Stellar Consensus by Instantiation

Giuliano Losa*
Galois, Inc.
giuliano@galois.com

Eli Gafni†
UCLA
eli@ucla.edu

David Mazières‡
Stanford
http://www.scs.stanford.edu/~dm/addr/

January 24, 2020

Abstract

Stellar introduced a new type of quorum system called a Federated Byzantine Agreement System. A major difference between this novel type of quorum system and a threshold quorum system is that each participant has its own, personal notion of a quorum. Thus, unlike in a traditional BFT system, designed for a uniform notion of quorum, even in a time of synchrony one well-behaved participant may observe a quorum of well-behaved participants, while others may not.

To tackle this new problem in a more general setting, we abstract the Stellar Network as an instance of what we call *Personal Byzantine Quorum Systems*. Using this notion, we streamline the theory behind the Stellar Network, removing the clutter of unnecessary details, and we refute the conjecture that Stellar's notion of intact set is optimally fault-tolerant. Most importantly, we develop a new consensus algorithm for the new setting.

*Funding: BSF Grant 2014226, NSF Grant 1655166, a gift from the Stellar Development Foundation, and Galois, Inc.

†Funding: BSF Grant 2014226, NSF Grant 1655166, and a gift from the Stellar Development Foundation

‡Funding: Stanford Center for Blockchain Research

1 Introduction

We study the consensus problem in a new type of quorum system that we call a Personal Byzantine Quorum System (abbreviated PBQS). In a PBQS, each participant has its own, private notion of what a quorum is, subject to the requirement that if Q_p is a quorum of p and $p' \in Q_p$ then there is a quorum $Q_{p'}$ of p' inside Q_p. Justifying this rather strong requirement on the intuitive level, Q being a quorum of p has the connotation that p trusts the members of Q collectively. Hence, Q should contain at least one quorum of each $p' \in Q$.

In contrast to PBQSs, traditional Byzantine quorum systems are uniform, in the sense that a quorum is a public notion common to all participants. Under the assumptions of quorum intersection (i.e., that every two quorums intersect at a well-behaved participant) and quorum availability (i.e., that at least one quorum is exclusively well-behaved), one can implement consensus under eventual synchrony [6]. However, traditionally, the ability to implement consensus using quorums is all or nothing; as soon as two quorums fail to intersect at a well-behaved participant, or if no quorum is available, no subset of the participants can solve consensus.

In a PBQS, it is possible that a subset S of the participants has intersecting quorums, in which case we say S is intertwined, while the system as a whole does not. Relying on quorum intersection to ensure safety to S is straightforward. However, suppose S_1 and S_2 are each intertwined but $S_1 \cup S_2$ is not. In this case there is no way to keep $S_1 \cup S_2$ in agreement, but we can still keep each set internally in agreement. Additonally, we may have $S_1 \cap S_2 \neq \emptyset$; in this case, can a consensus algorithm ensure liveness to S_1 and S_2? This seems impossible since, if S_1 and S_2 diverge, a participant belonging to both S_1 and S_2 has to pick a side and violate safety on the other side in order to make progress. Those observations raise the problem of determining, given an instance of PBQS and a set B of malicious participants, for which family of sets both safety and liveness are achievable, and whether there is an optimal such family. Of course, participants have no knowledge of what B is. In Section 2, we give necessary conditions for a family of sets to enjoy consensus and we define the notion of a *consensus cluster*, for which we show how to solve consensus in Section 3.

Another crucial technical difference between PBQSs and traditional Byzantine quorum systems is that since participants do not know what constitutes a quorum for another participant, even in a synchronous period, we face the asynchronous phenomenon that one well-behaved member observes a quorum of well-behaved participants, while others do not. This phenomenon was previously encountered only during periods of asynchrony.

Why is it important to study PBQSs? As we show in section 4, PBQSs abstract a deployed, real-world system: the Stellar Network. We found designing a BFT consensus algorithm which is both safe and live under these condition to be challenging. Indeed, the Stellar Consensus Protocol [14] (SCP) has only been proved non-blocking when there are Byzantine failures. Here, we propose an algorithm which is safe and live, albeit impractical. Nevertheless, it serves our purpose of showing that while the Stellar network is optimally fault-tolerant for safety, Stellar's family of intact sets, which enjoy both safety and liveness, is not optimal as previously conjectured. Furthermore, our algorithm guarantees termination in the eventually synchronous model. Whether a practical protocol can achieve these properties is still an open question.

In addition to introducing the PBQS model, we make the following contributions:

- We design an unauthenticated BFT consensus algorithm using idea from Dwork et al.[6] to solve consensus for the Stellar Network's consensus clusters.
- We refute the conjecture made in the Sellar Whitepaper [14] that intact sets are optimal for consensus. Indeed we suspect that our generalization of intact sets called consensus clusters are optimal.
- We show that the Stellar Network may harbor several disjoint consensus clusters which can nevertheless remain internally in agreement and live. Past work on federated Byzantine agreement systems [14, 7] (FBAS) assumes global quorum intersection and leaves the reader pondering whether all guarantees collapse should this assumption be violated.

Finally, we formalize the static properties of PBQSs and Stellar's federated Byzantine agreement systems in Isabelle/HOL; the formal theory is available in the Archive of Formal Proofs [12].

2 Personal Byzantine Quorum Systems

In this section we formalize the Personal Byzantine Quorum System Model (the PBQS Model), we define what it means to solve consensus in this model, we observe that global consensus is impossible even without faults, we give lower bounds on what subsets of participants can possibly enjoy consensus, and we define the notion of a consensus cluster. In a consensus algorithm, different consensus clusters may diverge, but, as we show in the next section, consensus is solvable under eventual synchrony within a consensus cluster. The main technical result of this section is that maximal consensus clusters are disjoint, as it is an obvious requirement for consensus.

Definition 1. A PBQS consists of a set of participants P, a set $B \subseteq P$ of Byzantine participants, a set $W = P \setminus B$ of well-behaved participants, and a function mapping a participant p to its non-empty set of quorums, which are subsets of P. The participants' quorums must be such that:

Property 1 (Quorum sharing)**.** *If Q_p is a quorum of p and $p' \in Q_p$ then there exists a quorum $Q_{p'}$ of p' such that $Q_{p'} \subseteq Q_p$.*

In other words, property 1 states that a quorum Q of some participant p must contain a quorum of every one of its members. As we show in Lemma 4, this remarkably simple property is sufficient to give a mathematically pleasing structure, obviously required if each consensus cluster is to be internally consistent, to PBQSs: Maximal consensus clusters are disjoint.

