and not safety should depend on message timing.

4 Optimal resilience

Whether or not nodes enjoy safety and liveness depends on several factors: what quorum slices they have chosen, which nodes are ill-behaved, and of course the concrete consensus protocol and network behavior. As is common for asynchronous systems, we assume the network eventually delivers messages between well-behaved nodes, but can otherwise arbitrarily delay or reorder messages.

This section answers the following question: given a specific $\langle \mathbf{V}, \mathbf{Q} \rangle$ and particular subset of $\mathbf{V}$ that is ill-behaved, what are the best safety and liveness that any federated Byzantine agreement protocol can guarantee regardless of the network? We first discuss quorum intersection, a property without which safety is

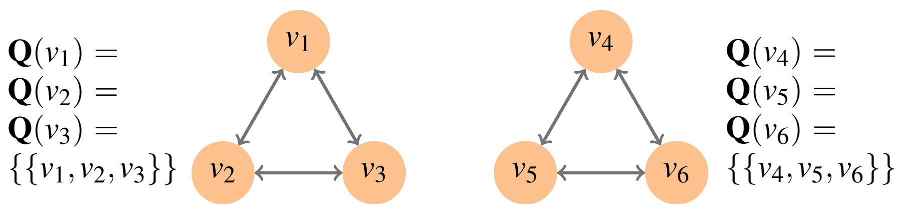


Figure 6: FBAS lacking quorum intersection

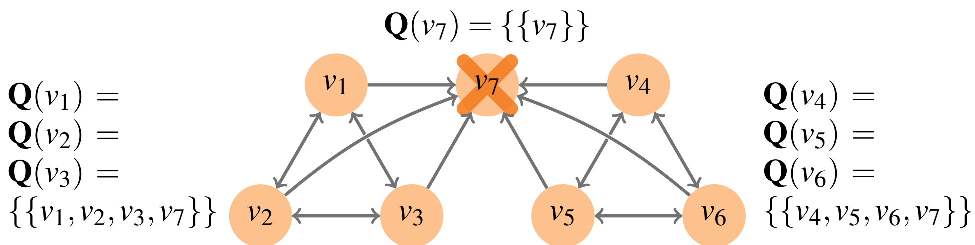


Figure 7: Ill-behaved node v_7 can undermine quorum intersection.

impossible to guarantee. We then introduce a notion of dispensable sets—sets of failed nodes in spite of which it is possible to guarantee both safety and liveness.

4.1 Quorum intersection

A protocol can guarantee agreement only if the quorum slices represented by function $\mathbf{Q}$ satisfy a validity property we call quorum intersection.

Definition (quorum intersection)**.** An FBAS enjoys *quorum intersection* iff any two of its quorums share a node—i.e., for all quorums U_1 and U_2, $U_1 \cap U_2 \neq \emptyset$.

Figure 6 illustrates a system lacking this property, where $\mathbf{Q}$ permits two quorums, $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$, that do not intersect. Disjoint quorums can independently agree on contradictory statements, undermining system-wide agreement. When many quorums exist, quorum intersection fails if any two do not intersect. For example, the set of all nodes $\{v_1, \ldots, v_6\}$ in Figure 6 is a quorum that intersects the other two, but the system still lacks quorum intersection.

No protocol can guarantee safety in the absence of quorum intersection, since such a configuration can operate as two different FBAS systems that do not know anything about each other. However, even with quorum intersection, safety may be impossible to guarantee in the presence of ill-behaved nodes. Compare Figure 6, in which there are two disjoint quorums, to Figure 7, in which two quorums intersect at a single node v_7, and v_7 is ill-behaved. If v_7 makes inconsistent statements to the left and right quorums, the effect is equivalent to disjoint quorums.

In fact, since ill-behaved nodes contribute nothing to safety, no protocol can guarantee safety without the well-behaved nodes enjoying quorum intersection on their own. After all, in a worst-case scenario for safety, all ill-behaved nodes can constantly make inconsistent statements in an attempt to make quorums diverge. Two quorums overlapping only at ill-behaved nodes will again be able to operate like two different FBAS systems thanks to the duplicity of the ill-behaved nodes. In short, FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ can survive Byzantine failure by a set of nodes $B \subseteq \mathbf{V}$ iff $\langle \mathbf{V}, \mathbf{Q} \rangle$ enjoys quorum intersection after deleting the nodes in B from $\mathbf{V}$ and from all slices in $\mathbf{Q}$. More formally:

Definition (delete)**.** If $\langle \mathbf{V}, \mathbf{Q} \rangle$ is an FBAS and $B \subseteq \mathbf{V}$ is a set of nodes, then to *delete* B from $\langle \mathbf{V}, \mathbf{Q} \rangle$, written $\langle \mathbf{V}, \mathbf{Q} \rangle^B$, means to compute the modified FBAS $\langle \mathbf{V} \setminus B, \mathbf{Q}^B \rangle$ where $\mathbf{Q}^B(v) = \{\, q \setminus B \mid q \in \mathbf{Q}(v) \,\}$.

It is the responsibility of each node v to ensure $\mathbf{Q}(v)$ does not violate quorum intersection. One way to do so is to pick conservative slices that lead to large quorums. Of course, a malicious v may intentionally pick $\mathbf{Q}(v)$ to violate quorum intersection. But a malicious v can also lie about the value of $\mathbf{Q}(v)$ or ignore $\mathbf{Q}(v)$ to make arbitrary assertions. In short, $\mathbf{Q}(v)$'s value is not meaningful when v is ill-behaved. This is why the necessary property for safety—quorum intersection of well-behaved nodes after deleting ill-behaved nodes—is unaffected by the slices of ill-behaved nodes.

Suppose Figure 6 evolved from a three-node FBAS v_1, v_2, v_3 with quorum intersection to a six-node FBAS without. When v_4, v_5, v_6 join, they maliciously choose slices that violate quorum intersection and no protocol can guarantee safety for $\mathbf{V}$. Fortunately, deleting the bad nodes to yield $\langle \mathbf{V}, \mathbf{Q} \rangle^{\{v_4, v_5, v_6\}}$ restores quorum intersection, meaning at least $\{v_1, v_2, v_3\}$ can enjoy safety. Note that deletion is conceptual, for the sake of describing optimal safety. A protocol should guarantee safety for v_1, v_2, v_3 without their needing to know that v_4, v_5, v_6 are ill-behaved.

4.2 Dispensable sets (DSets)

We capture the fault tolerance of nodes' slice selections through the notion of a *dispensible set* or *DSet*. Informally, the safety and liveness of nodes outside a DSet can be guaranteed regardless of the behavior of nodes inside the DSet. Put another way, in an optimally resilient FBAS, if a single DSet encompasses every ill-behaved node, it also contains every failed node, and conversely all nodes outside the DSet are correct. As an example, in a centralized PBFT system with $3f+1$ nodes and quorum size $2f+1$, any f or fewer nodes constitute a DSet. Since PBFT in fact survives up to f Byzantine failures, its robustness is optimal.

In the less regular example of Figure 3, $\{v_1\}$ is a DSet, since one top tier node can fail without affecting the rest of the system. $\{v_9\}$ is also a DSet because no other node depends on v_9 for correctness. $\{v_6,\dots,v_{10}\}$ is a DSet, because neither v_5 nor the top tier depend on any of those five nodes. $\{v_5,v_6\}$ is *not* a DSet, as it is a slice for v_9 and v_{10} and hence, if entirely malicious, can lie to v_9 and v_{10} and convince them of assertions inconsistent with each other or the rest of the system.

To prevent a misbehaving DSet from affecting the correctness of other nodes, two properties must hold. For safety, deleting the DSet cannot undermine quorum intersection. For liveness, the DSet cannot deny other nodes a functioning quorum. This leads to the following definition:

Definition (DSet)**.** Let $\langle \mathbf{V}, \mathbf{Q} \rangle$ be an FBAS and $B \subseteq \mathbf{V}$ be a set of nodes. We say B is a dispensible set, or *DSet*, iff:

1. *(quorum intersection despite B)* $\langle \mathbf{V}, \mathbf{Q} \rangle^B$ enjoys quorum intersection, and
2. *(quorum availability despite B)* Either $\mathbf{V} \setminus B$ is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle$ or $B = \mathbf{V}$.

Quorum availability despite B protects against nodes in B refusing to answer requests and blocking other nodes' progress. Quorum intersection despite B protects against the opposite—nodes in B making contradictory assertions that enable other nodes to externalize inconsistent values for the same slot. Nodes must balance the two threats in slice selection. All else equal, bigger slices lead to bigger quorums with greater overlap, meaning fewer failed node sets B will undermine quorum intersection when deleted. On the other hand, bigger slices are more likely to contain failed nodes, endangering quorum availability.

The smallest DSet containing all ill-behaved nodes may encompass well-behaved nodes as well; this reflects the fact that a sufficiently large set of ill-behaved nodes can cause well-behaved nodes to fail. For instance, in Figure 3, the smallest DSet containing v_5 and v_6 is $\{v_5,v_6,v_9,v_{10}\}$. As a special case, the set of all nodes, $\mathbf{V}$, is a DSet. The motivation for this special case is that, if all nodes fail, the remaining (zero) nodes are vacuously correct. Given sufficiently many ill-behaved nodes, the set $\mathbf{V}$ of all nodes may be the smallest DSet to contain all ill-behaved ones, which means no protocol can guarantee anything better than complete system failure.

well-behaved / ill-behaved	Local property of nodes, independent of other nodes (except for the validity of slice selection).
intact / befouled	Property of nodes given their quorum slices and a particular set of ill-behaved nodes. Befouled nodes are ill-behaved or depend, possibly indirectly, on too many ill-behaved nodes.
correct / failed	Property of nodes given their quorum slices, a concrete protocol, and actual network behavior. The goal of a consensus protocol is to guarantee node correctness whenever possible.

Figure 8: Key properties of FBAS nodes

The set of DSets in an FBAS is determined *a priori* by the quorum function $\mathbf{Q}$. Which nodes are well- and ill-behaved depends on runtime behavior, such as machines getting compromised. The DSets we care about are those that encompass all ill-behaved nodes, as they help us distinguish nodes that should be guaranteed correct from ones for which such a guarantee is impossible. To this end, we introduce the following terms:

Definition (intact)**.** A node v in an FBAS is *intact* iff there exists a DSet B containing all ill-behaved nodes and such that $v \notin B$.

