6.857 - Final Project
Secure Transactions without Mining or Central Authority

Andre Saraiva, Bruno Almeida, Samuel Barroso
{andresan, bcalm, sfsouza}@mit.edu

May 2015

1 Introduction

In our project we implemented a system from transferring money without using Proof-of-Work, but consensus algorithms. First we implemented it using Paxos[11], that does not support Byzantine failures and needs a central authority to decide which nodes participate on the agreements. Then we reimplemented it using Stellar Consensus Protocol[2], that has no central authority and allows Byzantine nodes, given that Safety and Liveness assumptions are guarantee.

The actual Stellar Network has many features that we did not implement for simplicity. While at Stellar Network anyone is allowed to be a gateway, entities that can create their own coins, our system has only one gateway. This way, there is no need to implement Distributed Exchange. Also, we only put one transaction on each slot of our ledger.

This report is divided in 6 sections. Section 1 is this introduction. Section 2 gives an overview of existing models to implement a secure financial system with support to transactions. In section 3 we give an overview at Byzantine Agreements and Federated Byzantine Agreements. In section 4 we talk about how the Stellar Consensus Protocol works. Section 5 is about our implementation. Section 6 is the conclusion.

We will not prove any theorem of these systems. We recommend looking at the references.

2 Cryptocurrency

In our course we saw how Bitcoin[1] gives us a new way to transfer money using Proof-of-Work. The decentralized nature of Bitcoin is something desirable, but this model has some issues such as wasting a huge amount of resources on mining to assure security. Bitcoin is protected from 51% attacks because it is backed by

a huge computing power, once it is the leading cryptocurrency, but smaller ones have been victim of these kind of attacks [4] [5]. These 51% attacks are derived from the fact that, in Proof-of-Work, the participants don't have any other choice than trust that 51% of the computational power will behave correctly, i.e. it lacks flexible trust. Besides all this, Bitcoin has high latency: to put one block on the chain, it can take up to 10 minutes, and to be sure that your transaction will not be forgotten, the user has to wait up to 6 blocks on the chain, what means waiting for 1 hour. It also lacks asymptotic security, because it is possible to have computer power enough to break the system by doing 51% attacks.

Some alternatives to Proof-of-Work have been proposed such as Proof of Stake [6], Byzantine Agreement [9] and Tendermint [8]. We will not dive in more details of each one of these schemes, but there are some features of these system that are important to know.

Proof-of-Stake also lacks flexible trust. It is based on the fact that parties that posted collateral with behave correctly, because they receive rewards for doing so, but also because they can be penalized if they do not [7]. The parallel of 51% attacks on this scheme are the "nothing at stake" attacks, in which parties that previously posted collateral spend their money then can reverse the system to one state where they still had the money.

Byzantine Agreements seems to be a good solution. First, it solves the problem of flexible trust: someone with few resources can play a major role on ensuring the security of the system only because it was chosen to do so, not because it has a lot of computing power or financial resources. Second, it solves the problem of latency, once an agreement can be reached in a few seconds, something closer to the actual financial system. Third, it has asymptotic security: even if an extremely powerful enemy will not be able to convince the system of something unless it can break digital signatures. The problem is that the participants of the system must be chosen by a central authority, what can not be done in our actual financial system: there is no party trusted by everyone to play this role. If you let it open to anyone to join, it may suffer Sybil attacks [10], where an attacker can create a lot of peers to overrule the system, once, in these systems, an agreement is reached when a certain amount of peers, that constitutes a quorum, agrees on it.

Tendermint removes the central authority by tying Proof-of-Stake and Byzantine Agreement. A party have its vote on the agreement proportional to the collateral it has posted. The problem with this is that it lacks flexible trust again.

Ripple came out with the idea of letting the participants on the agreement to be able to agree on adding someone. It solve the problem of central authority and keeps the features of Byzantine Agreement. However, in practice, the participants were not willing to edit the list participants with fear of losing the security assumptions.

In this context, David Mazières from *Stellar Development Foundation* came with the idea of Federated Byzantine Agreement[2] in which we can have all good points from Byzantine Agreements and also have no central authority

to decide who participates on the system, without permitting Sybil attacks, however, differently from Ripple, anyone can join the system. We will talk more out SCP, that is an implementation o FBA, in section 3.