2.1 Consensus Algorithms in PBQSs

We assume that the participants communicate via a fully-connected point-to-point message-passing network. (In the Stellar network this is accomplished using an overlay network and signatures.) This means that a participant always knows the identity of the well-behaved sender of a message that it receives. However, message content is not authenticated (in keeping with the current Stellar Modus Operandi of not forwarding signatures) and therefore a participant cannot trust what a sender p says it heard from sender q. Well-behaved participants take steps according to the algorithm they are given, while Byzantine participants may take arbitrary steps. Each well-behaved participant is scheduled infinitely often and a message sent from a well-behaved participant to a well-behaved participant is eventually delivered.

A consensus algorithm consists of a non-terminating sequential program run by each participant in the system. The program can send and receive messages as well as take local computation steps. Initially, a participant starts with a unique identifier, a set of quorums, the set of all participants (used for round-robin leader election, which is replaced by a probabilistic election algorithm in the Stellar Network), and an input value, all of which are accessible to its program. Crucially, a participant does not know a priori the quorums of other participants (it only knows its own set of quorums). In the Stellar Network, a participant learns one of its own quorums only when it receives messages from all members of that quorum, but this difference is not of consequence. A participant also does not know which participants are Byzantine. At any point, a participant's program may produce a unique, irrevocable decision value.

Definition 2 (Intertwined)**.** We say that a set S of well-behaved participants is intertwined when for every two sets Q and Q' which are both quorums of some (possibly different) members of S, we have $Q \cap Q' \cap W \neq \emptyset$.

Note that, by definition, two intertwined participants cannot have empty quorums.

Definition 3 (Quorum-based algorithm)**.** We say that a consensus algorithm is quorum-based when:

1. If a well-behaved participant p decides, then there must be a quorum Q of p such that p received at least one message from each member of Q.

2. If Q is a quorum of a participant p, $p \in W$, and v is a possible input value, then there exists an execution in which only p and members of Q take steps, and p eventually outputs v.

As we have already noted, a PBQS may, for example, harbor two intertwined sets S_1 and S_2 such that $S_1 \cup S_2$ is not intertwined. As implied by the following lemma, in this case no quorum-based algorithm can solve consensus for $S_1 \cup S_2$.

Lemma 1. *Consider two participants p and p', $p \neq p'$, and two quorums Q, Q' such that Q is a quorum of p and Q' is a quorum of p' and $(Q \cap Q') \setminus B = \emptyset$. Then no quorum-based algorithm can guarantee agreement between p and p'.*

Proof. By definition of quorum-based algorithm, there are two executions e and e' such that (a) only p and members of Q take steps in e and p decides value v in e, and (b) only p' and members of Q' take steps in e' and p' decides value $v' \neq v$ in e'. Because Q and Q' are disjoint, the execution $e \cdot e'$ consisting of the concatenation of e and e' is also an execution. Moreover, agreement is violated in $e \cdot e'$. □

Lemma 1 shows that, in general, consensus in a PBQS is not solvable globally. Instead, we reformulate the consensus problem such that, given a PBQS $\mathcal{U}$ and a family of sets of participants depending on $\mathcal{U}$ (and thus on the quorum slices and on W), the traditional properties of consensus have to be guaranteed only to each set in the family.

Definition 4 (The PBQS Consensus Problem)**.** In the PBQS consensus problem for a PBQS $\mathcal{U}$ and a family of sets of participants $\{S_i\}$ (depending on $\mathcal{U}$), we require that for every set S_i in the family:

- **Agreement**: no two members of S_i decide different values.
- **Liveness**: every member of S_i eventually decides some value.
- **Non-triviality**: if only well-behaved participants take steps and a member of S_i decides, then it decides the input value of some well-behaved participant.

Note that the definition above does not preclude any participant from taking steps in the algorithm; instead, the definition gives guarantees only to sets in the family.

In Section 2.3, we define the family of consensus clusters, and we show in Section 3 that PBQS consensus is solvable for consensus clusters. Another, more restrictive, family for which PBQS consensus is solvable is the family of intact sets, as defined in the Stellar Whitepaper. In Section 5, we show that every intact set is a consensus cluster but that the reverse is not true. In this sense, it shows that intact sets cannot be optimal for PBQS consensus. Definition 4 also raises the question of whether there exists an optimal family (in the sense of inclusion) for which PBQS consensus is solvable. We leave this question open, although we conjecture that the consensus clusters family is optimal.

2.2 A Necessary Condition for Liveness

Next we observe that if every quorum Q of a participant p contains a Byzantine node, then it is impossible to guarantee liveness for p because malicious participants can always remain silent. This is formalized using the notion of blocking set:

Definition 5 (Blocking)**.** If R is a set of participants, we say that p is blocked by R, or equivalently that R blocks p, when every quorum of p intersects R. We denote the set of participants blocked by R by $\mathit{BlockedBy}(R)$, and the set of sets that each blocks p, called p's blocking sets, by $\mathit{Blocking}(p)$.

Lemma 2. *If p is blocked by B then no quorum-based algorithm can ensure liveness to p.*

Proof. If all malicious participants remain silent, then there is no quorum Q such that p eventually receives a message from every member of Q. Therefore, by requirement 1, p never decides. □

An interesting question is whether q who is blocked by $\mathit{BlockedBy}(B)$ shares the same fate as p who is blocked by B. The answer is positive and a consequence of the quorum sharing property, as implied by the following lemma.

Lemma 3. *In a personal quorum system, for every set of participants R, we have*

$$\mathit{BlockedBy}(\mathit{BlockedBy}(R)) = \mathit{BlockedBy}(R).$$

Proof. Suppose that $p \in \mathit{BlockedBy}(\mathit{BlockedBy}(R))$ but $p \notin \mathit{BlockedBy}(R)$. Hence, there is a quorum Q of p that does not intersect R. However, since $p \in \mathit{BlockedBy}(\mathit{BlockedBy}(R))$, Q must contain p' which is $\mathit{BlockedBy}(R)$. By the quorum sharing property, Q contains a quorum Q' of p', and by the virtue of p' being blocked by R, Q' contains a member of R. Since $Q' \subseteq Q$, we conclude that Q contains a member of R, and this is a contradiction.