Definition (befouled)**.** A node v in an FBAS is *befouled* iff it is not intact.

A befouled node v is surrounded by enough failed nodes to block its progress or poison its state, even if v itself is well-behaved. No FBAS can guarantee the correctness of a befouled node. However, an optimal FBAS guarantees that every intact node remains correct. Figure 8 summarizes the key properties of nodes. The following theorems facilitate analysis by showing that the set of befouled nodes is always a DSet in an FBAS with quorum intersection.

Theorem 1. *Let U be a quorum in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$, let $B \subseteq \mathbf{V}$ be a set of nodes, and let $U' = U \setminus B$. If $U' \neq \emptyset$ then U' is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$.*

Proof. Because U is a quorum, every node $v \in U$ has a $q \in \mathbf{Q}(v)$ such that $q \subseteq U$. Since $U' \subseteq U$, it follows that every $v \in U'$ has a $q \in \mathbf{Q}(v)$ such that $q \setminus B \subseteq U'$. Rewriting with deletion notation yields $\forall v \in U', \exists q \in \mathbf{Q}^B(v)$ such that $q \subseteq U'$, which, because $U' \subseteq \mathbf{V} \setminus B$, means that U' is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. □

Theorem 2. *If B_1 and B_2 are DSets in an FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ enjoying quorum intersection, then $B = B_1 \cap B_2$ is a DSet, too.*

Proof. Let $U_1 = \mathbf{V} \setminus B_1$ and $U_2 = \mathbf{V} \setminus B_2$. If $U_1 = \emptyset$, then $B_1 = \mathbf{V}$ and $B = B_2$ (a DSet), so we are done. Similarly, if $U_2 = \emptyset$, then $B = B_1$, and we are done. Otherwise, note that by quorum availability despite DSets B_1 and B_2, U_1 and U_2 are quorums in $\langle \mathbf{V}, \mathbf{Q} \rangle$. By definition, the union of two quorums is also a quorum. Hence $\mathbf{V} \setminus B = U_1 \cup U_2$ is a quorum and we have quorum availability despite B.

We must now show quorum intersection despite B. Let U_a and U_b be any two quorums in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. Let $U = U_1 \cap U_2 = U_2 \setminus B_1$. By quorum intersection of $\langle \mathbf{V}, \mathbf{Q} \rangle$, $U = U_1 \cap U_2 \neq \emptyset$. But then by Theorem 1, $U = U_2 \setminus B_1$ must be a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_1}$. Now consider that $U_a \setminus B_1$ and $U_a \setminus B_2$ cannot both be empty, or else $U_a \setminus B = U_a$ would be. Hence, by Theorem 1, either $U_a \setminus B_1$ is a quorum in $\left(\langle \mathbf{V}, \mathbf{Q} \rangle^B\right)^{B_1} = \langle \mathbf{V}, \mathbf{Q} \rangle^{B_1}$, or $U_a \setminus B_2$ is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_2}$, or both. In the former case, note that if $U_a \setminus B_1$ is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_1}$, then by quorum intersection of $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_1}$, $(U_a \setminus B_1) \cap U \neq \emptyset$; since $(U_a \setminus B_1) \cap U = (U_a \setminus B_1) \setminus B_2$, it follows that $U_a \setminus B_2 \neq \emptyset$, making $U_a \setminus B_2$ a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_2}$. By a similar argument, $U_b \setminus B_2$ must be a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^{B_2}$. But then quorum intersection despite B_2 tells us that $(U_a \setminus B_2) \cap (U_b \setminus B_2) \neq \emptyset$, which is only possible if $U_a \cap U_b \neq \emptyset$. □

Theorem 3. *In an FBAS with quorum intersection, the set of befouled nodes is a DSet.*

Proof. Let $B_{\min}$ be the intersection of every DSet that contains all ill-behaved nodes. It follows from the definition of *intact* that a node v is intact iff $v \notin B_{\min}$. Thus, $B_{\min}$ is precisely the set of befouled nodes. By Theorem 2, DSets are closed under intersection, so $B_{\min}$ is also a DSet. □

5 Federated voting

This section develops a federated voting technique that FBAS nodes can use to agree on a statement. At a high level, the process for agreeing on some statement a involves nodes exchanging two sets of messages. First, nodes *vote* for a. Then, if the vote was successful, nodes *confirm* a, effectively holding a second vote on the fact that the first vote succeeded.

From each node's perspective, the two rounds of messages divide agreement on a statement a into three phases: unknown, accepted, and confirmed. Initially, a's status is completely *unknown* to a node v—a could end up true, false, or even *stuck* in a permanently indeterminate state. If the first vote succeeds, v may come to *accept* a. No two intact nodes ever accept contradictory statements, so if v is intact and accepts a, then a cannot be false.

For two reasons, however, v accepting a does not make a true. First, the fact that v accepted a does not mean all intact nodes can; a could be stuck for other nodes. Second, if v is befouled, then accepting a means nothing—a may be false at intact nodes. Yet even if v is befouled—which v does not know—the system may still enjoy quorum intersection of well-behaved nodes, in which case for optimal safety v needs greater assurance of a. Holding a second vote addresses both problems. If the second vote succeeds, v moves to the *confirmed* phase in which it can finally deem a true and act on it.

The next few subsections detail the federated voting process. Because voting does not rule out the possibility of stuck statements, Section 5.6 discusses how to cope with them. Section 6 will turn federated voting into a consensus protocol that avoids the possibility of stuck slots for intact nodes.

5.1 Voting with open membership

A correct node in a Byzantine agreement system accepts a statement a only when it knows that other correct nodes will never agree to statements contradicting a. Most protocols employ voting for this purpose. Well-behaved nodes vote for a statement a only if it is valid. Well-behaved nodes also never change their votes. Hence, in centralized Byzantine agreement, it is safe to accept a if a majority of well-behaved nodes—a quorum—has voted for it. We say a statement is *ratified* once it has received the necessary votes.

In a federated setting, we must adapt voting to accommodate open membership. One difference is that a quorum no longer corresponds to a majority of well-behaved nodes. However, the majority requirement primarily serves to ensure quorum intersection of well-behaved nodes, which Section 4.1 already adapted to FBA. Another implication of open membership is that nodes must discover what constitutes a quorum as part of the voting process. To implement quorum discovery, a protocol can include $\mathbf{Q}(v)$ in all messages from v or provide some other means of querying v for $\mathbf{Q}(v)$.

Definition (vote)**.** A node v *votes* for an (abstract) statement a iff

1. v asserts a is valid and consistent with all statements v has accepted, and

2. v asserts it has never *voted against* a—i.e., voted for a statement that contradicts a—and v promises never to vote against a in the future.

Definition (ratify)**.** A quorum U_a *ratifies* a statement a iff every member of U_a votes for a. A node v *ratifies* a iff v is a member of a quorum U_a that ratifies a.

Theorem 4. *Two contradictory statements a and $\bar{a}$ cannot both be ratified in an FBAS that enjoys quorum intersection and contains no ill-behaved nodes.*

Proof. By contradiction. Suppose quorum U_1 ratifies a and quorum U_2 ratifies $\bar{a}$. By quorum intersection, $\exists v \in U_1 \cap U_2$. Such a v must have illegally voted for both a and $\bar{a}$, violating the assumption of no ill-behaved nodes. □

Theorem 5. *Let $\langle \mathbf{V}, \mathbf{Q} \rangle$ be an FBAS enjoying quorum intersection despite B, and suppose B contains all ill-behaved nodes. Let v_1 and v_2 be two nodes not in B. Let a and $\bar{a}$ be contradictory statements. If v_1 ratifies a then v_2 cannot ratify $\bar{a}$.*

Proof. By contradiction. Suppose v_1 ratifies a and v_2 ratifies $\bar{a}$. By definition, there must exist a quorum U_1 containing v_1 that ratified a and quorum U_2 containing v_2 that ratified $\bar{a}$. By Theorem 1, since $U_1 \setminus B \neq \emptyset$ and $U_2 \setminus B \neq \emptyset$, both must be quorums in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$, meaning they ratified a and $\bar{a}$ respectively in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. But $\langle \mathbf{V}, \mathbf{Q} \rangle^B$ enjoys quorum intersection and has no ill-behaved nodes, so Theorem 4 tell us a and $\bar{a}$ cannot both be ratified. □

Theorem 6. *Two intact nodes in an FBAS with quorum intersection cannot ratify contradictory statements.*

Proof. Let B be the set of befouled nodes. By Theorem 3, B is a DSet. By the definition of *DSet*, $\langle \mathbf{V}, \mathbf{Q} \rangle$ enjoys quorum intersection despite B. By Theorem 5, two nodes not in B cannot ratify contradictory statements. □

5.2 Blocking sets

In centralized consensus, liveness is an all-or-nothing property of the system. Either a unanimously well-behaved quorum exists, or else ill-behaved nodes can prevent the rest of the system from accepting new statements. In FBA, by contrast, liveness may differ across nodes. For instance, in the tiered quorum example of Figure 3, if middle tier nodes v_6, v_7, v_8 crash, the leaf tier will be blocked while the top tier and node v_5 will continue to enjoy liveness.

An FBA protocol can guarantee liveness to a node v only if $\mathbf{Q}(v)$ contains at least one quorum slice comprising only correct nodes. A set B of failed nodes can violate this property if B contains at least one member of each of v's slices. We term such a set B v-blocking, because it has the power to block progress by v.

Definition (v-blocking)**.** Let $v \in \mathbf{V}$ be a node in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$. A set $B \subseteq \mathbf{V}$ is *v-blocking* iff it overlaps every one of v's slices—i.e., $\forall q \in \mathbf{Q}(v), q \cap B \neq \emptyset$.

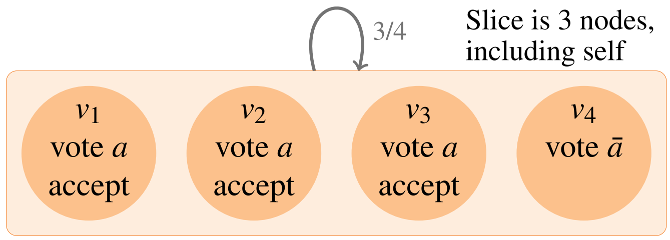


Figure 9: v_4 voted for $\bar{a}$, which contradicts ratified statement a.

