Fast and secure global payments with Stellar

Marta Lokhava, Giuliano Losa*, David Mazières, Graydon Hoare,
Nicolas Barry, Eli Gafni†, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb
Stellar Development Foundation

Abstract

International payments are slow and expensive, in part because of multi-hop payment routing through heterogeneous banking systems. Stellar is a new global payment network that can directly transfer digital money anywhere in the world in seconds. The key innovation is a secure transaction mechanism across untrusted intermediaries, using a new Byzantine agreement protocol called SCP. With SCP, each institution specifies other institutions with which to remain in agreement; through the global interconnectedness of the financial system, the whole network then agrees on atomic transactions spanning arbitrary institutions, with no solvency or exchange-rate risk from intermediary asset issuers or market makers. We present SCP's model, protocol, and formal verification; describe the Stellar payment network; and finally evaluate Stellar empirically through benchmarks and our experience with several years of production use.

CCS Concepts • **Security and privacy → Distributed systems security**; • **Computer systems organization → Peer-to-peer architectures**; • **Information systems →** *Electronic funds transfer*.

Keywords blockchain, BFT, quorums, payments

ACM Reference Format:
Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, Jed McCaleb. 2019. Fast and secure global payments with Stellar. In *SOSP '19: Symposium on Operating Systems Principles, October 27–30, 2019, Huntsville, ON, Canada.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3341301.3359636

1 Introduction

International payments are notoriously slow and costly [32]. Consider the impracticality of sending $0.50 from the U.S. to Mexico, two neighboring countries. End users pay nearly $9 for the average such transfer [32], and a bilateral agreement brokered by the countries' central banks could only reduce the underlying bank cost to $0.67 per item [2]. On top of fees, the *latency* of international payments is generally counted in days, making it impossible to get money abroad quickly in emergencies. In countries where the banking system doesn't work or doesn't serve all citizens, or where fees are intolerable, people resort to sending payments by bus [38], by boat [19], and occasionally now by Bitcoin [55], all of which incur risk, latency, or inconvenience.

While there will always be compliance costs, evidence suggests a significant amount is lost to lack of competition [21], which is exacerbated by inefficient technology. Where people can innovate, prices and latencies go down. For instance, remittances from bank accounts in Q2 2019 cost an average of 6.99%, while the figure for mobile money was only 4.88% [13]. An open, global payment network that attracts innovation and competition from non-bank entities could drive down costs and latencies at all layers, including compliance [83].

This paper presents Stellar, a blockchain-based payment network specifically designed to facilitate innovation and competition in international payments. Stellar is the first system to meet all three of the following goals (under a novel but empirically valid "Internet hypothesis"):

1. **Open membership** – Anyone can issue currency-backed digital tokens that can be exchanged among users.
2. **Issuer-enforced finality** – A token's issuer can prevent transactions in the token from being reversed or undone.
3. **Cross-issuer atomicity** – Users can atomically exchange and trade tokens from multiple issuers.

Achieving the first two is easy. Any company can unilaterally offer a product such as Paypal, Venmo, WeChat Pay, or Alipay and ensure the finality of payments in the virtual currencies they have created. Unfortunately, transacting atomically across these currencies is impossible. In fact, despite Paypal having acquired Venmo's parent company in 2013, it is still impossible for end users to send Venmo dollars to Paypal users [78]. Only recently can merchants even accept both with a single integration.

Goals 2 and 3 can be achieved in a closed system. In particular, a number of countries have efficient domestic payment networks, typically overseen by a universally trusted regulatory authority. However, membership is limited to a closed set of chartered banks and the networks are limited to the reach of a country's regulatory authority.

*Galois, Inc.
†UCLA

SOSP '19, October 27–30, 2019, Huntsville, ON, Canada

ACM ISBN 978-1-4503-6873-5/19/10…$15.00
https://doi.org/10.1145/3341301.3359636

Goals 1 and 3 have been achieved in mined blockchains, most notably in the form of ERC20 tokens on Ethereum [3]. The key idea of these blockchains is to create a new *cryptocurrency* with which to reward people for making settled transactions hard to revert. Unfortunately, this means token issuers do not control transaction finality. If software errors cause transactions history to be reorganized [26, 73], or when the spoils of defrauding people exceed the cost of reorganizing history [74, 97], issuers may be liable for tokens they have already redeemed for real-world money.

The Stellar blockchain has two distinguishing properties. First, it natively supports efficient markets between tokens from different issuers. Specifically, anyone can issue a token, the blockchain provides a built-in orderbook for trade between any pair of tokens, and users can issue *path payments* that atomically trade across several currency pairs while guaranteeing an end-to-end limit price.

Second, Stellar introduces a new Byzantine agreement protocol, SCP (Stellar Consensus Protocol), through which token issuers designate specific *validator* servers to enforce transaction finality. So long as no one compromises an issuer's validators (and the underlying digital signatures and cryptographic hashes remain secure), the issuer knows exactly which transactions have occurred and avoids the risk of losses from blockchain history reorganization.

SCP's key idea is that most asset issuers benefit from liquid markets and want to facilitate atomic transactions with other assets. Hence, validator administrators configure their servers to agree with other validators on the exact history of all transactions on all assets. A validator v_1 can be configured to agree with v_2, or v_2 can be configured to agree with v_1, or both may be configured to agree with each other; in all cases, neither will commit to a transaction history until it knows the other cannot commit to a different history.

By transitivity, if v_1 cannot disagree with v_2 and v_2 cannot disagree with v_3 (or vice versa), v_1 cannot disagree with v_3, whether or not v_3 represents assets v_1 has even heard of. Under the hypothesis that these agreement relationships transitively connect the whole network, SCP guarantees global agreement, making it a global Byzantine agreement protocol with open membership. We call this new connectedness assumption the Internet hypothesis, and note that it holds of both "*the* Internet" (which everyone understands to mean the single largest transitively connected IP network) and legacy international payments (which are hop-by-hop non-atomic, but leverage a transitively connected, global network of financial institutions).

Stellar has been in production use since September, 2015. To keep the blockchain length manageable, the system runs SCP at 5-second intervals—fast by blockchain standards, but far slower than typical applications of Byzantine agreement. Though the primary use has been payments, Stellar has also proven appealing for non-money fungible tokens that benefit from immediate secondary markets (see Section 7.1).

The next section discusses related work. Section 3 presents SCP. Section 4 describes our formal verification of SCP. Section 5 describes Stellar's payment layer. Section 6 relates some of our deployment experience and lessons learned. Section 7 evaluates the system. Section 8 concludes.

2 Related work

The system described in this paper is collectively our third attempt to realize a decentralized payment network. The first attempt, by one of the authors, resulted in the Ripple blockchain. While still in existence, we believe Ripple cannot realize the vision of a truly open payment network for both governance and technical reasons. The governance reasons derive from tension between shareholder interests and network users, but are beyond the scope of this paper. The technical issue is that Ripple's Byzantine agreement protocol [89] requires nearly closed membership [33], making it functionally similar to conventional Byzantine agreement.

The second attempt, a short-lived fork of the Ripple code base called *stellard*, went live in July, 2014. We had planned to develop SCP and slot it into *stellard* when ready, but instead decided to build a new system from scratch around SCP, called *stellar-core* and described in this paper. *stellar-core* was released along with a description and proof of SCP [68] in April, 2015, and entered production use 5 months later. An Internet draft [20] later described SCP for implementors.

SCP's motivation and structure are similar to asymmetric trust in multiparty computation [40], though that model requires nodes to have a global view of the adversary structure. By introducing a notion of quorum slices, SCP enjoys discoverability and dynamic membership that are important to Stellar's deployment. Personal Byzantine quorum systems [47] and asymmetric distributed trust [29] generalize SCP's quorums and simplify comparison to traditional systems. In a more restricted setting, Gotsman et al. [49] also study a generalization of Stellar's quorums.

Several other blockchains have adopted SCP, including MobileCoin [7] and NCNT. Ripple also more recently proposed an open-membership Byzantine agreement protocol called Cobalt [66]. SCP's safety is optimal, so SCP is safe in any failure scenario where Cobalt is, while the converse is unclear. However, Cobalt claims its safety condition is easier to understand and thus less prone to misconfiguration, which will be interesting to evaluate if Cobalt gets deployed.

2.1 Systems without open membership

Like many consensus protocols, SCP is based on voting [50, 94] in a quorum system. Previous work on quorum systems [52, 67, 72], sometimes called "survivor sets" [57], studies quorum systems that are static, i.e., fixed throughout system execution, and uniform, i.e., where all nodes have the same notion of what a quorum is. SCP's setting is different in that different nodes accept different and evolving quorums.

Heterogeneous fast consensus [91] allows more nuanced notions of trust to be expressed via labels in a lattice and leverages this to optimize communication, but it assumes closed membership.

Other efforts to adapt Byzantine agreement protocols to Blockchain-like settings, including HoneyBadger [70], SBFT [53], and HotStuff [99], have focused on scalability to many participants and performance over wide-area networks, retaining the traditional closed-membership model. Blockchains targeting closed membership have leveraged such protocols, including most recently Libra [11].

2.2 Systems without issuer-enforced finality

Bitcoin [71] was the first fungible digital asset without a trusted authority to track balances. It simultaneously solved two problems: how to distribute a new asset so people believe it has value, and how to provide irreversible transactions that prevent double-spend attacks. Bitcoin creates assets through proof-of-work [43], which its author compares to gold miners investing resources to increase the supply of gold. The proved work depends on transaction history, which thwarts double-spend attacks because rolling back transactions requires work comparable to creating the original history.

Attacks on proof-of-work blockchains become feasible once an attacker controls 1/3 of mining power [46, 48]. Proof-of-work is particularly risky for new or smaller blockchains [15, 23]; established miners have been known to divert mining power to smaller chains for profit [10, 54, 74, 97, 98] or just to create problems [25, 37] (so-called "Goldfinger attacks" [60]). At times, miners have benevolently mounted such "51% attacks" to roll back the undesired effects of bugs [95]. New blockchains can gain some protection from 51% attacks by leveraging more established blockchains with merged mining [6] or proof of burn [79].

