Deconstructing Stellar Consensus

Álvaro García-Pérez
IMDEA Software Institute, Madrid, Spain

Maria A. Schett
University College London, United Kingdom

Abstract

Some of the recent blockchain proposals, such as Stellar and Ripple, allow for open membership while using quorum-like structures typical for classical Byzantine consensus with closed membership. This is achieved by constructing quorums in a decentralised way: each participant independently chooses whom to trust, and quorums arise from these individual decisions. Unfortunately, the consensus protocols underlying such blockchains are poorly understood, and their correctness has not been rigorously investigated. In this paper we rigorously prove correct the Stellar Consensus Protocol (SCP), with our proof giving insights into the protocol structure and its use of lower-level abstractions. To this end, we first propose an abstract version of SCP that uses as a black box Stellar's federated voting primitive (analogous to reliable Byzantine broadcast), previously investigated by García-Pérez and Gotsman [7]. The abstract consensus protocol highlights a modular structure in Stellar and can be proved correct by reusing the previous results on federated voting. However, it is unsuited for realistic implementations, since its processes maintain infinite state. We thus establish a refinement between the abstract protocol and the concrete SCP that uses only finite state, thereby carrying over the result about the correctness of former to the latter. Our results help establish the theoretical foundations of decentralised blockchains like Stellar and gain confidence in their correctness.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Blockchain, Consensus protocol, Stellar, Byzantine quorum systems

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2019.5

Related Version `http://arxiv.org/abs/1911.05145`

Funding This work was supported by ERC Starting Grant RACCOON, Spanish State Research Agency project BOSCO, and Madrid Regional Government project BLOQUES.

Acknowledgements We thank Ilya Sergey, Giuliano Losa and Alexey Gotsman for their comments.

1 Introduction

Permissioned blockchains are becoming increasingly popular due to the low-energy consumption and hard guarantees they provide on when a transaction can be considered successfully committed. Such blockchains are often based on classical Byzantine fault-tolerant (BFT) consensus protocols, like PBFT [4]. In these protocols consensus is reached once a *quorum* of participants agrees on the same decision. Quorums can be defined as sets containing enough nodes in the system (e.g., $2f + 1$ out of $3f + 1$, assuming at most f failures) or by a more general structure of a Byzantine quorum system (BQS) [12]. Unfortunately, defining quorums in this way requires fixing the number of participants in the system, which prevents decentralisation.

Some of the recent blockchain proposals, such as Stellar [13] and Ripple [15], allow for open membership while using quorum-like structures typical for classical Byzantine consensus with closed membership. This is achieved by constructing quorums in a decentralised way: each protocol participant independently chooses whom to trust, and quorums arise from these individual decisions. In particular, in Stellar trust assumptions are specified using a *federated Byzantine quorum system (FBQS)*, where each participant selects a set of *quorum* *slices* – sets of nodes each of which would convince the participant to accept the validity of a given statement (§2). Quorums are defined as sets of nodes U such that each node in U has some quorum slice fully within U, so that the nodes in a quorum can potentially reach an agreement. Consensus is then implemented by a fairly intricate protocol whose key component is *federated voting* – a protocol similar to Bracha's protocol for reliable Byzantine broadcast [1, 2]. Unfortunately, even though Stellar has been deployed as a functioning blockchain, the structure of the consensus protocol underlying it is poorly understood and its correctness has not been rigorously investigated. In this paper we aim to close this gap, rigorously defining and proving correct the Stellar Consensus Protocol (SCP). Apart from giving more confidence in the correctness of the protocol, our proof is structured in such a way as to give insights into its structure and its use of lower-level abstractions.

In more detail, the guarantees provided by SCP are nontrivial. When different participants in an FBQS choose different slices, only a subset of the participants may take part in a subsystem in which every two quorums intersect in a correct node – a property required for achieving consensus. The system may partition into such subsystems, and SCP will guarantee agreement within each of them. In blockchain terms, the blockchain may fork, but in this case each fork will be internally consistent, a property that is enough for business applications of the Stellar blockchain. The subsystems where agreement is guaranteed are characterised by Mazières et al. [14] through the notion of *intact sets*. Our proof of correctness establishes safety and liveness properties of SCP relative to such intact sets (§3).

As a stepping stone in the proof, we first propose an *abstract version* of SCP that uses as a black box Stellar's federated voting primitive (analogous to reliable Byzantine broadcast) previously investigated by García-Pérez and Gotsman [7] (§5). This abstract formulation allows specifying the protocol concisely and highlights the modular structure present in it. This allows proving the protocol by reusing the previous results on federated voting [7] (reviewed in §4). However, the abstract protocol is unsuited for realistic implementations, since its processes maintain infinite state. To address this, we formulate a realistic version of the protocol – a *concrete SCP* – that uses only finite state. We then prove a refinement between the abstract and concrete SCP, thereby carrying over the result about the correctness of former to the latter (§6).

A subtlety in SCP is that its participants receive information about quorum slices of other participants directly from them. Hence, Byzantine participants may lie to others about their choices of quorum slices, which may cause different participants to disagree on what constitutes a quorum. Our results also cover this realistic case (§7).

Overall, our results help establish the theoretical foundations of decentralised blockchains like Stellar and gain confidence in their correctness. Due to space constraints, proofs are deferred to an extended version of the paper [8].

2 Background: System Model and Federated Byzantine Quorum Systems

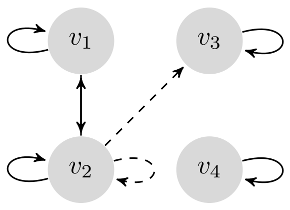
System model. We consider a system consisting of a finite *universe* of *nodes* $\mathbf{V}$ and assume a Byzantine failure model where *faulty* nodes can deviate arbitrarily from their specification. All other nodes are called *correct*. Nodes that are correct, or that only deviate from their specification by stopping execution, are called *honest*. Nodes that deviate from their specification in ways other than stopping are called *malicious*. We assume that any two nodes can communicate over an authenticated perfect link. We assume a partial synchronous network, which guarantees that messages arrive within bounded time after some unknown, finite *global stabilisation time* (*GST*). Each node has a local timer and a timeout service that can be initialised with an arbitrary delay Δ. We assume that after GST the clock skew of correct nodes is bounded, *i.e.*, after GST two correct nodes can only disagree in the duration of a given delay Δ by a bounded margin.

Federated Byzantine quorum systems. Given a finite universe $\mathbf{V}$ of nodes, a *federated Byzantine quorum system* (*FBQS*) [13, 7] is a function $\mathcal{S} : \mathbf{V} \to 2^{2^{\mathbf{V}}} \setminus \{\emptyset\}$ that specifies a non-empty set of *quorum slices* for each node, ranged over by q. We require that a node belongs to all of its own quorum slices: $\forall v \in \mathbf{V}.\, \forall q \in \mathcal{S}(v).\, v \in q$. Quorum slices reflect the trust choices of each node. A non-empty set of nodes $U \subseteq \mathbf{V}$ is a *quorum* in an FBQS $\mathcal{S}$ iff U contains a slice for each member, *i.e.*, $\forall v \in U.\, \exists q \in \mathcal{S}(v).\, q \subseteq U$.