Theorem 7. *Let $B \subseteq \mathbf{V}$ be a set of nodes in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$. $\langle \mathbf{V}, \mathbf{Q} \rangle$ enjoys quorum availability despite B iff B is not v-blocking for any $v \in \mathbf{V} \setminus B$.*

Proof. "$\forall v \in \mathbf{V} \setminus B, B$ is not v-blocking" is equivalent to "$\forall v \in \mathbf{V} \setminus B, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq \mathbf{V} \setminus B$." By the definition of *quorum*, the latter holds iff $\mathbf{V} \setminus B$ is a quorum or $B = \mathbf{V}$, the exact definition of *quorum availability despite B*. □

As a corollary, the DSet of befouled nodes is not v-blocking for any intact v.

5.3 Accepting statements

When an intact node v learns that it has ratified a statement, Theorem 6 tells v that other intact nodes will not ratify contradictory statements. This condition is sufficient for v to accept a, but we cannot make it necessary. Ratifying a statement requires voting for it, and some nodes may have voted for contradictory statements. In Figure 9, for example, v_4 votes for $\bar{a}$ before learning that the other three nodes ratified the contradictory statement a. Though v_4 cannot now vote for a, we would still like it to accept a to be consistent with the other nodes.

A key insight is that if a node v is intact, then no v-blocking set B can consist entirely of befouled nodes. Now suppose B is a v-blocking set and every member of B claims to accept statement a. If v is intact, at least one member of B must be, too. The intact member will not lie about accepting a; hence, a is true and v can accept it. Of course, if v is befouled, then a might not be true. But a befouled node can accept anything and vacuously not affect the safety of intact nodes.

Definition (accept)**.** An FBAS node v *accepts* a statement a iff it has never accepted a statement contradicting a and it determines that either

1. There exists a quorum U such that $v \in U$ and each member of U either voted for a or claims to accept a, or

2. Each member of a v-blocking set claims to accept a.

Though a well-behaved node cannot vote for contradictory statements, condition 2 above allows a node to *vote* for one statement and later *accept* a contradictory one.

Theorem 8. *Two intact nodes in an FBAS that enjoys quorum intersection cannot accept contradictory statements.*

Proof. Let $\langle \mathbf{V}, \mathbf{Q} \rangle$ be an FBAS with quorum intersection and let B be its DSet of befouled nodes (which exists by Theorem 3). Suppose an intact node accepts statement a. Let v be the first intact node to accept a. At the point v accepts a, only befouled nodes in B can claim to accept it. Since by the corollary to Theorem 7, B cannot be v-blocking, it must be that v accepted a through condition 1. Thus, v identified a quorum U such that every node claimed to vote for or accept a, and since v is the first intact node to accept a, it must mean all nodes in $U \setminus B$ voted for a. In other words, v ratified a in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. Generalizing, any statement accepted by an intact node in $\langle \mathbf{V}, \mathbf{Q} \rangle$ must be ratified in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. Because B is a DSet, $\langle \mathbf{V}, \mathbf{Q} \rangle^B$ enjoys quorum intersection. Because additionally B contains all ill-behaved nodes, Theorem 4 rules out ratification of contradictory statements. □

5.4 Accepting is not enough

Unfortunately, for nodes to assume the truth of accepted statements would yield suboptimal safety and liveness guarantees in a federated consensus protocol. We discuss the issues with safety and liveness in turn. To provide some context, we then explain why these issues are thornier in FBA than in centralized Byzantine agreement.

5.4.1 Safety

Consider an FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ in which the only quorum is unanimous consent—i.e., $\forall v, \mathbf{Q}(v) = \{\mathbf{V}\}$. This ought to be a conservative choice for safety—don't do anything unless everyone agrees. Yet since every node is v-blocking for every v, any node can single-handedly convince any other node to accept arbitrary statements.

The problem is that accepted statements are only safe among intact nodes. But as discussed in Section 4.1, the only condition necessary to guarantee safety is quorum intersection of well-behaved nodes, which might hold even in the case that some well-behaved nodes are befouled. In particular, when $\mathbf{Q}(v) = \{\mathbf{V}\}$, the only DSets are $\emptyset$ and $\mathbf{V}$, meaning any node failure befouls the whole system. By contrast, quorum intersection holds despite every $B \subseteq \mathbf{V}$.

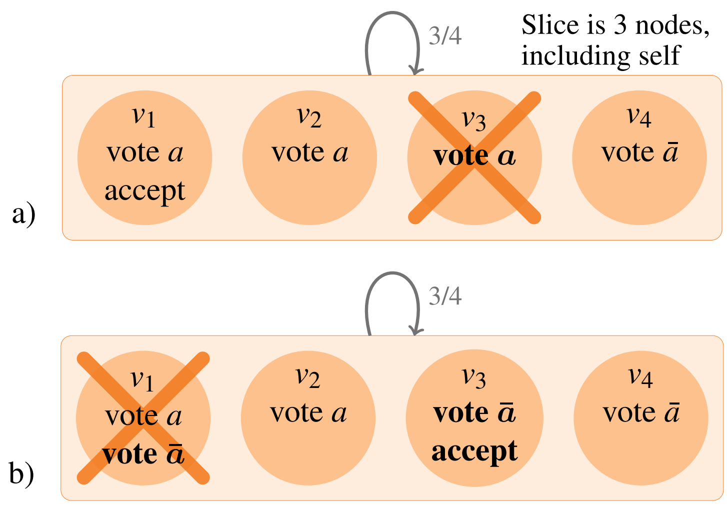


Figure 10: Scenarios indistinguishable to v_2 when v_2 does not see bold messages

5.4.2 Liveness

Another limitation of accepted statements is that other intact nodes may be unable to accept them. This possibility makes reliance on accepted statements problematic for liveness. If a node proceeds to act on a statement because it accepted the statement, other nodes could be unable to proceed in a similar fashion.

Consider Figure 10a, in which node v_3 crashes after helping v_1 ratify and accept statement a. Though v_1 accepts a, v_2 and v_4 cannot. In particular, from v_2's perspective, the situation depicted is indistinguishable from Figure 10b, in which v_3 voted for $\bar{a}$ and is well-behaved but slow to respond, while v_1 is ill-behaved and sent v_3 a vote for $\bar{a}$ (thereby causing v_3 to accept $\bar{a}$) while illegally also sending v_2 a vote for a.

To support a protocol-level notion of liveness in cases like Figure 10a, v_1 needs a way to ensure every other intact node can eventually accept a before v_1 acts on a. Once this is the case, it makes sense to say the system agrees on a.

Definition (agree)**.** An FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ *agrees* on a statement a iff, regardless of what subsequently transpires, once sufficient messages are delivered and processed, every intact node will accept a.

5.4.3 Comparison to centralized voting

To understand why the above issues arise in federated voting, consider a centralized Byzantine agreement system of N nodes with quorum size T. Such a system enjoys quorum availability with $f_L = N - T$ or fewer node failures. Since any two quorums share at least $2T - N$ nodes, quorum intersection of well-behaved nodes holds up to $f_S = 2T - N - 1$ Byzantine failures.

Centralized Byzantine agreement systems typically set $N = 3f + 1$ and $T = 2f + 1$ to yield $f_L = f_S = f$, the equilibrium point at which safety and liveness have the same fault tolerance. If safety is more important than liveness, some protocols increase T so that $f_S > f_L$ [32]. In FBA, because quorums arise organically, systems are unlikely to find themselves at equilibrium, making it far more important to protect safety in the absence of liveness.

Now consider a node v that votes against a ratified statement a in a centralized system. If v hears $f_S + 1$ nodes claim a was ratified, v knows that either one of them is well-behaved or all safety guarantees have collapsed. Either way, v can immediately act on a with no loss of safety. The FBA equivalent would be to hear from a set B where B, if deleted, undermines quorum intersection of well-behaved nodes. Identifying such a B is a tricky proposition for three reasons: one, quorums are discovered dynamically; two, ill-behaved nodes may lie about slices; and three, v does not know which nodes are well-behaved. Instead, we defined federated voting to accept a when a v-blocking set does. The v-blocking property has the advantage of being easily checkable, but is equivalent to hearing from $f_L + 1$ nodes in a centralized system when we really want $f_S + 1$.

To guarantee agreement among all well-behaved nodes in a centralized system, one merely needs $f_L + f_S + 1$ nodes to acknowledge that a statement was ratified. If more than f_L of them fail, we do not expect liveness anyway. If f_L or fewer fail, then we know $f_S + 1$ nodes remain willing to attest to ratification, which will in turn convince all other well-behaved nodes. Again, the reliance on f_S has no easy analogue in the FBA model.

Put another way, at some point nodes need to believe a statement strongly enough to depend on its truth for safety. A centralized system offers two ways to reach this point for a statement a: ratify a first-hand, or reason backwards from $f_S + 1$ nodes claiming a was ratified, figuring safety is hopeless if they have all lied. FBA lacks the latter approach; the only tool it has for safety among well-behaved nodes is first-hand ratification. Since nodes still need a way to overcome votes against ratified statements, we introduced a notion of accepting, but it provides a weaker consistency guarantee limited to intact nodes.

5.5 Statement confirmation

Both limitations of accepted statements stem from the fact that an intact node v may end up voting against a statement a that is nonetheless ratified. After voting against a, v cannot vote for it, preventing v from ever ratifying a. To provide v a means of accepting a after voting against it, the definition of *accept* has a second criterion based on v-blocking sets. But the second criterion is weaker than ratification, offering no guarantees to befouled nodes that enjoy quorum intersection.

Now if a statement a has the property that no intact node ever votes against it, then we have no need to accept it. Intact nodes can all simply ratify a, and we can require them to do so before acting on a. We call such statements irrefutable.

Definition (irrefutable)**.** A statement a is *irrefutable* in an FBAS if no intact node can ever vote against it.

Theorem 8 tells us that two intact nodes cannot accept contradictory statements. Thus, while some nodes may vote against a statement a that was accepted by an intact node, the statement *an intact node accepted a* is irrefutable. This suggests holding a second vote to ratify the fact that an intact node accepted a.

Definition (confirm)**.** A quorum U_a in an FBAS *confirms* a statement a iff $\forall v \in U_a$, v claims to accept a. A node *confirms a* iff it is in such a quorum.