□

Corollary 1. *If p is well-behaved and is not blocked by B, then p has a quorum consisting exclusively of well-behaved participants that are not blocked by B.*

2.3 Consensus Clusters

In this section we define consensus clusters and we show that maximal consensus clusters are disjoint. We also define the notion of strong consensus clusters, which are a subset of the consensus clusters and for which we propose a simpler consensus algorithm in Section 3 than for general consensus clusters.

Consensus clusters can be thought of as disjoint islands which can be kept internally consistent and live by a consensus algorithm, but which may diverge from each other.

Definition 6 (Consensus cluster)**.** A subset $S \subseteq W$ of the well-behaved participants is a consensus cluster when:

- Quorum Intersection: S is intertwined.
- Quorum Availability: If $p \in S$ then there is a quorum Q_p of p such that $Q_p \subseteq S$.

Note that, by quorum availability, a member of a consensus cluster must have a quorum, and, by quorum intersection, all its quorums must be non-empty.

Definition 7 (Strong Consensus cluster)**.** A subset $S \subseteq W$ of the well-behaved participants is a strong consensus cluster when:

- Quorum Intersection: The intersection of two quorums of members of S contains a member of S.
- Quorum Availability: If $p \in S$ then there is a quorum Q_p of p such that $Q_p \subseteq S$.

We now show that maximal consensus clusters are disjoint.

Definition 8. A consensus cluster C is maximal when no strict superset of C is a consensus cluster.

Lemma 4. *Consider a personal quorum system. If C_1 and C_2 are two consensus clusters and $C_1 \cap C_2 \neq \emptyset$, then $C_1 \cup C_2$ is a consensus cluster.*

Proof. Consider $p \in C_1$ and $q \in C_2$. It suffices to show that p and q are intertwined (quorum availability is immediate). Consider two quorums Q_p and Q_q of p and q, and a quorum Q_m of a participant $m \in C_1 \cap C_2$ such that $Q_m \subseteq C_1$. Since m and q are intertwined by virtue of belonging to C_2, it follows that Q_q and Q_m have non-empty intersection in C_1. Let $n \in C_1$ be a member of this intersection. By the quorum sharing property, Q_q contains a quorum Q_n of n. Since both n and p belong to C_1 they are intertwined. Consequently Q_p and Q_n intersect at a well-behaved participant. Since $Q_n \subseteq Q_q$, we get that Q_p and Q_q intersect at a well-behaved participant as required.

□

Corollary 2. *Maximal consensus clusters are disjoint.*

Finally, we present the two properties, Properties 2 and 3, that, as shown in the next section, are sufficient to solve PBQS consensus for any consensus cluster C.

Property 2 (quorum of member of C, blocks all members of C)**.** *If C is a consensus cluster and Q is a quorum of a member of C, then $Q \cap W$ blocks every member of C.*

Proof of Property 2. Consider $p \in C$. By the virtue of C being intertwined, all quorums of p intersect Q at a well-behaved participant. Thus $Q \cap W$ intersects all quorums of p, and we conclude that $Q \cap W$ blocks p. □

Property 3 (blocking set of member of C contains a member of C)**.** *If C is a consensus cluster, $p \in C$, and R blocks p, then $R \cap C \neq \emptyset$.*

Proof of Property 3. By definition of blocking sets, R intersects all quorums of p. Moreover, by the quorum-availability property of consensus clusters, p has a quorum $Q_p \subseteq C$. Thus, R intersects C. □

3 Solving Consensus under Eventual Synchrony in a PBQS

3.1 The Key Insight

Most eventually-synchronous BFT consensus algorithms [6, 3, 11, 5, 1, 8], whether they use authenticated messages or not, rely for liveness on the fact that if two participants p, p' receive the same messages then p observes a quorum (or blocking set) if and only if p' does. For example, this is used by PBFT's leader to convince other participants to prepare its value by attaching signed messages that prove that the value cannot contradict a past decision. In the unauthenticated BFT algorithm of Dwork et al. [6] (Algorithm 3), liveness is ensured by the fact that, during synchrony, a participant that locks a value at the highest round causes all other locks to be released because, thanks to reliable broadcast, the corresponding quorum is observed by all in a timely manner.

Unfortunately, those techniques fail in a PBQS because the notion of quorum is not shared by the participants: even if all participants receive the same messages, one may observe a quorum while the other does not.

The key observation that we make to solve this problem is the following. Consider a consensus cluster C. If, instead of just observing a quorum, a member p of C observes a quorum Q such that each member q of Q observed a quorum R_q unanimously making statement s, then all members of C that receive the same messages as p can derive that there is a unanimous quorum of some member of C making statement s. This is because, by Property 2 and Property 3, Q contains a member c of C, which can be trusted when it reports that the quorum R_c unanimously makes statement s.

3.2 The Consensus Algorithm

We follow the approach of Dwork et al.[6], i.e. we use a clock-synchronization protocol that simulates a synchronous model and then give a consensus protocol for the synchronous model. In the synchronous model, computation proceeds in rounds $1, 2, 3, \ldots$ where, in each round, each participant first broadcasts its local state and then receives a subset of the messages broadcast by other participants in that round, and updates its local state accordingly. We assume that each participants starts with a local state containing its input value. Moreover, in each round, each participant is given a leader for the current round.

We start by giving an algorithm for strong consensus clusters, and we later modify it to solve consensus for consensus clusters in general. Consider a strong consensus cluster C. We assume eventual synchrony, i.e., that there is a time GST after which (a) the messages between well-behaved participants are reliably delivered within a time bound Δ and (b) the relative rate of the clocks of any two well-behaved participants is bounded by a constant ρ. GST, Δ, and ρ are fixed but unknown to the participants. Under eventual synchrony, the clock-synchronization protocol ensures that there comes a round GSR such that, in round GSR and all subsequent rounds, well-behaved participants always receive all of each other's messages. We also assume that in round GSR and after, all members of C agree on a unique leader among C.

We now describe the consensus algorithm in the synchronous round model (a TLA+ specification appears in Appendix A). We group rounds into epochs, starting with epoch 1, consisting of 5 rounds. We say round $5(e-1)+i$ is phase i of epoch e. The local state of a participant consists of a boolean flag, a proposal for the epoch, and of an array indexed by epoch and phase where each position is either empty or contains a value. Initially, the flag is false, the proposal is the participant's input value, and the array is empty.