For simplicity, for now we assume that faulty nodes do not equivocate about their quorum slices, so that all the nodes share the same FBQS. In §7 we consider the more realistic *subjective FBQS* [7], where malicious nodes may lie about their slices and different nodes have different views on the FBQS. There we also lift the results on the subsequent sections of the paper to subjective FBQSes.

▶ **Example 1.** Consider a universe $\mathbf{V}$ with $3f+1$ nodes, and consider the FBQS $\mathcal{S}$ where for every node $v \in \mathbf{V}$, the set of slices $\mathcal{S}(v)$ consists of every set of $2f+1$ nodes that contains v itself. $\mathcal{S}$ encodes the classical cardinality-based quorum system of $3f+1$ nodes with failure threshold f, since every set of $2f+1$ or more nodes is a quorum.

▶ **Example 2.** Let the universe $\mathbf{V}$ contain four nodes v_1 to v_4, and consider the FBQS $\mathcal{S}$ in the diagram below.



$$\begin{aligned} \mathcal{S}(v_1) &= \{\{v_1, v_2\}\} \\ \mathcal{S}(v_2) &= \{\{v_1, v_2\}, \{v_2, v_3\}\} \\ \mathcal{S}(v_3) &= \{\{v_3\}\} \\ \mathcal{S}(v_4) &= \{\{v_4\}\} \end{aligned}$$

For each node, all the outgoing arrows with the same style determine one slice. Node v_2 has two slices, determined by the solid and dashed arrow styles respectively. The rest of the nodes have one slice. $\mathcal{S}$ has the following set of quorums $\mathcal{Q} =$

$$\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3\}, \{v_4\}, \{v_1, v_2, v_3\}, \{v_3, v_4\}, \{v_1, v_2, v_4\}, \{v_2, v_3, v_4\}, \{v_1, v_2, v_3, v_4\}\}.$$

A consensus protocol that runs on top of an FBQS may not guarantee global agreement, because when nodes choose slices independently, only a subset of the nodes may take part in a subsystem in which every two quorums intersect in at least one correct node – a basic requirement of a Byzantine quorum system [12] to ensure agreement. To formalise which parts of the system may reach agreement internally, we borrow the notions of *intertwined nodes* and of *intact set* from [14]. Two nodes v_1 and v_2 are *intertwined* iff they are correct and every quorum containing v_1 intersects every quorum containing v_2 in at least one correct node. Consider an FBQS $\mathcal{S}$ and a set of nodes I. The *projection* $\mathcal{S}|_I$ of $\mathcal{S}$ to I is the FBQS over universe I given by $\mathcal{S}|_I(v) = \{q \cap I \mid q \in \mathcal{S}(v)\}$. For a given set of faulty nodes, a set I is an *intact set* iff I is a quorum in $\mathcal{S}$ and every member of I is intertwined with each other in the projected FBQS $\mathcal{S}|_I$. The intact sets characterise those sets of nodes that can reach consensus, which we later show using the following auxiliary result.

▶ **Lemma 3.** *Let $\mathcal{S}$ be an FBQS and assume some set of faulty nodes. Let I be an intact set in $\mathcal{S}$ and consider any two quorums U_1 and U_2 in $\mathcal{S}$ such that $U_1 \cap I \neq \emptyset$ and $U_2 \cap I \neq \emptyset$. Then the intersection $U_1 \cap U_2$ contains some node in I.*

The maximal intact sets are disjoint with each other:

▶ **Lemma 4.** *Let $\mathcal{S}$ be an FBQS and assume some set of faulty nodes. Let I_1 and I_2 be two intact sets in $\mathcal{S}$. If $I_1 \cap I_2 \neq \emptyset$ then $I_1 \cup I_2$ is an intact set in $\mathcal{S}$.*

In SCP the system may split into different partitions – *i.e.*, the maximal intact sets – that may be inconsistent with each other, but which constitute independent systems each of which can reach consensus.

Consider the $\mathcal{S}$ from Example 1, which encodes the cardinality-based quorum system of $3f + 1$ nodes, and let $f = 1$, so that the universe $\mathbf{V}$ contains four nodes v_1 to v_4. If we assume that node v_3 is faulty, then the set $I = \{v_1, v_2, v_4\}$ is the only maximal intact set: I is a quorum in $\mathcal{S}$, and $\mathcal{S}|_I$ contains the quorums $\{\{v_1, v_2\}, \{v_2, v_4\}, \{v_1, v_4\}, \{v_1, v_2, v_4\}\}$, which enjoy quorum intersection. This ensures that every two nodes in I are intertwined in the projected system $\mathcal{S}|_I$.

Now consider the $\mathcal{S}$ from Example 2. If we assume that node v_3 is faulty, then the sets $I = \{v_1, v_2\}$ and $I' = \{v_4\}$ are the maximal intact sets: I and I' are quorums in $\mathcal{S}$, and the projected systems $\mathcal{S}|_I$ and $\mathcal{S}|_{I'}$ enjoy quorum intersection – $\mathcal{S}_I$ contains quorums $\{v_1, v_2\}$ and $\{v_2\}$, and $\mathcal{S}|_{I'}$ contains quorum $\{v_4\}$ – which ensures that every two nodes in either I or I' are intertwined in the projected systems $\mathcal{S}|_I$ and $\mathcal{S}_{I'}$ respectively. It is easy to check that adding any set of correct nodes to either I or I' results in sets that are not quorums in $\mathcal{S}$, or in projected systems that contain some pairs of nodes that are not intertwined.

3 Specifications

Assume a set $\mathbf{Val}$ of *consensus values*. In the consensus protocols that we study in §5–6, each correct node proposes some $x \in \mathbf{Val}$ through an invocation $\mathsf{propose}(x)$, and each node may decide some $x' \in \mathbf{Val}$ through an indication $\mathsf{decide}(x')$. We consider a variant of the *weak Byzantine consensus* specification in [2] that we call *non-blocking Byzantine consensus for intact sets*, which is defined as follows. Given a maximal intact set I,

(*Integrity*) no correct node decides twice,

(*Agreement for intact sets*) no two nodes in I decide differently,

(*Weak validity for intact sets*) if all nodes are honest and every node proposes x, then no node in I decides a consensus value different from x; furthermore, if all nodes are honest and some node in I decides x, then x was proposed by some node, and

(*Non-blocking for intact sets*) if a node v in I has not yet decided in some run of the protocol, then for every continuation of that run in which all the malicious nodes stop, node v eventually decides some consensus value.

The usual *Weak validity* property of consensus [2] ensures that if all nodes are correct and they propose the same consensus value, then no node can decide a consensus value different from the proposed one; and that if all nodes are correct, then a node can only decide a consensus value proposed by some node. Our *Weak validity for intact sets* above adapts this requirement to the nodes in a maximal intact set, and weakens its condition by assuming that all nodes are honest instead of correct. Notice that if every two quorums intersect our property entails the usual one because a correct node is also honest, and because if all nodes are correct then the maximal intact set is the universe. For instance, this condition holds in the cardinality-based quorum systems $(3f + 1)$.