Nodes express that they have accepted statement a by stating "accept(a)," an abbreviation of the statement, "An intact node accepted a." To confirm a means to ratify accept(a). A well-behaved node v can vote for accept(a) only after accepting a, as v cannot assume any other nodes are intact. If v itself is befouled, accept(a) might be false, in which case voting for it may cost v liveness, but a befouled node has no guarantee of liveness anyway.

The next theorem shows that nodes can rely on confirmed statements without losing optimal safety. Theorem 10 then shows that confirmed statements meet the definition of *agreement* from Section 5.4.2, meaning nodes can rely on confirmed statements without endangering the liveness of intact nodes.

Theorem 9. *Let $\langle \mathbf{V}, \mathbf{Q} \rangle$ be an FBAS enjoying quorum intersection despite B, and suppose B contains all ill-behaved nodes. Let v_1 and v_2 be two nodes not in B. Let a and $\bar{a}$ be contradictory statements. If v_1 confirms a, then v_2 cannot confirm $\bar{a}$.*

Proof. First note that accept(a) contradicts accept($\bar{a}$)—no well-behaved node can vote for both. Note further that v_1 must ratify accept(a) to confirm a. By Theorem 5, v_2 cannot ratify accept($\bar{a}$) and hence cannot confirm $\bar{a}$. □

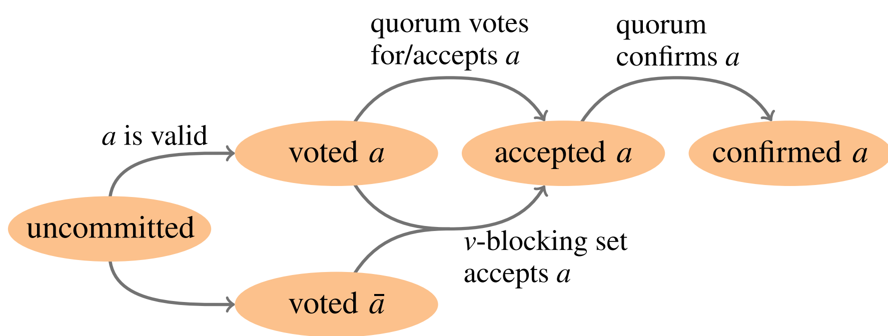


Figure 11: Possible states of an accepted statement a at a single node v

Theorem 10. *If an intact node in an FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ with quorum intersection confirms a statement a, then, whatever subsequently transpires, once sufficient messages are delivered and processed, every intact node will accept and confirm a.*

Proof. Let B be the DSet of befouled nodes and let $U_a \nsubseteq B$ be the quorum through which an intact node confirmed a. Let nodes in $U_a \setminus B$ broadcast accept(a). By definition, any node v, regardless of how it has voted, accepts a after receiving accept(a) from a v-blocking set. Hence, these messages may convince additional nodes to accept a. Let these additional nodes in turn broadcast accept(a) until a point is reached at which, regardless of future communication, no further intact nodes can ever accept a. Once this process is complete, let V^+ be the set of intact nodes that have accepted a, and let $V^- = (\mathbf{V} \setminus B) \setminus V^+$ be the intact nodes that cannot accept a. To show every intact node accepted a, we must show $V^- = \emptyset$.

By Theorem 1, $U_a \setminus B$ is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. Because V^+ is not v-blocking for any $v \in V^-$, then by Theorem 7 either $V^- = \emptyset$ or V^- is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle^B$. The latter case leads to a contradiction: Since V^- consists of only intact (and thus well-behaved) nodes, none of them would state accept(a) without first actually accepting a, meaning $(U_a \setminus B) \cap V^- = \emptyset$. But this is impossible, as quorum intersection despite B (a DSet) tells us the opposite, namely $(U_a \setminus B) \cap V^- \neq \emptyset$. The remaining possibility is that $V^- = \emptyset$. Once every intact node accepts a, all can vote to confirm a. The vote will succeed since intact nodes form a quorum. □

Figure 11 summarizes the paths an intact node v can take to confirm a. Given no knowledge, v might vote for either a or the contradictory $\bar{a}$. If v votes for $\bar{a}$, it cannot later vote for a, but can nonetheless accept a if a v-blocking set accepts it. A subsequent quorum of confirmation messages allows v to confirm a, which by Theorem 10 means the system agrees on a.

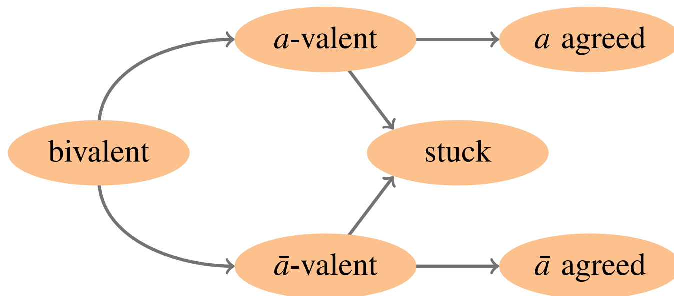


Figure 12: Possible system-wide status of a statement a

5.6 Liveness and neutralization

The main challenge of distributed consensus, whether centralized or not, is that a statement can get stuck in a permanently indeterminate state before the system reaches agreement on it. Hence, a protocol must not attempt to ratify externalized values directly. Should the statement “The value of slot i is x” get stuck, the system will be forever unable to agree on slot i, losing liveness. The solution is to craft the statements in votes carefully. It must be possible to break a stuck statement’s hold on the question we really care about, namely slot contents. We call the process of obsoleting a stuck statement *neutralization*.

More concretely, Figure 12 depicts the potential status a statement a can have system-wide. Initially, the system is *bivalent*, by which we mean there is one sequence of possible events through which all intact nodes will accept a, and another sequence through which all intact nodes will *reject* a (i.e., accept a statement $\bar{a}$ contradicting a). At some point, one of these two outcomes may cease to be possible. If no intact node can ever reject a, we say the system is *a-valent*; conversely, if no intact node can ever accept a, we say the system is *$\bar{a}$-valent*.

At the time an FBAS transitions from bivalent to a-valent, there is a possible outcome in which all intact nodes accept a. However, this might not remain the case. Consider a PBFT-like four-node system $\{v_1, \ldots, v_4\}$ in which any three nodes constitute a quorum. If v_1 and v_2 vote for a, the system becomes a-valent; no three nodes can ratify a contradictory statement. However, if v_3 and v_4 subsequently vote for $\bar{a}$ contradicting a, it also becomes impossible to ratify a. In this case, a’s state is permanently indeterminate, or *stuck*.

As seen in Figure 10a, even once an intact node accepts a, the system may still fail to reach system-wide agreement on a. However, by Theorem 10, once an intact node confirms a, all intact nodes can eventually come to accept it; hence the

Local state	**System-wide status of *a***
uncommitted	unknown (any)
voted a	unknown (any)
voted $\bar{a}$	unknown (any)
accepted a	stuck, a-valent, or a agreed
confirmed a	a agreed

Figure 13: What an intact node knows about the status of statement a

system has agreed upon a. Figure 13 summarizes what intact nodes know about the global state of a statement from their own local state.

To preserve the possibility of consensus, a protocol must ensure that every statement is either irrefutable, and hence cannot get stuck, or neutralizable, and hence cannot block progress if stuck. There are two popular approaches to crafting neutralizable statements: the *view-based* approach, pioneered by viewstamped replication [38] and favored by PBFT [11], and the *ballot-based* approach, invented by Paxos [26]. The ballot-based approach may be harder to understand [39]. Compounding confusion, people often call viewstamped replication "Paxos" or assert the two algorithms are the same when they are not [43].

View-based protocols associate the *slots* in votes with monotonically increasing view numbers. Should consensus get stuck on the ith slot in view n, nodes recover by agreeing that view n had fewer than i meaningful slots and moving to a higher view number. Ballot-based protocols associate the *values* in votes with monotonically increasing ballot numbers. Should a ballot get stuck, nodes retry with a higher ballot number, taking care to select values that do not contradict any previous indeterminate ballot.

This work takes a ballot-based approach, as doing so makes it easier to do away with the notion of a distinguished primary node or leader. For example, leader behavior can be emulated [28].

6 SCP: A federated Byzantine agreement protocol

This section presents the Stellar Consensus Protocol, SCP. At a high level, SCP consists of two sub-protocols: a nomination protocol and a ballot protocol. The nomination protocol produces candidate values for a slot. If run long enough, it eventually produces the same set of candidate values at every intact node, which means nodes can combine the candidate values in a deterministic way to produce a single composite value for the slot. There are two huge caveats, however. First, nodes have no way of knowing when the nomination protocol has reached the point of convergence. Second, even after convergence, ill-behaved nodes may be able to reset the nomination process a finite number of times.

When nodes guess that the nomination protocol has converged, they execute the ballot protocol, which employs federated voting to commit and abort ballots associated with composite values. When intact nodes agree to commit a ballot, the value associated with the ballot will be externalized for the slot in question. When they agree to abort a ballot, the ballot's value becomes irrelevant. If a ballot gets stuck in a state where one or more intact nodes cannot commit or abort it, then nodes try again with a higher ballot; they associate the new ballot with the same value as the stuck one in case any node believes the stuck ballot was committed. Intuitively, safety results from ensuring that all stuck and committed ballots are associated with the same value. Liveness follows from the fact that a stuck ballot can be neutralized by moving to a higher ballot.

The remainder of this section presents the nomination and ballot protocols. Each is described first in terms of conceptual statements, then as a concrete protocol with messages representing sets of conceptual statements. Finally, Section 6.3 shows the correctness of the protocol. SCP treats each slot completely independently and can be viewed as many separate instances of a single-slot consensus protocol (akin to the "single-decree synod" in Paxos [26]). Concepts such as candidate values and ballots must always be interepreted in the context of a particular slot even if much of the discussion leaves the slot implicit.

6.1 Nomination protocol

Because slots need only be partially ordered, some applications of SCP will have only one plausible ballot per slot. For example, in certificate transparency, each CA may have its own series of slots and sign exactly one certificate tree per slot. However, other applications admit many plausible values per slot, in which case it is helpful to narrow down the possible input values. Our strategy is to begin with a synchronous nomination protocol that achieves consensus under certain timing assumptions, and feed the output of the nomination protocol into an asynchronous ballot protocol whose safety does not depend on timing [27]. Such an initial synchronous phase is sometimes called a *conciliator* [3].