At the end of phase i of epoch e, a participant may update the corresponding index in its array with a value v; in this case we say that the participant adopts v. We say that the value that appears at the highest non-empty position in a participant's array is the participant's candidate, and we say that the highest index in the participant's row at which the candidate value appears is the candidate's index. When the locked flag is true, we say that the participant's candidate is locked. We say that v blocks participant p at phase i of epoch e when p learns that one of its blocking sets unanimously adopted v at phase i of epoch e.

In each epoch e, the algorithm proceeds as follows:

1. In phase 1, a participant p that hears from its leader l_p for the round adopts the candidate value of l_p unless p's candidate is locked and not equal to l_p's candidate.

2. In phases $i \in \{2, 3, 4, 5\}$, a participant adopts value v if it learns that v was adopted unanimously by one of its quorums in phase $i-1$ of epoch e.

3. A participant that adopts a value in phase 4 sets its flag to true, thereby locking its candidate.

4. A participant that adopts a value in phase 5 decides that value.

5. In phase 5, if no value blocks p, then p sets its proposal for epoch $e+1$ to its candidate. Otherwise, p determines the value v that blocks it at phase 3 in the highest epoch e_v among all the values that block p at phase 3, and p sets its proposal for $e+1$ to v.

6. In phase 5, if p's candidate is locked, then p determines the value v that blocks it at phase 2 in the highest epoch e_v among all the values that block p at phase 2. If v is different from p's candidate value and e_v is strictly bigger than p's candidate index, then p unlocks its candidate.

We now show that the algorithm is safe, i.e. no two members of C decide different values. This relies on the following crucial invariant:

Lemma 5. *If all well-behaved members of a quorum Q of a member of C have unanimously adopted v at phase* 4 *of epoch e, then no member of C ever adopts a different value in phases* 2*,* 3*,* 4*, or* 5 *of epoch e or higher.*

Proof. Suppose that Q is a quorum of a member of C whose well-behaved members have unanimously adopted v at phase 4 of epoch e. This means that all members of $Q \cap C$ locked v.

Moreover, no member of C ever unlocks v because it would require a unanimous quorum Q' for a different value at a higher epoch, which is impossible since, by the quorum-intersection property of strong consensus clusters, the intersection of Q and Q' must contain a member of C.

Finally, by the quorum-intersection property of strong consensus clusters, no member of C can observe a quorum Q' unanimous for a value different from v at an epoch greater or equal to e, and thus no member of C adopts any value different from v at phase 2, 3, 4, or 5 of any epoch greater or equal to e. □

The safety of the protocol follows immediately from Lemma 5.

Theorem 1. *No two members of C decide different values.*

Proof. Consider the first value v ever decided by a member of C and let e be the epoch during which v is decided by a member of C. It must be the case that there is a quorum Q of a member of C whose well-behaved members all locked v during epoch e (i.e. adopted v in phase 4 of epoch e). Thus, by Lemma 5, no other value may be decided by any member of C ever after. □

Next we show that the algorithm is live. More precisely, we show that all members of C decide in the first epoch that starts strictly after round GSR.

Lemma 6. *If phase* 4 *of epoch e occurs at or after GSR, then no two members of C have a lock on different values in phase* 1 *of epoch $e+1$.*

Proof. Suppose that, at the end of phase 5 of epoch e, participant $p \in C$ has a lock on its candidate v obtained at epoch e_1 and participant $p' \in C$ has a lock on $v' \neq v$ obtained at epoch e_2. By quorum intersection, we cannot have $e_1 = e_2$, because that would imply that a member of C adopted two different values in phase 3 of epoch e_1.

Suppose that $e_1 < e_2$. There must be a quorum Q of a member of C such that $Q \cap W$ unanimously adopted v' in phase 2 of epoch e_2. Since phase 5 of epoch e occurs at or after GSR, p hears from all members of $Q \cap W$ in phase 5 of epoch e. By Property 2, $Q \cap W$ is a blocking set for p. Thus v' blocks p at epoch $e_2 > e_1$, and by Item 6 above, p' unlocks its candidate by the end of epoch e. This is a contradiction. □

Lemma 7. *If phase* 4 *of epoch e occurs at or after GSR, if the leader $l \in C$ of epoch $e+1$ is well-behaved, and if a member c of C has a lock on value v at the end of phase* 5 *of epoch e, then l proposes v in epoch $e+1$.*

Proof. Note that l sets its proposal to the value v_l that blocks it at phase 3 in the highest possible epoch e_l. Suppose that c has a lock on its candidate v_c with candidate index e_c.

By quorum intersection, we cannot have $e_c = e_l$, because that would imply that a member of C adopted two different values in phase 2 of epoch e_1.

Suppose that $e_c > e_l$. Then, by property 2, l must be blocked by v_c at phase 3 of epoch e_c. Therefore, e_l cannot be the highest epoch at which l is blocked by some value, which is a contradiction.

Suppose that $e_l > e_c$. Then, by property 2, c must be blocked by v_l at phase 2 of epoch e_l. Thus c unlocks its candidate by the end of phase 5 of epoch e, which is a contradiction. □

Lemma 8. *If phase 4 of epoch e occurs at or after GSR, then every member of C decides in epoch $e + 1$.*

Proof. By Lemmas 6 and 7, since all members of C agree on a leader among themselves and communication is reliable and timely, all members of C adopt the leader's proposal (because any lock held must be for the leader's proposal) and decide it in phase 5. □

Note that termination only depends on the behavior of the members of C.

3.2.1 Extension to Consensus Clusters

If C is now a consensus cluster, and not a strong consensus cluster, then the algorithm above is not safe anymore. This is because two quorums Q_1 and Q_2 of members of C may not intersect in C: even though it is guaranteed that $Q_1 \cap Q_2 \cap W \neq \emptyset$, it is possible that $Q_1 \cap Q_2 \cap W \neq \emptyset$. With the latter, Lemma 5 does not hold anymore, and thus Theorem 1 may be violated too.

Intuitively, Lemma 5 fails because malicious participants can convince well-behaved participants which are not in C to unlock any value. To remedy the situation, participants are now required to send their complete history of lock acquisitions and releases with each message, and a participant ignores any message containing a history that it cannot independently verify. To verify a lock-unlock history, a participants p checks that, for each unlock event concerning value v at epoch e, p can derive that there is a unanimous quorum for a different value at a higher epoch, and thus that unlocking was safe from p's point of view.

