university of groningen / faculty of science and engineering

Experimental fault tolerance assessment of Replicated State Machines

Bachelor's Project Thesis

Wenxuan Huang, s2961296, w.huang.1@student.rug.nl,
Supervisors: Prof Dr D. Grossi

Abstract: From primitive trading to modern transaction based on digital currency, the methodology to implement 'trust' and credit reinforcement has evolved to adapt the form of transaction to counter exploitation. The implementation of 'trust' is increasingly essential since exploitation advanced. This paper aims to formulate an assessment procedure to test the level of 'trust' in financial platforms based on Federated Byzantine Agreement systems. The experiment is designed to trial the assessment's feasibility with two major replicated state machines under the Federal Byzantine Agreement systems. Additionally, 'trust' could be materialized by a measurable property networks possess while the experiment is centered around 'trust' evaluation (Byzantine fault tolerance). The result of a simulated transaction test provide indications on the effectiveness of some feature one network exclusively possess or the impact created by alteration of configurations of the network. Stellar, based on the result, has strong tolerance compared to Ripple with several feature advancement over Ripple.

1 Introduction

1.1 Evolution of transaction

Transaction is the constitutional mechanism in agreement-based economics. The mechanism functions between a buyer and a seller by exchanging a form of asset for payments. Replicated state machine is a form of transaction vehicle that leverages the Internet and blockchain technology.

Definition 1.1. The **state machine approach** is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas (Fred, 1990).

To further understand the idea of replicated state machine, this section explore the history of transaction from primitive phase to the application of digital asset.
Primitive transaction model quantifies the promise of future reimbursement based on intangible agreements between traders (Tymoigne and Henry, 2018). Moral principles dictate reinforcement of the agreement. With the implementation of authoritative intermediaries (e.g. bank and other financial institutes) and a credit system that promotes transactions between standardized currencies, it fosters demands of asset accumulation as intermediaries stabilize the value of credit as an asset (Stefan and Walter, 2002). The evolution of transaction from barter-based to credit-based agreement sees a shift from trust between traders to trust between the trader and authoritative institution.
Yet the transition further escalated trust-related issues. Firstly, trust placed on intermediaries needs to be reinforced since controlling the currency price could create a potential conflict of interest. More importantly, since assets are digitized into credit numbers, unauthorized modification, deletion and addition of credits could lead to credibility crisis.
In 1991, cryptography-backed structure 'block' is theorized as a securely hashed collection of information (Haber and Stornetta, 1991). The information includes a unique identifier of the last 'block', therefore forming a sequence of chronologically ordered state chain.

Definition 1.2. Blockchain is a sequence of connected blocks, which holds a complete list of transaction records like public ledger (Zheng, Xie, Dai, Chen, and Wang, 2017).

Since the validity of the block is verified by previous blocks while it guarantees safety of the next block, the information of previous blocks is generally resistant to modification and spoof (Nakamoto, 2008). Data stored in one block cannot be modified without changing subsequent chained blocks. The core principle of unmodifiable data sees a promising potential of improving credence between traders and central authorities.

1.2 Federated Byzantine Agreement

Blockchain information is decentralized through distributed networking in the form of replicate state machines. The register of blocks and their validation chain are available as copies stored separately by network participators. Different from protocols developed by current financial institutes, the decentralized network lacks authoritative participator as the decision maker, thus the power of authorization is distributed to selected or all participators, depends on the level of decentralization the network applies. By design, exploitation that targets single participator are often nullified or alleviated. Specifically, exploitation could be nullified since contaminating one or few copies of blockchain data cannot contaminate the final decision of that transaction outcome. It needs a form of majority consensus within the participators in order to externalize the final transaction outcome, instead of decisions from a single participators.
Federated Byzantine Agreement Systems (FBAs) is a model for the decentralized network. The transaction outcome is determined by consensus agreement between nodes in quorum and quorum slices. Since decentralized models lack central participator, every participator of the network in FBAs is a node of the network. Each node in the FBAs network agrees on one value for one transaction instance and exchange the value with other nodes. For each node to publish a value, it selects a set of nodes within the network as 'trusted peers' and if values published by 'peers' is consistent with the node's own proposal without any malfunctioned node, the value will be subsequently published. This set of nodes are referred to 'quorum slices'.

Definition 1.3. Federated Byzantine Agreement system is a protocol consisting of a loose confederation of a set of nodes and one or more quorum slices for each node(Mazieres, 2015).

While quorum slices determines the agreement of one particular node, the 'Quorum' is the set of nodes to reach an agreement on one transaction outcome for the network. Contrasting to other protocols based on Byzantine theory, Federated Byzantine Agreement Systems enable each node to select 'trusted peers' to form its 'quorum slices'. The network model of FBAs is implemented in both Stellar and Ripple, which are two major decentralized services used in personal and corporate finance.

1.2.1 Implementation of FBAs

Ripple network is a payment infrastructure within a decentralized network of established financial institutions. It is a production-level FBAs network. Stellar, on the other hand, is a PAAS (Platform-as-a-service) payment service which can be adapted to both corporate infrastructure and personal transaction. The security features of both Stellar Consensus Protocol and Ripple's Ledger Consensus Protocol are based on FBAs. They are both replicated state machines which distribute payment information on each node in the network by externalizing transactions to respective ledgers (Baliga, 2017). While Stellar adapts for live and personal transaction cases, Ripple serves as backbone of established financial institutions. Albeit the difference in scales, both apply deterministic rules on the validity of transactions and ledger state updates.
Additionally, evidence suggest that Stellar Consensus Protocol is designed to be more secure and tolerant to failure in actual implementation of the consensus model (Mazieres, 2015). Ripple Consensus is based on probabilistic voting (see Figure 2 in appendixC) where nodes reach agreements for the transaction outcome if the pre-configured percentage of nodes agree with the outcome (Schwartz, Youngs, and Britto, 2014) while Stellar has more customizable configurations available for each node in the network. The latter offers an adjustable percentage to cope with different purposes in terms

of failure tolerance (see appendixB). Compared to probabilistic voting, Stellar provides dynamic combinations of configuration to adapt for differnt applications of the consensus.
Stellar and Ripple also differ from the approach of decentralization.

Definition 1.4. Decentralized system, in system hierarchy, refers to the mechanism where lower level units operate locally and separately to perpetrate global tasks (Bekey, 2005).

Ripple consensus protocol operates in a permissioned ledger where the ledger validates transactions mainly based on Ripple's own list of nodes (Moreno-Sanchez, Modi, Songhela, Kate, and Fahmy, 2017). Ripple, as a primary party of the consensus network, acts as an authoritative node to evolve its ledger. Stellar, however, operates on a free ledger where any participator within the network who voluntarily labeled themselves as 'validator' can contribute to the consensus (Baliga, 2017). Since Stellar enables each node to be potential validator, there is no authoritative parties that centralize the network control. In this perspective, Stellar consensus is more decentralized than Ripple consensus.

1.3 Byzantine Fault Tolerance

In both Consensus protocol, Byzantine failure compromise both nodes and the consensus network. From the perspective of a decentralized consensus, a network failure refers to the failure of one or more nodes.

Definition 1.5. Byzantine failures are 'arbitrary deviations' of a mechanism from its assumed functions based on the running algorithm and the received inputs (Agbaria and Friedman, 2003).