The nomination protocol works by converging on a set of candidate values for a slot. Nodes then deterministically combine these candidates into a single *composite* value for the slot. Exactly how to combine values depends on the application. By way of example, the Stellar network uses SCP to choose a set of transactions and a ledger timestamp for each slot. To combine candidate values, Stellar takes the union of their transaction sets and the maximum of their timestamps. (Values with invalid timestamps will not receive enough nominations to become candidates.) Other possible approaches include combining sets by intersection or simply picking the candidate value with the highest hash.

Nodes produce a candidate value x through federated voting on the statement *nominate* x.

Definition (candidate)**.** A node v considers a value x to be a *candidate* when v has confirmed the statement *nominate* x—i.e., v has ratified accept(*nominate* x).

So long as node v has no candidate values, v may vote in favor of *nominate* x for any value x that passes application-level validity checks (such as timestamps not being in the future). In fact, v should generally re-nominate any values that it sees other nodes nominate, with some rate-limiting discussed below to avoid an explosion of candidates. As soon as v has a candidate value, however, it must cease voting to *nominate* x for any new values x. It should still continue to accept *nominate* statements for new values (when accepted by a v-blocking set) and confirm new *nominate* statements as prescribed by the federated voting procedure.

The nomination protocol enjoys several properties when a system has intact nodes (meaning it has avoided complete failure). Specifically, for each slot:

1. Intact nodes can produce at least one candidate value.
2. At some point, the set of possible candidate values stops growing.
3. If any intact node considers x to be a candidate value, then eventually every intact node will consider x to be a candidate value.

Now consider how the nomination protocol achieves its three properties. Property 1 is achieved because *nominate* statements are irrefutable. Nodes never vote against nominating a particular value, and until the first candidate value is confirmed, intact nodes can vote to nominate any value. So long as any value x passes application-level validity checks, intact nodes can vote for and confirm *nominate* x. Property 2 is ensured because once each intact node confirms at least one candidate value—which will happen in a finite amount of time—no intact nodes will vote to nominate any new values. Hence, the only values that can become candidates are those that already have votes from intact nodes. Property 3 is a direct consequence of Theorem 10.

The nomination process will be more efficient if fewer combinations of values are in play. Hence, we assign nodes a temporary priority and have each node, when possible, nominate the same values as a higher-priority node. More concretely, let H be a cryptographic hash function whose range can be interpreted as a set of integers $\{0,\ldots,h_{\max}-1\}$. (H might be SHA-256 [37], in which case $h_{\max}=2^{256}$.) Let $G_i(m)=H(i,x_{i-1},m)$ be a slot-specific hash function for slot i, where x_{i-1} is the value chosen for the slot preceding i (or the sorted set of values of all immediate dependencies of slot i when slots are governed by a partial order). Given a slot i and a *round number* n, each node v computes a set of *neighbors* and a *priority* for each neighbor as follows:

$$\begin{aligned}\mathrm{weight}(v,v') &= \frac{\big|\{\, q \mid q\in \mathbf{Q}(v) \ \wedge\ v'\in q \,\}\big|}{\big|\mathbf{Q}(v)\big|}\\ \mathrm{neighbors}(v,n) &= \big\{\, v' \mid G_i(\textsc{n},n,v') < h_{\max}\cdot \mathrm{weight}(v,v') \,\big\}\\ \mathrm{priority}(n,v') &= G_i(\textsc{p},n,v')\end{aligned}$$

N and P are constants to produce two different hash functions. The function $\mathrm{weight}(v,v')$ returns the fraction of slices in $\mathbf{Q}(v)$ containing v'. By using weight as the probability over n that v' appears in $\mathrm{neighbors}(v,n)$, we also reduce the chance that nodes without a lot of trust will dominate a round.

Each node v should initially find a node $v_0\in\mathrm{neighbors}(v,0)$ that maximizes $\mathrm{priority}(0,v_0)$ among nodes it can communicate with, then vote to *nominate* the same values as v_0. Only if $v=v_0$ should v introduce a new value to nominate. v should use timeouts to decide on new *nominate* statements to vote for. After n timeouts, v should find a node $v_n\in\mathrm{neighbors}(v,n)$ maximizing $\mathrm{priority}(n,v_n)$ and vote to nominate everything v_n has voted to nominate.

Theorem 11. *Eventually, all intact nodes will have the same composite value.*

Proof. The theorem follows from the three properties of the nomination protocol. Each intact node will only ever vote to nominate a finite number of ballots. In the absence of action by ill-behaved nodes, intact nodes will converge on the same set of candidate values, call it Z. To forestall this convergence, ill-behaved nodes may introduce new candidate values, which for a period may be candidates at some but not all intact nodes. Such values will need to have garnered votes from well-behaved nodes, however, which limits them to a finite set. Eventually, ill-behaved nodes will either stop perturbing the system or run out of new candidate values to inject, in which case intact nodes will converge on Z. □

Variable	Meaning
X	The set of values node v has voted to nominate
Y	The set of values node v has has accepted as nominated
Z	The set of values that node v considers candidate values
N	The set of the latest NOMINATE message received from each node

Figure 14: Nomination state maintained by node v for each slot

NOMINATE $v\ i\ X\ Y\ D$

This is a message from node v nominating values for slot i. D is v's quorum slice $\mathbf{Q}(v)$ or a collision-resistant hash of $\mathbf{Q}(v)$. X and Y are from v's state. The concrete message encodes the following conceptual messages:

- $\{\, \textit{nominate}\ x \mid x \in X \,\}$ (votes to nominate each value in X)
- $\{\, \text{accept}(\textit{nominate}\ x) \mid x \in Y \,\}$ (votes to confirm nominations in Y)

Figure 15: Message in nomination protocol

6.1.1 Concrete nomination protocol

Figure 14 lists the nomination protocol state a node v must maintain for each slot. X is the set of values x for which v has voted *nominate* x, Y is the set of values for which v has accepted *nominate* x, and Z is the set of candidate values—i.e., all values for which a quorum including v has stated accept(*nominate* x). Finally, v maintains N, the latest concrete message from each node. (Technically, X, Y, and Z can all be recomputed from N, but it is convenient to be able to reference them directly.) All four fields are initialized to the empty set. Note that all three of X, Y, and Z are growing over time—nodes never remove a value from these sets.

Figure 15 shows the concrete message that constitutes the nomination protocol. Because X and Y grow monotonically over time, it is possible to determine which of multiple NOMINATE messages from the same nodes is the latest, independent of network delivery order, so long as D does not change mid-nomination (or D has to be versioned). Only one remote procedure call (RPC) is needed for nomination—the argument is the sender's latest NOMINATE message and the return value is the receiver's. If D or the nominated values are cryptographic hashes, a second RPC should permit retrieval of uncached hash preimages as needed.

Because nodes cannot tell when the nomination protocol is complete anyway, SCP must cope with different composite values at different nodes. As an optimization, then, nodes can attempt to predict the final composite value before they even have a candidate value. To do this, the composite value can be taken as combine(Z) when $Z \neq \emptyset$, otherwise combine(Y) when $Y \neq \emptyset$, otherwise combine(X) when $X \neq \emptyset$. This means the highest-priority node can optimistically initiate balloting at the same time as nomination, piggybacking its first ballot message PREPARE (described below) on its first NOMINATE message.

6.2 Ballot protocol

Once nodes have a composite value, they engage in the ballot protocol, though nomination may continue to update the composite value. A ballot b is a pair of the form $b = \langle n, x \rangle$, where $x \neq \perp$ is a value and b is a referendum on externalizing x for the slot in question. The value $n \geq 1$ is a counter to ensure higher ballot numbers are always available. We use C-like notation $b.n$ and $b.x$ to denote the counter and value fields of ballot b, so that $b = \langle b.n, b.x \rangle$. Ballots are totally ordered, with $b.n$ more significant than $b.x$. For convenience, a special invalid null ballot $\mathbf{0} = \langle 0, \perp \rangle$ is less than all other ballots, and a special counter value ∞ is greater than all other counters.

We speak of committing and aborting a ballot b as a shorthand for using federated voting to agree on the statements *commit b* and *abort b*, respectively. For a given ballot, *commit* and *abort* are contradictory, so a well-behaved node may vote for at most one of them. In the notation of Section 5, the opposite of *commit b* would be "$\overline{\textit{commit } b}$," but *abort b* is a more intuitive notation.

Because at most one value can be chosen for a given slot, all committed and stuck ballots must contain the same value. Roughly speaking, this means *commit* statements are invalid if they conflict with lower-numbered unaborted ballots.

Definition (compatible)**.** Two ballots b_1 and b_2 are *compatible*, written $b_1 \sim b_2$, iff $b_1.x = b_2.x$ and *incompatible*, written $b_1 \nsim b_2$, iff $b_1.x \neq b_2.x$. We also write $b_1 \lesssim b_2$ or $b_2 \gtrsim b_1$ iff $b_1 \leq b_2$ (or equivalently $b_2 \geq b_1$) and $b_1 \sim b_2$. Similarly, $b_1 \lnsim b_2$ or $b_2 \gnsim b_1$ means $b_1 \leq b_2$ (or equivalently $b_2 \geq b_1$) and $b_1 \nsim b_2$.

Definition (prepared)**.** A ballot b is *prepared* iff every statement in the following set is true: $\{\, \textit{abort } b_{\text{old}} \mid b_{\text{old}} \lnsim b \,\}$.

More precisely, then, *commit b* is valid to vote for only if b is confirmed prepared, which nodes ensure through federated voting on the corresponding *abort*

Variable	Meaning
φ	Current phase: one of PREPARE, CONFIRM, or EXTERNALIZE
b	Current ballot that node v is attempting to prepare and commit
p', p	The two highest ballots accepted as prepared such that $p' \lnapprox p$, where $p' = \mathbf{0}$ or $p = p' = \mathbf{0}$ if there are no such ballots
P	In PREPARE: highest ballot v confirmed as prepared, or $\mathbf{0}$ if none In CONFIRM: highest ballot for which v accepted *commit* P In EXTERNALIZE: highest ballot for which v confirmed *commit* P
c	In PREPARE: lowest ballot for which v has voted *commit* c and not accepted *abort* c, or $\mathbf{0}$ if none In CONFIRM: lowest ballot with *commit* c accepted In EXTERNALIZE: lowest ballot with *commit* c confirmed Invariant: if $c \neq \mathbf{0}$ then $c \lesssim P \lesssim b$.
M	Set of the latest non-NOMINATE message seen from each node

Figure 16: Ballot state maintained by each node v for each slot

statements. It is convenient to vote on these statements en masse, so wherever we write "b is prepared," the surrounding context applies to the whole set of *abort* statements. In particular, a node votes, accepts, or confirms that b is prepared iff it votes for, accepts, or confirms, respectively, all of these *abort*s.