To make sure that this does not block the progress of members of C, we split phase 2 into two phases 2' and 2, where phase 2' comes before phase 2 (we now have 6 phases 1, 2', 2, 3, 4, and 5). The rule for adopting a value in a phase remains unchanged, and a participants p verifies that an unlock event concerning value v at epoch e is valid by checking that there is an epoch $e' > e$ and a value $v' \neq v$ such that v' blocks p at phase 2' of epoch e' (which implies that a quorum of p adopted v' in phase 1 of epoch e').

Crucially, note that, by properties of consensus clusters, if a member of C unlocks a value, then, after GSR, it is guaranteed that other members of C will learn of a corresponding blocking set in phase 2'; thus, after GSR, members of C

always successfully validate each other's histories, and liveness is therefore ensured because, as noted in the previous section, it depends only on the behavior of members of C.

3.3 Clock Synchronization

We now describe a clock-synchronization algorithm adapted from the Stellar Consensus Protocol [14], which is simpler than the algorithm of Dwork et al. A participant p running the clock-synchronization protocol continuously advertises its current round $r[p]$ to all other participants, and it updates its round according to the following rules:

1. If p hears from a quorum whose members all advertise a round greater or equal to $r[p]$, then p arms a timer of duration $r[p] \cdot T_0$, where T_0 is some base timeout (e.g., 1 second).

2. If p's timer fires, p increments its current round.

3. If there is a round $r' > r[p]$ such that p hears from a blocking set whose members all advertise a round greater or equal to r', then p cancels any pending timeout and advances $r[p]$ to r'.

Now consider a consensus cluster C. By Property 2, rule 3 ensures that, after GST, any members of C that straggle in lower rounds catch up in constant time d_1 to the highest round that is advertised unanimously by the well-behaved portion of a quorum Q of C (because $Q \cap W$ is a blocking set for members of C). Since a blocking set must contain a member of C, rule 3 cannot be used by Byzantine participants to bring well-behaved participants to a round that was not already started by a member of C. Finally, rules 1 and 2 ensure that, despite Byzantine behavior, the first member of C to enter round r stays in round r for a duration proportional to r. Thus, round progression slows down linearly with time, and there eventually comes a round GSR after which rounds are long enough for all members of C to receive each other's messages. Note the timer duration in Rule 1 can be change, e.g., to obtain an exponential increase in round duration.

4 Consensus in Federated Byzantine Agreement Systems

In this section we show that, despite their seemingly unrealistic features, PBQSs are a useful model of Stellar's federated Byzantine agreement systems (FBASs). More precisely:

- We instantiate the consensus algorithm of Section 3 to FBASs, providing effective ways to implement its steps.

- Given a FBAS, we define a corresponding PBQS and we show that, under eventual synchrony, the instantiated consensus algorithm behaves similarly to its counterpart in the PBQS model.

The results of this section show that consensus clusters can be kept safe and live in a federated Byzantine agreement system that does not enjoy system-wide quorum intersection, whereas previous work on the subject made the assumption of system-wide quorum intersection. This is important because, in practice, misconfigured participants, rival factions, or compromised participants could, in violating quorum intersection, yield several disjoint consensus clusters.

4.1 Federated Byzantine Agreement Systems

In a FBAS, each participant chooses a set of slices, which are sets of participants. A participant p considers a set Q to be a quorum when (a) p has at least one slice inside Q and (b) every member of Q has a slice that is a subset of Q. Practical aspects of FBASs are beyond the scope of this paper, and we refer the reader to Mazières [14] for such matters. What we will say is that it is intended that a participant will trust any information unanimously agreed upon by any of its slices, and thus a quorum is, intuitively, a set that trusts itself.

Slice-based quorums have the advantage that any new participant can join or leave the system without coordination (to join, all it needs to do is join the communication substrate; in practice, this is an overlay network emulating a point-to-point network using public-key cryptography). Moreover, any participant can also reconfigure its slices unilaterally, without coordination, e.g., to remove participants it deems unreliable or to add newcomers. On the flip side, without further assumptions, there is no guarantee that quorums will intersect, and the set of participants at a given time is generally unknown. For the analysis that follows, we assume that the set of participants is unknown but fixed and that the participants' slices do not change throughout an execution.

Three key aspects of federated Byzantine agreement systems prevent a straightforward analogy with PBQSs for the purpose of solving consensus:

1. Since each participants self-declares its set of slices (e.g., by broadcasting it), participants discover their quorums as they receive the slices of other participants. Byzantine participants have the opportunity to declare arbitrary slices and shape the quorums of well-behaved participants.

2. The algorithms of Section 3 require checking whether a set of participants is a blocking set. Doing this check by enumerating quorums is not practical even if all slices are known because the number of quorums of a participant may be exponential in the size of the system.

3. The set of participants is unknown, and thus round-robin leader-election is impossible.

4.2 Abstracting Federated Byzantine Agreement Systems

In a FBAS, participants discover quorums as they learn about the slices of other participants. Therefore, for a participant, the notion of quorum is not fixed; instead, it is augmented with new quorums as the participant learns about the slices of other participants. We call the quorums of a participant p at time t the observed quorums of p at time t. We now define a fixed notion of abstract quorums, which form a PBQS, and relate them to observed quorums.

Definition 9 (Abstract Quorums)**.** A set Q is an abstract quorum of participant p when $p \in B$ or p has a slice contained in Q, and every well-behaved member of Q has a slice contained in Q.

Note that the definition of abstract quorum places requirements only on well-behaved nodes. Hence it is not computable by the participants, who do not know which participants are well-behaved. The following three lemmas are direct consequences of the definition of abstract quorum.

Lemma 9. *Abstract quorums form a PBQS.*

Proof. From the definition of abstract quorum we immediately get that if Q is an abstract quorum of p and $p' \in Q$, then Q is an abstract quorum of p'. □

Lemma 10. *If Q is an observed quorum of a well-behaved participant p at some time t, then Q is an abstract quorum.*

Lemma 11. *Assume that Q is an abstract quorum of $p \in W$ consisting exclusively of well-behaved participants. Then, under eventual synchrony and assuming that participants do advertise their slices: shortly after GST, Q is an observed quorum of p.*

Proof. Since Q is exclusively well-behaved, shortly after GST, all well-behaved participants receive the slices of the members of Q and can check whether Q is a quorum of theirs. □

Lemma 10 shows that the set of abstract quorums is an over-approximation of the observed quorums. Because all the algorithms presented so far use the notion of quorum only positively (i.e. adding quorums can only enable more behaviors), Lemma 10 implies that abstract quorums are a safe abstraction of the Stellar Network when considering those algorithms, and substituting the notion of observed quorum for quorum in those algorithms does not compromise their safety properties. Lemma 11 shows that, after GST, a well-behaved participant has observed all its abstract quorums. Since the liveness of the consensus algorithm depends only on the behavior of its maximal consensus cluster, we conclude that the instantiation of the algorithm to the FBAS model preserves liveness.