As of this writing, a 51% attack is estimated to cost $823,000/hour on Bitcoin, $93,000/hour on Ethereum, and under $20,000/hour for all other tracked blockchains [12]. Ultimately, these costs depend on the value of the cryptocurrency incentivizing the mining, which is independent of the value of issued assets. By contrast, the cost of compromising Stellar validators is somewhat under the control of their administrators; SCP is amenable to high-assurance implementations, such as using hardware security modules to prevent a validator from signing contradictory messages. As an additional benefit, Stellar has no mining costs to recoup through transaction fees or seigniorage.

Bitcoin's massive energy consumption [76] has motivated more energy-efficient variations of proof of work such as proof of space [16, 17, 44, 87], proof of storage [18, 56, 86, 90], and proof of elapsed time [4].

Proof of work can be slow, with blocks of transactions taking minutes to confirm and multiple blocks required for stronger security. However, the work to mine a block can be amortized over an arbitrary-sized batch of transactions, constrained only by political issues and compatibility. Bitcoin-NG [45] uses proof of work to elect a leader who coordinates a larger batch of transactions. Thunderella [84] can use proof-of-work as a fallback "slow path" and optimistically confirm transactions much faster using an asynchronous consensus protocol when an accelerator node and 3/4 of an elected committee are honest. Though SCP is faster than proof-of-work, it requires more messages than Thunderella's fast path, but could potentially be a useful Thunderella slow path.

An alternative to proof of work is proof of stake [59], in which miners obtain influence either directly or indirectly from asset ownership. In some cases, influence increases with the time assets have been held. Some schemes encourage honesty by confiscating miners' stake if they misbehave [27, 28, 41]. Delegated proof-of-stake [9, 61] has nodes vote their stake to elect representatives for a conventional Byzantine agreement protocol such as PBFT [31]. Algorand [51] uses verifiable random functions to elect representative nodes pseudorandomly, with probability proportional to cryptocurrency holdings. Snow White [39] formalizes the security properties required in proof of stake, accommodating "sleepy" nodes that go on and off line.

Though most proof-of-stake schemes cannot be brute-forced with expensive computation, the cost to acquire stake or bribe miners is still related to the value of the underlying cryptocurrency and outside an asset issuer's control. Even in the absence of malicious miners, bugs can also cause blockchain reorganization [26, 73]. Hence, unlike with SCP, issuers have no way to ensure blockchain finality before redeeming on-chain tokens for a real-world wire transfer or cash withdrawal.

3 Stellar consensus protocol

The Stellar consensus protocol (SCP) is a quorum-based Byzantine agreement protocol with open membership. Quorums emerge from the combined local configuration decisions of individual nodes. However, nodes only recognize quorums to which they belong themselves, and only after learning the local configurations of all other quorum members. One benefit of this approach is that SCP inherently tolerates heterogeneous views of what nodes exist. Hence, nodes can join and leave unilaterally with no need for a "view change" protocol to coordinate membership.

3.1 Federated Byzantine agreement

The traditional Byzantine agreement problem consists of a closed system of N nodes, some of which are faulty and may behave arbitrarily. Nodes receive input values and exchange messages to decide on an output value among the inputs. A Byzantine agreement protocol is *safe* when no two well-behaved nodes output different decisions and the unique decision was a valid input (for some definition of valid agreed

upon beforehand). A protocol is *live* when it guarantees that every honest node eventually outputs a decision.

Typically, protocols assume $N = 3f + 1$ for some integer $f > 0$, then guarantee safety and some form of liveness so long as at most f nodes are faulty. At some stage in these protocols, nodes vote on proposed values and a proposal receiving $2f + 1$ votes, called a quorum of votes, becomes the decision. With $N = 3f + 1$ nodes, any two quorums of size $2f + 1$ overlap in at least $f + 1$ nodes; even if f of these overlapping nodes are faulty, the two quorums share at least one non-faulty node, preventing contradictory decisions. However, this approach only works if all nodes agree on what constitutes a quorum, which is impossible in SCP where two nodes may not even know of each other's existence.

With SCP, each node v unilaterally declares sets of nodes, called its **quorum slices**, such that (a) v believes that if all members of a slice agree about the state of the system, then they are right, and (b) v believes that at least one of its slices will be available to provide timely information about the state of the system. We call the resulting system, consisting of nodes and their slices, a Federated Byzantine Agreement (FBA) system. As we will see next, a quorum system emerges from nodes' slices.

Informally, an FBA node's slices express with whom the node requires agreement. E.g., a node may require agreement with 4 specific organizations, each running 3 nodes; to accommodate downtime, it may set its slices to be all sets consisting of 2 nodes from each organization. If this "requires agreement with" relation transitively relates any two nodes, we get global agreement. Otherwise, we can get divergence, but only between organizations neither of which requires agreement with the other. Given the topology of today's financial system, we hypothesize that widespread convergence will keep producing a singe ledger history people call "*the* Stellar network," much as we speak of *the* Internet.

Quorums arise from slices as follows. Every node specifies its quorum slices in every message it sends. Let S be the set of nodes from which a set of messages originated. A node considers the set of messages to have reached quorum threshold when every member of S has a slice included in S. By construction, such a set S, if unanimous, satisfies the agreement requirements of each of its members.

A faulty peer may advertise slices crafted to change what well-behaved nodes consider quorums. For the sake of protocol analysis, we define a **quorum** in FBA to be a non-empty set S of nodes encompassing at least one quorum slice of each *non-faulty* member. This abstraction is sound, as any set of messages purporting to represent a unanimous quorum actually does (even if it contains messages from faulty nodes), and it is precise when S contains only well-behaved nodes. In this section, we also assume that nodes' slices do not change. Nevertheless, our results transfer to the changing-slice case because a system in which slices change is no less safe than a fixed-slice system in which a node's slices consist of all the slices it ever uses in the changing-slices case (see Theorem 13 in [68]). As explained in Section 4, liveness depends on well-behaved nodes eventually removing unreliable nodes from their slices.

Because different nodes have different agreement requirements, FBA precludes a global definition of safety. We say non-faulty nodes v_1 and v_2 are **intertwined** when every quorum of v_1 intersects every quorum of v_2 in at least one non-faulty node. An FBA protocol can ensure agreement only between intertwined nodes; since SCP does so, its fault tolerance for safety is optimal. The **Internet hypothesis**, underlying Stellar's design, states that the nodes people care about will be intertwined.

We say a set of nodes I is **intact** if I is a uniformly non-faulty quorum such that every two members of I are intertwined even if every node outside of I is faulty. Intuitively, then, I should remain impervious to the actions of non-intact nodes. SCP guarantees both non-blocking liveness [93] and safety to intact sets, though nodes themselves do not need to know (and may not be able to know) which sets are intact. Furthermore, the union of two intact sets that intersect is an intact set. Therefore, intact sets define a partition of the well-behaved nodes, where each partition is safe and live (under some conditions), but different partitions may output divergent decisions.

3.1.1 Safety vs. Liveness considerations in FBA

With limited exceptions [64], most closed Byzantine agreement protocols are tuned to the equilibrium point at which safety and liveness have the same fault tolerance. In FBA, that means configurations in which, regardless of failures, all intertwined sets are also intact. Given that FBA determines quorums in a decentralized way, it is unlikely that individual slice choices will lead to this equilibrium. Moreover, at least in Stellar, equilibrium is not desirable: the consequences of a safety failure (namely double-spent digital money) are far worse than those of a liveness failure (namely delays in payments that anyway took days before Stellar). People therefore should and do select large quorum slices such that their nodes are more likely to remain intertwined than intact.

Further tipping the scales, it is easier to recover from typical liveness failures in an FBA system than in a traditional closed one. In closed systems, all messages must be interpreted with respect to the same set of quorums. Hence, adding and removing nodes to recover from failure requires reaching consensus on a reconfiguration event, which is difficult once consensus is no longer live. By contrast, with FBA, any node can unilaterally adjust its quorum slices at any time. In response to an outage at a systemically important organization, node administrators can adjust their slices to work around the problem, a bit like coordinating responses to BGP catastrophes [63] (though without the constraints of routing over physical network links).

3.1.2 The cascade theorem

SCP follows the template of the basic round model [42]; nodes progress through a series of numbered ballots, each attempting three tasks: (1) identify a "safe" value not contradicted by any decision in a previous ballot (often termed *preparing* the ballot), (2) agree on the safe value, and (3) detect that agreement was successful. However, FBA's open membership stymies several common techniques, making it impossible to "port" traditional closed protocols to the FBA model by simply changing the definition of quorum.

One technique employed by many protocols is rotating through leader nodes in round-robin fashion following timeouts. In a closed system, round-robin leader selection ensures that eventually a unique honest leader ends up coordinating agreement on a single value. Unfortunately, round-robin cannot work in an FBA system with unknown membership.

Another common technique that fails with FBA is assuming a particular quorum can convince all nodes. For instance, if everyone recognizes any $2f + 1$ nodes as a quorum, then $2f + 1$ signatures suffice to prove protocol state to all nodes. Similarly, if a node receives a quorum of identical messages through reliable broadcast [24], the node can assume all non-faulty nodes will also see a quorum. In FBA, by contrast, a quorum means nothing to nodes outside the quorum.

Finally, non-federated systems often employ "backwards" reasoning about safety: if $f + 1$ nodes are faulty, all safety guarantees are lost. Hence, if node v hears $f + 1$ nodes all state some fact F, v can assume at least one is telling the truth (and hence that F is true) with no loss of safety. Such reasoning fails in FBA because safety is a property of *pairs* of nodes, so a node that has lost safety to some peers can always lose safety to more nodes by assuming bad facts.