To commit a ballot b and externalize its value $b.x$, SCP nodes first accept and confirm b is prepared, then accept and confirm *commit* b. Before the first intact node votes for *commit* b, the prepare step, though federated voting, ensures all intact nodes can eventually confirm b is prepared. When an intact node v accepts *commit* b, it means $b.x$ will eventually be chosen. However, as discussed in Section 5.4.1, v must confirm *commit* before acting on it in case v is befouled.

6.2.1 Concrete ballot protocol

Figure 16 illustrates the per-slot state maintained by each node. A node v stores: its current ballot b; the two most recent incompatible ballots it has prepared (p, p'); the lowest ballot c, if any, it has voted to *commit* (or confirm *commit* in later phases) and for which it has not subsequently accepted an *abort*; the highest ballot P confirmed prepared; the latest message received from each node (M); and a phase φ. Ballots b, p, p', and P are non-decreasing within a phase. In addition, if

PREPARE v i b p p' n_c n_P D

This is a message from node v about slot i. D specifies $\mathbf{Q}(v)$. The other fields reflect v's state, where $n_P = P.n$ and $n_c = c.n$. This concrete message encodes a host of conceptual statements, as follows:

- $\{\, \mathit{abort}\ b' \vee \text{accept}(\mathit{abort}\ b') \mid b' \lessapprox\!\!\!\!/\ b \,\}$ (a vote to prepare b)
- $\{\, \text{accept}(\mathit{abort}\ b') \mid b' \lessapprox\!\!\!\!/\ p \,\}$ (a vote to confirm p is prepared)
- $\{\, \text{accept}(\mathit{abort}\ b') \mid b' \lessapprox\!\!\!\!/\ p' \,\}$ (a vote to confirm p' is prepared)
- $\{\, \mathit{commit}\ \langle n, b.x\rangle \mid 0 \neq n_c \leq n \leq n_P \,\}$ (a vote to commit $c, \ldots, P$)

CONFIRM v i n_p c n_P D

Sent by v to externalize $c.x$ for slot i. Implies $b = \langle \infty, c.x\rangle$, $p = \langle n_p, c.x\rangle$, and $P = \langle n_P, b.x\rangle$ in v's state. For convenience, we also say $p' = \mathbf{0}$ (as older prepared ballots are now irrelevant). D specifies $\mathbf{Q}(v)$ as above. Encodes:

- Everything implied by PREPARE v i b p $\mathbf{0}$ $c.n$ ∞ D
- $\{\, \text{accept}(\mathit{commit}\ b') \mid c \lesssim b' \lesssim P \,\}$ (a vote to confirm *commit* $c, \ldots, P$)

EXTERNALIZE v i c n_P

After v confirms *commit* c for slot i and externalizes value $c.x$, this message helps other nodes externalize it. Implies $b = p = \langle \infty, c.x\rangle$, $P = \langle n_P, c.x\rangle$, and for convenience we say $p' = \mathbf{0}$, and $D = \{\{v\}\}$. Encodes:

- Everything implied by CONFIRM v i ∞ c n_P $\{\{v\}\}$

Figure 17: Messages in SCP's ballot protocol

$c \neq \mathbf{0}$—meaning v may have participated in ratifying *commit* c—code must ensure $c \lesssim P \lesssim b$. This invariant guarantees a node can always vote to prepare the current ballot b.

Figure 17 shows the protocol messages. Note "$a \vee \text{accept}(a)$" is what each node must assert for a quorum to accept a under condition 1 of the definition of *accept*. Each node initializes its slot state by setting $b \leftarrow \langle 1, \text{combine}(Z)\rangle$, $p \leftarrow \mathbf{0}$, $p' \leftarrow \mathbf{0}$, $P \leftarrow \mathbf{0}$, $c \leftarrow \mathbf{0}$, $M \leftarrow \emptyset$, and $\varphi \leftarrow$ PREPARE. Nodes then repeatedly exchange messages with peers, sending whichever message is indicated by φ. Upon adding a newly received message to M, a node v updates its state as follows:

1. If $\varphi = $ PREPARE and the received message lets v accept new ballots as prepared, update p and p'. Afterwards, if $c \neq \mathbf{0}$ and $p \underset{\nsim}{>} P$ or $p' \underset{\nsim}{>} P$, set $c \leftarrow \mathbf{0}$.

2. If $\varphi = $ PREPARE and v confirms new ballots prepared, increase P. Afterwards, if $c = \mathbf{0}$, $P \geq b$, and neither $p \underset{\nsim}{>} P$ nor $p' \underset{\nsim}{>} P$, then set $c \leftarrow P$ and $b \leftarrow P$ (though usually $b = P$ already).

3. If $\varphi = $ PREPARE and v accepts *commit* for one or more compatible ballots, set c to the lowest such ballot, P to the highest ballot such that v accepts all $\{\mathit{commit}\ b' \mid c \lesssim b' \lesssim P\}$, $b \leftarrow \langle \infty, c.x \rangle$, and $\varphi \leftarrow $ CONFIRM.

4. If $\varphi = $ CONFIRM and the received message lets v accept new ballots prepared, raise p to the highest accepted prepared ballot such that $p \sim c$.

5. If $\varphi = $ CONFIRM and v accepts more compatible *commit* messages, raise P to the highest ballot such that v accepts all $\{\mathit{commit}\ b' \mid c \lesssim b' \lesssim P\}$.

6. If $\varphi = $ CONFIRM and v confirms *commit* c' for any c', set c and P to the lowest and highest such ballots, set $\varphi \leftarrow $ EXTERNALIZE, externalize $c.x$, and terminate.

While $c = \mathbf{0}$, the above protocol implements federated voting to confirm b is prepared. Once $c \neq \mathbf{0}$, the protocol implements federated voting on *commit* c—actually, the range of compatible ballots between c and P. For the confirmation phase, once a well-behaved node accepts *commit* c, the node never accepts, and hence never attempts to confirm, *commit* c' for any $c' \nsim c$. Hence, intuitively, once a *commit* is confirmed, its value is safe to externalize so long as nodes enjoy quorum intersection.

All messages from a node are totally ordered by $\langle \varphi, b, p, p', P \rangle$, with φ the most significant and P the least significant field. All PREPARE messages precede all CONFIRM messages, which in turn precede the single EXTERNALIZE message for a given slot. PREPARE messages contain these four fields explicitly, while CONFIRM and EXTERNALIZE contain the values described in Figure 17. The ordering makes it possible to ensure M contains only the latest ballot from each node without relying on timing to order the messages, since the network may re-order messages.

A few details of the protocol merit explanation. The statements implied by PREPARE of the form "*abort* b' $\vee$ accept(*abort* b')" do not specify whether v is voting for or confirming *abort* b'. The distinction is unimportant for the definition of *accept*. Glossing over the distinction allows v to forget about old ballots it voted to commit (and hence cannot vote to abort), so long as it accepted an *abort* message for them.

Both p and p' are needed in order to ensure that nodes converge on P, as Theorem 10 requires that nodes rebroadcast what they have accepted. It follows from the definition of *prepare* that the two highest incompatible ballots a node has accepted as prepared subsume all ballots the node has accepted as prepared.

At the time v sends an EXTERNALIZE message, it has actually accepted an entire range of commit statements $\{\, commit\ b' \mid b' \gtrsim c \,\}$. However, rather than simply set $P.n = \infty$ in the implicit CONFIRM message, thereby asserting accept($commit\ b'$) for every $b' \gtrsim c$, v sets P to assert acceptance of only the ballots it confirmed committed. These suffice because once a single intact node confirms *commit c*, Theorem 10 tells us all intact nodes eventually will. Focusing on confirmed ballots has the added benefit that an EXTERNALIZE message only asserts things v has already ratified, making $\mathbf{Q}(v)$ irrelevant. This means a single, static EXTERNALIZE message is useful for nodes to catch up arbitrarily far in the future, even if quorum slices have changed significantly in the meantime.

Only one RPC is needed to exchange ballot messages. The argument is the sender's latest message and the return value is the receiver's latest message. As with NOMINATE, if D or the values x in ballots are cryptographic hashes, then a separate RPC is needed to retrieve uncached hash preimages.

6.2.2 Ballot selection

If all nodes set b to the same ballot, then steps 1 to 6 on pages 32–33 are sufficient to agree on the ballot's value and externalize it. However, the nomination protocol may not yet have produced identical values everywhere before the ballot protocol starts, in which case nodes may fail to commit the first ballot they try. If a ballot fails or takes long enough that it may fail because of unresponsive nodes, then a node must increment its ballot counter to try again with a higher ballot.

When moving to a new ballot, node v sets $b_v \leftarrow \langle b_v.n+1, z\rangle$, where z is determined as follows: if $P \neq \mathbf{0}$, then $z = P.x$. Otherwise, z is the composite value combine(Z), where Z are the candidate values described in Section 6.1.1. Since c is always initialized from $P = b$, prioritizing $P.x$ ensures the invariant at each node that if $c \neq \mathbf{0}$ then $c \lesssim P \lesssim b$.

For convenience, we identify fields belonging to particular nodes and messages with subscripts. If v is a node, then let $b_v, p_v, p'_v \ldots$ denote the value of $b, p, p', \ldots$ in node v's state as described in Figure 16. We also let z_v be the value for v's next ballot as described above—i.e., z_v is $P_v.x$ if $P \neq \mathbf{0}$ and v's composite value otherwise. Similarly, let b_m, p_m, p'_m, P_m, c_m denote the corresponding fields implied by a network message m as described in Figure 17.

Definition (self-validating)**.** A set of messages $S \subseteq M_v$ is *self-validating* at node v when v is one of the senders in S and the set of all senders in S is a quorum under the declared quorum sets in D fields of messages.

A node v should give up on the current ballot b_v and proceed to a higher ballot when $\varphi = \text{PREPARE}$ and one of the following conditions holds:

1. M_v contains a set S of messages from a v-blocking set of senders such that $\forall m \in S, b_v.n < b_m.n$ and either $b_m.n \neq \infty$ or $P_v \lesssim c_m$ (meaning v's ballot counter has fallen behind), or

2. A timeout has expired since M_v first contained a self-validating set S in which $\forall m \in S$, $b_v.n \leq b_m.n$ (meaning b_v will likely not commit, and v hopes it has received enough messages to pick a better new ballot).