The consensus protocols we consider in this paper specify the behaviour of SCP's ballot protocol [13, 14] with one of its suggested strategies for managing timeouts (Strategy 1 from [14]). As discussed in [14], in SCP malicious nodes with good network timing could permanently delay the termination of the nodes in an intact set, and thus the protocol does not provide the usual *Termination* guarantee that every correct node eventually decides some consensus value [2]. Instead, we consider the weaker liveness guarantee of *Non-blocking for intact sets*, which we have obtained by adapting the *Non-blocking* property in [16]. *Non-blocking* requires that some continuation of a given run *exists* in which every correct node terminates. Our *Non-blocking for intact sets* adapts this requirement to the nodes in a maximal intact set, and requires that they terminate in *every* continuation of the run in which malicious nodes are stopped. It is easy to check that if every correct node is in an intact set, then *Non-blocking for intact sets* entails *Non-blocking* in [16]. For instance, this condition holds in the cardinality-based quorum systems ($3f + 1$). Besides, if every correct node is honest, then *Non-blocking for intact sets* entails the usual *Termination* property that guarantees that every correct node eventually decides some consensus value.

The *non-blocking Byzantine consensus for intact sets* above entails the *weak Byzantine consensus* specification [2] in the cardinality-based quorum systems ($3f+1$), which guarantees the *Integrity* property above, as well as the usual *Agreement* property that ensures that no two correct nodes decide differently, and the usual *Weak validity* and *Termination* properties that we have recalled in the paragraphs above.

One of the core components of the consensus protocol in §5 is *federating voting* (*FV*) [13, 14]. Assume a set of *voting values* $\mathbf{A}$ that could be disjoint with the set $\mathbf{Val}$ of consensus values (we typically let $\mathbf{A}$ be the set of Booleans $\mathsf{Bool} \equiv \{true, false\}$). FV allows each correct node to vote for some $a \in \mathbf{A}$ through an invocation $\mathsf{vote}(a)$, and each node may deliver some $a' \in \mathbf{A}$ through an indication $\mathsf{deliver}(a')$. The interface of FV is akin to that of consensus, where each node activates itself through the primitive $\mathsf{vote}(a)$. However, FV has weaker liveness guarantees than consensus, which are reminiscent to those of *Byzantine reliable broadcast* from [2] and *weakly reliable Byzantine broadcast* from [7]. Here, we consider a variant of the latter specification that we call *reliable Byzantine voting for intact sets*, which is defined as follows. Given a maximal intact set I,

(*No duplication*) every correct node delivers at most one voting value,
(*Totality for intact sets*) if a node in I delivers a voting value, then every node in I eventually delivers a voting value,
(*Consistency for intertwined nodes*) if two intertwined nodes v and v' deliver a and a' respectively, then $a = a'$, and
(*Validity for intact sets*) if all nodes in I vote for a, then all nodes in I eventually deliver a.

The ability of each node to activate itself independently in the specification above simulates a malicious sender that may send different voting values to each node in the specification of *weakly reliable Byzantine broadcast* from [7].

4 Federated Voting

In this section we recall *federated voting* (*FV*) from [13], which also corresponds to the *Stellar broadcast* considered in [7]. We prove that FV implements the specification of reliable Byzantine voting for intact sets, thereby generalising the results of [7] to the case of multiple intact sets within the system. The consensus protocol that we study in the next section uses multiple instances of FV independent from each other. This is done by letting each node run a distinct process for each instance of FV, which is identified by a *tag* t from some designated set $\mathbf{Tag}$ of tags.

Algorithm 1 below depicts FV over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$. A node v runs a process federated-voting(v, t) for each tag t. The messages exchanged by such a process are also tagged with t, in order to distinguish them from the messages exchanged for instances of FV associated with tags different from t.

Algorithm 1 Federated voting (FV) over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$.

```
1  process federated-voting($v \in \mathbf{V}, t \in \mathbf{Tag}$)
2      voted, ready, delivered $\leftarrow$ false $\in$ Bool;
3      vote($a$)
4          if not voted then
5              voted $\leftarrow$ true;
6              send VOTE($t, a$) to every $v' \in \mathbf{V}$;
7      when received VOTE($t, a$) from every $u \in U$ for some $U \in \mathcal{Q}$ such that
         $v \in U$ and not ready
8          ready $\leftarrow$ true;
9          send READY($t, a$) to every $v' \in \mathbf{V}$;
10     when received READY($t, a$) from every $u \in B$ for some $v$-blocking $B$ and
         not ready
11         ready $\leftarrow$ true;
12         send READY($t, a$) to every $v' \in \mathbf{V}$;
13     when received READY($t, a$) from every $u \in U$ for some $U \in \mathcal{Q}$ such that
         $v \in U$ and not delivered
14         delivered $\leftarrow$ true;
15         trigger deliver($a$);
```

FV adapts *Bracha's protocol* for reliable Byzantine broadcast [1], which works over the cardinality-based quorum systems of $3f + 1$ nodes, to the federated setting of the FBQSs. In FV nodes process each other's messages in several stages, where for each tag t progress is denoted by several Boolean flags (line 2 of Algorithm 1). When a node v votes a for tag t for the first time, the node sends VOTE(t, a) to every node (including itself, for uniformity; lines 3–6). When a node v receives a VOTE(t, a) message from a quorum to which v itself belongs, it sends a READY(t, a) message to every node, signalling its willingness to deliver the value a for tag t (lines 7–9). Note that, for each tag t, two nodes in the same intact set I cannot send READY messages with two different voting values through the rule in lines 7–9. Indeed, this would require two quorums of VOTE messages, each with a node in I, with different voting values for the same tag. But by Lemma 3 these quorums would intersect in a node in I, which is by definition correct and cannot send contradictory VOTE messages for the same tag. When a node v receives the message READY(t, a) from a quorum to which v itself belongs, it delivers a for tag t (lines 13–15).

The exchange of READY messages in the protocol is necessary to establish liveness guarantees. It ensures that, if a node in an intact set I delivers a voting value for some tag, other nodes in I have enough information to also deliver a voting value for the same tag. This relies on the rule in lines 10–12, which uses the notion of *v-blocking set* [13]. Given a node v, a set B is *v-blocking* iff B overlaps each of v's slices, *i.e.*, $\forall q \in \mathcal{S}(v).\ q \cap B \neq \emptyset$. (To illustrate this notion, in Example 1 every set of $f + 1$ nodes is v-blocking for every v, and in

Node v_1	Node v_2	Node v_3	Node v_4
vote($false$)	vote($false$)		vote($true$)
VOTE($t, false$)	VOTE($t, false$)	VOTE($t, false$)	VOTE($t, true$)
READY($t, false$)	READY($t, false$)		
			READY($t, false$)
deliver($false$)	deliver($false$)		deliver($false$)

Figure 1 Execution of the instance of FV for tag t.

Example 2 the set $\{v_1, v_3\}$ is v_2-blocking and the set $\{v_2\}$ is v_1-blocking.) Lines 10–12 allow a node to send a READY(t, a) message even if it previously voted for a different voting value for tag t: this is done if v receives READY(t, a) from each member of a v-blocking set. If v is in an intact set I, the following lemma guarantees that in this case v has received at least one READY(t, a) message from some node in I.