4.3 Checking Whether a Set is Blocking

The algorithms of Section 3 depend on the ability for a participant p to compute whether a given set R is one of its blocking sets. Even if all slices were known, doing so by enumerating p's quorums is not practical because, by virtue of how quorums are defined in a FBAS, p may have a number of quorums that is exponential is the size of the system. Instead, we now show that there is a recursive algorithm to check whether a set is a blocking set (a) without enumerating quorums and (b) relying only on the knowledge of the slices of well-behaved participants. This algorithm can be run locally or as a distributed algorithm, e.g., as in Stellar's Federated Voting algorithm [14]. It relies on the notion of slice-blocking.

Definition 10 (Slice-Blocking)**.** We say that the set of participants R slice-blocks p when R intersects each slice of p.

Definition 11 (Inductively Blocked)**.** If R is a set of participants, the set of participants inductively blocked by R, denoted R^*, is defined computationally as follows. Start with $R^* = \emptyset$. While a fixpoint is not reached, repeat the following step: add to R^* all the participants that are slice-blocked by $R^* \cup R$.

A participant can compute locally whether some set R is blocking based on its knowledge of other's slices. However, if its knowledge of slices is incomplete, it might wrongly believe that R is not blocking. This can only remove behaviors in the algorithms of Section 3, because blocking set is used only positively, and thus, with Lemma 12, the substitution of inductively blocking for blocking does not impact safety.

Finally, Lemma 13 shows that, after GST, well-behaved blocking sets are reliably identified by well-behaved participants using the notion of inductively blocking. Thus, liveness is also preserved when substituting inductively blocking for blocking.

Lemma 12. *At any time, if R inductively blocks $p \in W$ then R blocks p in the abstract quorum system.*

Proof. Assume by induction that if p' is in a slice of p and p' is inductively blocked by R, then all quorums of p' intersect R.

Now suppose by contradiction that R does not block p in the abstract system, i.e. that Q is an abstract quorum of p and $R \cap Q = \emptyset$. Since Q is an abstract quorum of p, there must be a slice s_p of p such that $s_p \subseteq Q$. Moreover, since R inductively blocks p, then s_p must have a member p' that is inductively blocked by R. By the quorum-sharing property, Q contains an abstract quorum of p'. Thus $Q \cap R \neq \emptyset$, which is a contradiction. □

Lemma 13. *If $p \in W$ and $R \subseteq W$ blocks p in the abstract quorum system, then, shortly after GST, R inductively blocks p.*

Proof. First, observe that, shortly after GST, p knows all the slices of the well-behaved participants. Thus, suppose that p knows all the slices of the well-behaved participants.

Suppose that R does not inductively block p according to p. Then, by definition, there is a slice s_p of p whose members are not inductively blocked by R and such that $s_p \cap R = \emptyset$. Since the members of s_p are not inductively blocked by R, then, for every $p' \in s_p \setminus B$, we also have that there is a slice s'_p of p' whose members are not inductively blocked by R and such that $s'_p \cap R = \emptyset$ (we have to exclude B from s_p since p might not know the slices of Byzantine participants; in the worst case, none of those are observed inductively blocked). Continuing inductively in this fashion, we obtain an abstract quorum Q of p which does not intersect R, and we have only used the slices of well-behaved participants. This contradicts the fact that R blocks p in the abstract quorum system. □

4.4 Leader Election

As noted before, round-robin leader-election is impossible in a FBAS because the set of participants is in general unknown. In this section we show how to probabilistically elect a leader. However, we give no bound on the probability of success, except that it is non-zero. Devising an efficient leader-election mechanism, or, more generally, a conciliator[2] mechanism, is left open.

To agree on a common leader among C with non-zero probability, every participant p selects at random a participant p' from one of its slices or itself. If $p = p'$, then p elects itself as leader and broadcasts (leader, p). Otherwise, it waits to receive a broadcast of the form (leader, p'') from p', and then elects the participant p'' as leader and broadcasts (leader, p'').

We now show that, through this process, members of C agree on a common leader taken among C with non-zero probability.

Definition 12. Graph $D(S)$ If S is a set of participants, the directed graph $D(S)$ is defined as the graph whose set of vertices is S, and where there is an edge from n_1 to n_2 when $n_2 \neq n_1$ and n_2 is in a slice of n_1.

Lemma 14. *If C is a consensus cluster, $p \in C$, and Q is a quorum of a member of C, then Q is reachable from p in $D(C)$.*

Proof. Since $p \in C$ and C is a consensus cluster, there is a quorum Q' of p such that $Q' \subseteq C$. Now suppose that Q is not reachable from p in $D(C)$. Then, with $Q' \subseteq C$, we get that $Q' \cap Q = \emptyset$. This contradicts the assumption that C is a consensus cluster. □

Definition 13. Elementary quorum An elementary quorum is a quorum Q such that no strict subset of Q is a quorum.

Note that, by definition, every quorum contains an elementary quorum.