The node is labeled as failed if the node is crashed or unresponsive. It entails that the node will either not publish any transaction outcome in the current cycle of the consensus process or to publish arbitrary, compromised outcome that does not accurately reflect the nature of the transaction (Driscoll, Hall, Sivencrona, and Zumsteg, 2003).
Node failure attributes to network failure if the number of failed nodes increment to a point where consensus cannot continue to provide services (which is to publish accurate transaction outcome in a transaction process). In this case, the failed part of the consensus network may be perceived as functional for other active nodes (Stelling, DeMatteis, Foster, Kesselman, Lee, and von Laszewski, 1999), which disrupts the formulation of consistent consensus agreement across the network. To design and operate consensus networks function as a infrastructure of financial institutions, node failure and subsequent network Byzantine failure can lead to data loss and value inconsistency.
In order to assess the robustness against Byzantine failure, a standardized assessment that tests failure resistance among FBAs-based protocols contributes to the overall quality evaluation for potential adopters of the decentralized network (Castro and Liskov, 2002). The main component of the assessment is to test the Byzantine fault tolerance for each consensus algorithm. Fault tolerance (see definition 1.3.) is an inherit property of replicated state machine as a representation of the concept of 'Trust' in the blockchain-backed network. It is an indication of the dependability in terms of failure resistance against node failures (Laprie, 1985).

Definition 1.6. Byzantine fault tolerance is a measure of subsistence that is possessed by distributed computer networks against node failures and incomplete information about node failures (Lamport, Shostak, and Pease, 1982).

In events where part of the network failed and the remaining network functions for its original service, fault tolerance enables the network to reach accurate consensus nevertheless (Castro and Liskov, 2002). The level of Byzantine fault tolerance reflects the robustness of consensus protocol since node failure is the main threat to correct consensus and value externalization (Srivatsa and Liu, 2004).
The thesis sought to evaluate Byzantine Fault Tolerance of Ripple Consensus Network and Stellar Consensus Network by assembling an assessment work-flow applicable to networks based on Federated Byzantine Agreement systems. Tolerance is to be evaluated by imitating actual transaction processes in this experiment.

1.4 Paper overview

From primitive trading to blockchain transaction, trust and consensus evolves into different forms while the assessments of trust and consensus adapt continuously.
This research project aims to evaluate the reinforcement of trust in the Federated Byzantine Agreement network through Byzantine fault tolerance assessment in blockchain-backed replicated state machines like Stellar and Ripple. In the Method section, exploratory researches including the data flow of the protocol. Additionally, the Method section discuss in specific the methodology to materialize the concept of tolerance into measurable elements in the assessment process, in order to provide a feasible solution to the aim of the research (research question). The Experiment section details the practical information in the assessment procedure, including modifications on parameters, system-wide test procedures and additional efficiency improvement on the flow of the experiment. The result of the experiment and pragmatic discussion on limitation, implication and potential improvement follows after the Experiment section, which conclude the paper.

2 Method

2.1 Related theories

This section will materialize previously-mentioned concepts and theories in mathematical expression that are applied by the production-level software.

Definition 2.1. Federated Byzantine Agreement system (FBAs) is a set of $< V, Q >$ consisting a collection of node and a quorum function Q where '$Q : 2^{2^V} \backslash \{\emptyset\}$' subject to all quorum slices of each node, where the node belongs to all its quorum slices (Mazieres, 2015).

$$\forall v \in V, \forall q \in Q, v \in q$$

Definition 2.2. Quorum is a set of nodes U in $< V, Q >$ iff $U \neq \emptyset$ and there is quorum slice for each member in U (Mazieres, 2015).

$$U \neq \emptyset, \forall v \in U, q \in Q \text{ such that } q \in U$$

Definition 2.3. Quorum intersection refers to the state where every two quorums of a FBAs network share a node (Mazieres, 2015).

$$\text{For all quorums } U_1 \text{ and } U_2,\ U_1 \cap U_2 \neq \emptyset$$

Definition 2.4. Correctness (Ripple) (under 80% threshold percentage) is reflected by the impossibility for a fraudulent transaction to be confirmed in the consensus, unless the number of faulty nodes exceeds that correctness. (Schwartz et al., 2014).

$$f \leq (n-1)/5$$

where:
f = Number of byzantine failures
n = Number of nodes in Unique Node List

2.2 Platform selection

The production-level software of both consensus protocol is subject to the assessment for this thesis. Production-level software refers to software that is designed for actual production and applications. To put in context, the assessment conducts on Stellar and Ripple software (Stellar-core and Rippled as software names respectively) that is already implemented by organizations and corporations (Sankar, Sindhu, and Sethumadhavan, 2017), instead of simulations and models build for research purpose. The choice of platform is based on:

1. **Relevance:** Byzantine Fault Tolerance on production-level software is one of the factors when corporations and solution providers select the optimal consensus protocol.

2. **Applicability:** Production-level software is potentially bounded to the limitation of current Blockchain and software technology (Udokwu, Kormiltsyn, Thangalimodzi, and Norta, 2018). Therefore, it is less ideal compared to theoretical models and simulations in terms of following guidelines and principles stated in their respective white papers. Thus, An assessment on production-level software provides insight for future improvement. Stellar-core and Rippled are selected since it is the official release of the organizations and most institutions use them as a template for

their blockchain services. These two software are open-sourced software that provide viable methods for developers to access internal messages for testing and debugging.

2.3 Data flow

The data flow (see Figure 1 in appendixC) of each transaction contains a state, an input, and an output. The ledger is a state recorder while transaction and history record is generated as one transaction outcome. The transaction will be kept in the history of the ledger note. By tracing the input, output and the intermediate messages from both Ripple and Stellar, it connects fault tolerance as a concept and its implementation in the production-level software. Ledger messages and outcome generated can be used to identify potential Byzantine failures (Cachin, 2016). Since Stellar-core and Rippled wrapped state, input and output files into External Data Representation, it is not accessible in plain text. Instead, an SQLite database is implemented to export messages and outcome in the production level software and transaction data stored for each node will be shown in the database viewing tools. In this case, ledger state, input, and output data of the transaction will be accessible in plain text to provide insight into the possible approach for tolerance assessment.

2.4 Materialization of tolerance

In a decentralized network, each node is either well-behaved or ill-behaved. A well-behaved node chooses reasonable 'trusted peers' to form quorum slices and actively processes requests from the protocol (Mazieres, 2015). Ill-behaved nodes suffer from node crash and therefore not able to fulfill the responsibilities of the protocol. Byzantine fault tolerance reflects a measurable resistance against the effect of ill-behaved nodes for consensus network based on FBAs. Even for well-behaved nodes, Byzantine failure can still take place when nodes cannot output value (blocked nodes), or output inconsistent values (divergent nodes) (Fischer, Lynch, and Paterson, 1985). Since node failure of either ill-behaved and well-behaved node is a major contributor of Byzantine failure, fault tolerance of the network can be seen as an indication of the robustness where the network may have against node crashes. By exploring the data flow(see Figure 1 in appendixC) from the database example created, several outputs of the software provided a significant indication of Byzantine failure. The specific instances of indication will be mentioned in later sections. In summary, the indication of fault tolerance in the implementation of both Ripple and Stellar protocol can generally be divided into two components - Safety and Liveness. Therefore, the fault tolerance could be operationalized by evaluating the Safety and Liveness performance of both protocols.