FBA *can*, however, reason backwards about liveness. Define a ***v*-blocking** set as a set of nodes that intersects every slice of v. If a v-blocking set B is unanimously faulty, B can deny node v a quorum and cost it liveness. Hence, if B unanimously states fact F, then v knows that either F is true or v is not intact. However, v still needs to see a full quorum to know that intertwined nodes won't contradict F, which leads to a final round of communication in SCP and other FBA protocols [47] that is not required in analogous closed-membership protocols. The result is that we have three possible levels of confidence in potential facts: indeterminate, safe to assume among intact nodes (which we will term *accepted* facts), and safe to assume among intertwined nodes (which we will term *confirmed* facts).

Node v can efficiently determine whether a set B is v-blocking by checking whether B intersects all its slices. Interestingly, if nodes always announce the statements they accept and a full quorum accepts a statement, it sets off a cascading process by which statements propagate throughout intact sets. We call the key fact underlying this propagation the **cascade theorem**, which sates the following: If I is an intact set, Q is a quorum of any member of I, and S is any superset of Q, then either $S \supseteq I$ or there is a member $v \in I$ such that $v \notin S$ and $I \cap S$ is v-blocking. Intuitively, were this not the case, the complement of S would contain a quorum that intersects I but not Q, violating quorum intersection. Note that if we start with $S = Q$ and repeatedly expand S to include all nodes it blocks, we obtain a cascading effect until, eventually, S encompasses all of I.

3.2 Protocol description

SCP is a partially synchronous consensus protocol [42] consisting of a series of attempts to reach consensus called ballots. Ballots employ timeouts of increasing duration. A ballot-synchronization protocol ensures that nodes stay on the same ballot for increasing periods of time until the ballots are effectively synchronous. Termination is not guaranteed until ballots are synchronous, but two synchronous ballots in which faulty members of well-behaved nodes' slices do not interfere are sufficient for SCP to terminate.

A *balloting protocol* specifies the actions taken during each ballot. A ballot begins with a prepare phase, in which nodes try to determine a value to propose that does not contradict any previous decision. Then, in a commit phase, nodes try to make a decision on the prepared value.

Balloting employs an agreement sub-protocol called *federated voting*, in which nodes vote on abstract statements that may eventually be *confirmed* or get stuck. Some statements might be designated contradictory, and the safety guarantee of federated voting is that no two members of an intertwined set confirm contradictory statements. Confirmation of a statement is not guaranteed except for an intact set whose members all vote the same way. However, if a member of an intact set does confirm a statement, federated voting guarantees that all members of the intact set eventually confirm that statement. Hence, taking irreversible steps in response to confirming statements preserves liveness for intact nodes.

Nodes initially propose values obtained from a *nomination protocol* that increases the chances of all members of an intact set proposing the same value, and that eventually converges (though with no way to determine convergence is complete). Nomination combines federated voting with leader selection. Because round-robin is impossible in FBA, nomination uses a probabilistic leader-selection scheme.

The cascade theorem plays a crucial role both in ballot synchronization and in avoiding blocked states from which termination is no longer possible.

3.2.1 Balloting

SCP nodes proceed through a series of numbered ballots, employing federated voting to agree on statements about which values are or are not decided in which ballots. If asynchrony or faulty behavior prevents reaching a decision in ballot n, nodes time out and try again in ballot $n + 1$.

Recall federated voting might not terminate. Hence, some statements about ballots can get stuck in a permanently indeterminate state where nodes can never determine if they are still in progress or stuck. Because nodes cannot rule out the possibility of indeterminate statements later proving true, they must never attempt federated voting on new statements contradicting indeterminate ones.

In each ballot n, nodes use federated voting on two types of statement:

- **PREPARE** $\langle n, x \rangle$ – states that no value other than x was or will ever be decided in any ballot $\leq n$.
- **COMMIT** $\langle n, x \rangle$ – states x is decided in ballot n.

Importantly, note that PREPARE $\langle n, x \rangle$ contradicts COMMIT $\langle n', x' \rangle$ when $n \geq n'$ and $x \neq x'$.

A node starts ballot n by attempting federated voting on a statement PREPARE $\langle n, x \rangle$. If any previous PREPARE statement was successfully confirmed through federated voting, the node chooses x from the confirmed PREPARE of the highest ballot. Otherwise, the node sets x to the output of the nomination protocol described in the next subsection.

If and only if a node successfully confirms PREPARE $\langle n, x \rangle$ in ballot n, it attempts federated voting on COMMIT $\langle n, x \rangle$. If that succeeds, it means SCP has decided, so the node outputs the value from the confirmed COMMIT statement.

Consider an intertwined set S. Since at most one value can be confirmed prepared by members of S in a given ballot, no two different values may be confirmed committed by members of S in a given ballot. Moreover, if COMMIT $\langle n, x \rangle$ is confirmed, then PREPARE $\langle n, x \rangle$ was confirmed too; since PREPARE $\langle n, x \rangle$ contradicts any earlier commit for a different value, by the agreement guarantees of federated voting we get that no different value may be decided in an earlier ballot by members of S. By induction on ballot numbers, we therefore get that SCP is safe.

For liveness, consider an intact set I and a long enough synchronous ballot n. If faulty nodes appearing in the slices of well-behaved nodes do not interfere in n, then by ballot $n+1$ all members of I have confirmed the same set $\mathcal{P}$ of PREPARE statements. If $\mathcal{P} = \emptyset$ and ballot n was long enough, the nomination protocol will have converged on some value x. Otherwise, let x be the value from the prepare with the highest ballot in $\mathcal{P}$. Either way, I will uniformly attempt federated voting on PREPARE $\langle n+1, x \rangle$ in the next ballot. Therefore, if $n+1$ is also synchronous, a decision for x inevitably follows.

3.2.2 Nomination

Nomination entails federated voting on statements:

- **NOMINATE** x – states x is a valid decision candidate.

Nodes may vote to nominate multiple values—different NOMINATE statements are not contradictory. However, once a node confirms any NOMINATE statement, it stops voting to nominate new values. Federated voting still allows a node to confirm new NOMINATE statements it didn't vote for, which allows members of an intact set to confirm one another's nominated values while still withholding new votes.

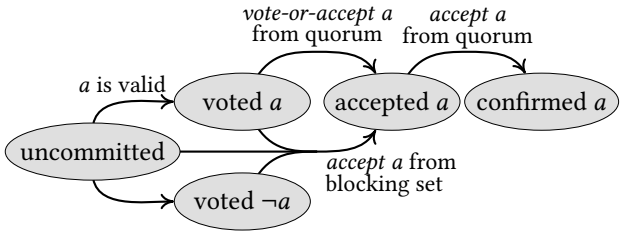

Figure 1. Stages of federated voting

The (evolving) result of nomination is a deterministic combination of all values in confirmed NOMINATE statements. If x represents a set of transactions, nodes can take the union of sets, the largest set, or the one with the highest hash, so long as all nodes do the same. Because nodes withhold new votes after confirming one NOMINATE statement, the set of confirmed statements can contain only finitely many values. The fact that confirmed statements reliably spread through intact sets means intact nodes eventually converge on the same set of nominated values and hence nomination result, though at an unknown point arbitrarily late in the protocol.

Nodes employ federated leader selection to reduce the number of different values in NOMINATE statements. Only a leader who has not already voted for a NOMINATE statement may introduce a new x. Other nodes wait to hear from leaders and just copy their leaders' (valid) NOMINATE votes. To accommodate failure, the set of leaders keeps growing as timeouts occur, though in practice only a few nodes introduce new values of x.

3.2.3 Federated voting

Federated voting employs a three-phase protocol shown in Figure 1. Nodes try to agree on abstract statements by first voting, then accepting, and finally confirming statements.

A node v may *vote* for any valid statement a that does not contradict its other outstanding votes and accepted statements. It does so by broadcasting a signed vote message. v then *accepts* a if a is consistent with other accepted statements and either (case 1) v is a member of a quorum in which each node either votes for a or accepts a, or (case 2) even if v didn't vote for a, a v-blocking set accepts a. In case 2, v may have previously cast votes contradicting a, which have now been *overruled*. v is allowed to forget about overruled votes and pretend it never cast them because if v is intact, it knows overruled votes cannot complete a quorum through case 1. v broadcasts that it accepts a, then *confirms* a when it is in a quorum that unanimously accepts a. Figure 2 shows the effect of v-blocking sets and the cascade theorem during federated voting.

Two intertwined nodes cannot confirm contradictory statements, as the two required quorums would have to share a

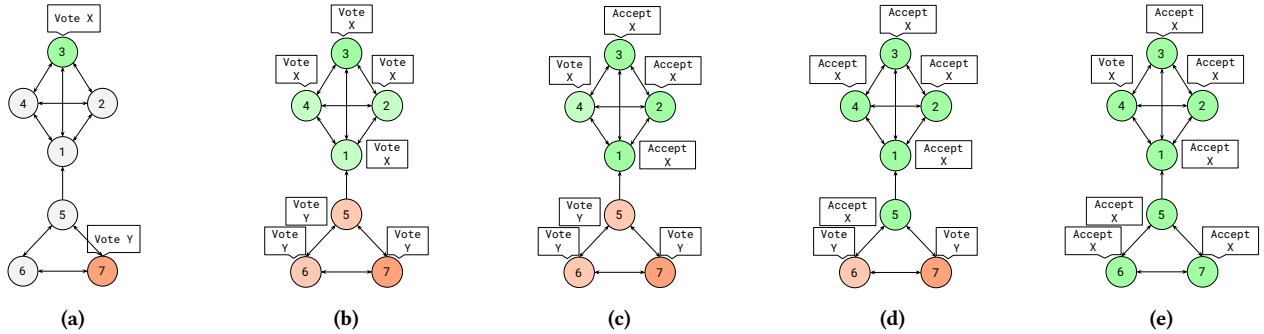

Figure 2. Cascade effect in federated voting. Each node has one quorum slice indicated by arrows to members of the slice. (a) Contradictory statements X and Y are introduced. (b) Nodes vote for valid statements. (c) Node 1 accepts X after its quorum $\{1, 2, 3, 4\}$ unanimously votes for X. (d) Nodes 1, 2, 3, and 4 all accept X; set $\{1\}$ is 5-blocking, so node 5 accepts X, overruling its previous vote for Y. (e) Set $\{5\}$ is 6- and 7-blocking, so 6 and 7 both accept X.

non-faulty node that could not accept contradictory statements. Confirmation of a statement is not guaranteed: in case of a split vote, both statements may be permanently stuck waiting for a quorum in the voting phase. However, if a node in an intact set I confirms a statement, the cascade theorem and accept case 2 ensure that all of I will eventually confirm that statement.