3 Federated Byzantine Agreements

In this section, we give some background to understanding the Stellar Consensus Protocol, once it is an implementation of Federated Byzantine Agreements. We start by explaining traditional Byzantine Agreements at section 3.1. In section 3.2 we show some important definitions in FBA and quickly discuss the security assumptions.

3.1 Byzantine Agreements

In a Byzantine Agreement System with N nodes, any T nodes constitutes a quorum. A quorum is a set of nodes that can reach consensus by themselves. When a agreement is reached the nodes can externalize the value, in our context, they can tell the users that their transactions are confirmed.

Differently from failures at Paxos, Byzantine failures can have random behavior, including lying. For example, one node can propose an invalid value, such as containing double spending transactions.

To make this kind of system usable as a financial system, we must have Safety and Liveness, described in sections 3.1.1 and 3.1.2. They limit the number of Byzantines nodes we can have in our system. The Sybil attacks [10] break these assumptions by inserting a lot of Byzantines nodes in the system.

3.1.1 Safety

To assure that two well behaved nodes will never externalize conflicting values, such as conflicting transactions, we need any two quorums always to share one good node. So the number of Byzantine failures has to be smaller than 2T-N.

3.1.2 Liveness

To assure that our system can always reach an agreement, i.e. will not be blocked, we need at least one entirely honest quorum to exists. So the number of Byzantine failures has to be at most N-T.

We can notice that there is a trade-off between Safety and Liveness by choosing T. The algorithm from Miguel Castro and Barbara Liskov [3] chooses T to be $\lfloor \frac{n+1}{3} \rfloor$ so that it can tolerate $\lfloor \frac{n-1}{3} \rfloor$ and still have both Safety and Liveness.

3.2 Federated Byzantine Agreement

The main idea of Federated Byzantine Agreements are the definition of quorum slices. Each node chooses its own quorum slices. If well behaved nodes chooses mainly well behaved nodes to their quorum slices, Safety and Liveness can be assured for all them. In this section we will give some definitions that are important to understand Stellar Consensus Protocol.

3.2.1 Quorum slices

Each node v can have any number of quorum slices. Each quorum slice, basically, is a set of nodes that v trust. When choosing its quorum slices, v has to keep in mind the same idea that the traditional quorums from Byzantine Agreements: once a quorum slice all agrees on something, v will can externalize it. Actually, it's not how it works, as we will see in the following sections, but if v keeps that in mind, the Safety assumptions will be achieved.

3.2.2 Quorum in FBA

In a FBA system, a quorum is not defined by the number of nodes, as in traditional Byzantine Agreements. Instead, a quorum is defined as a set of nodes S such that $\forall v \in S$, $\exists Q(v)$ such that $Q(v) \subset S$, where $Q(v)$ is a quorum slice of v.

3.2.3 Quorum intersection

Quorum intersection is a property of a set of nodes. Given a set of nodes S, for any $u, v \in S$, if Q, P are quorums such that $u \in P$ and $v \in Q$, then P and Q share at least one node.

3.2.4 Safety

The definition of Safety is the same of Byzantine Agreements: two well behaved nodes will never externalize conflicting values. We also need the same assumption: any two quorums always share one good node.

So, we need that after "deleting" all malicious nodes, we still have *quorum intersection*.

3.2.5 Liveness

As in Safety, the definition of Liveness is the same as before: the system can always reach an agreement. The assumption is the same too: at least one entirely honest quorum exists.

It is important to note that to an entirely honest quorum exists, each honest node needs at least one quorum slice only with honest nodes. As we will define later on section 3.2.5, we can not have a v-blocking set of malicious nodes for honest node v.

3.2.6 v-blocking set

For any given node v, a v-blocking set is a set that contains at least one node in each quorum slice of v. In other words, S is a v-blocking set if $\forall Q(v)$ quorum slice of v, $\exists u \in S$ such that $u \in Q(v)$.

4 Stellar Consensus Protocol

Given the definitions from section 3.2 we can now explain the SCP. First, we will explain how a round of federated voting works. Second, we will discuss how Ballots solve the situation in which the system can get stuck. Finally we explain how the protocol really works and which messages it exchanges.