▶ **Lemma 5.** *Let $\mathcal{S}$ be an FBQS and assume a set of faulty nodes. Let I be an intact set in $\mathcal{S}$ and $v \in I$. Then, no v-blocking set B exists such that $B \cap I = \emptyset$.*

By Lemma 5, the first node in I to ever send a READY(t, a) message for a tag t has to do it through the rule in lines 7–9, and hence the value a has been cross-checked by a quorum.

If the condition $v \in U$ in lines 7 and 13 of Algorithm 1 was dropped, this could violate *Agreement for intact sets* as follows. Take the $\mathcal{S}$ from Example 2 and consider a run of FV for some tag t where v_3 is malicious. Node v_3 could respectively send READY(t, a) and READY(t, a') with $a \neq a'$ to correct nodes v_1 and v_2. Since $\{v_3\} \in \mathcal{Q}$, these nodes will respectively deliver a and a' by lines 13–15 of Algorithm 1 without condition $v \in U$.

Our first contribution is to generalise the results of [7] to establish the correctness of FV within each of the maximal intact sets of an FBQS, as captured by Theorem 6 below.

▶ **Theorem 6.** *Let $\mathcal{S}$ be an FBQS and t be a tag. The instance for t of FV over $\mathcal{S}$ satisfies the specification of reliable Byzantine voting for intact sets.*

FV also guarantees the property stated by the following lemma, which helps establish the liveness properties of the consensus protocol that we introduce in §5.

▶ **Lemma 7.** *Let $\mathcal{S}$ be an FBQS and t be a tag. Consider an execution of the instance for t of FV over $\mathcal{S}$. Let I be an intact set in $\mathcal{S}$ and assume that GST has expired. If a node $v \in I$ delivers a voting value then every node in I will deliver a voting value within bounded time.*

We write δ_I for the time that a node in I takes to deliver some voting value after GST and provided that some other node in I already delivered some voting value. The delay δ_I – which is determined by $\mathcal{S}$ and I – is unknown, but Lemma 7 guarantees that it is finite.

▶ **Example 8.** Consider the $\mathcal{S}$ from Example 1, which encodes the cardinality-based quorum system $3f + 1$, and let $f = 1$ such that the universe $\mathbf{V}$ contains four nodes v_1 to v_4. Every set of three or more nodes is a quorum, and every set of two or more nodes is v-blocking for every $v \in \mathbf{V}$. Let us fix a tag t and consider an execution of the instance of FV for tag t where we let the voting values be the Booleans. Assume that nodes v_1, v_2 and v_4 are correct, which constitute the maximal intact set. In the execution, nodes v_1 and v_2 vote *false*, and node v_4 votes *true*. Malicious node v_3 sends the message VOTE$(t, false)$ to every node (highlighted in red) thus helping the correct nodes to deliver *false*.

Figure 1 depicts a possible execution of FV described above, from which a trace can be constructed as follows: all the events in each row may happen concurrently, and any two events in different rows happen in real time, where time increases downwards; in those cells that are tagged with a message, the node sends the message to every node, and in a given cell a node has received all the messages from every node in the rows above it. (These conventions are only for presentational purposes, and should not be mistaken with the *perfectly synchronised round-based model* of [5], which we do not use.) The quorum $\{v_1, v_2, v_3\}$ sends VOTE(t, *false*) to every node, which makes nodes v_1 and v_2 send READY(t, *false*) to every node through lines 7–9 of Algorithm 1. However, there exists not a quorum U such that $v_4 \in U$ and every member of U sends a message VOTE(t, a) with the same Boolean a, and thus node v_4 sends READY(t, *false*) through lines 10–12 of Algorithm 1, only after receiving corresponding ready messages from the v_4-blocking set $\{v_1, v_2\}$. Observe how node v_4 changes its original vote *true* and sends *false* in the READY message. After every correct node receives READY(t, *false*) from the quorum $\{v_1, v_2, v_4\}$, they all deliver *false*.

5 Abstract Stellar Consensus Protocol

In this section we introduce the *abstract SCP* (*ASCP*), which concisely specifies the mechanism of SCP [13, 14] and highlights the modular structure present in it[1]. Like *Paxos* [9], ASCP uses *ballots* – pairs $\langle n, x\rangle$, where $n \in \mathbb{N}^+$ a natural positive *round number* and $x \in \mathbf{Val}$ a *consensus value*. We assume that **Val** is totally ordered, and we consider a special *null ballot* $\langle 0, \perp\rangle$ and let $\mathbf{Ballot} = (\mathbb{N}^+ \times \mathbf{Val}) \cup \{\langle 0, \perp\rangle\}$ be the set of ballots. (We write $b.n$ and $b.x$ respectively for the round and consensus value of ballot b.) The set **Ballot** is totally ordered, where we let $b < b'$ iff either $b.n < b'.n$, or $b.n = b'.n$ and $b.x < b'.x$.

To better convey SCP's mechanism, we let the abstract protocol use FV as a black box where nodes may hold a binary vote on each of the ballots: we let the set of voting values **V** be the set of Booleans and the set of tags **Tag** be the set of ballots, and let the protocol consider a separate instance of FV for each ballot. A node voting for a Boolean a for a ballot b that carries the consensus value $b.x$ encodes the aim to either *abort* the ballot (when $a = \mathit{false}$) or to *commit* it (when $a = \mathit{true}$) thus deciding the consensus value $b.x$. From now on we will unambiguously use "Booleans", "ballots" and "values" instead of "voting values", "tags" and "consensus values", respectively.

We have dubbed ASCP "abstract" because, although it specifies the protocol concisely, it is unsuited for realistic implementations. On the one hand, each node v maintains infinite state, because it stores a process federated-voting(v, b) for each of the infinitely many ballots b in the array ballots (line 2 of Algorithm 2). On the other hand, each node v may need to send or receive an infinite number of messages in order to progress (lines 6, 8, 15 and 21 of Algorithm 2, which are explained in the detailed description of ASCP below). This is done by assuming a *batched network semantics* (*BNS*) in which the network exchanges *batches*, which are (possibly infinite) sequences of messages, instead of exchanging individual messages: the sequence of messages to be sent by a node when processing an event is batched per recipient, and each batch is sent at once after the atomic processing of the event; once a batch is received, the recipient node atomically processes all the messages in the batch in sequential order. By convention, we let the statement **forall** in lines 7 and 21 of Algorithm 2 consider the ballots b' in ascending ballot order. In §6 we introduce a "concrete" version of SCP that is amenable to implementation, since nodes in it maintain finite state and exchange a finite number of messages; however, this version does not use FV as a black box.

[1] More precisely, in this paper we focus on Stellar's core *balloting* protocol, which aims to achieve consensus. We abstract from Stellar's *nomination* protocol – which tries to converge (best-effort) on a value to propose – by assuming arbitrary proposals to consensus.