3.2.4 Ballot synchronization

If nodes are unable to confirm a COMMIT statement for the current ballot, they give up after a timeout. The timeout gets longer with each ballot so as to adjust to arbitrary bounds on network delay.

However, timeouts alone are not sufficient to synchronize ballots of nodes that did not start at the same time or got desynchronized for other reasons. To achieve synchronization, nodes start the timer only once they are part of a quorum that is all at the current (or a later) ballot n. This slows down nodes that started early and ensures that no member of an intact set stays too far ahead of the group. Moreover, if a node v ever notices a v-blocking set at a later ballot, it immediately skips to the lowest ballot such that this is no longer the case, regardless of any timers. The cascade theorem then ensures that all stragglers catch up. The result is that ballots are roughly synchronized throughout an intact set once the system becomes synchronous.

3.2.5 Federated leader selection

Leader selection allows each node to pick leaders in such a way that nodes generally only choose one or a small number of leaders. To accommodate leader failure, leader selection proceeds through rounds. If leaders of the current round appear not to be fulfilling their responsibilities, then after a certain timeout period nodes proceed to the next round to expand the set of leaders that they follow.

Each round employs two unique cryptographic hash functions, H_0 and H_1, that output integers in the range $[0, h_{\max})$. For instance, Stellar uses $H_i(m) = \text{SHA256}(i \| b \| r \| m)$, where b is the overall SCP instance (block or ledger number), r is the leader selection round number, and $h_{\max} = 2^{256}$. Within a round, we define the priority of node v as:

$$\text{priority}(v) = H_1(v)$$

One strawman would be for each node to choose as leader the node v with the highest priority(v). This approach works well with nearly identical quorum slices, but doesn't properly capture the importance of nodes in imbalanced configurations. For instance, if Europe and China each contribute 3 nodes to every quorum, but China runs 1,000 nodes and Europe 4, then China will have the highest-priority node 99.6% of the time.

We therefore introduce a notion of slice weight, where $\text{weight}(u, v) \in [0, 1]$ is the fraction of node u's quorum slices containing node v. When node u is selecting a new leader, it only considers neighbors, defined as follows:

$$\text{neighbors}(u) = \{\, v \mid H_0(v) < h_{\max} \cdot \text{weight}(u, v) \,\}$$

A node u then starts with an empty set of leaders, and at each round adds to it the node v in neighbors(u) with the highest priority(v). If the neighbors set is empty in any round, u instead adds the node v with lowest value of $H_0(v)/\text{weight}(u, v)$.

4 Formal verification of SCP

To eliminate design errors, we formally verified SCP's safety and liveness properties (see [65]). Specifically, we verified that intertwined nodes never disagree and that, under conditions discussed below, every member of an intact set eventually decides. Interestingly, verification revealed that the conditions under which SCP guarantees liveness are subtle, and stronger than initially thought [68]: as discussed below, malicious nodes that manipulate timing without otherwise deviating from the protocol may need to be manually evicted from quorum slices.

To ensure that the properties proved hold in all possible FBA configurations and executions, we consider an arbitrary number of nodes with arbitrary local configurations. This includes scenarios with disjoint intact sets, as well as potentially infinitely long executions. The drawback is that we face the challenging problem of verifying a parameterized infinite-state system.

To keep verification tractable, we modeled SCP in first-order logic (FOL) using Ivy [69] and the methodology of [82]. The verification process consists of manually providing inductive conjectures that are then automatically checked by Ivy. The FOL model of SCP abstracts over some aspects of FBA systems that are difficult to handle in FOL (e.g., the cascade theorem is taken as an axiom), so we verify the soundness of the abstraction using Isabelle/HOL [75].

After expressing the verification problem in FOL, we verify safety by providing an inductive invariant. The inductive invariant consists of a dozen one-line conjectures for about 150 lines of protocol specification. We then specify SCP's liveness properties in Ivy's Linear Temporal Logic and use the liveness to safety reduction of [80, 81] to reduce the liveness verification problem to the problem of finding an inductive invariant. While SCP's safety is relatively straightforward to prove, SCP's liveness argument is much more intricate and consists of around 150 single-line invariants.

Proving liveness required a precise formalization of the assumptions under which SCP ensures termination. We initially thought an intact set I would always terminate if all members removed faulty nodes from their slices [68]. However, this turned out to be insufficient: a well-behaved (but not intact) node in a quorum of a member of I can, under the influence of faulty nodes, prevent termination by completing a quorum just before the end of a ballot, thereby causing members of I to chose different values of x in the next ballot.

We must therefore additionally assume that, informally, each node in a quorum of a member of I eventually either becomes timely or doesn't send messages at all for a sufficient period. In practice, this means members of I may need to adjust their slices until the condition holds. This issue is not inherent to FBA systems: Losa et al. [47] present a protocol whose liveness depends on the strictly weaker assumptions of just eventual synchrony and eventual leader-election, without the need to remove faulty nodes from slices.

5 Payment network

This section describes Stellar's payment network, implemented as a replicated state machine [88] on top of SCP.

5.1 Ledger model

Stellar's ledger is designed around an *account* abstraction (in contrast to the more coin-centric unspent transaction output or UTXO model of Bitcoin). The ledger contents consists of a set of *ledger entries* of four distinct types: *accounts*, *trustlines*, *offers*, and *account data*.

Accounts are the principals that own and issue assets. Each account is named by a public key. By default, the corresponding private key can sign transactions for the account. However, accounts can be reconfigured to add other signers and deauthorize the key that names the account, with a "multisig" option to require multiple signers. Each account also contains: a sequence number (included in transactions to prevent replay), some flags, and a balance in a "native" pre-mined cryptocurrency called XLM, intended to mitigate some denial-of-service attacks and facilitate market making as a neutral currency.

Trustlines track the ownership of issued assets, which are named by a pair consisting of the issuing account and a short asset code (e.g., "USD" or "EUR"). Each trustline specifies an account, an asset, the account's balance in that asset, a limit above which the balance cannot rise, and some flags. An account must explicitly consent to holding an asset by creating a trustline, preventing spammers from saddling accounts with unwanted assets.

Know-your-customer (KYC) regulations require many financial institutions to know whose deposits they are holding, for instance by checking photo ID. To comply, issuers can set an optional AUTH_REQUIRED flag on their accounts, restricting ownership of the assets they issue to authorized accounts. To grant such authorization, the issuer sets an AUTHORIZED flag on customers' trustlines.

Offers correspond to an account's willingness to trade up to a certain amount of a particular asset for another at a given price on the order book; they are automatically matched and filled when buy/sell prices cross. Finally, *account data* consists of account, key, value triples, allowing account holders to publish small metadata values.

To prevent ledger spam, there is a minimum XLM balance, called the reserve. An account's reserve increases with each associated ledger entry and decreases when the ledger entry disappears (e.g., when an order is filled or canceled, or when a trustline is deleted). Currently the reserve grows by 0.5 XLM (~$0.03) per ledger entry. Regardless of the reserve, it is possible to reclaim the entire value of an account by deleting it with an AccountMerge operation.

A *ledger header*, shown in Figure 3, stores global attributes: a ledger number, parameters such as the reserve balance per ledger entry, a hash of the previous ledger header (actually several hashes forming a skiplist), the SCP output including a hash of new transactions applied at this ledger, a hash of the results of those transactions (e.g., success or failure for each), and a snapshot hash of all ledger entries.

Because the snapshot hash includes all ledger contents, validators need not retain history to validate transactions. However, to scale to hundreds of millions of anticipated accounts, we cannot rehash all ledger entry tables on every ledger close. Moreover, it is not practical to transfer a ledger

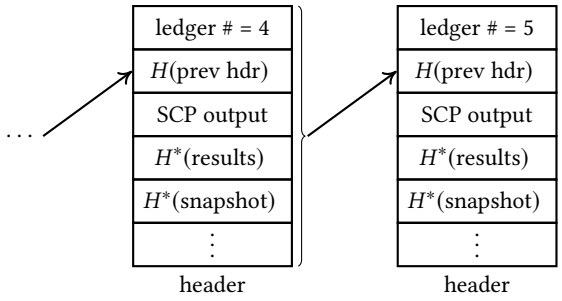

Figure 3. Ledger contents. H is SHA-256, while H^* represents hierarchical or recursive application of H. SCP output also depends the previous header hash.

CreateAccount	Create and fund new account ledger entry
AccountMerge	Delete account ledger entry
SetOptions	Change account flags and signers
Payment	Pay specific quantity of asset to dest. acct.
PathPayment	Like Payment, but pay in different asset (up to limit); specify up to 5 intermediary assets
ManageOffer	Create/delete/change offer ledger entry,
-PassiveOffer	with passive variant to allow zero spread
ManageData	Create/delete/change acct. data ledger entry
ChangeTrust	Create/delete/change trustline
AllowTrust	Set or clear AUTHORIZED flag on trustline
BumpSequence	Increase seq. number on account

Figure 4. Principal ledger operations

of that size every time a node has been disconnected from the network for too long. The snapshot hash is therefore designed to optimize both hashing and state reconciliation.

Specifically, the snapshot stratifies ledger entries by time of last modification in a set of exponentially-sized containers called *buckets*. The collection of buckets is called the *bucket list*, and bears some similarity to log-structured merge-trees (LSM-trees) [77]. The bucket list is not read during transaction processing (see Section 5.4). Hence, certain design aspects of LSM-trees can be relaxed. In particular, random access by key is not required, and buckets are only ever read sequentially as part of merging levels. Hashing the bucket list is done by hashing each bucket as it is merged and calculating a new cumulative hash of the bucket hashes (a small, fixed index of reference hashes) at each ledger close. Reconciling the bucket list after disconnection requires downloading only buckets that differ.