For the protocol to guarantee termination, the timeout in condition 2 must eventually be long enough for all intact nodes to exchange several rounds of messages. To achieve this in a concrete implementation without predicting network latencies, the timeout should be set to increase with $b_v.n$. Note also that when node v has fallen behind, rather than iterate through the tests for each counter, it can simply increase $b_v.n$ to the lowest value such that condition 1 is not true.

6.3 Correctness

An SCP node cannot pledge to confirm *commit b* until it has pledged to confirm *abort* for all lower-numbered incompatible ballots. Because a well-behaved node cannot accept (and hence pledge to confirm) contradictory statements, this means that for a given $\langle \mathbf{V}, \mathbf{Q} \rangle$, Theorem 5 ensures a set S of well-behaved nodes cannot externalize contradictory values so long as S enjoys quorum intersection despite $\mathbf{V} \setminus S$. This safety holds if $\mathbf{V}$ and $\mathbf{Q}$ change only between slots, but what if they change mid-slot (for instance, in reaction to node crashes)? To reason about safety under reconfiguration, we conservatively join all old and new quorum slice sets, reflecting the fact that nodes may make decisions based on a combination of messages from different configuration eras. Hence, to be very conservative we would say a node is intact only if it is intact in every configuration used by the current slot. But we can relax this and say a node is intact if it is intact in the most recent configuration and in no previous configuration received messages from a unanimously ill-behaved v-blocking set.

Theorem 12. *Let* $\langle \mathbf{V}_1, \mathbf{Q}_1 \rangle, \ldots, \langle \mathbf{V}_k, \mathbf{Q}_k \rangle$ *be the set of configurations an FBAS has experienced during agreement on a single slot. Let* $\mathbf{V} = \mathbf{V}_1 \cup \cdots \cup \mathbf{V}_k$ *and* $\mathbf{Q}(v) = \{\, q \mid \exists j, v \in \mathbf{V}_j \ \wedge\ q \in \mathbf{Q}_j(v) \,\}$. *Let* $B \subseteq \mathbf{V}$ *be a set such that all* B *contains all ill-behaved nodes that have sent illegal messages, though* $\mathbf{V} \setminus B$ *may still contain crashed (unresponsive) nodes. Suppose* $v_1 \notin B$ *externalizes* x_1*, and* $v_2 \notin B$ *externalizes* x_2*. If* $\langle \mathbf{V}, \mathbf{Q} \rangle^B$ *enjoys quorum intersection, then* $x_1 = x_2$*.*

Proof. For v_1 to externalize x_1, it must have ratified accept(*commit* $\langle n_1, x_1 \rangle$) in collaboration with a pseudo-quorum $U_1 \subseteq \mathbf{V}$. We say pseudo-quorum because U_1 might not be a quorum in $\langle \mathbf{V}_j, \mathbf{Q}_j \rangle$ for any particular j, as ratification may have involved messages spanning multiple configurations. Nonetheless, for ratification to succeed $\forall v \in U_1, \exists j, \exists q \in \mathbf{Q}_j(v)$ such that $q \subseteq U_1$. It follows from the construction of $\mathbf{Q}$ that $q \in \mathbf{Q}(v)$. Hence U_1 is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle$. By a similar argument a pseudo-quorum U_2 must have ratified accept(*commit* $\langle n_2, x_2 \rangle$), and U_2 must be a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle$. By quorum intersection of $\langle \mathbf{V}, \mathbf{Q} \rangle^B$, there must exists some $v \in \mathbf{V} \setminus B$ such that $v \in U_1 \cap U_2$. By assumption, such a $v \notin B$ could not claim to accept incompatible ballots. Since v confirmed accepting *commit* for ballots with both x_1 and x_2, it must be that $x_1 = x_2$. □

For liveness of a node v, we care about several things when an FBAS has undergone a series of reconfigurations $\langle \mathbf{V}_1, \mathbf{Q}_1 \rangle, \ldots, \langle \mathbf{V}_k, \mathbf{Q}_k \rangle$ within a single slot. First, the safety prerequisites of Theorem 12 must hold for v and the set of nodes v cares about, since violating safety undermines liveness and Theorem 10 requires quorum intersection. Second, the set of ill-behaved nodes in the latest state, $\langle \mathbf{V}_k, \mathbf{Q}_k \rangle$, must not be v-blocking, as this could deny v a quorum and prevent it from ratifying statements. Finally, v's state cannot have been poisoned by a v-blocking set falsely claiming to accept a statement in one of $\langle \mathbf{V}_1, \mathbf{Q}_1 \rangle, \ldots,$ $\langle \mathbf{V}_{k-1}, \mathbf{Q}_{k-1} \rangle$. To summarize, then, we consider v intact when the following conditions hold. First, the union of past configurations for the current slot must have quorum intersection when ill-behaved nodes are deleted. Second, v must be intact in the latest view $\langle \mathbf{V}_k, \mathbf{Q}_k \rangle$ by the previous criteria for static configurations. Finally, v must never have received messages from a unanimously befouled v-blocking set in a previous configuration $\langle \mathbf{V}_1, \mathbf{Q}_1 \rangle, \ldots, \langle \mathbf{V}_{k-1}, \mathbf{Q}_{k-1} \rangle$.

Theorem 13. *In an FBAS with quorum intersection, if an intact node with ballot counter* $b.n = n \neq \infty$ *starts the timer in condition 2 on the previous page, then all intact nodes can raise their ballot counters to at least* n *without the need for any timers to expire.*

Proof. A node will only start the timer when it sees messages from a quorum U, where every member of U has a ballot counter n or higher. If U contains every intact node, we are done. Otherwise, let $U' \neq \emptyset$ be the intact subset of U. By quorum intersection despite the DSet of befouled nodes, there must be at least one intact node $v \notin U$ such that U' is v-blocking. (The proof of Theorem 10 shows a contradiction otherwise.)

If $\forall v' \in U'$, $\varphi_{v'} = \text{PREPARE}$, meaning $\forall v' \in U', b_{v'}.n \neq \infty$, then by condition 1 on page 35, v will simply raise b_v until $b_v.n \geq n$. Otherwise, suppose $\exists v' \in U'$ such that $\varphi_{v'} \neq \text{PREPARE}$. This means v' (or another intact node) confirmed $c_{v'}$ prepared and accepted *commit* $c_{v'}$. Hence, by Theorem 10, v too will confirm $c_{v'}$ is prepared when it receives messages from enough other intact nodes. Moreover, by Theorem 8, no intact node will ever accept as prepared any ballot $b' \gtrsim\!\!\!\!\!/\; c_{v'}$. This means once $P_v \gtrsim c_{v'}$ holds, it will continue to hold in perpetuity, so again by condition 1, v will increase b_v until $b_v.n \geq n$.

Now that $b_v.n \geq n$, if there are any intact nodes remaining with a lower ballot counter, the process described in the previous two paragraphs will raise the counter of at least one such node, again requiring no timeouts. The process will repeat until every intact node has ballot counter $\geq n$. □

Theorem 14. *In an FBAS with quorum intersection and a long enough timeout for intact nodes to exchange sufficient messages, all intact nodes will eventually externalize a value.*

Proof. By Theorem 11, all intact nodes will eventually have compatible sets Z of candidate values. Assume this point has passed and every intact node has the same composite value $z = \text{combine}(Z)$. If no intact node ever confirms any ballot b prepared without $b.x = z$, then all new ballots of intact nodes will have value z. Let $b = \langle n, z \rangle$ be the highest ballot at any intact node once the highest ballot contains z. Eventually other nodes will time out and catch up to b, until a quorum catches up and a timer is set. If the timer is sufficiently long, then by Theorem 13 other nodes will catch up and they will be able to complete the protocol, confirming *commit* b.

Otherwise, given long enough timeouts, by Theorem 10 eventually intact nodes will converge on the same P. At this point, the same argument as the previous paragraph applies if we substitute $z = P.x$ for $z = \text{combine}(Z)$. □

It is worth noting that Theorem 14 is not true if we remove the requirement about sufficient timeouts. In particular, there is a vulnerable period between when a set of intact nodes has cast sufficient votes to confirm a ballot prepared and the point at which nodes learn the ballot is indeed confirmed prepared. Given sufficient message delays, nodes can potentially alternate between different values of $P.x$—i.e., by the time anyone realizes P is confirmed prepared, everyone has moved on and voted to confirm $P' \gtrsim P$ is prepared. By the famous FLP impossibility result [21], some kind of perpetual preemption scenario along these lines is inevitable. Hence, the best we can hope for is liveness under a partial synchrony [18] assumption, which is precisely what Theorem 14 assures us.

7 Limitations

SCP can only guarantee safety when nodes choose adequate quorum slices. Section 3.2 discusses why we can reasonably expect them to do so. Nonetheless, when security depends upon a user-configurable parameter, there is always the possibility people will set it wrong.

Even when people set quorum slices correctly and SCP guarantees safety, safety alone does not rule out other security issues that may arise in a federated system. For example, in a financial market, widely trusted nodes could leverage their position in the network to gain information with which to engage in front running or other unethical activities.

Byzantine nodes may attempt to filter transactions on the input side of SCP while otherwise producing the correct output. If well-behaved nodes accept all transactions, the combine function takes the union of transactions, and there are intact nodes, then such filtering will eventually fail to block victim transactions with probability 1 but may nonetheless impose delays.

Though SCP's safety is optimal, its performance and communication latency are not. In the common case that nodes have not previously voted to commit ballots incompatible with the current one, it is possible to reduce the number of communication rounds by one. An earlier version of SCP did so, but the protocol description was more complex. First, it required nodes to cache and retransmit signed messages previously sent by failed nodes. Second, it was no longer possible to gloss over the distinction between votes and confirmations of *abort* statements in PREPARE messages, so nodes had to send around potentially unbounded lists of exceptions to their *abort* votes.