2.4.1 Safety

Definition 2.5. A set of nodes in the Federated Byzantine Agreement System enjoys **Safety** if no two nodes ever externalize different value for the same slot (Mazieres, 2015).

For one transaction instance, there can only be one value externalized, thus Safety ensures the value consistency of the decentralized network. An indication of Safety failure could be reflected by any one of the three types being modified without consistency. In case of a ledger file, the inconsistent hash value of entries $accountID$ and $ledgerHeader$ of the SQLite database may lead to divergence (The database serves only as a container of one particular transaction and is replaced by subsequent transaction). In the database, the consistency of entries $sellerID$, $offerID$, $assetCode$ and $assetType$ also relfects the integrity of data. Moreover, the entries $createEntry$ and $modifyEntry$ will give a clear picture of whether divergence exists in the process of externalizing transaction outcome by consensus. By combining the information retrieved from several data entries, evaluator can visualize potential node failure in a transaction process, and whether it is caused by the divergent node.

2.4.2 Liveness

Another instance of node failure is the lack of liveness in the quorum. Liveness, in the real implementation, can be the property of one node, or the network as an entity.

Definition 2.6. In principle, a node enjoys

liveness if it can externalize new values without the participation of any failed (including ill-behaved) nodes (Mazieres, 2015).

For one transaction instance, there need to have a sufficient amount of nodes with liveness in order for a quorum to externalize a value outcome (Cachin and Vukolić, 2017). This paper does not concern individual liveness in this experiment, since it is part of the liveness of the quorum, and it is unable to be identified in the production-level software. The experiment identifies liveness for overall quorum through terminal messages for each transaction round. A terminal message is a built-in messaging system in both Stellar-core and Rippled. Its purpose is to notify users whether one transaction is successful by giving human-readable phrases (*Succ*, *Fail* or *Miss*). The phrase 'Miss' exclusively indicates node failure of the consensus network within one round of transaction, indicates that the number of nodes with sufficient liveness for externalizing value does not meet the requirement.

3 Experiment

3.1 Test configurations

Safety and liveness of the quorum determine the Byzantine Fault Tolerance of the consensus network. Since node configurations in Ripple and Stellar affect safety and liveness of these networks, this experiment aims to evaluate the fault tolerance of both networks in alternating configuration sets. Node configuration is a configuration file collection that determines the properties of one node. It controls peers the node trust as quorum slices, the threshold for the node to publish a value outcome, and the eligibility of the node being a validation node. There are two functions of node configuration:

- Node configuration affects the level of safety and liveness the node possess. In this context, varying a diversity of node configurations in the same software (Ripple or Stellar) test the fault tolerance of one consensus network in different node condition.

- The experiment also tests the correlation between the flexibility of node configuration and fault tolerance. Secondly, under the same (or at least similar) configuration, it enables accurate comparison between the fault tolerance of two consensus networks, since the confounding variables of two consensus networks are alleviated by the same configuration setup.

The configuration file (see appendixB) consists of several parameters to be edited by users. The node configuration file for Stellar and Ripple are similar in general. The difference will be detailed in the Test procedure section of this paper. Here is a list of the configurable parameters in the file that potentially modify the safety and liveness of the network:

- **Target Peer Connections:** Minimum numbers of trusted peers the node will connect during one consensus.

- **Known Peers:** A list of trusted peers known by the node that will be called if connected nodes are fewer than Target Peer Connections (Back-up of the quorum slice members).

- **Node Validator:** A binary switch for the validator eligibility for this node.

- **Failure Safety:** Maximum numbers of failures from the trusted peers that forms a quorum slice for this node. If the number of failures in one transaction is more than the number, the procedure will not run.

- **Unsafe Quorum:** A binary switch for allowing theoretically unsafe quorum slice set to form. This is set to false for this experiment since the set of configurations used in the test are all theoretically safe, while the actual performance of safety needs to be determined under either Stellar or Ripple.

- **Validators/Unique Node List:** A set of trusted peers that forms a quorum slice of one node.

- **Threshold Percentage:** The threshold percentage indicates the number of trusted peers in the quorum slices that needs to agree on a value before the node publish the value outcome. (e.g. If the percentage is set to 50%, half of the nodes in the quorum slice needs to agree on the same value before the node can externalize that outcome value to the network).

3.2 Consensus procedure

3.2.1 Selection of quorum slices

The selection of quorum slices per node in Ripple and Stellar production-level software is similar. In Ripple, the quorum slices per node are selected through Unique Node List. In the Unique Node List for each node, a collection of other nodes is presented through the public key of those nodes . There are several principles for choosing the member of the Unique Node List (Schwartz et al., 2014):

- The nodes have to be maintained by different organizations (i.e. the certificate of each node should be as decentralized as possible).
- The Unique Node list is recommended to be the nomination of the nodes trusted peers and nodes suggested by the starter Unique Node List which will include the nodes trusted by Ripple organization itself. Each node in the Unique Node List will be assigned a Trust Score by the node who owns the list, which indicates the level of trust the node has on each of these nodes.
- Nodes should be chosen to reflect different political affiliation, regime, legality and corporate category. The same applies to Stellar.

More importantly, for each node, the quorum slices are directly listed in the configuration file each node has.
The selection of quorum slices for Stellar Consensus protocol is also mainly manual, with recommendations from Stellar organizations and other institutions. In Stellar, there is a dedicated dashboard that recommend viable and responsive node candidate for quorum slices. The same principle that applies to Ripple equally applies to Stellar. In the actual protocol, there is no hierarchy between nodes. Yet Stellar organizations introduced the concept of Tier where some reputable nodes have a higher priority in terms of the popularity of being selected as nodes quorum slices (Wang, Vergne, and Hsieh, 2017). The priority here is perceived by users instead of imposed by consensus protocol itself.
The selection of quorum slices is manual in the actual implementation of both Ripple and Stellar (Baliga, 2017), and a set of principles and lists of recommended nodes from both official organizations and communities.

3.2.2 Voting procedure

The transaction is accepted by consensus protocol as input and the verification and acceptance of the input as part of the ledger record are processed by the consensus network (Peters and Panayi, 2016). The voting procedure for Stellar take place on two levels:

- **Node level voting:** The externalization of a node refers to the process of a node publishes a transaction outcome (Mazieres, 2015), in the financial perspective. The externalization requires the node to consult the value externalized by peers in its quorum slices. In Stellar, the configuration 'Threshold percentage' can be set by its users and testers, and if the number of peers that agrees with the value is more than that percentage, the node will externalize the value for that transaction. Therefore, the consistency of quorum slices of each node determines the voting procedure on the node level.
  The implementation of node voting is through Stellar's internal message function and configuration files. The network operates on Horizon network, which provide the message exchange capability between peers in the quorum slices. The threshold percentage is specified in the configuration file. The decision of externalization is summarized as follows:

  $$AP = \frac{N_{externalize\ candidate}}{N_{quorumslice}} \tag{3.1}$$

  where:
  AP = The percentage of peers in the quorum slices that agree on the value proposed by the node
  $N_{externalize\ candidate}$ = numbers of peers in the quorum slices that agree on the value proposed by the node
  $N_{quorumslice}$ = numbers of peers in the quorum slices

  If AP is larger than the 'Threshold percentage' value configured in the configuration file, the value can be externalized by the node.
- **Network level voting:** The network (quorum) observes the value externalization of each node constantly until every node cease

to externalize. The voting procedure of the consensus network determines the value that the network will recognize for that transaction. First, each node in the network 'accepts' the transaction outcome.