Lemma 15. *If n_1 and n_2 are members of an elementary quorum q consisting exclusively of well-behaved participants, then there is a path in $D(q)$ from n_1 to n_2.*

Proof. Suppose q is an elementary quorum and that $n_1, n_2 \in q$ and n_2 is not reachable from n_1 in $D(q)$. Then consider the set S of participants that are reachable from n_1 in $D(q)$. By our assumption above, n_2 does not belong to S. Thus S is a strict subset of q. Moreover, every member n of S has a slice $\mathrm{s}_n \subseteq q$. Additionally, consider that we must have that $\mathrm{s}_n \subseteq S$, as otherwise a participant outside S would be reachable from n_1. Thus every member of S has a slice in S, and therefore S is a quorum. Since S is a strict subset of q, this contradicts the fact that q is an elementary quorum. □

Lemma 16. *If C is a consensus cluster, then there exists a member of C that is reachable in $D(C)$ from every other participant in C.*

Proof. Since C is a quorum, C contains an elementary quorum Q. By Lemma 14, Q is reachable from every member n of C in $D(C)$. Moreover, by Lemma 15, every member of Q is reachable in $D(Q)$ from every other member of Q. Thus, because $D(Q) \subseteq D(C)$, every member of Q is reachable in $D(C)$ from every member of C. □

Lemma 17. *If C is a consensus cluster, then, with non-zero probability, every member of C elects the same leader $l \in C$.*

Proof. Note that the leader-election algorithm can be seen as randomly selection edges in $D(P)$ (where P is the set of participants). Because there is a member n of C reachable in D from all other members of C in $D(C)$ (and because well-behaved participant have a finite number of outgoing edges), then with non-zero probability the edges selected by the leader-election algorithm will form a sink tree rooted at n, who will be elected unique leader by all members of C. □

5 Related Work

Federated Byzantine quorum systems were first introduced in the Stellar Whitepaper by Mazières [14], who also proposes the notion of intact set and a consensus algorithm for intact sets, the Stellar Consensus Protocol (SCP). The epidemic propagation mechanism and the clock-synchronization protocol presented in the present paper are taken from the Stellar Whitepaper. Mazières also discusses more practical aspects of the Stellar Network.

One important contribution of the present paper is that Stellar's intact sets, conjectured in the Stellar Whitepaper to be optimal for consensus, are in fact not the biggest sets for which an algorithm can solve consensus. An intact set is a subset I of W such that, even if all participants outside I are malicious: (a) if Q and Q' are quorums of I, then $Q \cap Q' \cap I \neq \emptyset$; (b) I is a quorum. Comparing the definitions of consensus cluster and intact set, it is easy to see that any intact set is also a consensus cluster. However, as shown by the following lemma, there are some consensus clusters that are strictly bigger than any intact set.

Lemma 18. *There are some configurations in which a set S is a consensus cluster but S is not intact and S has no intact superset.*

Proof. Consider a system of three well-behaved participants p_1, p_2, and p_3 (note that there are no malicious participants) where p_1 has a single slice $\{p_1\}$, p_2 has two slices $\{p_1, p_2\}$ and $\{p_2, p_3\}$, and p_3 has two slices $\{p_1, p_3\}$ and $\{p_2, p_3\}$. According to those slices, the quorums are $\{p_1\}$, $\{p_1, p_2, p_3\}$, $\{p_2, p_3\}$, $\{p_1, p_2\}$, and $\{p_1, p_3\}$. In this system, $C = \{p_2, p_3\}$ is a consensus cluster but is not intact, because $Q_1 = \{p_1, p_2\}$ and $Q_2 = \{p_1, p_3\}$ intersect outside C. Moreover, the only strict superset of C, $\{p_1, p_2, p_3\}$, is not intact because the quorums $\{p_1\}$ and $\{p_2, p_3\}$ do not intersect. □

Another novel aspect of the present paper compared to the Stellar Whitepaper is that we do not assume global quorum intersection; nevertheless, we show that consensus clusters enjoy safe and live consensus. This is important because it shows that safety and liveness guarantees do not collapse system-wide in the face of misconfigurations or attacks.

We have studied federated quorum system under the assumption that well-behaved participants do not change their slices. However, in practice, well-behaved participants might change their slices to eliminate unreliable participants or add newcomers. The Stellar Whitepaper also analyzes this situation.

García-Pérez and Gotsman [7] study in details Stellar's federated Byzantine quorum systems and the implementation of broadcast abstractions therein. They also propose the notion of subjective dissemination quorum system (DQS) in which, like in a PBQS, each participant has its own set of quorums. However, subjective DQSs have two crucial differences compared to PBQSs: subjective DQSs have system-wide quorum intersection and they do not have Property 1 (which says that a quorum is a quorum for all its members). In the absence of system-wide quorum intersection, Property 1 of PBQSs ensures that maximal consensus clusters are disjoint (Lemma 4). Without it, maximal consensus clusters may intersect, which implies that consensus is not solvable even for consensus clusters (a participant in the intersection may have to violate safety on one side in order to make progress).

Ripple [15] introduced the first permissionless quorum-based consensus protocol. In the XRP Ledger Consensus Protocol, each participant p is responsible for configuring its own UNL, which is a list of participants that p accepts messages from. Moreover, p considers as a quorum any set of participants consisting of more than a fixed fraction (defined system-wide by the protocol, e.g. 80%) of its UNL. Maintaining agreement in Ripple's protocol rests on the assumption that participants will provide sufficiently overlapping UNLs (roughly 90% for every pair of participants, in the most adversarial model of Chase and MacBrough [4]).

Traditional Byzantine quorum systems are uniform, in the sense that every participant has the same notion of quorum. Uniform Byzantine quorum systems are studied in details by Malkhi and Reiter [13]. More complex types of uniform quorum systems are studied by Guerraoui and Vukolić [9]. General Byzantine adversaries [10] do not give rise to a PBQS because participants have global knowledge of the adversary in this model.

Acknowledgment: The authors are in debt to an anonymous reviewer who suspected that our algorithm had a flaw. Indeed, that suspicion was correct.

References

[1] Ittai Abraham, Guy Gueta, Dahlia Malkhi, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance: Thelma, Velma, and Zelma. *arXiv preprint arXiv:1801.10022*, 2018.

[2] James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. *Distributed Computing*, 25(2):179–188, 2012.

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

[4] Brad Chase and Ethan MacBrough. Analysis of the XRP ledger consensus protocol. *arXiv preprint arXiv:1802.07242*, 2018.

[5] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

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

[7] Álvaro García-Pérez and Alexey Gotsman. Federated byzantine quorum systems. In *22nd International Conference on Principles of Distributed Systems (OPODIS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[8] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 BFT protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376. ACM, 2010.

[9] Rachid Guerraoui and Marko Vukolić. Refined quorum systems. *Distributed Computing*, 23(1):1–42, 2010.

[10] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *PODC*, volume 97, pages 25–34, 1997.

[11] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[12] Giuliano Losa. Stellar quorum systems. *Archive of Formal Proofs*, August 2019. `http://isa-afp.org/entries/Stellar_Quorums.html`, Formal proof development.

[13] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed computing*, 11(4):203–213, 1998.

[14] David Mazieres. The Stellar Consensus Protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, page 32, 2015.