Algorithm 2 Abstract SCP (ASCP) over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$.

```
1  process abstract-consensus($v \in \mathbf{V}$)
2      ballots ← [new process federated-voting($v, b$)]$^{b \in \mathbf{Ballot}}$;
3      candidate, prepared ← $\langle 0, \bot \rangle \in \mathbf{Ballot}$;
4      round ← $0 \in \mathbb{N}^+ \cup \{0\}$;
5      propose($x$)
6          candidate ← $\langle 1, x \rangle$;
7          for all $b' \lesssim\!\!\!\!/\,$ candidate do ballots[$b'$].vote($false$);
8      when triggered ballots[$b'$].deliver($false$) for every $b' \lesssim\!\!\!\!/\, b$ and prepared $< b$
9          prepared ← $b$;
10         if candidate ≤ prepared then
11             candidate ← prepared;
12             ballots[candidate].vote($true$);
13     when triggered ballots[$b$].deliver($true$)
14         trigger decide($b.x$);
15     when exists $U \in \mathcal{Q}$ such that $v \in U$ and for each $u \in U$ exist
        $\mathtt{M}_u \in \{\mathtt{VOTE}, \mathtt{READY}\}$ and $b_u \in \mathbf{Ballot}$ such that round $< b_u.n$ and either
        received $\mathtt{M}_u(b_u, true)$ from $u$ or received $\mathtt{M}_u(b', false)$ from $u$ for every
        $b' \in [z_u, b_u)$ with $z_u < b_u$
16         round ← $\min\{b_u.n \mid u \in U\}$;
17         start-timer($F$(round));
18     when triggered timeout
19         if prepared = $\langle 0, \bot \rangle$ then candidate ← $\langle$round + 1, candidate.$x\rangle$;
20         else candidate ← $\langle$round + 1, prepared.$x\rangle$;
21         for all $b' \lesssim\!\!\!\!/\,$ candidate do ballots[$b'$].vote($false$);
```

ASCP uses the following *below-and-incompatible-than* relation on ballots. We say ballots b and b' are *compatible* (written $b \sim b'$) iff $b.x = b'.x$, and *incompatible* (written $b \nsim b'$) otherwise, where we let $\bot \neq x$ for any $x \in \mathbf{Val}$. We say ballot b is *below and incompatible than* ballot b' (written $b \lesssim\!\!\!\!/\, b'$) iff $b < b'$ and $b \nsim b'$. In a nutshell, ASCP works as follows: each node uses FV to *prepare* a ballot b which carries the candidate value $b.x$, this is, it aborts every ballot $b' \lesssim\!\!\!\!/\, b$, which prevents any attempt to decide a value different from $b.x$ at a round smaller than $b.n$; once b is prepared, the node uses FV again to commit ballot b, thus deciding the candidate value $b.x$.

ASCP is depicted in Algorithm 2 above. We assume that each node v creates a process federated-voting(v, b) for each ballot b, which is stored in the infinite array ballots[b] (line 2). The node keeps fields candidate and prepared, which respectively contain the ballot that v is trying to commit and the highest ballot prepared so far. Both candidate and prepared are initialised to the null ballot (line 3). The node also keeps a field round that contains the current round, initialised to 0 (line 4). Once v proposes a value x, the node assigns the ballot $\langle 1, x \rangle$ to candidate and tries to prepare it by invoking FV's primitive vote($false$) for each ballot below and incompatible than candidate (lines 5–7). This may involve sending an infinite number of messages, which by BNS requires sending finitely many batches. Once v

prepares some ballot b by receiving FV's indication deliver(*false*) for every ballot below and incompatible than b, and if b exceeds prepared, the node updates prepared to b (lines 8–9). The condition in line 8 may concern an infinite number of ballots, but it may hold after receiving a finite number of batches by BNS. If prepared reaches or exceeds candidate, then the node updates candidate to prepared, and tries to commit it by voting *true* for that ballot (lines 10–12). Once v commits some ballot b by receiving FV's indication deliver(*true*) for ballot b, the node decides the value $b.x$ (lines 13–14) and stops execution.

If the candidate ballot of a node v can no longer be aborted nor committed, then v resorts to a timeout mechanism that we describe next. The primitive start-timer(Δ) starts the node's local timer, such that a timeout event will be triggered once the specified delay Δ has expired. (Invoking start-timer(Δ') while the timer is already running has the effect of restarting the timer with the new delay Δ'.) In order to start the timer, a node v needs to receive, from each member of a quorum that contains v itself, messages that endorse either committing or preparing ballots with rounds bigger than round (line 15 of Algorithm 2). Since the domain of values can be infinite, the condition in line 15 requires that for each node u in some quorum U that contains v itself, there exists a ballot b_u with round $b_u.n >$ round, and either v receives from u a message endorsing to commit b_u, or otherwise v receives from u messages endorsing to abort every ballot in some non-empty, right-open interval $[z_u, b_u)$, whose upper bound is b_u. This condition may require receiving an infinite number of ballots, but it may hold after receiving a finite number of batches by BNS. Once the condition in line 15 holds, the node updates round to the smallest n such that every member of the quorum endorses to either commit or prepare some ballot with round bigger or equal than n, and (re-)starts the timer with delay F(round), where F is an unbound function that doubles its value with each increment of n (lines 16–17). If the candidate ballot can no longer be aborted or committed, then timeout will be eventually triggered (line 18) and the node considers a new candidate ballot with the current round increased by one, and with the value candidate.x if the node never prepared any ballot yet (line 19) or the value prepared.x otherwise (line 20). Then v tries to prepare the new candidate ballot by voting *false* for each ballot below and incompatible than it (line 21). This may involve sending an infinite number of messages, which by BNS requires sending finitely many batches.

The condition for starting the timer in line 15 does not strictly use FV as a black box. However, this use is warranted because line 15 only "reads" the state of the network. ASCP makes every other change to the network through FV's primitives.

ASCP guarantees the safety properties of *non-blocking Byzantine consensus* in §3. Since a node stops execution after deciding some value, *Integrity for intact sets* holds trivially. The requirement in lines 8–12 of Algorithm 2 that a node prepares the candidate ballot before voting for committing it, enforces that if a voting for committing some ballot within the nodes of an intact set I succeeds, then some node in I previously prepared that ballot:

▶ **Lemma 9.** *Let $\mathcal{S}$ be an FBQS and consider an execution of ASCP over $\mathcal{S}$. Let I be an intact set in $\mathcal{S}$. If a node $v_1 \in I$ commits a ballot b, then some node $v_2 \in I$ prepared b.*

Aborting every ballot below and incompatible than the candidate one prevents that one node in an intact set I prepares a ballot b_1, and concurrently another node in I sends READY(b_2, *true*) with b_2 below and incompatible than b_1:

▶ **Lemma 10.** *Let $\mathcal{S}$ be an FBQS and consider an execution of ASCP over $\mathcal{S}$. Let I be an intact set in $\mathcal{S}$. Let v_1 and v_2 be nodes in I and b_1 and b_2 be ballots such that $b_2 \lessapprox b_1$. The following two things cannot both happen: node v_1 prepares b_1 and node v_2 sends READY(b_2, true).*

Agreement for intact sets holds as follows: assume towards a contradiction that two nodes in I respectively commit ballots b_1 and b_2 with different values. A node in I prepared the bigger of the two ballots by Lemma 9, which results in a contradiction by Lemma 10.