Definition 3.1. A node in the consensus network **accepts** a value for one transaction if and only if it had not accept any value that contradict the value, and all nodes (including this node) in a quorum accept this value (Mazieres, 2015).

If a value is accepted by every node in the quorum, the network voting procedure moves on to the 'confirm' phase.

Definition 3.2. A consensus network **confirms** a value for one transaction if and only if every node in the quorum claims to accept the value for that transaction (Mazieres, 2015).

If a value is confirmed for that transaction, it will be externalized by the network as the verified copy of input that will be recorded in the ledger.

The voting procedure of Ripple consensus protocol is similar to Stellar, since both are based on FBAs. There are three differences between Ripple and Stellar:

- **Flexibility:** Unlike Stellar, the threshold percentage for quorum slices cannot be altered by testers and users. It is not available as a entry in the configuration file.
- **Vote counting:** While Stellar counts the number of agreed nodes in the quorum slices, Ripple node observe multiple candidate the peers have, and count the candidates separately, in the implemented voting procedure of the software (Schwartz et al., 2014). The peer node can have multiple candidate values, and each value will be counted. The value with the highest AP ratio will be externalized.
- **Ballot system:** In the network level, stellar consensus network has an additional ballot network. The ballot network contributes to Byzantine failure tolerance towards live-less nodes (Saraiva, Almeida, and Barroso, 2015). If one node is blocked by its quorum slice, Stellar consensus allows node to remove the statement that is blocking the node to externalize a value.

Definition 3.3. The node has a **blocked** state, if and only if sufficient peers from every quorum slices the node has agree on different values than the value node intend to externalize (Bracha and Toueg, 1985).

In actual implementation, if Stellar-core observed a blocked node with inconsistent quorum slices output, a value candidate is removed in a random selection in order for the node to start externalizing values.

3.3 Test procedures

In the context where both node configuration and consensus network are ready for the experiment, Stellar and Ripple nodes start to process transaction information within the data flow. As mentioned before (Section 1.2), as state machines, consensus network keep their ledger up to dates through accepting inputs as transactions. The transaction will be applied to the ledger, thus forming a new version of the ledger. As a result, a history outcome will be generated by the application process. Within the data flow of the transaction, there are three checkpoints. Checkpoints are internal tools Stellar-core and Rippled provided to determine whether the network has adequate safety and liveness level. Three checkpoints indicate the different context of safety/liveness level, and ultimately fault tolerance:

1. **Bucket Apply checkpoint:** The checkpoint is used to determine whether a transaction input can form before applies it to the ledger. Only a 'Succ' flag of the checkpoint indicates that the transaction is accepted as an input. This is the premise of any possible consensus, and will be considered when evaluating fault tolerance of the network.
2. **Ledger Apply Checkpoint:** The checkpoint is used to determine whether the input transaction has quorum agreement. Only if the quorum reaches agreement on the input value

for one transaction instance, the flag of this checkpoint will be 'Succ'. This checkpoint reflect directly on the performance of fault tolerance, since either insufficient safety or liveness could compromise quorum agreement. If network safety is compromised, the flag of the checkpoint will be 'fail' due to the lack of unanimous agreement under well-behaved nodes in the quorum. Similarly, the flag will be 'missed' if the quorum does not generate agreement message, indicates network liveness is compromised.

3. **Catch-up Checkpoint:** Catch-up is a built-in function of both Stellar-core and Rippled for keeping ledgers up-to-date. In production level software, the histories of a transaction are stored in the blockchain, and the latter will be downloaded and applies to the ledger before new transaction input get validated. This is also the premise of any possible consensus since new transaction must be processed after all old transaction is scripted in the ledger for consistency. If in a given time (time can be modified in each node configuration), the ledger of the node did not manage to catch-up to latest state, the ledger will not apply that transaction instance.

3.3.1 Transaction model

The experiment uses a terminal message from three checkpoints to determine Byzantine Fault Tolerance of one consensus network. Several pieces of research have been conducted for Byzantine failure in business transaction instances with various limitations in terms of controlling confounding variables (Castro, Liskov, et al., 1999; Silva, Prata, Rela, and Madeira, 1998; Haeberlen, Kouznetsov, and Druschel, 2007). Since transaction inputs generated by working and test network varies in both the availability and transaction amount, calculation for the probability of successful transaction based on available transactions (in total capacity of aproximately 18 billion lumens available for transaction) (stellar.org/stats) could produce an inconsistent conclusion. For the experiment of evaluating Byzantine fault tolerance, a simulated transaction package is made as input transactions. There are two reasons why using a simulated transaction is feasible:

- A simulated transaction model creates a similar setup for both Stellar and Ripple, and therefore checkpoint message from both Ripple and Stellar are more comparable. A simulated transaction does not indicate a simulated model or experiment since we are still testing tolerance on production level software.
- Furthermore, simulated transaction alleviate confounding variables between a large number of transactions that are used for the experiment. The simulated transactions share the same node publisher, same hash value for the transaction outcome, and most importantly, same public and private key for verification. It makes sure that the transaction instance only fails because of either safety and liveness issues before, in and after consensus.

3.4 Test variation

The experiment sets up a transaction model with simulated transaction instance. In order to determine the relationship between node configuration and the probability of consensus agreement (fault tolerance), the test is conducted in 5 different configuration environment:

1. 10 nodes in network, 5 validators as quorum set, threshold percent set at 60%
2. 10 nodes in network, 5 validators as quorum set, threshold percent set at 70%
3. 10 nodes in network, 3 validators as quorum set, threshold percent set at 60%
4. 5 nodes in network, 5 validators as quorum set, threshold percent set at 60%
5. 10 nodes in network, 8 validators as quorum set, threshold percent set at 60%

Adjustment of the value of *thresholdpercent* and *validators* could lead to significantly different results. Decreasing the minimum number of nodes in agreement with value outcome (lowering *thresholdpercent*) is an implementation of sacrificing safety over liveness (Martin and Alvisi, 2006). In contrast, increasing the *thresholdpercent* could be indicated as 'sacrificing liveness over safety'

(Martin and Alvisi, 2006). Thus it is essential to know to what extent the above-mentioned parameters implicate the overall failure of one transaction instance, in order to determine the optimal liveness/safety balance in the production-level software. Numbers of validators, on the other hand, determines how many potential nodes can be selected as quorum slices for one node in the network. The experiment aims to explore whether a higher percentage of validators in the network could lead to a higher probability of nodes reaching agreement within the network.

3.5 Test efficiency

When one simulated transaction input is generated, it will be applied to the network and ledger. When one transaction goes through three built-in checkpoints in Stellar and Ripple, the network will broadcast resulting messages. Transactions which obtained all three positive message will be applied to the ledger and the overall transaction is successful. However, the experiment is formulated to utilize over hundreds number of transaction per test and calculate the probability of successful transactions as evidence of the strength of Byzantine Fault Tolerance. One limitation the built-in test algorithm has is that it can only perform one transaction test with one configuration by default. Secondly, there is no overall message per test to conclude whether one transaction test is successful. In response to the limitation, a wrapper function *TestWrapper* (see appendixA) is implemented in order to perform an iterative test in different configurations and reports the test result back as plain words.

