Safety vs. Liveness in the Stellar Network

David Mazières

April, 2019

Introduction

Recently three researchers from KAIST, Minjeong Kim, Yujin Kwon, and Yongdae Kim, published a paper called Is Stellar As Secure As You Think? Stellar is a public blockchain based on SCP, a protocol I designed to provide consensus in a new model I call federated Byzantine agreement or FBA. The contributions claimed by the KAIST researchers are:

1) We analyze FBA and prove that it is not superior to PBFT in terms of safety and liveness.

2) We conduct a data analysis on the Stellar system, and show that the structure of quorum slices is highly centralized.

3) We study cascading failures considering the current quorum slices. Our results imply that validators cannot achieve a consensus after deleting only two nodes run by the Stellar foundation.

While technically true at the time their paper was submitted, a more informative way of characterizing the conclusions would be:

1) We analyzed FBA and proved that if an FBA system's safety has more fault-tolerance than PBFT, then the FBA system's liveness will have less fault-tolerance than PBFT. However, it is easier to recover from typical liveness failures with FBA than with PBFT.

2) Stellar's FBA configuration is highly centralized.

3) Stellar's FBA configuration prioritizes safety over liveness.

Conclusions #1 and #3 are actually good things, while #2 is a limitation of how people currently use Stellar. We are in the process of improving #2, and are glad the authors drew attention to this fact. Things have already improved considerably from the configuration analyzed in the paper—for instance the Stellar Development Foundation (SDF) can no longer halt the network, and no two nodes can affect liveness. I just wish the paper had disentangled point #2 from points #1 and #3, the latter of which distract from addressing #2.

The goal of this blogpost is to explain what safety and liveness are in a Byzantine agreement

protocol and why Stellar needs greater fault-tolerance with respect to safety than with respect to liveness.

Byzantine agreement

Byzantine agreement is the problem of nodes in a distributed system deciding on a value when some of the nodes may be faulty and behave arbitrarily (e.g., be taken over by attackers who deviate from the protocol with the express goal of undermining agreement). As typically formulated, each node starts with an input value, nodes exchange a bunch of messages, and each node may at some point produce a single, write-once output value called the *decision*. A Byzantine agreement protocol should provide *safety* consisting of two properties:

- *Agreement* – No two non-faulty nodes should output different decisions.
- *Validity* – The decision should be valid (e.g., in Stellar should reflect correctly executing legitimate transactions). Moreover, the output should actually equal one of the nodes' inputs.

A Byzantine agreement protocol would ideally also guarantee *liveness*, meaning that all non-faulty nodes eventually terminate and output a decision. Unfortunately, achieving all three of safety, guaranteed termination, and fault tolerance turns out to be impossible without some additional assumptions. Stellar thus guarantees a weaker non-blocking liveness property, which guarantees termination under some timing assumptions after malicious nodes stop sending messages or have been removed from configurations. This blog post is about the fault-tolerance under which Stellar guarantees liveness, however, rather than the specifics of the liveness property.

Byzantine agreement protocols rely on a notion of quorum and an assumption that any two quorums share at least one non-faulty node, a property known as *quorum intersection*. A decision requires a unanimous quorum, and the quorum intersection property thus guarantees that no two quorums can decide different values. Quorum size has opposite effects on fault tolerance when it comes to safety and liveness: The larger a system's quorums, the more failures it can withstand while still guaranteeing quorum intersection. On the other hand, guaranteeing liveness actually requires that there be a non-faulty quorum—a.k.a. *quorum availability*—and the larger the quorum size the smaller the number of failures required to undermine availability. For example, as illustrated in the figure, a symmetric system with N nodes where any $T > N/2$ nodes constitute a quorum will have quorum intersection with up to $f_S = 2T - N - 1$ failures, and quorum availability with up to $f_L = N - T$ failures.

With limited exceptions, most closed Byzantine agreement protocols are tuned to the equilibrium point at which safety and liveness have the same fault tolerance. With symmetric quorums, this means setting $N = 3f + 1$ and $T = 2f + 1$ for some positive integer f, yielding $f_L = f_S = f$.

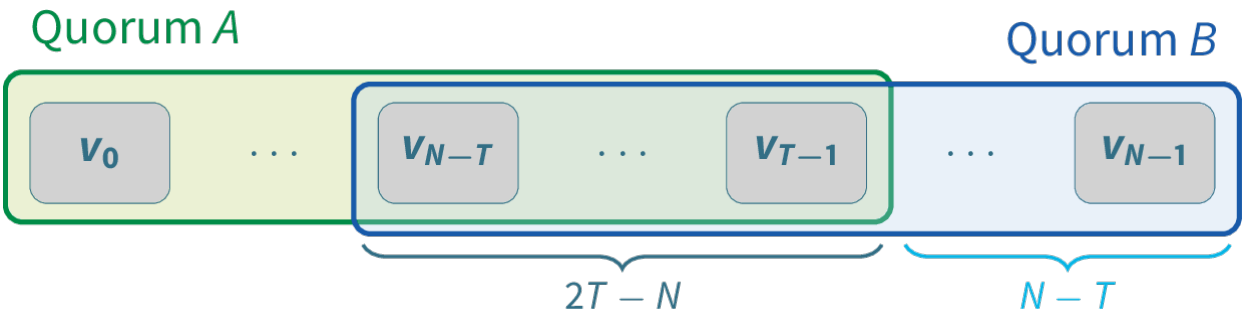

Figure 1: Symmetric Byzantine agreement system

Federated Byzantine agreement (FBA)