Lemma 11 below ensures that in line 20 it is safe to take as the new candidate value that of the largest prepared ballot, which helps to establish *Weak validity for intact sets.*

▶ **Lemma 11.** *Let $\mathcal{S}$ be an FBQS and consider an execution of ASCP over $\mathcal{S}$. Let b_1 be the largest ballot prepared by some node v_1 at some moment in the execution. If all nodes are honest, then some node v_2 proposed $b_1.x$.*

Now we examine the liveness properties of *non-blocking Byzantine consensus* in §3, which ASCP also meets. Recall from §4 the bounded interval δ_I that a node in an intact set I takes to deliver some Boolean for a given ballot, provided that some other node in I has already delivered a Boolean for the same ballot. Let v be a node in I that prepares some ballot b such that no other node in I has ever prepared a ballot with round bigger or equal than $b.n$. We call the interval of duration δ_I after v prepares b the *window for intact set I of round $b.n$*. Lemma 12 below guarantees that after some moment in the execution, no two consecutive windows ever overlap.

▶ **Lemma 12.** *Let $\mathcal{S}$ be an FBQS and consider an execution of ASCP over $\mathcal{S}$. Let I be an intact set in $\mathcal{S}$ and assume that all faulty nodes eventually stop. There exists a round n such that either every node in I decides some value before reaching round n, or otherwise the windows for I of all the rounds $m \geq n$ never overlap with each other, and in each window of round m the nodes in I that have not decided yet only prepare ballots with round m.*

Lemma 12 helps to establish *Non-blocking for intact sets* as follows. After the moment where no two consecutive windows overlap, either every node in I has the same candidate ballot at the beginning of the window of some round, or otherwise the highest ballots prepared by each node in I during that window coincide with each other. In either case all the nodes in I will eventually have the same candidate ballot, and they will decide a value in bounded time.

Correctness of ASCP is captured by Theorem 13 below:

▶ **Theorem 13.** *Let $\mathcal{S}$ be an FBQS. The ASCP protocol over $\mathcal{S}$ satisfies the specification of non-blocking Byzantine consensus for intact sets.*

6 Concrete Stellar Consensus Protocol

In this section we introduce *concrete SCP* (*CSCP*) which is amenable to implementation because each node v maintains finite state and only needs to send and receive a finite number of messages in order to progress. CSCP relies on *bunched voting* (*BV*) in Algorithm 3, which generalises FV and embodies all of FV's instances for each of the ballots. CSCP considers a single instance of BV, and thus each node v keeps a single process bunched-voting(v) (line 2 of Algorithm 4). In BV, nodes exchange messages that contain two kinds of statements: a *prepare statement* PREP b encodes the aim to abort the possibly infinite range of ballots that are lower and incompatible than b; and a *commit statement* CMT b encodes the aim to commit ballot b.

Algorithm 3 depicts BV over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$. A node v stores the highest ballot for which v has respectively voted, readied, or delivered a prepare statement in fields max-voted-prep, max-readied-prep, and max-delivered-prep (line 2). It also stores the set of ballots for which v has respectively voted, readied, or delivered a commit statement in fields ballots-voted-cmt, ballots-readied-cmt, and ballots-delivered-cmt (line 3). All these

Algorithm 3 Bunched voting (BV) over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$.

```
1  process bunched-voting($v \in \mathbf{V}$)
2      max-voted-prep, max-readied-prep, max-delivered-prep $\leftarrow \langle 0, \bot \rangle \in \mathbf{Ballot}$;
3      ballots-voted-cmt, ballots-readied-cmt, ballots-delivered-cmt $\leftarrow \emptyset \in 2^{\mathbf{Ballot}}$;
4      prepare($b$)
5          if max-voted-prep $< b$ then
6              max-voted-prep $\leftarrow b$;
7              send VOTE(PREP max-voted-prep) to every $v' \in \mathbf{V}$;
8      when exists maximum $b$ such that max-voted-prep $< b$ and exists $U \in \mathcal{Q}$
        such that $v \in U$ and for every $u \in U$ received VOTE(PREP $b_u$) where
        $b' \lnsim b_u$ for every $b' \lnsim b$
9          max-readied-prep $\leftarrow b$ ;
10         send READY(PREP max-readied-prep) to every $v' \in \mathbf{V}$;
11     when exists maximum $b$ such that max-readied-prep $< b$ and exists
        $v$-blocking $B$ such that for every $u \in B$ received READY(PREP $b_u$) where
        $b' \lnsim b_u$ for every $b' \lnsim b$
12         max-readied-prep $\leftarrow b$ ;
13         send READY(PREP max-readied-prep) to every $v' \in \mathbf{V}$;
14     when exists maximum $b$ such that max-delivered-prep $< b$ and exists $U \in \mathcal{Q}$
        such that $v \in U$ and for every $u \in U$ received READY(PREP $b_u$) where
        $b' \lnsim b_u$ for every $b' \lnsim b$
15         max-delivered-prep $\leftarrow b$;
16         prepared(max-delivered-prep);
17     commit($b$)
18         if $b \notin$ ballots-voted-cmt and max-voted-prep $= b$ then
19             ballots-voted-cmt $\leftarrow$ ballots-voted-cmt $\cup \{b\}$;
20             send VOTE(CMT $b$) to every $v' \in \mathbf{V}$ ;
21     when received VOTE(CMT $b$) from every $u \in U$ for some $U \in \mathcal{Q}$ such that
        $v \in U$ and $b \notin$ ballots-readied-cmt
22         ballots-readied-cmt $\leftarrow$ ballots-readied-cmt $\cup \{b\}$ ;
23         send READY(CMT $b$) to every $v' \in \mathbf{V}$ ;
24     when received READY(CMT $b$) from every $u \in B$ for some $v$-blocking $B$ and
        $b \notin$ ballots-readied-cmt
25         ballots-readied-cmt $\leftarrow$ ballots-readied-cmt $\cup \{b\}$ ;
26         send READY(CMT $b$) to every $v' \in \mathbf{V}$ ;
27     when received READY(CMT $b$) from every $u \in U$ for some $U \in \mathcal{Q}$ such that
        $v \in U$ and $b \notin$ ballots-delivered-cmt
28         ballots-delivered-cmt $\leftarrow$ ballots-delivered-cmt $\cup \{b\}$;
29         committed($b$) ;
```