Algorithm 3.1 Test wrapper

```
Input: data:
- checkpoint message from BUCKET-APPLY, APPLY-LEDGER and CATCHUP.
- simulated transaction
- Stellar/Ripple node configuration file
Output: message:
- Number of passed/failed test per configuration
import stellar or ripple configuration
integer passCount set to 0
integer failCount set to 0
num ⇐ number of transactions generated by transactionTemplate
for all configurations ∈ configuration set do
for all transactions ∈ transaction set for that configuration do
cfg ⇐ current configuration
input ⇐ transaction
getReporterOutput messages ⇐ stellar-core.consoleReporter(input)
if messages.applyBucket equal pass
and messages.applyLedger equal pass
and messages.catchup equal pass
outputCase ⇐ SUCCESS
passCount increase by 1
else
outputCase ⇐ FAIL
failCount increase by 1
end if
end for
output ⇐ passCount, failCount
end for
return output
```

4 Result

To evaluate the optimal fault tolerance for Stellar consensus network and Ripple consensus network, the experiment aggregated the number of successful and failed transaction externalizations, in each configuration for two networks. The result is the ratio between successful transactions and failed transactions for each configuration.

4.1 Result for Stellar consensus network

Test	N(node)	N(vldt)	Threshold (%)
1	10	5	60
2	10	5	70
3	10	3	60
4	5	5	60
5	10	8	60

Table 4.1: Setup summary for Stellar test cases

There are five cases implemented in the test environment, differs in the number of nodes, the number of validators and the threshold percentage to reach agreement across the quorum. All parameters are modifiable in the configuration files for each node, and the result is generated by the TestWrapper function. Table 4.1 is a summary of the configuration used in the Stellar network. Five configuration sets are implemented, and each serves a different purpose for comparison.

- **Test 1 and 2:** To determine the extent of effect a higher threshold percent could impose to the network's fault tolerance.
- **Test 1 and 3:** To determine the effect of fewer validators in the network.
- **Test 1 and 4:** To determine whether a higher validator ratio within the network could affect the tolerance.
- **Test 1 and 5:** To determine the effect of validators as network majority.

The number of test cases varies with different configurations and it is determined by the number of node operation, time complexity for nomination algorithm and other factors, thus it cannot be personalized by test configuration. This is reflected in table 4.2.
Table 4.2 only reflects the number of successful and failed transaction instances, as the TestWrapper function only records messages with content. The rest instances are considered 'missing'.

In order to compare the performance of fault tolerance for Stellar-core in different parameter set, the percentage of successful transaction cases and percentage of cases without node nominations (lack of liveness on the network level) within that test is shown in table 4.3.

Test	N(case)	N(succ)	N(fail)
1	207	183	6
2	255	209	16
3	195	142	13
4	349	325	9
5	321	308	3

Table 4.2: Results for 5 configurations (Stellar)

Test	P(succ)(%)	P(miss)(%)
1	88.4	2
2	82	6.3
3	72.8	6.6
4	93.1	2.6
5	96	0.9

Table 4.3: Results for success and miss nodes

The percentage of successful transaction is the ratio between the number of transaction instances that return message indicates a successful transaction and the overall number of transaction instances. The percentage of the missing transaction is the ratio between the number of transaction instances that return no message and the overall number of transaction instances.

- **Test 1 and 2:** By increasing the threshold percentage, the network requires more validators to agree on one transaction in order to externalize the outcome within the network. In this case, the safety level required is higher than the first test. Under these adjustments, the percentage of the successful transaction in dropped by approximately 6.4%, which is significantly lower. Since the tolerated number of nodes without nomination is decreased, the percentage of missing nodes is considerably increased with a higher percentage threshold.
- **Test 1 and 3:** Comparing to the standard environment in teat case 1, the chance of a successful transaction in test case 3 is approximately 15.6% smaller. The percentage of the missing transaction is increased by a significant 4.6%.

- **Test 1,4 and 5:** In the case where all nodes in the network are validators, there is a 4.7% increase in the percentage of successful transaction ratio in test case 4 compared to test case 1. However, due to a limited number of nodes in the network, liveness problem could potentially be amplified since the selection of nodes as validators are smaller than other test cases. This is reflected by a 0.6% increase in the percentage of transactions without one of the checkpoint messages. In terms of test case 5 where validator ratio increased with sufficient nodes in the network, there is a 7.6% increase in successful transaction ratio, and the percentage of missing transactions decreased by 1.1%.

4.2 Result for Ripple consensus network

Test	N(node)	N(vldt)
1	10	5
3	10	3
4	5	5
5	10	8

Table 4.4: Setup summary for Ripple test cases

There are four cases implemented in the test environment, differs in the number of nodes and number of validators. Threshold percent for Ripple network is fixed in each implementation of Ripple network. In production-level software of Ripple (Rippled), the threshold percentage for each node is set at 80%. The parameters are adjustable within multiple Rippled dependencies, including a node configuration file and a unique node list dedicated for the node's quorum arrangement. Table 4.4 showcased all the parameter combination used in the experiment for the Ripple network. The purpose of evaluating fault tolerance in one network is consistent with the experiment on the Stellar network, and the comparison between test cases are identical (see section 4.1). Since threshold percent for Ripple node is not modifiable, test case 2 will not be tested in Ripple consensus network.

In Ripple consensus network, the built-in terminal reporter broadcast messages in three checkpoints, similar to the Stellar network. However,

Test	N(case)	N(succ)	N(fail)
1	30	26	4
3	30	15	15
4	30	26	4
5	30	27	3

Table 4.5: Result of 4 configurations (Ripple)

Test	P(succ)(%)	P(fail)(%)
1	86.6	13.4
3	50	50
4	86.6	13.4
5	90	10

Table 4.6: Results for success and failed nodes

when the network generates an empty message for one transaction instance, it will be counted as a node failure instead of further categorized into safety or liveness issues in Stellar-core. Thus, the experiment only compares the percentage of the successful transaction between Stellar and Ripple for evaluation of Byzantine fault tolerance.

- **Test 1 and 3:** The node in one ripple network has fewer candidates eligible to form its trusted peers in the unique node list, similar to the case in Stellar where there is an insufficient number of validators. Compared to Stellar network, Ripple network's percentage of successful transaction plunged to 50%, from 86.6% in the standard test environment (test case 1). Compared to Stellar network, Ripple network is more sensitive to a lower validator ratio.

- **Test 1,4 and 5:** By increasing the ratio of validators in Ripple network, the node has more candidate for its list of trusted peers. While nodes in the network are provided with more validators as list candidates, the increase of successful ratio is smaller than Stellar's respective result. In both cases where validator ratio increases, the successful rate either remain constant or increase slightly for approximately 3.4%. It indicates that the Ripple network is less reliant on the validator ratio, compared to the Stellar network.

5 Discussion

5.1 Implication

Based on the result obtained for both Stellar and Ripple experiment, their respective percentage of successful transaction is compared with similar configuration, so as to compare its performance under the same configuration, and the effect of different mechanism employed on this two consensus protocol.