FBA, introduced by Stellar, generalizes Byzantine agreement to let different nodes recognize different quorums. Informally, we want to guarantee that each node agrees with the nodes it needs to agree with. Given transitive convergence of this "needs to agree with" relation, we get global agreement. Otherwise, we can get divergence, but only between nodes neither of which needs to agree with the other. Given the topology of today's financial system, we hypothesize that widespread convergence will continue to produce one ledger history that people call "*the* Stellar network," much as we speak of *the* Internet for the largest IPv4/v6 deployment.

Because we still want fault tolerance, FBA allows nodes to express redundant policies about whom they need to agree with. For example, a node might want to stay in agreement with at least one of the three nodes run by SDF, a majority of the nodes run by IBM, and two-thirds of nodes run by some consortium of anchors. A set of nodes that satisfies a node v's agreement requirements is called a *quorum slice* for v.

FBA then defines a *quorum* as a non-empty set of nodes that contains at least one *quorum slice* for each of its non-faulty members. If a quorum unanimously outputs the same decision, it by definition satisfies the agreement needs of all its non-faulty members, regardless of what non-members may decide. Faulty nodes have no agreement needs (they can output any value), so we ignore their slices in the definition.

A node specifies its quorum slices in every message it broadcasts. Hence, to be precise, in FBA a quorum isn't a set of nodes so much as a unanimous set of messages whose senders form a quorum under the specified quorum slices. Since nodes do not know which peers are faulty, a faulty peer may advertise slices that prevent honest nodes from recognizing some quorums. Conversely, however, any set of messages purporting to represent a unanimous quorum actually does.

Liveness in FBA

An FBA system generally will not end up with the same fault-tolerance with respect to both safety and liveness, meaning there are more failure scenarios in which non-faulty nodes stop

seeing new transactions than ones in which non-faulty nodes externalize conflicting decisions. One reason is that FBA determines quorums in a decentralized way, so it is unlikely that a bunch of individual slice choices will happen to arrive at equilibrium. More importantly in the case of Stellar, however, 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 take days through traditional non-blockchain payment rails). People therefore should and do select large quorum slices such that nodes are more likely to remain in agreement than live.

There's another important reason to prioritize safety over liveness with FBA, however, namely that it is far 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. But it's difficult to reach consensus when your consensus protocol 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 once they've confirmed it's not some kind of eclipse attack or catastrophic software bug that should be corrected before resuming operation.

Finally, one has to consider the fact that FBA is by it's very nature asymmetric. Very much by design, not all nodes have the same importance—this is why Sybill attackers can create quorums consisting entirely of fake nodes without undermining the safety or liveness of non-faulty nodes. The KAIST authors introduce a metric called (f, x)-fault-tolerance, where an (f, x)-FT configuration can survive the failure of any f nodes or any $x\%$ of all nodes. By this metric, the U.S. banking system would seem extremely precarious. After all, if the federal reserve takes down fedwire, the entire banking system will grind to a halt. So that makes the banking system only $(1, 0.0147)$-FT (assuming 6,799 banks). Yet in reality, banks fail all the time without affecting the overall liveness of the banking system. It matters *which* of the f banks are so important to liveness and how stable those important banks are.

Security of Stellar

Stellar's goal is to allow *anchors* (our term for asset issuers) to issue arbitrarily valuable digital assets. Someday a central bank might want to issue $1T worth of redeemable digital dollars. In such scenarios, it is critical that the issuer not face more redemption requests than they have issued digital assets. Stellar lets anchors protect themselves by running one or more validator nodes that participate in consensus. The anchor should advertise that they process redemptions through their own validators. Users who care about the value of those digital assets should then include the issuer's nodes in their quorum slices.

To the extent that Stellar is centralized, the question is whether the validators at the center of the network actually reflect the most important anchors and exchanges. This is currently a weak spot in the Stellar ecosystem. Not all anchors and exchanges run public validators, though many run private ones. Running a validator costs about $40/month on a cloud provider, which is a small price to pay for maintaining consistent, official records of a digital

asset you may be on the hook to redeem. Hence, I would encourage any anchor as well as any traditional crypto exchange processing Stellar assets (including XLM) to run a public validator and publish its public key in a `stellar.toml` file.

Of course, downtime does happen, so what if a set of important nodes goes down and the network halts? Ultimately, this is not unlike dealing with any large-scale federated system. For example, BGP failures have on occasion caused widespread Internet outages. In the face of such issues, administrators need to coordinate in forums such as the NANOG mailing list. When it comes to Stellar, anyone running an important validator (or who aspires to having their validator become important) should join the `#validators` channel of the `stellar.public` keybase group. Ultimately, if some important validator proves too unreliable, or if its administrator proves unresponsive to issues, other administrators will permanently remove the validator from their quorum slices. The unreliable validator will still stay in agreement with other nodes in its quorum slices, but the rest of the network will no longer wait on that validator to produce new ledgers.