fields are finite and thus v maintains only finite state. When a node v invokes $\mathsf{prepare}(b)$, if b exceeds the highest ballot for which v has voted a prepare, then the node updates $\mathsf{max\text{-}voted\text{-}prep}$ to b and sends $\mathtt{VOTE}(\text{PREP } b)$ to every other node (lines 4–7). The protocol then proceeds with the usual stages of FV, with the caveat that at each stage of the protocol only the maximum ballot is considered for which the node can send a message – or deliver an indication – with a prepare statement. In particular, when there exists a ballot b that exceeds $\mathsf{max\text{-}readied\text{-}prep}$ and such that v received a message $\mathtt{VOTE}(\text{PREP } b_u)$ from each member u of some quorum to which v belongs, then the node proceeds as follows: it checks that each b' lower and incompatible than b_u is also lower and incompatible than b (line 8). If b is the maximum ballot passing the previous check for every member u of the quorum, then the node updates the field $\mathsf{max\text{-}readied\text{-}prep}$ to b and sends the message $\mathtt{READY}(\text{PREP } b)$ to every other node (lines 9–10). The node v checks similar conditions for the case when it receives messages $\mathtt{READY}(\text{PREP } b_u)$ from each member u of a v-blocking set, and proceeds similarly by updating $\mathsf{max\text{-}readied\text{-}prep}$ to b and sending $\mathtt{READY}(\text{PREP } b)$ to every other node (lines 11–13). The node will update $\mathsf{max\text{-}delivered\text{-}prep}$ and trigger the indication $\mathsf{prepared}(b)$ when the same conditions are met after receiving messages $\mathtt{READY}(\text{PREP } b_u)$ from each member u of a quorum to which v belongs (lines 14–16). When a node v invokes $\mathsf{commit}(b)$ then the protocol proceeds with the usual stages of FV with two minor differences (lines 17–29). First, a node v only votes commit for the highest ballot for which v has voted a prepare statement (condition $\mathsf{max\text{-}voted\text{-}prep} = b$ in line 18). Second, the protocol uses the sets of ballots $\mathsf{ballots\text{-}voted\text{-}cmt}$, $\mathsf{ballots\text{-}readied\text{-}cmt}$ and $\mathsf{ballots\text{-}delivered\text{-}cmt}$ in order to keep track of the stage of the protocol for each ballot.

The structure of CSCP in Algorithm 4 directly relates to ASCP in Algorithm 2. A node proposes a value x in line 5. A node tries to prepare a ballot b by invoking $\mathsf{prepare}(b)$ in line 7, and receives the indication $\mathsf{prepared}(b)$ in line 8. A node tries to commit a ballot b by invoking $\mathsf{commit}(b)$ in line 12, and receives the indication $\mathsf{committed}(b)$ in line 13. A node decides a value x in line 14. Timeouts are set in lines 15–17 and triggered in line 18.

Next we establish a correspondence between CSCP in and ASCP in §5: the concrete protocol observationally refines the abstract one, which means that any externally observable behaviour of the former can also be produced by the latter [6]. Informally, the refinement shows that for an FBQS $\mathcal{S}$ and an intact set I, for every execution of CSCP over $\mathcal{S}$ there exists an execution of ASCP over $\mathcal{S}$ (with some behaviour of faulty nodes) such that each node in I decides the same value in both of the executions. The refinement result allows us to carry over the correctness of ASCP established in §5 to CSCP.

We first define several notions required to formalise our refinement result. A *history* is a sequence of the events $v.\mathsf{propose}(x)$ and $v.\mathsf{decide}(x)$, where v is a correct node and x a value. The specification of consensus assumes that v triggers an event $v.\mathsf{propose}(x)$, thus a history will have $v.\mathsf{propose}(x)$ for every correct node v. A *concrete trace* τ is a sequence of events that subsumes histories, and contains events $v.\mathsf{prepare}(b)$, $v.\mathsf{commit}(b)$, $v.\mathsf{prepared}(b)$, $v.\mathsf{committed}(b)$, $v.\mathsf{start\text{-}timer}(n)$, $v.\mathsf{timeout}$, $v.\mathsf{send}(m, v')$, and $v.\mathsf{receive}(m, v')$, where v is a correct node and v' is any node, b is a ballot, m is a message in $\{\mathtt{VOTE}(s), \mathtt{READY}(s)\}$ with s a statement in $\{\text{PREP } b, \text{CMT } b\}$, and n is a round. An *abstract trace* τ is a sequence of events that subsumes histories, and contains events $v.\mathsf{start\text{-}timer}(n)$, $v.\mathsf{timeout}$, and batched events $v.\mathsf{vote\text{-}batch}([b_i], a)$, $v.\mathsf{deliver\text{-}batch}([b_i], a)$, $v.\mathsf{send\text{-}batch}([m_i], v')$, and $v.\mathsf{receive\text{-}batch}([m_i], v')$,where v is a correct node and v' is any node, n is a round, $[b_i]$ is a sequence of ballots, a is a Boolean, and $[m_i]$ is a sequence of messages in $\{\mathtt{VOTE}(b, a), \mathtt{READY}(b, a)\}$. The sequences of ballots and messages above, which represent a possibly infinite number of "batched" events, ensure that the length of any abstract trace is bounded by ω. We may

Algorithm 4 Concrete SCP (CSCP) over an FBQS $\mathcal{S}$ with set of quorums $\mathcal{Q}$.

```
process concrete-consensus($v \in \mathbf{V}$)
    brs ← new process bunched-voting($v$);
    candidate, prepared ← $\langle 0, \bot \rangle \in$ Ballot;
    round ← $0 \in \mathbb{N}^+ \cup \{0\}$;
    propose($x$)
        candidate ← $\langle 1, x \rangle$;
        brs.prepare(candidate);
    when triggered brs.prepared($b$) and prepared $< b$
        prepared ← $b$;
        if candidate $\leq$ prepared then
            candidate ← prepared;
            brs.commit(candidate);
    when triggered brs.committed($b$)
        trigger decide($b.x$);
    when exists $U \in \mathcal{Q}$ such that $v \in U$ and for each $u \in U$ exist
     $\mathtt{M}_u \in \{\mathtt{VOTE}, \mathtt{READY}\}$ and $b_u \in$ Ballot such that round $< b_u.n$ and received
     $\mathtt{M}_u(\mathrm{S}_u\ b_u)$ from $u$ with $\mathrm{S}_u \in \{\mathrm{CMT}, \mathrm{PREP}\}$
        round ← $\min\{b_u.n \mid u \in U\}$;
        start-timer($F$(round));
    when triggered timeout
        if prepared $= \langle 0, \bot \rangle$ then candidate ← $\langle$round $+ 1$, candidate.$x\rangle$;
        else candidate ← $\langle$round $+ 1$, prepared.$x\rangle$;
        brs.prepare(candidate);
```

omit the adjective "concrete/abstract" from "trace" when it is clear from the context. Given a trace τ, a history $H(\tau)$ can be uniquely obtained from τ by removing every event in τ different from v.propose(x) or v.decide(x).

An execution of CSCP (respectively, ASCP) *entails* a *concrete trace* (respectively, *abstract trace*) τ iff for every invocation and indication as well as for every send or receive primitive in an execution of the protocol in Algorithm 4 (respectively, for every invocation, indication and primitive in an execution of the protocol in Algorithm 2, where the vote, deliver, send and receive events are batched together), τ contains corresponding events in the same order.

We are interested in traces that are relative to some intact set I. Given a trace τ, the *I-projected* trace $\tau|_I$ is obtained by removing the events v.ev $\in \tau$ such that $v \notin I$.