The third test case represents an environment for a simulated transaction where there is a fewer validators in the same quorum size. Comparing to the result of percentage of successful cases, Ripple dropped approximately 36% while Stellar declined around 6%. Amid all failed cases in Ripple, missed nodes that is caused by liveness issue consist a significant portion. For Stellar, since there is a general increase of missed nodes (approximately 4%), it grows along with the increase of failed nodes due to small validator ratio. The experiment implies that small validator ratio contribute to liveness issue, since the choice of validator within this network is narrowed. If in case validator nodes become blocked nodes, or unable to externalize values and messages due to other factors, the probability of nodes that choose them as part of the quorum slices is higher. However, Stellar has an additional mechanism to unblock blocked nodes and its blocked quorum slices. The use of the ballot system (see section 3.2.2) ensures that even node is blocked by a set of peer nodes of each quorum slice, the value that caused the blocking will be neutralized and other values will be nominated. This mechanism directly improved the liveness level of networks that is based on Stellar consensus protocol.

The fourth test case represents an environment where every node exists in the network is validator. In this case, Stellar sees a slight increment in terms of percentage of successful transaction cases while Ripple network is not affected by the alteration compared to its performance on test case 1. In this case of a network with higher validator ratio, both network stagnates in the growth of successful transaction. With a decrease of number of nodes, there are both potential safety and liveness issue due to a smaller quorum size. specifically, with one validator being ill-behaved to behave arbitrarily or be blocked, with a smaller number of nodes in the network, the percentage of nodes that is affected by this particular ill-behaved node is higher than a network with more nodes, since other nodes may not list the ill-behaved node as part of the quorum slice, and will be functional with its original service. The fifth test case represents an environment with more validator nodes than non-validating nodes (a higher validator ratio). In this case, the selection of validators as quorum slices is more decentralized than other test cases, since the choice of each node is more scattered. A higher validator number in the network took advantage of the decentralization feature the network has, and can effectively avoid exploitation that target one node, or a set of nodes. In this case, if there is one validator that has been compromised, a relatively smaller percentage of nodes (compared to network with lower validator ratio) will be affected since there are more validator candidate for nodes to enlist. Therefore, potential inconsistency caused ny safety issue of such exploitation can be alleviated.

5.2 Limitation

The experiment sought to evaluate the effect of node configurations to Byzantine fault tolerance for replicated state machine by cross-comparing safety and liveness performance in production level software that is based on state machines. In the process of formulating and executing the experiment, there are several limitations that could potentially affect the course of the results. The limitations are mainly categorized into two factors.

5.2.1 Confounding variables

Confounding variables refers to factors based on external influence and potentially affect the experiment outcome. There are several limitations observed in the course of the experiment:

- **Network time-out:** Since production-level software is used in the evaluation process, network timeout issue could potentially be attributed to missing transactions which the network does not publish messages. It turned out that instead of just using the hash value of the transaction as verification, a time function also limits the time the system waits for the messages. If there is a network delay on both the node and the blockchain server, transaction for that instance could be abandoned due to time out, even if there are either 'Succ' or 'Fail' message (Dwork, Lynch, and Stockmeyer, 1988). However, since the experiment runs a considerable amount of transaction test on each network, temporary network delay will have a restrained effect on the result.

- **Operating environment:** Both Stellar-core and Rippled implement their network in various operating environment. The same network operates on native operating systems or virtual environments have different performance in terms of Byzantine fault tolerance. Additionally, Stellar and Ripple software developed by third-party developers further diversified the range of performance within the same consensus network. In this experiment, the evaluation of fault tolerance only limited to the official implementation of Stellar and Ripple by their respective organizations. Thus the result may be less reliable to the same consensus networks on the different operating environment.

- **Number of nodes:** The experiment is capable of carrying out tolerance assessment for more than 10 nodes, for its scalability on adding more nodes as long as nodes or devices are linked by the same address. However, in this phase of the project, only a maximum of 10 nodes is used due to efficiency. With a limited infrastructure (i.e. performance of computers, bandwidth of network and number of devices), consensus model that has more than 10 nodes leads to significant network latency. Network latency could compromise the synchronization of decentralized network by affecting the liveness of each node, since a latency could jeopardize the externalization of value for a transaction (see 'Network timeout'). On the other hand, the essential idea for this experiment is to collect messages that reflect Byzantine fault tolerance level of the network. With a larger quorum, the size of the quorum slices and the complexity of slice structure will increment, which is counter-effective when the test procedure tries to limit it scope to one transaction case in the first place. For a potential extension of this project, both Stellar and Ripple network can operate on a server-based operating system and network, so that the test can be implemented with higher order of quorum structure and larger number of participators. The test can also be modified to accept either multiple simulated transactions or real-case transactions to take advantage to complex slice structure and larger quorum sizes.

5.2.2 Ripple and Stellar implementations

- **Level of decentralization:** As mentioned in section 1.2.1, Stellar consensus network are implemented with the higher level of decentralization. In the actual implementation of the network, the difference is reflected in the available options of validator/quorum slice choice. The validator nodes maintained by Ripple organization has a higher reputation, thus these nodes have a higher Trust Score that other nodes may not have. Stellar, however, only let node owner to determine their own node priority. The difference in the node priority could potentially favor the performance of the Ripple network since the safety level could be higher if the validators in Ripple network directly or indirectly have the 'approved' node as validators.

- **Parameter modification:** While Ripple configurations restrict threshold percent for each node, Stellar provides a flexible threshold in order to manually adjust the safety and liveness level. Although they are controlled variables in the experiment, the result may not be conclusive based on one threshold percentage since it offers more possible threshold values. Secondly, Ripple will count 'Missing' transaction into 'Failed transaction', therefore, it is not visible whether the failure is caused by Safety Issue or Liveness issue. Thus, the only percentage of the successful transaction can be used as the main data for comparison between Stellar-core and Rippled. Additionally, while the test number for each configuration in Rippled can be customized, the number of test per configurations in Stellar-core can vary. It is determined by the number of nodes and time complexity in nomination algorithm. Therefore, different test number in Stellar and Ripple experiment could introduce a potentially biased result.

5.3 Conclusion

The recent events of network forking due to safety issue in both Stellar and Ripple has inspired the experiment. This experiment identifies the Byzantine fault tolerance as one of the evident indicators for network consistency and compared it in two replicated state machines based on Federal Byzantine Agreement system. In the later period, due to the difficulties caused by inefficient testing process, the experiment attempts to formulate an efficient test protocol for production-level software based on FBAs. The test protocol is demonstrated by fault tolerance evaluation between Stellar consensus network and Ripple consensus network. There are three conclusions based on the experiment:

- **Fault tolerance:** Compared to Ripple network, Stellar-core is more resilient with insufficient validators (see section 5.1). By decreasing the number of validators, the node has fewer candidates eligible to form quorum slice.
  Due to the nature of slice selection in the production-level network (see section 3.2), lack of sufficient validator could run the risk of losing liveness due to the lack of validator diversity.
  On the other hand, insufficient validators could lead to safety issues due to a potential risk of missing quorum intersection. In this case, the selected validators for each node in the network may not intersect.Quorum intersection in a quorum network is achieved by a larger candidate size for quorum slice and a priority to choose validators with higher validity. Conversely, potential lack of quorum intersection caused by small candidate size could increase the weight one validator has. If one validator attempt to exploit the network by feeding false transaction value, smaller candidate size will increase the probability this particular ill-behaved validator get selected. Thus resulting in a lower fault tolerance.
  Fault tolerance, in actual implementation, is case-specific since each case of consensus network applies to different safety and liveness requirement. Stellar enjoys a higher flexibility to engineer tolerance by an adjustable threshold that controls the degree of restraint put on the number of consistent nodes in the quorum slices. In this case, a controllable tolerance is an advancement of Stellar over Ripple and other FBAs-based network.

- **Safety:** Compared to the Ripple network, Stellar enjoys a higher safety level with higher validator ratio. By increasing the ratio of validators within the quorum, the node has more candidates for its quorum slice. Since the diversity of candidate selection is increased, exploitation by one compromised validator node will have a smaller impact. It remains valid in an actual implementation of Stellar-core. It is evident that Stellar assures safety for nodes that are both well-behaved and in quorum intersections. The first advancement of Stellar over Ripple is reflected in the transaction validation process in an actual transaction instance. In a Stellar run, a function called 'quorum set sanity checker' will ensure that the number of nodes in the quorum set is larger than the node number of the v-blocking set. This indicates that for each consensus, the transaction will be approved by one particular node if all nodes in the quorum set all agree on the same slot. In the practical cases, the quorum slice is determined by the threshold of validator nodes, and the v-blocking sets are calculated by both threshold percentage, number of validators and inner sets. The implementation of this 'sanity check' could improve fault tolerance for Stellar Consensus Protocol over Ripple's counterpart.

References

A. Agbaria and R. Friedman. Overcoming byzantine failures using checkpointing. *Coordinated Science Laboratory Report no. UILU-ENG-03-2228, CRHC-03-14*, 2003.

A. Baliga. Understanding blockchain consensus models. *Persistent*, 2017.

George A Bekey. *Autonomous robots: from biological inspiration to implementation and control.* MIT press, 2005.

G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

C. Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

Christian Cachin and Marko Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

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

Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg. Byzantine fault tolerance, from theory to reality. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2003.

C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32 (2):374–382, 1985.

S. Fred. Implementing fault-tolerant services using the state machine approach: A tutorial. *Acm Computing Surveys*, 22(4):299–319, 1990.

S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2): 99111, 1991.

A. Haeberlen, P. Kouznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6):175–188, 2007.

L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4/3:382–401, 1982.

J. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995,*, page 2. IEEE, 1985.

J. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.

P. Moreno-Sanchez, N. Modi, R. Songhela, A. Kate, and S. Fahmy. Mind your credit: Assessing the health of the ripple credit network. *arXiv preprint arXiv:1706.02358*, 2017.

S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

G. W Peters and E. Panayi. Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. In *Banking Beyond Banks and Money*, pages 239–278. Springer, 2016.

Lakshmi S. Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on*, pages 1–5. IEEE, 2017.

A. Saraiva, B. Almeida, and S. Barroso. Secure transactions without mining or central authority. 2015.

D. Schwartz, N. Youngs, and A. Britto. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5, 2014.

João G. Silva, P. Prata, M. Rela, and H. Madeira. Practical issues in the use of abft and a new failure model. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 26–35. IEEE, 1998.

M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 252–261. IEEE, 2004.

T. Stefan and K. Walter. Asset accumulation, interdependence and technological change: evidence from pharmaceutical drug discovery. *Strategic Management Journal*, 23(7):619–635, 2002.

P. Stelling, C. DeMatteis, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128, 1999.

E. Tymoigne and J. Henry. Primitive trade relations: A proposed solution. 2018.

C. Udokwu, A. Kormiltsyn, K. Thangalimodzi, and A. Norta. An exploration of blockchain enabled smart-contracts application in the enterprise. Technical report, Technical Report, 2018.

S. Wang, Jean-Philippe JP Vergne, and Y. Hsieh. The internal and external governance of blockchain-based organizations: Evidence from cryptocurrencies. In *Bitcoin and Beyond*, pages 48–68. Routledge, 2017.

Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 557–564. IEEE, 2017.

A Appendix: Test wrapper

```
#include "lib/catch.hpp"

struct TestWrapper : public ConsoleReporter {

    Prefs getPreferences() const override {
        Prefs prefs;
        prefs.shouldRedirectStdOut = false;
        return prefs;
    } // function that allocate built-in functions in software to
     retrieve Stellar/Ripple configurations

    void testInitiation(TestInfo const& ti) override {
        ConsoleReporter::testInitiation(ti);
        stream << "|"" << ti.name << "|" " << ti.lineInfo << std
    ::endl;
    } // function that prepare info of a transaction instance

    void sectionInitiation(SectionInfo const& _sectionInfo)
    override {
        ConsoleReporter::sectionInitiation(_sectionInfo);
    } // function that read built-in message from built-in
    function consoleReporter

    void assertionInitiation(AssertionInfo const& ai) override {
        mLastAssertInfo = std::make_unique<AssertionInfo>(ai);
    }

    bool assertionTermination(AssertionStats const&
    _assertionStats) override {
        bool res = _assertionStats.assertionResult.isOk();
        if (!res) {
            ConsoleReporter::assertionInitiation(*mLastAssertInfo
    );
            res = ConsoleReporter::assertionTermination(
    _assertionStats);
        }
        mLastAssertInfo.reset();
        return res;
    }

    void testTermination(TestCase const&) {
        stream << "<done>";
        printNewLine();
    }

    static std::string getReporterOutput<reporter::gtest_reporter
    >::convert(reporter::gtest_reporter const& os) {
        return fmt::format(
            "test cases: {} | {} passed | {} failed",
            reporter::gtest_reporter(info.report_info),
            reporter::gtest_reporter(success.report_success),
            reporter::gtest_reporter(failure.report_failure));
    } // functions that generate overall test message for one
    configuration case

    static std::string getAssertionOutput<reporter::
    gtest_reporter>::convert(reporter::gtest_reporter const& os)
    {
        return fmt::format(
            "assertions: {} | {} passed | {} failed",
            reporter::gtest_reporter(info.assertion_info),
            reporter::gtest_reporter(success.assertion_success),
            reporter::gtest_reporter(failure.assertion_failure));
    }

    static std::string getDescription() {
        return "Repeat test-case for conclusive output";
    }

  private:
    std::unique_ptr<AssertionInfo> mLastAssertInfo;

    void printNewLine() {
        stream << '\n';
        mDots = 0;
    }
};
```

B Appendix: Stellar Configuration

```
LOG_FILE_PATH="/var/log/stellar/stellar-core.log"
BUCKET_DIR_PATH="/var/stellar/buckets"
DATABASE="sqlite3://stellar.db"
// a copy of readable text of the history and transaction details
     is stored in the database of choice (sqlite or postgres)

TARGET_PEER_CONNECTIONS=10
//how aggressive the network will connect to peers

PEER_TIMEOUT=30
//maximum time before dropping the coonection with the peer due
    to liveness problems

NODE_NAMES=[
"GC5A5WKAPZU5ASNMLNCAMLW7CVHMLJJAKHSZZHE2KWGAJHZ4EW6TQ7PB
    stellarport_ohio",
"GAXP5DW4CVCW2BJNPFGTWCEGZTJKTNWFQQBE5SCWNJIJ54BOHR3WQC3W  moni",
"GBFZFQRGOPQC5OEAWO76NOY6LBRLUNH4I5QYPUYAK53QSQWVTQ2D4FT5  dzham"
    ,
"GAOO3LWBC4XF6VWRP5ESJ6IBHAISVJMSBTALHOQM2EZG7Q477UWA6L7U  eno",
"GCJCSMSPIWKKPR7WEPIQG63PDF7JGGEENRC33OKVBSPUDIRL6ZZ5M7OO  tempo"
    ,
"GCCW4H2DKAC7YYW62H3ZBDRRE5KXRLYLI4T5QOSO6EAMUOE37ICSKKRJ
    sparrow",
"GD7FVHL2KUTUYNOJFRUUDJPDRO2MAZJ5KP6EBCU6LKXHYGZDUFBNHXQI  umbrel
    ",
"GDAXAGWQNTOUIGTAJDYIL4QCM3Q6HM67SKEAJNSOW6G2Z3QPPKGAVJFW  cowrie
    ",
"GDRA72H7JWXAXWJKOONQOPH3JKNSH5MQ6BO5K74C3X6FO2G3OG464BPU
    ibm_norway",
"GCDLFPQ76D6YUSCUECLKI3AFEVXFWVRY2RZH2YQNYII35FDECWUGV24T  snt.
    lux",
"GBAR4OY6T6M4P344IF5II5DNWHVUJU7OLQPSMG2FWVJAFF642BX5E3GB
    telindus",
// non-validating nodes
"GCGB2S2KGYARPVIA37HYZXVRM2YZUEXA6S33ZU5BUDC6THSB62LZSTYH
    sdf_watcher1",
"GCM6QMP3DLRPTAZW2UZPCPX2LF3SXWXKPMP3GKFZBDSF3QZGV2G5QSTK
    sdf_watcher2",
"GABMKJM6I25XI4K7U6XWMULOUQIQ27BCTMLS6BYYSOWKTBUXVRJSXHYQ
    sdf_watcher3"
]
//a list of nodes with their symbolized names
```

```
PREFERRED_PEERS=[]

PREFERRED_PEER_KEYS=[
"$sdf_watcher1",
"$sdf_watcher2",
"$sdf_watcher3",
"$dzham",
"$$moni",
"$$cowrie"
]
//This node will try to always stay connected to the other peers
    on this list.

KNOWN_PEERS=[
"core-live-a.stellar.org:11625",
"core-live-b.stellar.org:11625",
"core-live-c.stellar.org:11625"
]
//It connect to the enlisted nodes when it is below
    TARGET_PEER_CONNECTIONS

NODE_SEED="SXXXXXX....XXXX"
//each node has a unique node seed as private key to serve as a
    verification of the node (for safety)

NODE_IS_VALIDATOR=true
//a switch to enable one node to be a validator

FAILURE_SAFETY=1
// Maximum number of validator failures from your QUORUM_SET that
     the network tolerate.

UNSAFE_QUORUM=false
If true, it enables to specify a unsafe quorum set.

// Local history storage (use online storage instead)
// [HISTORY.local]
// get="cp /var/stellar/history/vs/{0} {1}"
// put="cp {0} /var/stellar/history/vs/{1}"
// mkdir="mkdir -p /var/stellar/history/vs/{0}"

// Stellar.org history storage
[HISTORY.sdf1]
get="curl -sf http://history.stellar.org/prd/core-live/
    core_live_001/{0} -o {1}"
```

```
[HISTORY.sdf2]
get="curl -sf http://history.stellar.org/prd/core-live/
    core_live_002/{0} -o {1}"

[HISTORY.sdf3]
get="curl -sf http://history.stellar.org/prd/core-live/
    core_live_003/{0} -o {1}"

[QUORUM_SET]
THRESHOLD_PERCENT=66
VALIDATORS=[
"$moni",
"$eno",
"umbrel",
"$dzham",
"$sparrow",
"$cowrie",
"self",
"tempo",
"ibm_norway"
]

//THRESHOLD_PERCENT: how many percentage of nodes have to agree
    on one value

// MAINTENANCE_ON_STARTUP=true
```

C Appendix: Graphic illustration

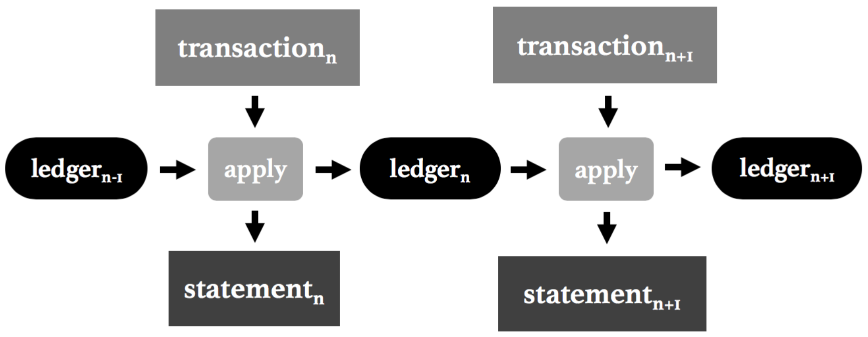

Figure 1: Data flow of the consensus network

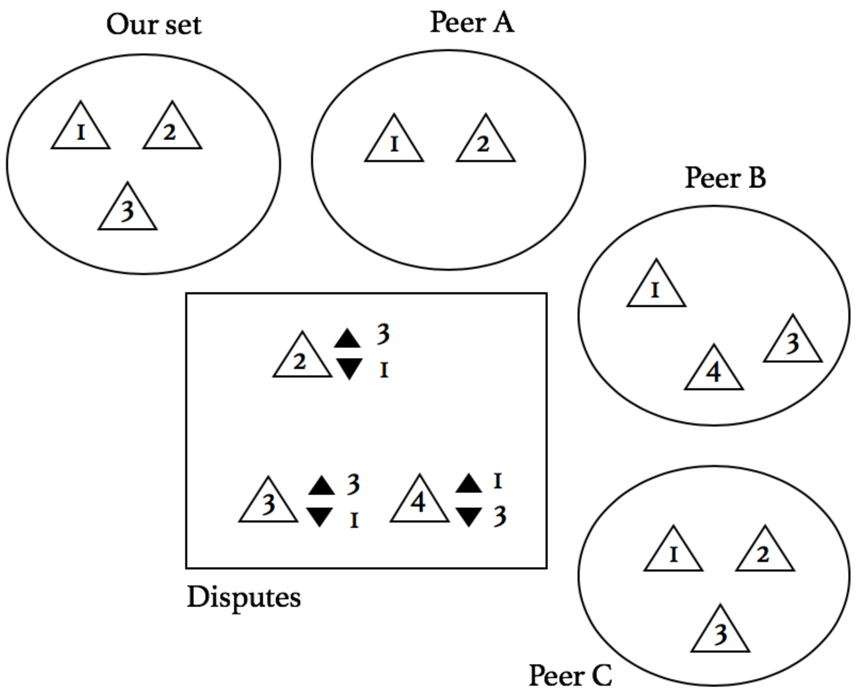

Figure 2: Probabilistic voting for Ripple consensus network