Unfortunately, changing slices mid-slot to accommodate failed nodes is problematic for liveness if a well-behaved node v has ever experienced a wholly malicious and colluding v-blocking set. The good news is that Theorem 12 guarantees safety to any set S of well-behaved nodes enjoying quorum intersection despite

$\mathbf{V} \setminus S$, even when S has befouled members. The bad news is that updating $\mathbf{Q}$ may be insufficient to unblock v if v was tricked into voting to confirm a bad *commit* message. In such a situation, v needs to disavow past votes, which it can do only by rejoining the system under a new node name $v' \neq v$. There may exist a way to automate such recovery, such as having other nodes recognize reincarnated nodes and automatically replace v with v' in slices.

The FBA model requires continuity of participants over time. Should all nodes simultaneously and permanently leave, restarting consensus would require central coordination or human-level agreement. By contrast, a proof-of-work system such as Bitcoin could undergo sudden complete turnover yet continue to operate with little human intervention. On the other hand, if nodes do return, an FBAS can recover from an arbitrarily long outage, while a proof-of-work scheme would face the possibility of an attacker working on a fork during the outage.

An intriguing possibility is to leverage SCP to mediate tussles [13] by voting on changes to configuration parameters or upgrades to an application protocol. One way to do this is to nominate special messages that update parameters. Candidate values could then consist of both a set of values and a set of parameter updates. A big limitation of this approach is that a set of malicious nodes large enough to deny the system a quorum but not large enough to undermine safety could nonetheless trigger arbitrary configuration changes (by lying and putting configuration changes in Y that were never ratified). It remains an open question how to vote on parameter changes in a way that requires the consent of a full quorum but also never jeopardizes liveness.

8 Summary

Byzantine agreement has long enabled distributed systems to achieve consensus with efficiency, standard cryptographic security, and flexibility in designating trusted participants. More recently, Bitcoin introduced the revolutionary notion of decentralized consensus, leading to many new systems and research challenges. This paper introduces federated Byzantine agreement (FBA), a model for achieving decentralized consensus while preserving the traditional benefits of Byzantine agreement. The key distinction between FBA and prior Byzantine agreement systems is that FBA forms quorums from participants' individual trust decisions, allowing an organic growth model similar to that of the Internet. The Stellar Consensus Protocol (SCP) is a construction for FBA that achieves optimal resilience against ill-behaved participants.

Acknowledgments

Jed McCaleb inspired this work and provided feedback, terminology suggestions, and help thinking through numerous conjectures. Jessica Collier collaborated on writing the paper. Stan Polu created the first implementation of SCP and provided invaluable corrections, suggestions, simplifications, and feedback in the process. Jelle van den Hooff provided the key idea to restructure the paper around quorum intersection and federated voting, as well as other crucial suggestions for terminology, organization, and presentation. Nicolas Barry found several bugs in the paper as he implemented the protocol, as well as identifying necessary clarifications. Ken Birman, Bekki Bolthouse, Joseph Bonneau, Mike Hamburg, Graydon Hoare, Joyce Kim, Tim Makarios, Mark Moir, Robert Morris, Lucas Ryan, and Katherine Tom slogged through drafts of the paper, identifying errors and sources of confusion as well as providing helpful suggestions. Eva Gantz provided helpful motivation and references. Winnie Lim provided guidance on figures. The reddit community and Tahoe-LAFS group pointed out a censorship weakness in an earlier version of SCP, leading to the improved nomination protocol. Finally, the author would like to thank the whole Stellar team for their support, feedback, and encouragement.

Disclaimer

Professor Mazières's contribution to this publication was as a paid consultant, and was not part of his Stanford University duties or responsibilities.

References

[1] Fraudulent digital certificates could allow spoofing. Microsoft Security Advisory 2798897, January 2013. `https://technet.microsoft.com/en-us/library/security/2798897.aspx`.

[2] Eduardo A. Alchieri, Alysson Neves Bessani, Joni Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, pages 22–40, December 2008.

[3] James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. In *Proceedings of the 29th Symposium on Principles of Distributed Computing*, pages 460–467, July 2010.

[4] Rachel Banning-Lover. Boatfuls of cash: how do you get money into fragile states? February 2015. `http://www.theguardian.com/global-development-professionals-network/2015/feb/19/boatfuls-of-cash-how-do-you-get-money-into-fragile-states`.

[5] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393, November 2014.

[6] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the 2nd Symposium on Principles of Distributed Computing*, pages 27–30, June 1983.

[7] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research perspectives and challenges for bitcoin and cryptocurrencies. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, May 2015.

[8] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, October 1985.

[9] Danny Bradbury. Feathercoin hit by massive attack, June 2013. `http://www.coindesk.com/feathercoin-hit-by-massive-attack/`.

[10] Vitalik Buterin. Slasher: A punitive proof-of-stake algorithm, January 2014. `https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/`.

[11] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186, February 1999.

[12] CGAP. Making money transfers work for microfinance institutions, March 2008. `http://www.cgap.org/sites/default/files/CGAP-Technical-Guide-Making-Money-Transfers-Work-for-Microfinance-Institutions-A-Technical-Guide-to-Developing-and-Delivering-Money-Transfers-Mar-2008.pdf`.

[13] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: Defining tomorrow's internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, June 2005.

[14] crazyearner. Terracoin attack over 1.2TH attack confirmd [sic], July 2013. `https://bitcointalk.org/index.php?topic=261986.0`.

[15] Kourosh Davarpanah, Dan Kaufman, and Ophelie Pubellier. NeuCoin: the first secure, cost-efficient and decentralized cryptocurrency, March 2015. `http://www.neucoin.org/en/whitepaper/download`.

[16] Asli Demirguc-Kunt, Leora Klapper, Dorothe Singer, and Peter Van Oudheusden. The global findex database 2014 measuring financial inclusion around the world. Policy Research Working Paper 7255, World Bank, April 2015. `http://www-wds.worldbank.org/external/default/WDSContentServer/WDSP/IB/2015/04/15/090224b082dca3aa/1_0/Rendered/PDF/The0Global0Fin0ion0around0the0world.pdf`.

[17] John R. Douceur. The Sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, March 2002.

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[19] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, August 1992.

[20] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable, November 2013. `http://arxiv.org/abs/1311.0243`.

[21] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[22] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917, October 2012.

[23] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perring, Collin Jackson, and Virgil Gligor. Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 679–690, May 2013.

[24] Sunny King and Scott Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake, August 2012. http://peercoin.net/assets/paper/peercoin-paper.pdf.

[25] Jae Kwon. Tendermint: Consensus without mining, 2014. http://tendermint.com/docs/tendermint.pdf.

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

[27] Leslie Lamport. Brief announcement: Leaderless byzantine paxos. In *Proceedings of the 25th International Conference on Distributed Computing*, pages 141–142, September 2011.

[28] Leslie Lamport. Byzantizing paxos by refinement. In *Proceedings of the 25th International Conference on Distributed Computing*, pages 211–224, September 2011.

[29] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401, July 1982.

[30] Adam Langley. Maintaining digital certificate security, March 2015. http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html.

[31] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. RFC 6962, Internet Engineering Task Force (IETF), June 2013. http://tools.ietf.org/html/rfc6962.

[32] Jinyuan Li and David Mazières. Beyond one-third faulty replicas in Byzantine fault tolerant systems. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation*, pages 131–144, April 2007.

[33] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Michael J. Freedman, and Edward W. Felten. CONIKS: A privacy-preserving consistent key service for secure end-to-end communication. Cryptology ePrint Archive, Report 2014/1004, December 2014. http://eprint.iacr.org/2014/1004.

[34] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://bitcoin.org/bitcoin.pdf.

[35] William B. Norton. The art of peering: The peering playbook, August 2010. http://drpeering.net/white-papers/Art-Of-Peering-The-Peering-Playbook.html.

[36] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *Irish Signals and Systems Conference*, pages 280–285, Limerick, Ireland, June 2014.

[37] National Institute of Standards and Technology. Secure hash standard (SHS). Federal Information Processing Standards Publication 180-4, March 2012. `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`.

[38] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the 7th Symposium on Principles of Distributed Computing*, pages 8–17, August 1988.

[39] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pages 305–319, June 2014.

[40] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[41] Claire Provost. Why do africans pay the most to send money home? January 2013. `http://www.theguardian.com/global-development/2013/jan/30/africans-pay-most-send-money`.

[42] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple protocol consensus algorithm, 2014. `https://ripple.com/files/ripple_consensus_whitepaper.pdf`.

[43] Robbert van Renesse, Nicolas Schiper, and Fred B. Schneider. Vive la différence: Paxos vs. Viewstamped Replication vs. Zab. *IEEE Transactions on Dependable and Secure Computing*, September 2014.

A Glossary of notation

Notation	Name	Definition
iff		An abbreviation of "if and only if"
$f(x)$	function	The result of calculating function f on argument x
$\bar{a}$	complement	An overbar connotes the opposite, i.e., $\bar{a}$ is the opposite of a.
$\langle a_1, \ldots, a_n \rangle$	tuple	A structure (compound value) with field values $a_1, \ldots, a_n$
$A \wedge B$	logical and	Both A and B are true.
$A \vee B$	logical or	At least one, possibly both, of A and B are true.
$\exists e, C(e)$	there exists	There is at least one value e for which condition $C(e)$ is true.
$\forall e, C(e)$	for all	$C(e)$ is true of every value e.
$\{a, b, \ldots\}$	set	A set containing the listed elements $(a, b, \ldots)$
$\{\, e \mid C(e) \,\}$	set-builder	The set of all elements e for which $C(e)$ is true
$\emptyset$	empty set	The set containing no elements
$\lvert S \rvert$	cardinality	The number of elements in set S
$e \in S$	element of	Element e is a member of set S.
$A \subseteq B$	subset	Every member of set A is also a member of set B.
$A \subsetneq B$	strict subset	$A \subseteq B$ and $A \neq B$.
2^A	powerset	The set of sets containing every possible combination of members of A, i.e., $2^A = \{\, B \mid B \subseteq A \,\}$
$A \cup B$	union	The set containing all elements that are members of A or members of B, i.e., $A \cup B = \{\, e \mid e \in A \vee e \in B \,\}$
$A \cap B$	intersection	The set containing all elements that are members of both A and B, i.e., $A \cap B = \{\, e \mid e \in A \wedge e \in B \,\}$
$A \setminus B$	set difference	The set containing every element of A that is not a member of B, i.e., $A \setminus B = \{\, e \mid e \in A \wedge e \notin B \,\}$
$/$	not	Negates a symbol's meaning. E.g., $e \notin A$ means $e \in A$ is false, while $\nexists e, C(e)$ means no e exists such that $C(e)$ is true.