▶ **Theorem 14.** *Let $\mathcal{S}$ be an FBQS and I be an intact set. For every execution of CSCP over $\mathcal{S}$ with trace τ, there exists an execution of ASCP over $\mathcal{S}$ with trace ρ and $H(\tau|_I) = H(\rho|_I)$.*

Proof sketch. We define a *simulation function* σ from concrete to abstract traces. Theorem 14 can be established by showing that, for every finite prefix τ of a trace entailed by CSCP, the simulation $\sigma(\tau)$ is a prefix of a trace entailed by ASCP. ◀

Every execution of ASCP enjoys the properties of *Integrity*, *Agreement for intact sets*, *Weak validity for intact sets* and *Non-blocking for intact sets*, and so does every execution of CSCP by refinement.

▶ **Corollary 15.** *Let $\mathcal{S}$ be an FBQS. The CSCP protocol over $\mathcal{S}$ satisfies the specification of non-blocking Byzantine consensus for intact sets.*

7 Lying about Quorum Slices

So far we have assumed the unrealistic setting where faulty nodes do not equivocate their quorum slices, so all nodes share the same FBQS $\mathcal{S}$. We now lift this assumption. To this end, we use a generalisation of FBQS called *subjective FBQS* [7], which allows faulty nodes to lie about their quorum slices. Assuming that $\mathbf{V}_{\text{ok}}$ is the set of correct nodes, the *subjective FBQS* $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ is an indexed family of FBQSes where the different FBQSes agree on the quorum slices of correct nodes, *i.e.*, $\forall v_1, v_2, v \in \mathbf{V}_{\text{ok}}.\ \mathcal{S}_{v_1}(v) = \mathcal{S}_{v_2}(v)$. For each correct node v, the FBQS $\mathcal{S}_v$ is the *view* of node v, which reflects the choices of trust communicated to v. We can run either ASCP or CSCP over a subjective FBQS $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ by letting each correct node v act according to its view $\mathcal{S}_v$.

We generalise the definition of intact set to subjective FBQSes, and we lift our results so far to the subjective FBQSes. Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS. A set I is an *intact set* iff for each $v \in \mathbf{V}_{\text{ok}}$ the set I is a quorum in $\mathcal{S}_v$ that only contains correct nodes, and every member of I is intertwined with each other in the projected FBQS $\mathcal{S}_v|_I$.

▶ **Lemma 16.** *Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS. For any node $v \in \mathbf{V}_{\text{ok}}$, a set I is an intact set in $\mathcal{S}_v$ iff I is an intact set in $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$.*

Since Lemma 16 above guarantees that every view has the same intact sets, which also coincide with the intact sets of the subjective FBQS, from now on we may say "an intact set I" and omit to which system (a particular view, or the subjective FBQS) I belongs .

Using the fact that nodes agree on the slices of correct nodes, we can prove Lemma 17 below, which is the analogue to Lemma 3 and states sufficient safety conditions for the nodes in an intact set I to reach agreement when each node acts according to its own view.

▶ **Lemma 17.** *Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS and for each correct node v let $\mathcal{Q}_v$ be the set of quorums in the view $\mathcal{S}_v$. Let I be an intact set and consider two quorums U_1 and U_2 in $\bigcup_{v\in\mathbf{V}_{\text{ok}}} \mathcal{Q}_v$. If $U_1 \cap I \neq \emptyset$ and $U_2 \cap I \neq \emptyset$, then $U_1 \cap U_2 \cap I \neq \emptyset$.*

Using arguments similar to those in the previous sections, we can establish the correctness of ASCP and CSCP over subjective FBQSes.

▶ **Theorem 18.** *Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS. The ASCP protocol over $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ satisfies the specification of non-blocking Byzantine consensus for intact sets.*

▶ **Theorem 19.** *Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS and I be an intact set. For every execution of CSCP over $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ with trace τ, there exists an execution of ASCP over $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ with trace ρ and $H(\tau|_I) = H(\rho|_I)$.*

▶ **Corollary 20.** *Let $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ be a subjective FBQS. The CSCP protocol over $\{\mathcal{S}_v\}_{v\in\mathbf{V}_{\text{ok}}}$ satisfies the specification of non-blocking Byzantine consensus for intact sets.*

8 Related Work

García-Pérez and Gotsman [7] have previously investigated Stellar's federated voting and its relationship to Bracha's broadcast over classical Byzantine quorum systems. They did not address the full Stellar consensus protocol. Our proof of SCP establishes the correctness of federated voting by adjusting the results in [7] to multiple intact sets within the system.

Losa et al. [10] have also investigated consensus over FBQSs. They propose a generalisation of Stellar's quorums that does not prescribe constructing them from slices, yet allows different participants to disagree on what constitutes a quorum. They then propose a protocol solving consensus over intact sets in this setting that provides better liveness guarantees than SCP, but is impractical. Losa et al.'s work is orthogonal to ours: they consider a more general setting than Stellar's and a theoretical protocol, whereas we investigate the practical protocol used by Stellar.

The advent of blockchain has given rise to a number of novel proposals of BFT protocols; see [3] for a survey. Out of these, the most similar one to Stellar is Ripple [15]. In particular, Ripple have recently proposed a protocol called Cobalt that allows for a federated setting similar to Stellar's [11]. We hope that our work will pave the way to investigating the correctness of this and similar protocols.

References

1 Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

2 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

3 Christian Cachin and Marko Vukolic. Blockchain Consensus Protocols in the Wild. In *International Symposium on Distributed Computing (DISC)*, 2017.

4 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002.

5 Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Bastian Pochon. The perfectly synchronized round-based model of distributed computing. *Information and Computation*, 205(5):783–815, 2007.

6 Ivana Filipovic, Peter W. O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theoretical Computer Science*, 411(51-52):4379–4398, 2010.

7 Álvaro García-Pérez and Alexey Gotsman. Federated Byzantine Quorum Systems. In *International Conference on Principles of Distributed Systems (OPODIS)*, volume 125 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl, 2018.

8 Álvaro García-Pérez and Maria A. Schet. Deconstructing Stellar Consensus (extended version). *CoRR*, abs/1911.05145, 2019. URL: `http://arxiv.org/abs/1911.05145`.

9 Leslie Lamport. The Part-Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

10 Giuliano Losa, Eli Gafni, and David Mazières. Stellar consensus by instantiation. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl, 2019.

11 Ethan MacBrough. Cobalt: BFT Governance in Open Networks. *CoRR*, abs/1802.07240, 2018. URL: `http://arxiv.org/abs/1802.07240`.

12 Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

13 David Mazières. The Stellar consensus protocol: a federated model for internet-Level consensus, 2015. URL: `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`.

14 David Mazières, Giuliano Losa, and Eli Gafni. Simplifed SCP, 2019. URL: `http://www.scs.stanford.edu/~dm/blog/simplified-scp.html`.

15 David Schwartz, Noah Youngs, and Arthur Britto. The Ripple Protocol Consensus Algorithm, 2014. URL: `https://ripple.com/files/ripple_consensus_whitepaper.pdf`.

16 Yee Jiun Song, Robbert van Renesse, Fred B. Schneider, and Danny Dolev. The Building Blocks of Consensus. In *International Conference on Distributed Computing and Networking (ICDCN)*, 2008.