5.2 Transaction model

A transaction consists of a source account, validity criteria, a memo, and a list of one or more *operations*. Figure 4 lists available operations. Each operation has a source account, which defaults to that of the overall transaction. A transaction must be signed by keys corresponding to every source account in an operation. Multisig accounts can require higher signing weight for some operations (such as SetOptions) and lower for others (such as AllowTrust).

Transactions are atomic—if any operation fails, none of them execute. This simplifies multi-way deals. Suppose an issuer creates an asset to represent land deeds, and user A wants to exchange a small land parcel plus $10,000 for a bigger land parcel owned by B. The two users can both sign a single transaction containing three operations: two land payments and one dollar payment.

A transaction's main validity criterion is its sequence number, which must be one greater than that of the transaction's source account ledger entry. Executing a valid transaction (successfully or not) increments the sequence number, preventing replay. Initial sequence numbers contain the ledger number in the high bits to prevent replay even after deleting and re-creating an account.

The other validity criterion is an optional limit on when a transaction can execute. Returning to the land and dollar swap above, if A signs the transaction before B, A may not want B to sit on the transaction for a year before submitting it, and so could place a time limit invalidating the transaction after a few days. Multisig accounts can also be configured to give signing weight to the revelation of a hash pre-image, which, combined with time bounds, permits atomic cross-chain trading [1].

A transaction's source account pays a trivial fee in XLM, 10^{-5} XLM unless there is congestion. Under congestion, the cost of operations is set by Dutch auction. Validators are not compensated by fees because validators are analogous to Bitcoin full nodes, not miners. Rather than destroy XLM, fees are recycled and distributed proportionally by vote of existing XLM holders, which in retrospect might or might not have been worth the complexity.

5.3 Consensus values

For each ledger, Stellar uses SCP to agree on a data structure with three fields: a transaction set hash (including a hash of the previous ledger header), a close time, and upgrades. When multiple values are confirmed nominated, Stellar takes the transaction set with the most operations (breaking ties by total fees, then transaction set hash), the union of all upgrades, and the highest close time. A close time is only valid if it is between the last ledger's close time and the present, so nodes do not nominate invalid times.

Upgrades adjust global parameters such as the reserve balance, minimum operation fee, and protocol version. When combined during nomination, higher fees and protocol version numbers supersede lower ones. Upgrades effect governance through a federated-voting tussle space [34], neither egalitarian nor centralized. Each validator is configured as either governing or non-governing (the default), according to whether its operator wants to participate in governance.

Governing validators consider three kinds of upgrade: desired, valid, and invalid (anything the validator does not

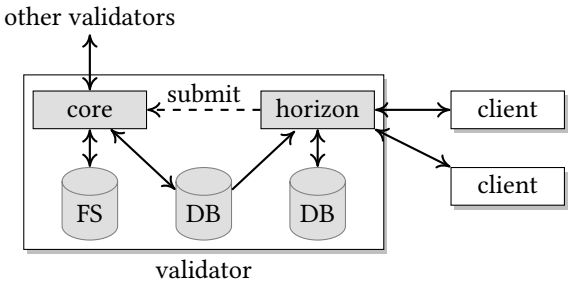

Figure 5. Stellar validator architecture

know how to implement). Desired upgrades are configured to trigger at a specific time, intended to be coordinated among operators. Governing nodes always vote to nominate desired upgrades, accept but do not vote to nominate valid upgrades (i.e., go along with a blocking quorum), and never vote for or accept invalid upgrades. Non-governing validators echo any vote they see for a valid upgrade, essentially delegating the decision on what upgrades are desired to those who opt for a governance role.

5.4 Implementation

Figure 5 shows Stellar's validator architecture. A daemon called *stellar-core* (~92k lines of C++, not counting third-party libraries) implements the SCP protocol and the replicated state machine. Producing values for SCP requires reducing large numbers of ledger entries to small cryptographic hashes. By contrast, transaction validation and execution requires looking up account state and order matching at the best price. To serve both functions efficiently, *stellar-core* keeps two representations of the ledger: an *external* representation containing the bucket list, stored as binary files that can be efficiently updated and incrementally rehashed, and an *internal* representation in a SQL database (PostgreSQL for production nodes).

Stellar-core creates a write-only *history archive* containing each transaction set that was confirmed and snapshots of buckets. The archive lets new nodes bootstrap themselves when joining the network. It also provides a record of ledger history—there needs to be some place one can look up a transaction from two years ago. Since history is append-only and accessed infrequently, it can be kept in cheap places such as Amazon Glacier or any service allowing one to store and retrieve flat files. Validator hosts typically do not host their own archives so as to avoid any impact on validation performance from serving history.

To keep *stellar-core* simple, it is not intended to be used directly by applications and exposes only a very narrow interface for the submission of new transactions. To support clients, most validators run a daemon called *horizon* (~18k lines of Go) that provides an HTTP interface for submitting and learning of transactions. *horizon* has read-only access to *stellar-core*'s SQL database, minimizing the risk of *horizon* destabilizing *stellar-core*. Features such as payment path finding are implemented entirely in *horizon* and can be upgraded unilaterally without coordinating with other validators.

Several optional higher-layer daemons are clients to *horizon*, rounding out the ecosystem. A *bridge* server facilitates integration of Stellar with existing systems, e.g., posting notifications of all payments received by a specific account. A *compliance* server provides hooks for financial institutions to exchange and approve of sender and beneficiary information on payments, for compliance with sanctions lists. Finally, a *federation server* implements a human-readable naming system for accounts.

6 Deployment experience

Stellar grew for several years into a state with a moderate number of reasonably-reliable full node operators. However, nodes' configurations were such that liveness (though not safety) depended on us, the Stellar Development Foundation (SDF); had SDF suddenly disappeared, other node operators would have needed to intervene and manually remove us from quorum slices for the network to continue.

While we and many others want to reduce SDF's systemic importance, this goal received increasing priority after researchers [58] quantified and publicized the network's centralization without differentiating the risks to safety and liveness. A number of operators reacted with active configuration adjustments, primarily increasing the size of their quorum slices in an effort to dilute SDF's importance; ironically this only increased the risk to liveness.

Two problems exacerbated the situation. First, a popular third-party Stellar monitoring tool [5] was systematically overestimating validator uptime by not actually verifying that *stellar-core* was running; this lead people to include unreliable nodes in their quorum slices. Second, a bug in *stellar-core* meant once a validator moved to the next ledger, it didn't adequately help remaining nodes complete the previous ledger in the event of lost messages. As a result, the network experienced 67 minutes of downtime and required manual coordination by validator administrators to restart.

Worse, while attempting to restart the network, simultaneous rushed reconfigurations on multiple nodes resulted in a collective misconfiguration that allowed some nodes to diverge, requiring a manual shutdown of those nodes and resubmission of the transactions accepted during the divergence. Luckily, this divergence was caught and corrected quickly and contained no conflicting transactions, but the risk of the network failing to enjoy quorum intersection—splitting while continuing to accept potentially conflicting transactions, simply due to misconfiguration—was made very concrete by this incident.

Reviewing these experiences led to two major conclusions and corresponding corrective actions.

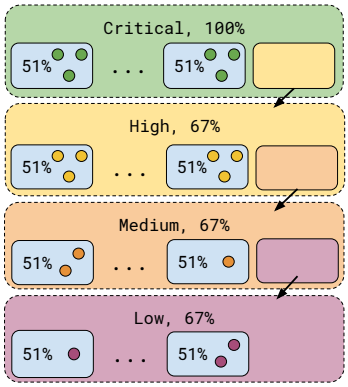

Figure 6. Validator quality hierarchy. Highest quality nodes require the highest threshold of 100%, whereas lower qualities are configured to 67% threshold. Nodes within a single organization require a simple 51% majority.

6.1 Configuration complexity and fragility

Stellar expresses quorum slices as nested *quorum sets* consisting of n entries and a threshold k where any set of k entries constitutes a quorum slice. Each of the n entries then is either a validator public key or, recursively, another quorum set.

While flexible and compact, we realized nested quorum sets simultaneously afforded node operators too much flexibility and too little guidance: it was easy to write unsafe (or even nonsensical) configurations. The criteria for grouping nodes into sets, for organizing subsets into a hierarchy, and for choosing thresholds were all insufficiently clear and contributed to operational failures. It wasn't clear whether to treat a "level" in the nested-set hierarchy as a level of trust, or an organization, or both; many configurations in the field mixed these concepts, in addition to specifying dangerous or meaningless thresholds.

We therefore added a simpler configuration mechanism that separates two aspects of nested quorum sets: grouping nodes together by *organization*, and labeling each organization with a simple trust classification (LOW, MEDIUM, HIGH, or CRITICAL). Organizations at and above HIGH are required to publish history archives. The new system synthesizes nested-quorum sets in which each organization is represented as a 51% threshold set, and organizations are grouped into sets with 67% or 100% thresholds (depending on group quality). Each group is a single entry in the next (higher quality) group, as illustrated in Figure 6. This simplified model reduces the likelihood of misconfiguration, both in terms of the structure of the synthesized nested sets and the thresholds chosen for each set.

6.2 Proactive detection of misconfiguration

Second, we realized that detecting collective misconfiguration by waiting to observe its negative effects is too late. Especially with respect to misconfigurations that can *diverge*—a more serious failure mode than halting—the network needs to be able to detect misconfiguration immediately so that operators can revert it before any divergence actually hapens.

To address this need, we built a mechanism into the validator software that continuously gathers the collective configuration state of all the peers in the node's transitive closure and detects the potential for divergence—i.e., disjoint quorums—within that collective configuration.