4.1 Federated voting

One important stage of SCP if the Federated voting, since it is compound of a series of rounds of Federated voting. The Federated voting round if based on a statement. Each node can vote for or against that statement. We will call the statement a and its opposite $\bar{a}$.

Each Federated voting round has 4 states: voting, accepting, confirmed or stuck. Lets discuss each one of them. The definitions below can found at [12] and [2].

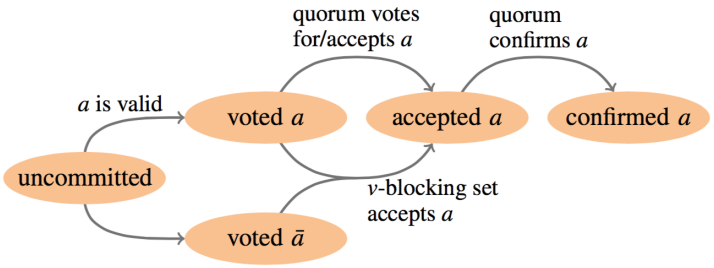

Figure 1: Possible states of a federated voting. Figure from [2]

4.1.1 Voting

Node v can vote for a or $\bar{a}$. If v is a well behaved node it will vote for a iff:

- a is valid and consistent with past statements v accepted
- v has never voted $\bar{a}$ and promises it never will

4.1.2 Accepting

If node v is a well behaved node it accept a iff:

- Each member of a v-blocking set claims to accept a, or
- A quorum containing v all either voted for or accepted a

4.1.3 Confirming

If node v is a well behaved node it confirm a iff:

- A quorum containing v all accepted a

4.1.4 Stuck

During one of the previous states, the system can get stuck. For example, if we have only one quorum and some nodes accepted a and some nodes accepted $\bar{a}$. That's why SCP does not vote only on each transaction x.

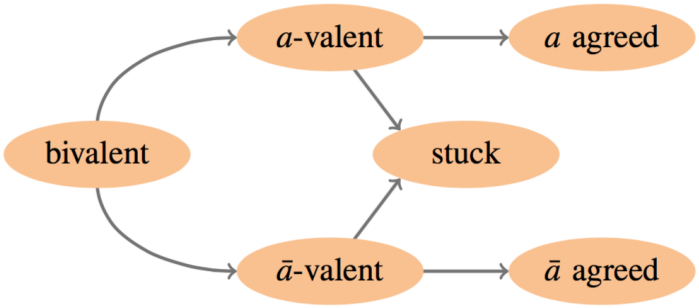

Figure 2: A system can get stuck. Figure from [2]

4.2 Ballots

Ballots $< n, x >$ are compound by a number n, equivalent to propose numbers at Paxos[13], and a value x, that can be the value to be inserted at given slot of the ledger history. SCP instantiate poll on the statements $PREPARE < n, x >$ and $COMMIT < n, x >$.

$PREPARE < n, x >$ means abort all ballots $< n', x' >$ such that $n' < n$ and $x' \neq x$. Well behaved nodes will only accept a commit at ballot b if it was prepared before.

If the system get stuck while trying to prepare or commit a given ballot b, the system just has to create a new ballot b' with a larger n.

4.3 The Protocol

This section is much better described at the Stellar Consensus Protocol white paper [2] at section 6.1. We strongly recommend you to read the original paper. All information here as extracted from it.

The protocol is independent for each slot i of the ledger. So, all we will discuss bellow if for only one slot. Each node stores 4 ballot $b, p, p' and c$ and the phase it is at the moment ϕ. b is the most recent ballot it has voted for prepare. p and p' are the most recent ballots with different values x that it has accepted. And c is the ballot it has voted to commit. ϕ can be PREPARE, FINISH, or EXTERNALIZE.

Each node also stores the most recent state it knows from each other node it knows about. So that it can know the conditions to accept or confirm preparing or confirming a ballot.

Note that, for a well behaved node, b, p, p' and c will be non-decreasing.

The nodes will exchange messages sending their state to the other nodes and will update their state with the following conditions:

1. If ϕ=PREPARE and a message allows v to accept that new ballots are prepared by either of accept's two criteria, then update p and p' accordingly. If afterwards $c \neq 0$ but p or p' invalidate c, then set $c = 0$.