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

A TLA+ Specification of the PBQS Consensus Algorithm

MODULE *SynchronousAlgo*

This is a specification of the FBQS consensus algorithm for strong consensus clusters. The algorithm is given in a synchronous round-by-round model implemented by the "scheduler" process below.

```
EXTENDS PBQS, Utilities

CONSTANTS
    V,
    InitVal,
     \* The set of epoch. This should be the set of natural numbers, but it can be replaced by a finite set for model-check
    Epoch,
    C  \* A strong consensus cluster.

ASSUME IsStrongConsensusCluster(C)

Phase ≜ 1 .. 5

--algorithm Consensus{
    variables
        adopted = [p ∈ P ↦ [e ∈ Epoch ↦ [i ∈ Phase ↦
            IF e = 1 ∧ i = 1 THEN InitVal[p] ELSE ⟨⟩]]],
        proposal = [p ∈ P ↦ [e ∈ Epoch ↦ IF e = 1 THEN InitVal[p] ELSE ⟨⟩]],
        locked = [p ∈ P ↦ FALSE],
        epoch = 1;
    define {
         \* A function to convert a PID to a processor :
        NumToProc ≜ CHOOSE f ∈ [1 .. Cardinality(P) → P] :
            ∀ p ∈ P : ∃ i ∈ DOMAIN f : f[i] = p
         \* The inverse of NumToProc :
        ProcToNum(p) ≜ CHOOSE n ∈ 1 .. Cardinality(P) : NumToProc[n] = p

        Candidate(p) ≜
            LET tuples ≜ {t ∈ [epoch : Epoch, phase : Phase, val : V] :
                        adopted[p][t.epoch][t.phase] = t.val}
            IN  MaxLexico(tuples)

         \* A record describing the highest value that blocks p at phase i of some epoch :
        MaxBlocking(p, i, H) ≜
            LET tuples ≜ {t ∈ [epoch : Epoch, phase : i .. 5, val : V] :
                        ∨ ∃ B ∈ Blocking(p) : B ⊆ H ∧ ∀ q ∈ B :
                              adopted[q][t.epoch][t.phase] = t.val}
            IN  IF tuples ≠ {} THEN MaxLexico(tuples) ELSE ⟨⟩

         \* The set of values that reached quorum threshold in phase i :
        GotQuorum(p, H, i) ≜ {v ∈ V :
            ∃ Q ∈ Quorums(p) : Q ⊆ H ∧
```

$\forall\, q \in Q : adopted[q][epoch][i] = v\}$

$HeardFrom \triangleq \{H \in (\textsc{subset}\ P) \setminus P : Cardinality(H) \neq 1 \land H \neq \{\}\}$

Here we remove some sets to speed up model − checking

$Leader(e) \triangleq NumToProc[(e \% Cardinality(P)) + 1]$ *We use a rotating leader*

$Safety \triangleq \forall\, p, q \in C : \forall\, e1, e2 \in Epoch :$
$\quad \forall\, v1, v2 \in V : adopted[p][e1][5] = v1 \land adopted[q][e2][5] = v2 \Rightarrow v1 = v2$

$Inv1 \triangleq \forall\, p \in C : \forall\, Q \in Quorums(p) : \forall\, e \in Epoch : \forall\, v, v2 \in V : \forall\, e2 \in Epoch :$
$\quad (\forall\, q \in Q \cap WellBehaved : adopted[p][e][4] = v) \land e2 > e \Rightarrow \forall\, q \in Q : adopted[p][e2][1] = v$

}

We now specify what a participant does upon changing round.proc is the processor ID, phase is the current phase(1, 2, 3, 4, or 5), and H is the set of processors that proc hears from in the phase phase

procedure $changeRound(\ proc, phase, H\ )$ **{**

l0: **if** ($phase = 1$) **{**
    adopt the leader s proposal unless the lock prevents it.
l1: **with** ($v = proposal[Leader(epoch)][epoch]$) **{**
    if ($\land\ Leader(epoch) \in H$
    $\land\ \ \lor \neg locked[proc]$
    $\quad\ \lor\ Candidate(proc).val = v$)
    $adopted[proc][epoch][1] := v$ **;**
    }
  } ;

l2: **if** ($phase \in 2 \,..\, 5$) **{**
    if ($GotQuorum(proc, H, phase - 1) \neq \{\}$)
    some value has a unanimous quorum with phase *phase* − 1
    with ($v \in GotQuorum(proc, H, phase - 1)$) **{** *pick one such value*
    $adopted[proc][epoch][phase] := v$ **;** *adopt it for the current round*
    if ($phase = 4$) $locked[proc] := \textsc{true}$ *in phas4, lock the candidate*
    }
  } ;

l3: **if** ($phase = 5$) **{**
    with ($b2 = MaxBlocking(proc, 2, H)$)
    unlock if the max phase − 2 blocking value contradicts the candidate :
    if ($b2 \neq \langle\rangle \land b2.val \neq Candidate(proc).val \land b2.epoch > Candidate(proc).epoch$)
    $locked[proc] := \textsc{false}$ **;**
    if ($proc = Leader(epoch + 1) \land epoch + 1 \in Epoch$)
    set the proposal to the max phase − 3 blocking value
    epoch + 1 ∈ Epoch prevents the model − checker from generating an epoch that does not belong to
    with ($b3 = MaxBlocking(proc, 3, H)$)
    if ($b3 \neq \langle\rangle$) $proposal[proc][epoch + 1] := b3.val$ **;**

```
                  else proposal[proc][epoch + 1] := Candidate(proc) ;
        } ;
l4:     return
      }

    process ( scheduler = "sched" )
        variables procNum = 1, phase = 1 ;
    {
l5:     while ( epoch ∈ Epoch ) {
l6:         while ( phase ≤ 5 ) {
l7:             while ( procNum ∈ DOMAIN NumToProc ) {
ll:                 with ( Heard  ∈ HeardFrom )
                          call changeRound(NumToProc[procNum], phase, Heard) ;
l8:                 procNum := procNum + 1
                } ;
                phase := phase + 1 ;
                procNum := 1 ;
            } ;
            epoch := epoch + 1 ;
            phase := 1
        }
    }
}
```

* *Modification History*
* *Last modified Tue Jan* 07 11 : 49 : 27 *PST* 2020 *by nano*
* *Created Thu Nov* 01 10 : 46 : 36 *PDT* 2018 *by nano*