6.2.1 Checking quorum intersection

While gathering quorum slices is easy, finding disjoint quorums among them is co-NP-hard [62]. However, we adopted a set of algorithmic heuristics and case-elimination rules proposed by Lachowski [62] that check typical instances of the problem several orders of magnitude faster than the worst-case cost. Practically speaking, the current network's quorum slice transitive closures are on the order of 20–30 nodes and, with Lachowski's optimizations, typically check in a matter of seconds on a single CPU. Should the need arise to enhance performance, we may parallelize the search.

6.2.2 Checking risky configurations

Detecting that the network admits disjoint quorums is a step in the right direction, but flags the danger uncomfortably late for such a critical issue. Ideally, we want node operators to receive warnings when the network's collective configuration is merely *approaching* a risky state.

We therefore extended the quorum-intersection checker to detect a condition we call *criticality*: when the current collective configuration is one misconfiguration away from a state that admits disjoint quorums. To detect criticality, the checker repeatedly replaces each organization's configuration with a *simulated* worst-case misconfiguration, then re-runs the inner quorum intersection checker on the result. If any such critical misconfiguration exists one step away from the current state, the software issues a warning and reports the organization posing a misconfiguration risk.

These changes give the community of operators two layers of notice and guidance to insulate against the worst forms of collective misconfiguration.

7 Evaluation

To understand Stellar's suitablity as a global payment and trading network, we evaluated the state of the public network and ran controlled experiments on a private experimental network. We focused on the following questions:

- What does the production network topology look like? How many messages are broadcast on average, and how does SCP experience timeouts?
- Do consensus and ledger update latencies remain independent of the number of accounts?

- How are latencies affected by increasing (a) transactions per second (and, consequently, transactions per ledger), and (b) the number of validator nodes?
- What is the cost of running a node in terms of CPU, memory, and network bandwidth?

Payment networks have low transaction rates compared to other types of distributed system. The leading blockchains, Bitcoin and Ethereum, confirm up to 15 transactions/second, less than Stellar. Moreover, these systems take minutes to an hour to confirm a transaction securely, because proof-of-work requires waiting for several blocks to be mined. The non-blockchain SWIFT network averaged only 420 transactions per second on its peak day [14]. We therefore chose to compare our measurements against the 5-second target ledger interval, a more aggressive target. Our results show that latencies are comfortably below this limit even with several unimplemented optimizations still in the pipeline.

7.1 Anchors

The top traded assets by volume include currency (e.g., 3 USD anchors, 2 CNY), a Bitcoin anchor, a real-estate-backed security token [92], and an in-app currency [8]. Different anchors have different policies. For instance, one USD anchor, Stronghold, sets AUTH_REQUIRED and requires a know-your-customer (KYC) process for every account that holds their assets. Another, AnchorUSD, let's anyone receive and trade their USD (making it literally possible to send $0.50 to Mexico in 5 seconds with a fee of $0.000001). However, AnchorUSD *does* require KYC and fees to purchase or redeem their USD with conventional wire transfers. In the Philippines, where bank regulations are laxer for incoming payments, `coins.ph` supports cashing out PHP at any ATM machine [36]. In addition to the aforementioned security token and in-app currency, there's a range of non-currency tokens ranging from commercial bonds [22] and carbon credits [85, 96] to more esoteric assets such as a token incentivizing collaborative car repossession [35].

7.2 Public network

As of this writing, there are 126 active full nodes, 66 of which participate in consensus by signing vote messages. Figure 7 (generated by [5]) visualizes the network, with a line between two nodes if one appears in the other's quorum slices and a darker blue line to show bi-directional dependence. At the center is a cluster of 17 de facto "tier-one validators" run by SDF, SatoshiPay, LOBSTR, COINQVEST, and Keybase.

Four months ago, before the events of Section 6, there were 15 systemically important nodes: 3 from seemingly tier-one organizations and several random singletons. The graph also looked much less regular. Hence, the new configuration mechanism and/or better operator decisions seem to be contributing to a healthier network topology. Without great financial resources (and corresponding shareholder obligations), it would have been difficult to recruit 5 tier one organizations from the start, however. This suggests quorum slices play a useful role in network bootstraping: anyone can join with the goal of becoming an important player because there are no gatekeepers to pairwise agreement.

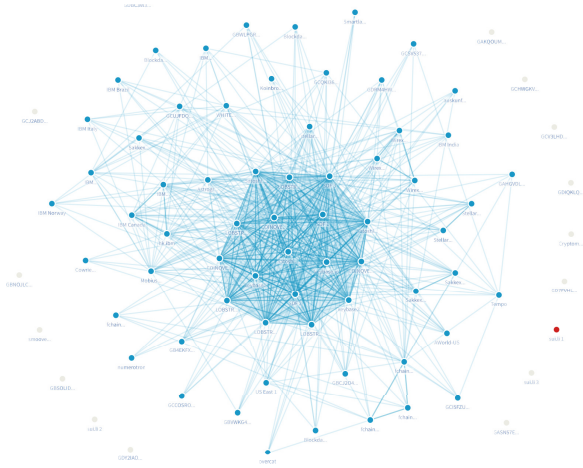

Figure 7. Quorum slice map

There are currently over 3.3M accounts in the ledger. Over a recent 24-hour period, Stellar averaged 4.5 transactions and 15.7 operations per second. Reviewing recent ledgers, most transactions seem to have a single operation, while every few ledgers we see transactions containing many operations that appear to come from market makers managing offers. The mean times to achieve consensus and update the ledger were 1061 ms and 46 ms, respectively. The 99th percentiles were 2252 ms and 142 ms (the former reflecting a 1-second timeout in nomination leader selection). Note SCP's performance is mostly independent of transactions per second, since SCP agrees on a hash of arbitrarily many transactions. Bottlenecks are more likely to arise from propagating candidate transactions during nomination, executing and validating transactions, and merging buckets. We have not yet needed to parallelize *stellar-core*'s transaction processing over multiple CPU cores or disk drives.

We also evaluated the number of SCP messages broadcast on the production network. In the normal case with a single leader elected to nominate a value, we expect seven logical messages to be broadcast: two messages to vote and accept a NOMINATE statement, two messages to accept and confirm a PREPARE statement, two message to accept and confirm a COMMIT statement, and finally, an EXTERNALIZE message (sent after committing a new ledger to disk to help stragglers catch up). The implementation combines confirm COMMIT and EXTERNALIZE messages as an optimization, since it is safe to externalize a value after it is committed. We then analyze metrics gathered on a production Stellar validator. Over the course of 68 hours, 1.3 messages/second were emitted, averaging to 6-7 messages per ledger. We note that the total

Percentile	Number of Timeouts	
	Nomination	Balloting
75%	0	0
99%	1	0
Max	4	1

Figure 8. Timeouts per ledger over 68 hours

count of messages broadcast by validators is larger, since in addition to federated voting messages, nodes also broadcast any transactions they learn about.

Figure 8 shows the timeouts experienced by a production validator over a period of 68 hours. Nomination timeouts are a measure of the (in)effectiveness of the leader election function, while ballot timeouts depend heavily on the network and potential message delays. The timeouts are consistent with the number of messages emitted: six messages in the best case scenario, and at least seven messages if an additional nomination round is needed.

7.3 Controlled experiments

We ran controlled experiments in containers packed onto Amazon EC2 c5d.9xlarge instances with 72 GiB of RAM, 900 GB of NVMe SSD, and 36 vCPUs. Each instance was in the same EC2 region and had a fixed bandwidth of 10 Gbps. We used SQLite as a store. (Stellar also supports PostgreSQL, but that has asynchronous tasks that inject noise into measurements.)

Stellar provides a built-in runtime query, *generateload*, that allows generating synthetic load at a specific target transaction/second rate. Although Stellar supports various trading features, such as order book and cross-asset path payments, we focused on simple payments.

Confirming transactions consists of multiple steps, so we recorded the measurements for each of the following:

- *Nomination*: time from nomination to first prepare
- *Balloting*: time from first prepare to confirming a ballot committed
- *Ledger update*: time to apply consensus value
- *Transaction count*: confirmed transactions per ledger

Each of our experiments was defined by three parameters: the number of account entries in the ledger, the amount of load (in the form of XLM payments) submitted per second, and the number of validators. We configured every validator to know about every other validator (a worst-case scenario for SCP), with quorum slices set to any simple majority of nodes (so as to maximize the number of different quorums).

Baseline Our baseline experiment measured Stellar with 100,000 accounts, four validators, and the load generation rate of 100 transactions/second. We observed 507 transactions per ledger on average, with standard deviation of 49 (9.7%). Note that no transactions were dropped; the slight

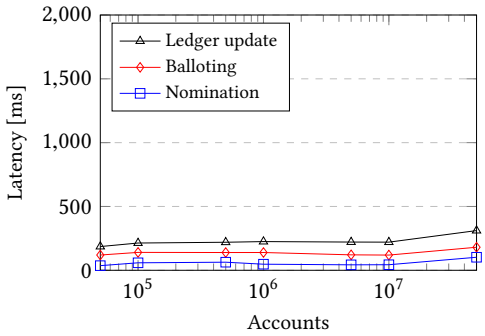

Figure 9. Latency as number of accounts increases

variance is due to scheduling limitations of the load generator. We observed that the number of transactions per ledger was consistent with our load generation rate, given ledger closing every 5 seconds. Nomination, balloting, and ledger update showed mean latencies of 82.53 ms, 95.96 ms, and 174.08 ms, respectively. We observed that nomination latency 99th percentile is consistently under 61ms, with occasional spikes of roughly 1 second, corresponding to the first step in the timeout function of leader selection.

Given the baseline performance, we looked at the effects of varying each of the test setup parameters.

Accounts The data in Figure 9 suggests that Stellar scales well as the number of accounts increases. Generation of test accounts became a lengthy process, as bucket creation and merging prevented us from simply populating the database with accounts directly via SQL. Therefore, we conducted our experiments for up to 50,000,000 accounts. While there is minimal impact on consensus and ledger update latencies, we note that increasing accounts creates an overhead of merging buckets, which get larger.

Transaction rate Transaction rate impacts the amount of traffic multicast among validators, the number of transactions included in each ledger, and the size of the top level buckets. To understand the effects of increasing transaction load, we ran an experiment with 100,000 accounts and 4 validators.

Figure 10 shows slow growth in the consensus latency, while the majority of time was spent updating the ledger. Not surprisingly, as the transaction set increases in size, it takes longer to commit it to the database. We also note that ledger update latency is heavily implementation-dependent, and is affected by the choice of the database.

Validator nodes To see how increasing the number of tier-one validators impacts performance, we ran experiments with 100,000 accounts, 100 transactions/second, and a varying number of validators from 4 to 43. All validators appeared in all validators' quorum slices; smaller quorum slices would have a lesser impact on performance.

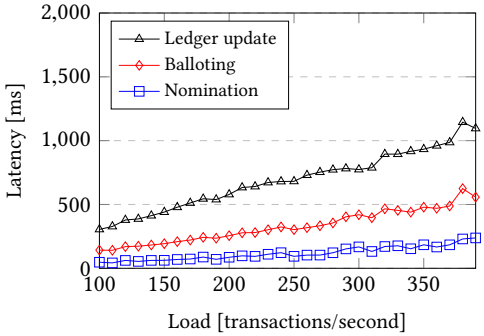

Figure 10. Latency as transaction load increases

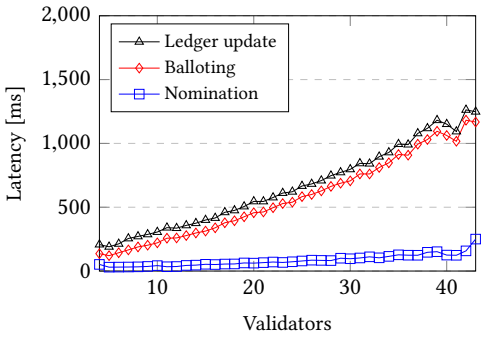

Figure 11. Latency as number of nodes increases

Changing the number of validating nodes on the network impacts the number of SCP messages exchanged as well as the number of potential values during nomination. Figure 11 shows nomination time growing at a relatively small rate. While the data suggests that balloting is the bottleneck, we believe many scaling issues can be addressed by improving Stellar's overlay network to optimize network traffic. As expected, ledger update latency remained independent of the number of nodes.

Close rate Lastly, we wanted to measure Stellar's end-to-end performance by measuring how often ledgers are confirmed and whether Stellar meets its 5-second target without dropping any transactions. We observed average ledger close times of 5.03 s, 5.10 s, and 5.15 s as we increased account entries, transaction rate, and number of nodes, respectively. The results suggest that Stellar can consistently close ledgers under high load.

7.4 Running a validator

One of the important features of Stellar is the low cost of running a validator, as anchors should run (or contract with) validators to enforce finality. SDF runs 3 production validators, all on c5.large AWS instances, which have two cores, 4 GiB of RAM and Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz processors. Inspecting resource usage on one of these machines, we observed the Stellar process using around 7% of CPU and 300 MiB of memory. In terms of network traffic, with 28 connections to peers and a quorum size of 34, the incoming and outgoing rates were 2.78 Mbit/s and 2.56 Mbit/s, respectively. Hardware required to run such a process is inexpensive. In our case, the cost is $0.054/hour or about $40/month.

7.5 Future work

These experiments suggest Stellar can easily scale 1–2 orders of magnitude beyond today's network usage. Because the performance demands have been so modest to date, Stellar leaves room for many straight-forward optimizations using well-known techniques. For example, transactions and SCP messages are broadcast by validators using a naïve flooding protocol, but should ideally use more efficient, structured peer-to-peer multicast [30]. Additionally, database-heavy ledger update time can be improved through standard batching and prefetching techniques.

8 Conclusion

International payments are expensive and take days. Fund custody passes through multiple financial institutions including correspondent banks and money transfer services. Because each hop must be fully trusted, it is difficult for new entrants to gain market share and compete. Stellar shows how to send money around the world cheaply in seconds. The key innovation is a new open-membership Byzantine agreement protocol, SCP, that leverages the peer-to-peer structure of the financial network to achieve global consensus under a novel *Internet hypothesis*. SCP lets Stellar atomically commit irreversible transactions across arbitrary participants who don't know about or trust each other. That in turn guarantees new entrants access to the same markets as established players, makes it secure to get the best available exchange rates even from untrusted market makers, and dramatically reduces payment latency.

Acknowledgments

Stellar would not be where it is today without the early leadership of Joyce Kim or the tremendous contributions of Scott Fleckenstein and Bartek Nowotarski in building and maintaining *horizon*, the Stellar SDK, and other key pieces of the Stellar ecosystem. We also thank Kolten Bergeron, Henry Corrigan-Gibbs, Candace Kelly, Kapil K. Jain, Boris Reznikov, Jeremy Rubin, Christian Rudder, Eric Saunders, Torsten Stüber, Tomer Weller, the anonymous reviewers, and our shepherd Justine Sherry for their helpful comments on earlier drafts.

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

References

[1] [n.d.]. Atomic cross-chain trading. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading.

[2] [n.d.]. Directo a México Frequently Asked Questions. https://www.frbservices.org/assets/resources/financial-services/directomexicofaq.pdf.

[3] [n.d.]. ERC20 Token Standard. https://theethereum.wiki/w/index.php/ERC20_Token_Standard.

[4] [n.d.]. PoET 1.0 Specification. https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html.

[5] [n.d.]. Stellar network visibility. https://stellarbeat.io.

[6] 2015. Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification.

[7] 2017. MobileCoin. https://www.mobilecoin.com/whitepaper-en.pdf.

[8] 2017. Mobius: A Universal Protocol Suite for the Blockchain Ecosystem and Real World Data. https://crushcrypto.com/wp-content/uploads/2017/11/MOBI-Whitepaper.pdf.

[9] 2018. The BitShares Blockchain. https://github.com/bitshares-foundation/bitshares.foundation/blob/master/download/articles/BitSharesBlockchain.pdf.

[10] 2018. ZenCash Statement On Double Spend Attack. https://blog.zencash.com/zencash-statement-on-double-spend-attack/.

[11] 2019. An Introduction to Libra. https://libra.org/en-US/white-paper/.

[12] 2019. PoW 51% Attack Cost. https://www.crypto51.app (retrieved Aug. 29, 2019).

[13] 2019. Remittance Prices Worldwide. Issue 30. https://remittanceprices.worldbank.org/sites/default/files/rpw_report_june_2019.pdf.

[14] 2019. SWIFT in Figures July 2019. https://www.swift.com/about-us/swift-fin-traffic-figures/monthly-figures.

[15] Hussam Abboud. 2018. The Realistic Lucrative Case of Ethereum Classic attack—Today. (May 2018). https://medium.com/@HusamABBOUD/the-realistic-lucrative-case-of-ethereum-classic-attack-with-1mm-today-8fa0430a7c25.

[16] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. 2017. Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space. Cryptology ePrint Archive, Report 2017/893. https://eprint.iacr.org/2017/893.

[17] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. 2014. Proofs of Space: When Space Is of the Essence. In *Security and Cryptography for Networks*, Michel Abdalla and Roberto De Prisco (Eds.). 538–557.

[18] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. 598–609. https://doi.org/10.1145/1315245.1315318

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

[20] Nicolas Barry, Giuliano Losa, David Mazieres, Jed McCaleb, and Stanislas Polu. 2018. *The Stellar Consensus Protocol (SCP)*. Internet-Draft draft-mazieres-dinrg-scp-05. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-mazieres-dinrg-scp-05.txt http://www.ietf.org/internet-drafts/draft-mazieres-dinrg-scp-05.txt.

[21] Thorsten Beck and María Soledad Martínez Peria. 2011. What Explains the Cost of Remittances? An Examination across 119 Country Corridors. *The World Bank Economic Review* 25, 1 (2011), 105–131.

[22] Bitbond. 2019. Securities Prospectus. https://www.bitbondsto.com/.

[23] Joseph Bonneau. 2018. Hostile blockchain takeovers. In *Proceedings of the 5th Workshop on Bitcoin and Blockchain Research*. http://www.jbonneau.com/doc/B18a-BITCOIN-why_buy_when_you_can_rent.pdf

[24] Gabriel Bracha. 1987. Asynchronous Byzantine Agreement Protocols. *Information and Computation* 75 (1987), 130–143.

[25] Danny Bradbury. 2013. Feathercoin hit by massive attack. http://www.coindesk.com/feathercoin-hit-by-massive-attack/.

[26] Vitalik Buterin. 2013. Bitcoin Network Shaken by Blockchain Fork. (March 2013). https://bitcoinmagazine.com/articles/bitcoin-network-shaken-by-blockchain-fork-1363144448/.

[27] Vitalik Buterin. 2014. Slasher: A Punitive Proof-of-Stake Algorithm. https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/.

[28] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. arXiv:1710.09437. http://arxiv.org/abs/1710.09437.

[29] Christian Cachin and Björn Tackmann. 2019. Asymmetric Distributed Trust. arXiv:arXiv:1906.09314 https://arxiv.org/abs/1906.09314.

[30] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. 2003. SplitStream: High-bandwidth Multicast in Cooperative Environments. In *Proceedings of the 19th Symposium on Operating Systems Principles*. 298–313.

[31] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems* 20, 4 (Nov. 2002), 398–461.

[32] Stephen Cecchetti and Kim Schoenholtz. 2018. The stubbornly high cost of remittances. https://voxeu.org/article/stubbornly-high-cost-remittances.

[33] Brad Chase and Ethan MacBrough. 2018. Analysis of the XRP Ledger Consensus Protocol. arXiv:1802.07242. https://arxiv.org/abs/1802.07242v1.

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

[35] REPO Coin. [n.d.]. Bringing the $1 billion auto repossession industry onto the blockchain. https://www.repocoin.io/docs/Repo%20Coin-WhitePaper.pdf.

[36] coins.ph. 2019. Cardless ATM Instant Cash Out. https://coins.ph/atm-cash-pickup/ (retrieved Aug. 29, 2019).

[37] crazyearner. 2013. TERRACOIN ATTACK OVER 1.2TH ATTACK CONFIRMD [sic]. https://bitcointalk.org/index.php?topic=261986.0.

[38] Jan Cronje. 2017. High bank charges force immigrants to send money home "hand-to-hand". https://www.groundup.org.za/article/high-bank-charges-force-immigrants-send-money-home-hand-hand/.

[39] Phil Daian, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919. https://eprint.iacr.org/2016/919.

[40] Ivan Damgård, Yvo Desmedt, Matthias Fitzi, and Jesper Buus Nielsen. 2007. Secure Protocols with Asymmetric Trust. In *Advances in Cryptology – ASIACRYPT 2007*, Kaoru Kurosawa (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 357–375.

[41] Kourosh Davarpanah, Dan Kaufman, and Ophelie Pubellier. 2015. NeuCoin: the First Secure, Cost-efficient and Decentralized Cryptocurrency. http://www.neucoin.org/en/whitepaper/download.

[42] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *Journal of the ACM* 35, 2 (April 1988), 288–323.

[43] Cynthia Dwork and Moni Naor. 1992. Pricing via Processing or Combatting Junk Mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. 139–147.

[44] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2013. Proofs of Space. Cryptology ePrint Archive, Report 2013/796. https://eprint.iacr.org/2013/796.

[45] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*. Santa Clara, CA, 45–59.

[46] Ittay Eyal and Emin Gün Sirer. 2013. Majority is not Enough: Bitcoin Mining is Vulnerable. http://arxiv.org/abs/1311.0243.

[47] Eli Gafni, Giuliano Losa, and David Mazières. 2019. Stellar Consensus By Instantiation. In *33nd International Symposium on Distributed Computing (DISC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[48] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015*. 281–310.

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

[50] David K. Gifford. 1979. Weighted Voting for Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles (SOSP '79)*. 150–162. https://doi.org/10.1145/800215.806583

[51] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. 51–68. https://doi.org/10.1145/3132747.3132757

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

[53] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2018. SBFT: a Scalable Decentralized Trust Infrastructure for Blockchains. arXiv:1804.01626v1. https://arxiv.org/abs/1804.01626v1.

[54] Dave Gutteridge. 2018. Japanese Cryptocurrency Monacoin Hit by Selfish Mining Attack. *CCN* (May 2018). https://www.ccn.com/japanese-cryptocurrency-monacoin-hit-by-selfish-mining-attack/.

[55] Carlos Hernández. 2019. Bitcoin Has Saved My Family. *The New York Times* (February 23 2019). https://www.nytimes.com/2019/02/23/opinion/sunday/venezuela-bitcoin-inflation-cryptocurrencies.html.

[56] Ari Juels and Burton S. Kaliski, Jr. 2007. Pors: Proofs of Retrievability for Large Files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. 584–597. https://doi.org/10.1145/1315245.1315317

[57] Flavio P. Junqueira, Keith Marzullo, Maurice Herlihy, and Lucia Draque Penso. 2010. Threshold protocols in survivor set systems. *Distributed Computing* 23, 2 (01 Oct 2010), 135–149. https://doi.org/10.1007/s00446-010-0107-3

[58] Minjeong Kim, Yujin Kwon, and Yongdae Kim. 2019. Is Stellar As Secure As You Think?. In *Proceedings of the IEEE Security & Privacy on the Blockchain*.

[59] Sunny King and Scott Nadal. 2012. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. http://peercoin.net/assets/paper/peercoin-paper.pdf.

[60] Joshua A Kroll, Ian C Davey, and Edward W Felten. 2013. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of the Workshop on th eEconomics of Information Security (WEIS)*.

[61] Jae Kwon. 2014. Tendermint: Consensus without Mining. http://tendermint.com/docs/tendermint.pdf.

[62] Łukasz Lachowski. 2019. Complexity of the Quorum Intersection Property of the Federated Byzantine Agreement System. https://arxiv.org/abs/1902.06493.

[63] Khin Thida Latt, Yasuhiro Ohara, Satoshi Uda, and Yoichi Shinoda. 2010. Analysis of IP Prefix Hijacking and Traffic Interception. *International Journal of Computer Science and Network Security* 10, 7 (July 2010), 22–31.

[64] Jinyuan Li and David Mazières. 2007. Beyond One-third Faulty Replicas in Byzantine Fault Tolerant Systems. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation*. 131–144.

[65] Giuliano Losa. 2019. Formal Verification of SCP. https://github.com/stellar/scp-proofs.

[66] Ethan MacBrough. 2018. Cobalt: BFT Governance in Open Networks. arXiv:1802.07240v1. https://arxiv.org/abs/1802.07240v1.

[67] Dahlia Malkhi and Michael Reiter. 1998. Byzantine quorum systems. *Distributed Computing* 11, 4 (01 Oct 1998), 203–213. https://doi.org/10.1007/s004460050050

[68] David Mazières. 2016. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. (February 2016). https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[69] Kenneth L. McMillan and Oded Padon. 2018. Deductive Verification in Decidable Fragments with Ivy. In *International Static Analysis Symposium*. Springer, 43–55.

[70] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. 31–42. https://doi.org/10.1145/2976749.2978399

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

[72] Moni Naor and Avishai Wool. 1998. The Load, Capacity, and Availability of Quorum Systems. *SIAM J. Comput.* 27, 2 (April 1998), 423–447. https://doi.org/10.1137/S0097539795281232

[73] Arvind Narayanan. 2015. Analyzing the 2013 Bitcoin fork: centralized decision-making saved the day. (July 2015). https://freedom-to-tinker.com/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/.

[74] Mark Nesbitt. 2019. *Deep Chain Reorganization Detected on Ethereum Classic (ETC)*. Technical Report. The Coinbase Block. https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de.

[75] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Vol. 2283. Springer Science & Business Media.

[76] Karl J. O'Dwyer and David Malone. 2014. Bitcoin Mining and its Energy Footprint. In *Irish Signals and Systems Conference*. 280–285.

[77] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (June 1996), 351–385.

[78] Oregon11. 2018. Re: Venmo to Paypal payments. https://www.paypal-community.com/t5/PayPal-Basics/Venmo-to-Paypal-payments/m-p/1520181#M15025.

[79] P4Titan. 2017. Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn. https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf.

[80] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, Mooly Sagiv, and Sharon Shoham. 2018. Reducing Liveness to Safety in First-Order Logic. In *45th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2018)*. Los Angeles.

[81] Oded Padon, Jochen Hoenicke, Kenneth L. McMillan, Andreas Podelski, Mooly Sagiv, and Sharon Shoham. 2018. Temporal Prophecy for Proving Temporal Properties of Infinite-State Systems. In *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 1–11.

[82] Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. Paxos Made EPR: Decidable Reasoning About Distributed Protocols. *Proc. ACM Program. Lang.* 1, OOPSLA (Oct. 2017), 108:1–108:31.

[83] José Parra Moyano and Omri Ross. 2017. KYC Optimization Using Distributed Ledger Technology. *Business & Information Systems Engineering* 59, 6 (01 Dec 2017), 411–423.

[84] Rafael Pass and Elaine Shi. 2017. *Thunderella: Blockchains with Optimistic Instant Confirmation*. Technical Report 2017/913.

Cryptography ePrint Archive. https://eprint.iacr.org/2017/913.pdf.
[85] Poseidon. 2019. Climate Rescue: Empowering People to Save the Planet with Every Purachase. https://poseidon.eco/assets/documents/Poseidon-Climate-Rescue.pdf.
[86] Protocol Labs. 2017. Proof of Replication. https://filecoin.io/proof-of-replication.pdf.
[87] Ling Ren and Srinivas Devadas. 2016. Proof of Space from Stacked Expanders. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography – Volume 9985*. 262–285. https://doi.org/10.1007/978-3-662-53641-4_11
[88] Fred B. Schneider. 1990. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (Dec. 1990), 299–319. https://doi.org/10.1145/98163.98167
[89] David Schwartz, Noah Youngs, and Arthur Britto. 2014. The Ripple Protocol Consensus Algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf.
[90] Hovav Shacham and Brent Waters. 2008. Compact Proofs of Retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08)*. 90–107. https://doi.org/10.1007/978-3-540-89255-7_7
[91] Isaac C. Sheff, Robbert van Renesse, and Andrew C. Myers. 2014. Distributed Protocols and Heterogeneous Trust: Technical Report. arXiv:1412.3136. https://arxiv.org/abs/1412.3136.
[92] Smartlands. [n.d.]. https://rise.smartlands.io/.
[93] Yee Jiun Song, Robbert van Renesse, Fred B. Schneider, and Danny Dolev. 2008. The Building Blocks of Consensus. In *Distributed Computing and Networking*, Shrisha Rao, Mainak Chatterjee, Prasad Jayanti, C. Siva Ram Murthy, and Sanjoy Kumar Saha (Eds.). 54–72.
[94] Robert H. Thomas. 1979. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems* 4, 2 (June 1979), 180–209. https://doi.org/10.1145/320071.320076
[95] Greg Thomson. 2019. Coinbase Sides with Bitcoin Cash (BCH) Miners on 51% Attack Smack. CCN. https://www.ccn.com/coinbase-bitcoin-cash-bch-miners-51-attack-smack/.
[96] Veridium Labs. [n.d.]. https://www.veridium.io/.
[97] Josiah Wilmoth. 2018. Bitcoin Gold Hit by Double Spend Attack, Exchanges Lose Millions. *CCN* (May 2018). https://www.ccn.com/bitcoin-gold-hit-by-double-spend-attack-exchanges-lose-millions/.
[98] Josiah Wilmoth. 2018. Privacy Coin Verge Succumbs to 51% Attack [Again]. *CCN* (May 2018). https://www.ccn.com/privacy-coin-verge-succumbs-to-51-attack-again/.
[99] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. HotStuff: BFT Consensus in the Lens of Blockchain. arXiv:1803.05069. https://arxiv.org/abs/1803.05069.