2. If $b \neq c$, $b = p$, ϕ=PREPARE, and v confirms b is prepared, then set $c = b$.

3. If $b = p = c$, ϕ=PREPARE, and v accepts commit b, set ϕ=FINISH

4. If $b = p = c$ and ϕ = FINISH and v confirms commit b, then set ϕ=EXTERNALIZE and externalize the value in b. I.e. tell the user that the transaction of b was done.

Many points were not clear on the paper [2]. For example, when the ballot b can be accepted or confirmed. We figured out that, to confirm b, you have to look at the b of each node in a quorum that you participates have the same b as you. We discuss these issues in more details on section 5.3.

5 Implementation

We made two implementations: one using Paxos and another using SCP. Both can be plugged at the ledger packager.

Comparing with the actual Stellar Network, we made some simplifications. For example, the Stellar Network allows anyone node to be a gateway, i.e. can create their own coins, so they have to implement a Distributed Exchange to exchange these coins. In our systems we have only one entity that can insert coins, as the Stellar coin, and all transactions are in this coin.

Our system uses unix RPC calls to exchange messages and to talk with the client.

1. PREPARE $v\ i\ b\ p\ p'\ c\ D$
   This is a message from node v about slot i. D is v's quorum slice $\mathbf{Q}(v)$ or a collision-resistant hash of $\mathbf{Q}(v)$. The other fields reflect v's state. This concrete message encodes a host of conceptual statements, as follows:
   - $\{\, abort\ b_{\text{old}} \lor \text{accept}(abort\ b_{\text{old}}) \mid b_{\text{old}} \lesssim b \,\}$ (a vote to prepare b)
   - $\{\, \text{accept}(abort\ b_{\text{old}}) \mid b_{\text{old}} \lesssim p \,\}$ (a vote to confirm p is prepared)
   - $\{\, \text{accept}(abort\ b_{\text{old}}) \mid b_{\text{old}} \lesssim p' \,\}$ (a vote to confirm p' is prepared)
   - if $c = \mathbf{0}$ then $\emptyset$, else $\{\, commit\ c \,\}$ (a vote to commit c)
2. FINISH $v\ i\ b\ D$
   This is a message from node v attempting to confirm *commit* b for slot i, and may legally be sent only if $b = p = c$. Again, D is $\mathbf{Q}(v)$ or a collision-resistant hash of $\mathbf{Q}(v)$. The concrete message encodes:
   - Every statement implied by PREPARE $v\ i\ b\ b\ \mathbf{0}\ b\ D$
   - $\{\, \text{accept}(commit\ b) \,\}$

Figure 3: Meaning of the messages in SCP. Figure from [2]

Our code is hosted at GitHub and can be found at: `https://github.com/Andresnds/go-stellar`

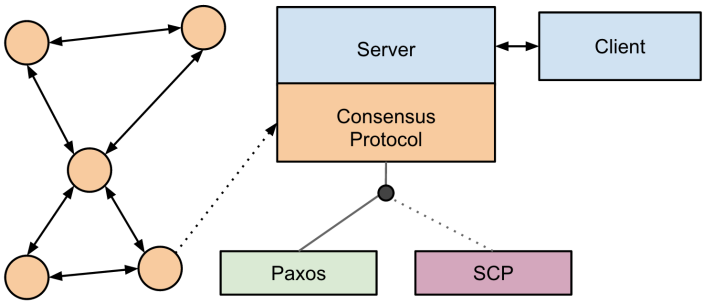

Figure 4: Basic architecture of our system. Each node has an instance of Ledger that can plugged with either Paxos or SCP

5.1 Ledger

The package ledger has an implementation of Server and Client. A client send RPC calls to with the transaction it wishes to make and the server uses the plugged consensus protocol, either Paxos or SCP, to put it in the distributed ledger. Each node verifies the signature and if that account has enough balance to make that transaction, if not, it rejects the transaction.

If the transaction seems valid but it was not possible to put it in a given slot because another transaction was inserted before. It just tries on the next slot and so on, until the transaction is inserted.

Each transaction is encoded as an operation that will be applied to the ledger sequentially, as a state machine.

When the client wants to query the balance of a given account, an get operation is also created, so that the ledger can be updated if it was not.

The account, as in Bitcoin [1], is defined by the public key used to verify its signatures.

5.2 Paxos

When creating a paxos instance, we need to pass all other server that are participating on the agreement. That's when we are playing the role of the central authority that defines the participants.

Each Paxos instance has an interface to be used by the server. It has methods to initiate reaching and agreement at a given slot and to get the status on a given moment. The server should wait until the status of a given slot is Decided.

5.3 SCP

The goal of using the SCP is the same as of the paxos: reaching a consensus on each slot of the ledger. But with SCP we have the advantages we already described earlier on this paper. The main challenge of implementing the Stellar Consensus Protocol, is the lack of information, given that it is a brand new algorithm and it's white paper is still a draft [2]. The procedure described in the paper fails to address many details, that are of utmost importance for the algorithm.

The major point that we had to figure out was how to send the messages. In a paxos-like algorithm, we have a proposer in charge of sending the messages to it's peers, but in SCP we concluded that it has a quite different style of communication: the messages are propagated, like in a chain reaction, until the messages starts fading when the nodes reach the commit point (which means, reaching ϕ=EXTERNALIZE). So basically in our implementation the proposer sends a message to it's peers, and for each peer that changes a state, they also send a message to all it's peers. When a node reaches the commit point, the behavior changes and it only replies to whoever sends him a message, instead of broadcasting that to everybody. With that, we are able to ensure that the proposer will eventually reach an agreement, and also that the messages exchanged will reach zero eventually.

Another important point that the paper does not explain is when and how update b. In the description of the SCP, we have 4 steps, which are described is section 4.3. If the reader is attentive, he will notice that at no moment it describes when the value of b would be updated. With that we created what we called the step 0, to fill this hole. We set b as the highest ballot seen, but compatible with c, so we maintain the invariance b c.

Also, for the steps 1 and 2, which are the steps in which we update p and c, it is not explained how they are updated. Given the definition of prepare, which is to abort any ballot less and not compatible, we could think about some optimizations, as the definitions of voting/accepting prepare cover more cases than that of voting/accepting to commit. In the end however, we noticed that the author expected for us to treat prepare as if it was a new statement in which we are voting, and that is one of the great ideas that he had, so we basically have two rounds of federated voting, while actually voting to abort various ballots and commit one. Given this conclusion, we proceed with the step 1 and 2 just like the step 3 and 4.

Another point that we needed to handle was when to propose a new ballot. For that we took an easy route: a timeout. So we have a goroutine that proposes a new ballot after a interval without reaching agreement.

In our implementation we could have more than one node proposing for the same slot on the ledger, but in they will all reach agreement on the same ballot, given that we meet the safe criteria.

6 Conclusion

Implementing SCP was a challenge, since the paper [2] was not clear in many points. We looked at the Stellar Foundation source code at github, but they made a lot of optimizations that we did not implement for simplicity, so that we ended up not really using it. We had to make ours own assumptions given what we read about SCP, but we are confident they are correct.

However, we were able to implement both consensus protocols and make transactions between accounts in each of them, that was the objective of this project.

6.1 Future work

We plan to create some malicious implementations of SCP nodes and see if the security assumptions hold in their presence. For example, they could accept any ballot and propose invalid ballots, like transactions with invalid signatures or from account with insufficient balance. If these transaction are externalized in good nodes, our attack will be successful.

6.2 Stellar

While preparing for this project, we had to read and understand a lot about the Stellar Network, and we are really hoping that it becomes very popular. A common misconception about Stellar is that it is a cryptocurrency to substitute Bitcoin, but we think it can be a bridge between Bitcoin and our actual financial system that can help Bitcoin to become more useful.

References

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

[2] David Mazières. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2015. https://www.stellar.org/papers/stellar-consensus-protocol.pdf

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance, 1999. http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

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

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

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

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

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

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

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

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

[12] David Mazières. Talk on The Stellar Consensus Protocol, April 21, 2015. https://www.stellar.org/wp-content/uploads/2015/04/scp-talk.pdf

[13] Leslie Lamport. Paxos Made Simple, Nov 2001. http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf