The Ripple Protocol Consensus Algorithm

David Schwartz
david@ripple.com

Noah Youngs
nyoungs@nyu.edu

Arthur Britto
arthur@ripple.com

This paper does not reflect the current state of the ledger consensus protocol or its analysis. We will continue hosting this draft for historical interest, but it SHOULD NOT be used as a reference. For an updated analysis and presentation of the consensus protocol, please refer to arXiv:1802.07242 (https://arxiv.org/abs/1802.07242), released 20 February 2018.

Abstract
While several consensus algorithms exist for the Byzantine Generals Problem, specifically as it pertains to distributed payment systems, many suffer from high latency induced by the requirement that all nodes within the network communicate synchronously. In this work, we present a novel consensus algorithm that circumvents this requirement by utilizing collectively-trusted subnetworks within the larger network. We show that the "trust" required of these subnetworks is in fact minimal and can be further reduced with principled choice of the member nodes. In addition, we show that minimal connectivity is required to maintain agreement throughout the whole network. The result is a low-latency consensus algorithm which still maintains robustness in the face of Byzantine failures. We present this algorithm in its embodiment in the Ripple Protocol.

Contents

1. Introduction

Interest and research in distributed consensus systems has increased markedly in recent years, with a central focus being on distributed payment networks. Such networks allow for fast, low-cost transactions which are not controlled by a centralized source. While the economic benefits and drawbacks of such a system are worthy of much research in and of themselves, this work focuses on some of the technical challenges that all distributed payment systems must face. While these problems are varied, we group them into three main categories: correctness, agreement, and utility.

By correctness, we mean that it is necessary for a distributed system to be able to discern the difference between a correct and fraudulent transaction. In traditional fiduciary settings, this is done through trust between institutions and cryptographic signatures that guarantee a transaction is indeed coming from the institution that it claims to be coming from. In distributed systems, however, there is no such trust, as the identity of any and all members in the network may not even be known. Therefore, alternative methods for correctness must be

utilized.

Agreement refers to the problem of maintaining a single global truth in the face of a decentralized accounting system. While similar to the correctness problem, the difference lies in the fact that while a malicious user of the network may be unable to create a fraudulent transaction (defying correctness), it may be able to create multiple correct transactions that are somehow unaware of each other, and thus combine to create a fraudulent act. For example, a malicious user may make two simultaneous purchases, with only enough funds in their account to cover each purchase individually, but not both together. Thus each transaction by itself is correct, but if executed simultaneously in such a way that the distributed network as a whole is unaware of both, a clear problem arises, commonly referred to as the "Double-Spend Problem" [1]. Thus the agreement problem can be summarized as the requirement that only one set of globally recognized transactions exist in the network.

Utility is a slightly more abstract problem, which we define generally as the "usefulness" of a distributed payment system, but which in practice most often simplifies to the latency of the system. A distributed system that is both correct and in agreement but which requires one year to process a transaction, for example, is obviously an inviable payment system. Additional aspects of utility may include the level of computing power required to participate in the correctness and agreement processes or the technical proficiency required of an end user to avoid being defrauded in the network.

Many of these issues have been explored long before the advent of modern distributed computer systems, via a problem known as the "Byzantine Generals Problem" [2]. In this problem, a group of generals each control a portion of an army and must coordinate an attack by sending messengers to each other. Because the generals are in unfamiliar and hostile territory, messengers may fail to reach their destination (just as nodes in a distributed network may fail, or send corrupted data instead of the intended message). An additional aspect of the problem is that some of the generals may be traitors, either individually, or conspiring together, and so messages may arrive which are intended to create a false plan that is doomed to failure for the loyal generals (just as malicious members of a distributed system may attempt to convince the system to accept fraudulent transactions, or multiple versions of the same truthful transaction that would result in a double-spend). Thus a distributed payment system must be robust both in the face of standard failures, and so-called "Byzantine" failures, which may be coordinated and originate from multiple sources in the network.

In this work, we analyze one particular implementation of a distributed payment system: the Ripple Protocol. We focus on the algorithms utilized to achieve the above goals of correctness, agreement, and utility, and show that all are met (within necessary and predetermined tolerance thresholds, which are well-understood). In addition, we provide code that simulates the consensus process with parameterizable network size, number of malicious users, and message-sending latencies.

2. Definitions, Formalization and Previous Work

We begin by defining the components of the Ripple Protocol. In order to prove correctness, agreement, and utility properties, we first formalize those properties into axioms. These properties, when grouped together, form the notion of *consensus*: the state in which nodes in the network reach correct agreement. We then highlight some previous results relating to consensus algorithms, and finally state the goals of consensus for the Ripple Protocol within our formalization framework.

2.1 Ripple Protocol Components

We begin our description of the ripple network by defining the following terms:

- **Server:** A server is any entity running the Ripple Server software (as opposed to the Ripple Client software which only lets a user send and receive funds), which participates in the consensus process.

- **Ledger:** The ledger is a record of the amount of currency in each user's account and represents the "ground truth" of the network. The ledger is repeatedly updated with transactions that successfully pass through the consensus process.

- **Last-Closed Ledger:** The last-closed ledger is the most recent ledger that has been ratified by the consensus process and thus represents the current state of the network.

- **Open Ledger:** The open ledger is the current operating status of a node (each node maintains its own open ledger). Transactions initiated by end users of a given server are applied to the open

ledger of that server, but transactions are not considered final until they have passed through the consensus process, at which point the open ledger becomes the last-closed ledger.

- **Unique Node List (UNL):** Each server, s, maintains a unique node list, which is a set of other servers that s queries when determining consensus. Only the votes of the other members of the UNL of s are considered when determining consensus (as opposed to every node on the network). Thus the UNL represents a subset of the network which when taken collectively, is "trusted" by s to not collude in an attempt to defraud the network. Note that this definition of "trust" does not require that each individual member of the UNL be trusted (see section 3.2).

- **Proposer:** Any server can broadcast transactions to be included in the consensus process, and every server attempts to include every valid transaction when a new consensus round starts. During the consensus process, however, only proposals from servers on the UNL of a server s are considered by s.

2.2 Formalization

We use the term *nonfaulty* to refer to nodes in the network that behave honestly and without error. Conversely, a *faulty* node is one which experiences errors which may be honest (due to data corruption, implementation errors, etc.), or malicious (Byzantine errors). We reduce the notion of validating a transaction to a simple binary decision problem: each node must decide from the information it has been given on the value 0 or 1.

As in Attiya, Dolev, and Gill, 1984 [3], we define consensus according to the following three axioms:

1. **(C1):** Every nonfaulty node makes a decision in finite time
2. **(C2):** All nonfaulty nodes reach the same decision value
3. **(C3):** 0 and 1 are both possible values for all nonfaulty nodes. (This removes the trivial solution in which all nodes decide 0 or 1 regardless of the information they have been presented).

2.3 Existing Consensus Algorithms

There has been much research done on algorithms that achieve consensus in the face of Byzantine errors. This previous work has included extensions to cases where all participants in the network are not known ahead of time, where the messages are sent asynchronously (there is no bound on the amount of time an individual node will take to reach a decision), and where there is a delineation between the notion of strong and weak consensus.

One pertinent result of previous work on consensus algorithms is that of Fischer, Lynch, and Patterson, 1985 [4], which proves that in the asynchronous case, non-termination is always a possibility for a consensus algorithm, even with just one faulty process. This introduces the necessity for time-based heuristics, to ensure convergence (or at least repeated iterations of non-convergence). We shall describe these heuristics for the Ripple Protocol in section 3.

The strength of a consensus algorithm is usually measured in terms of the fraction of faulty processes it can tolerate. It is provable that no solution to the Byzantine Generals problem (which already assumes synchronicity, and known participants) can tolerate more than $(n-1)/3$ byzantine faults, or 33% of the network acting maliciously [2]. This solution does not, however, require verifiable authenticity of the messages delivered between nodes (digital signatures). If a guarantee on the unforgeability of messages is possible, algorithms exist with much higher fault tolerance in the synchronous case.

Several algorithms with greater complexity have been proposed for Byzantine consensus in the asynchronous case. FaB Paxos [5] will tolerate $(n-1)/5$ Byzantine failures in a network of n nodes, amounting to a tolerance of up to 20% of nodes in the network colluding maliciously. Attiya, Doyev, and Gill [3] introduce a phase algorithm for the asynchronous case, which can tolerate $(n-1)/4$ failures, or up to 25% of the network. Lastly, Alchieri et al., 2008 [6] present BFT-CUP, which achieves Byzantine consensus in the asynchronous case even with unknown participants, with the maximal bound of a tolerance of $(n-1)/3$ failures, but with additional restrictions on the connectivity of the underlying network.

2.4 Formal Consensus Goals

Our goal in this work is to show that the consensus algorithm utilized by the Ripple Protocol will achieve consensus at each ledger-close (even if consensus is the trivial consensus of all transactions being rejected), and that the trivial consensus will only be reached with a known probability, even in the face of Byzantine failures.

Since each node in the network only votes on proposals from a trusted set of nodes (the other nodes in its UNL), and since each node may have differing UNLs, we also show that only one consensus will be reached amongst all nodes, regardless of UNL membership. This goal is also referred to as preventing a "fork" in the network: a situation in which two disjoint sets of nodes each reach consensus independently, and two different last-closed ledgers are observed by nodes on each node-set.

Lastly we will show that the Ripple Protocol can achieve these goals in the face of $(n-1)/5$ failures, which is not the strongest result in the literature, but we will also show that the Ripple Protocol possesses several other desirable features that greatly enhance its utility.

3. Ripple Consensus Algorithm

The Ripple Protocol consensus algorithm (RPCA), is applied every few seconds by all nodes, in order to maintain the correctness and agreement of the network. Once consensus is reached, the current ledger is considered "closed" and becomes the last-closed ledger. Assuming that the consensus algorithm is successful, and that there is no fork in the network, the last-closed ledger maintained by all nodes in the network will be identical.

3.1 Definition

The RPCA proceeds in rounds. In each round:

- Initially, each server takes all valid transactions it has seen prior to the beginning of the consensus round that have not already been applied (these may include new transactions initiated by end-users of the server, transactions held over from a previous consensus process, etc.), and makes them public in the form of a list known as the "candidate set".
- Each server then amalgamates the candidate sets of all servers on its UNL, and votes on the veracity of all transactions.
- Transactions that receive more than a minimum percentage of "yes" votes are passed on to the next round, if there is one, while transactions that do not receive enough votes will either be discarded, or included in the candidate set for the beginning of the consensus process on the next ledger.
- The final round of consensus requires a minimum percentage of 80% of a server's UNL agreeing on a transaction. All transactions that meet this requirement are applied to the ledger, and that ledger is closed, becoming the new last-closed ledger.

3.2 Correctness

In order to achieve correctness, given a maximal amount of Byzantine failures, it must be shown that it is impossible for a fraudulent transaction to be confirmed during consensus, unless the number of faulty nodes exceeds that tolerance. The proof of the correctness of the RPCA then follows directly: since a transaction is only approved if 80% of the UNL of a server agrees with it, as long as 80% of the UNL is honest, no fraudulent transactions will be approved. Thus for a UNL of n nodes in the network, the consensus protocol will maintain correctness so long as:

$$f \leq (n-1)/5 \tag{1}$$

where f is the number Byzantine failures. In fact, even in the face of $(n-1)/5+1$ Byzantine failures, correctness is still technically maintained. The consensus process will fail, but it will still not be possible to confirm a fraudulent transaction. Indeed it would take $(4n+1)/5$ Byzantine failures for an incorrect transaction to be confirmed. We call this second bound the bound for *weak* correctness, and the former the bound for *strong* correctness.

It should also be noted that not all "fraudulent" transactions pose a threat, even if confirmed during consensus. Should a user attempt to double-spend funds in two transactions, for example, even if both transactions are confirmed during the consensus process, after the first transaction is applied, the second will fail, as the funds are no longer available. This robustness is due to the fact that transactions are applied deterministically, and that consensus ensures that all nodes in the network are applying the deterministic rules to the same set of transactions.

For a slightly different analysis, let us assume that the probability that any node will decide to collude and join a nefarious cartel is p_c. Then the probability of correctness is given by p^*, where:

$$p^* = \sum_{i=0}^{\lceil(\frac{n-1}{5})\rceil} \binom{n}{i} p_c^i (1-p_c)^{n-i} \tag{2}$$

This probability represents the likelihood that the size of the nefarious cartel will remain below the maximal

threshold of Byzantine failures, given p_c. Since this likelihood is a binomial distribution, values of p_c greater than 20% will result in expected cartels of size greater than 20% of the network, thwarting the consensus process. In practice, a UNL is not chosen randomly, but rather with the intent to minimize p_c. Since nodes are not anonymous but rather cryptographically identifiable, selecting a UNL of nodes from a mixture of continents, nations, industries, ideologies, etc. will produce values of p_c much lower than 20%. As an example, the probability of the Anti-Defamation League and the Westboro Baptist Church colluding to defraud the network, is certainly much, much smaller than 20%. Even if the UNL has a relatively large p_c, say 15%, the probability of correctness is extremely high even with only 200 nodes in the UNL: 97.8%.

A graphical representation of how the probability of incorrectness scales as a function of UNL size for differing values of p_c is depicted in Figure 1. Note that here the vertical axis represents the probability of a nefarious cartel thwarting consensus, and thus lower values indicate greater probability of consensus success. As can be seen in the figure, even with a p_c as high as 10%, the probability of consensus being thwarted very quickly becomes negligible as the UNL grows past 100 nodes.

3.3 Agreement

To satisfy the agreement requirement, it must be shown that all nonfaulty nodes reach consensus on the same set of transactions, regardless of their UNLs. Since the UNLs for each server can be different, agreement is not inherently guaranteed by the correctness proof. For example, if there are no restrictions on the membership of the UNL, and the size of the UNL is not larger than $0.2 * n_{total}$ where n_{total} is the number of nodes in the entire network, then a fork is possible. This is illustrated by a simple example (depicted in figure 2): imagine two cliques within the UNL graph, each larger than $0.2 * n_{total}$. By cliques, we mean a set of nodes where each node's UNL is the selfsame set of nodes. Because these two cliques do not share any members, it is possible for each to achieve a correct consensus independently of each other, violating agreement. If the connectivity of the two cliques surpasses $0.2 * n_{total}$, then a fork is no longer possible, as disagreement between the cliques would prevent consensus from being reached at the 80% agreement threshold that is required.

An upper bound on the connectivity required to prove agreement is given by:

$$|UNL_i \cap UNL_j| \geq \frac{1}{5}\max(|UNL_i|, |UNL_j|) \forall i, j \quad (3)$$

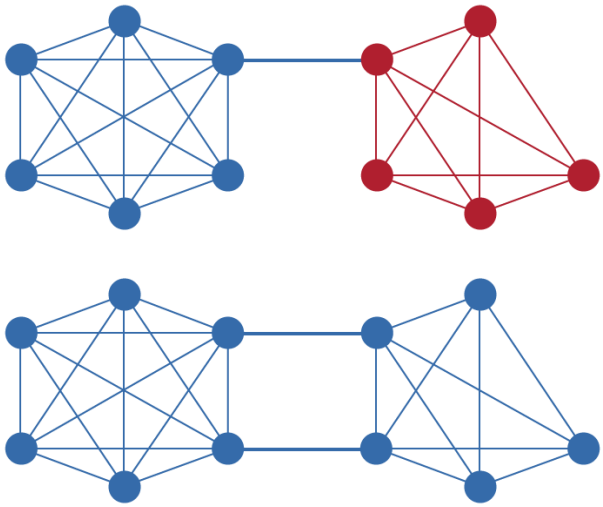

Figure 2. An example of the connectivity required to prevent a fork between two UNL cliques.

This upper bound assumes a clique-like structure of UNLs, i.e. nodes form sets whose UNLs contain other nodes in those sets. This upper bound guarantees that no two cliques can reach consensus on conflicting transactions, since it becomes impossible to reach the 80% threshold required for consensus. A tighter bound is possible when indirect edges between UNLs are taken into account as well. For example, if the structure of the network is not clique-like, a fork becomes much more difficult to achieve, due to the greater entanglement of the UNLs of all nodes.

It is interesting to note that no assumptions are made about the nature of the intersecting nodes. The intersection of two UNLs may include faulty nodes, but so long as the size of the intersection is larger than the bound required to guarantee agreement, and the total number of faulty nodes is less than the bound required to satisfy strong correctness, then both correctness and agreement will be achieved. That is to say, agreement is dependent solely on the size of the intersection of nodes, not on the size of the intersection of nonfaulty nodes.

3.4 Utility

While many components of utility are subjective, one that is indeed provable is convergence: that the consensus process will terminate in finite time.

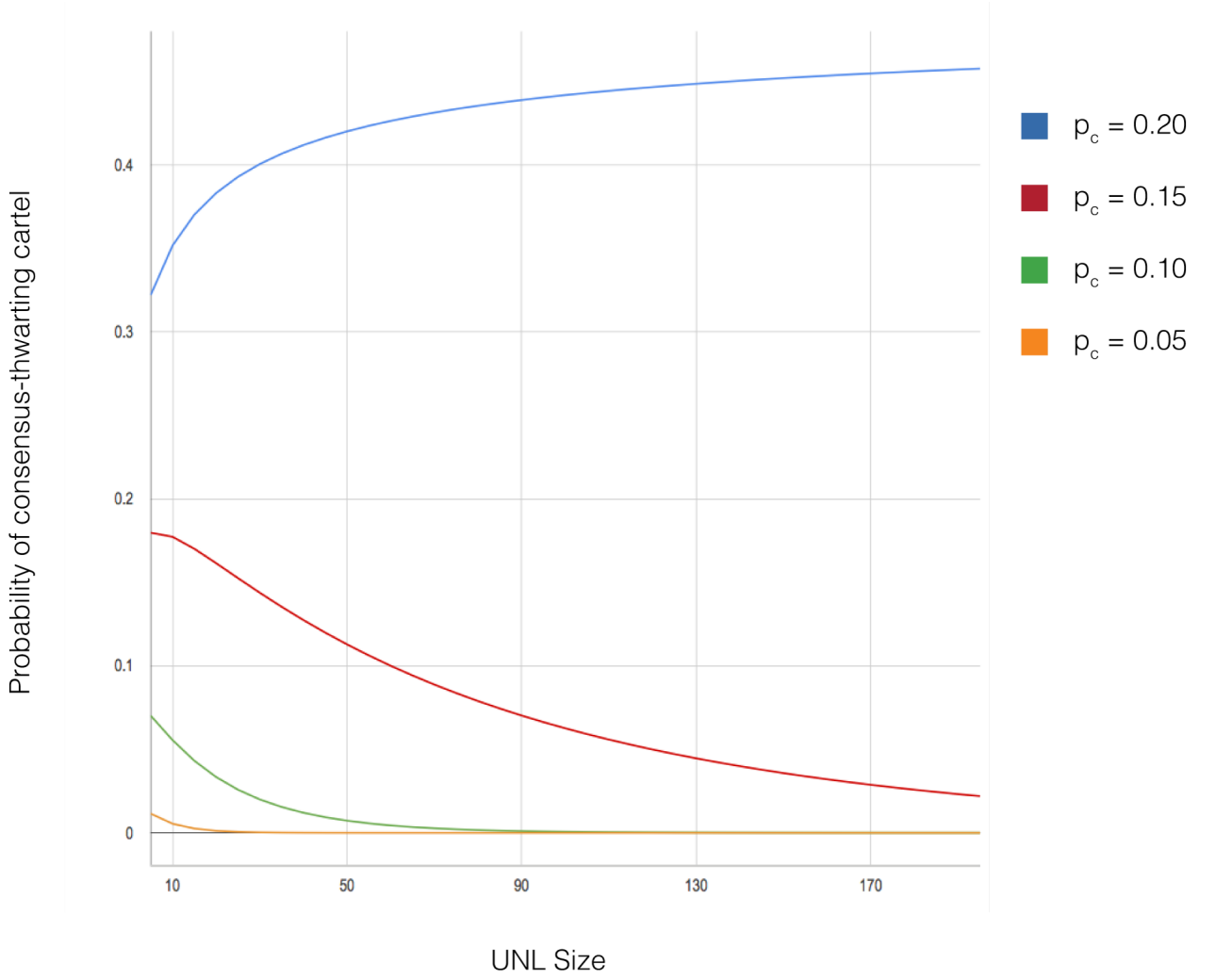

Figure 1. Probability of a nefarious cartel being able to thwart consensus as a function of the size of the UNL, for different values of p_c, the probability that any member of the UNL will decide to collude with others. Here, lower values indicate a higher probability of consensus success.

3.4.1 Convergence

We define convergence as the point in which the RPCA reaches consensus with strong correctness on the ledger, and that ledger then becomes the last-closed ledger. Note that while technically weak correctness still represents convergence of the algorithm, it is only convergence in the trivial case, as proposition **C3** is violated, and no transactions will ever be confirmed. From the results above, we know that strong correctness is always achievable in the face of up to $(n-1)/5$ Byzantine failures, and that only one consensus will be achieved in the entire network so long as the UNL-connectedness condition is met (Equation 3). All that remains is to show that when both of these conditions are met, consensus is reached in finite time.

Since the consensus algorithm itself is deterministic, and has a preset number of rounds, t, before consensus is terminated, and the current set of transactions are declared approved or not-approved (even if at this point no transactions have more than the 80% required agreement, and the consensus is only the trivial consensus), the limiting factor for the termination of the algorithm is the communication latency between nodes. In order to bound this quantity, the response-time of nodes is monitored, and nodes who's latency grows larger than a preset bound b are removed from all UNLs. While this guarantees that consensus will terminate with an upper bound of tb, it is important to note that the bounds described for correctness and agreement above must be met by the *final* UNL, after all nodes that will be

dropped have been dropped. If the conditions hold for the initial UNLs for all nodes, but then some nodes are dropped from the network due to latency, the correctness and agreement guarantees do not automatically hold but must be satisfied by the new set of UNLs.

3.4.2 Heuristics and Procedures

As mentioned above, a latency bound heuristic is enforced on all nodes in the Ripple Network to guarantee that the consensus algorithm will converge. In addition, there are a few other heuristics and procedures that provide utility to the RPCA.

- There is a mandatory 2 second window for all nodes to propose their initial candidate sets in each round of consensus. While this does introduce a lower bound of 2 seconds to each consensus round, it also guarantees that all nodes with reasonable latency will have the ability to participate in the consensus process.

- As the votes are recorded in the ledger for each round of consensus, nodes can be flagged and removed from the network for some common, easily-identifiable malicious behaviors. These include nodes that vote "No" on every transaction, and nodes that consistently propose transactions which are not validated by consensus.

- A curated default UNL is provided to all users, which is chosen to minimize p_c, described in section 3.2. While users can and should select their own UNLs, this default list of nodes guarantees that even naive users will participate in a consensus process that achieves correctness and agreement with extremely high probability.

- A network split detection algorithm is also employed to avoid a fork in the network. While the consensus algorithm certifies that the transactions on the last-closed ledger are correct, it does not prohibit the possibility of more than one last-closed ledger existing on different subsections of the network with poor connectivity. To try and identify if such a split has occurred, each node monitors the size of the active members of its UNL. If this size suddenly drops below a preset threshold, it is possible that a split has occurred. In order to prevent a false positive in the case where a large section of a UNL has temporary latency, nodes are allowed to publish a "partial validation", in which they do not process or vote on transactions, but declare that are still participating in the consensus process, as opposed to a different consensus process on a disconnected subnetwork.

- While it would be possible to apply the RPCA in just one round of consensus, utility can be gained through multiple rounds, each with an increasing minimum-required percentage of agreement, before the final round with an 80% requirement. These rounds allow for detection of latent nodes in the case that a few such nodes are creating a bottleneck in the transaction rate of the network. These nodes will be able to initially keep up during the lower-requirement rounds but fall behind and be identified as the threshold increases. In the case of one round of consensus, it may be the case that so few transactions pass the 80% threshold, that even slow nodes can keep up, lowering the transaction rate of the entire network.

4. Simulation Code

The provided simulation code demonstrates a round of RPCA, with parameterizable features (the number of nodes in the network, the number of malicious nodes, latency of messages, etc.). The simulator begins in perfect disagreement (half of the nodes in the network initially propose "yes", while the other half propose "no"), then proceeds with the consensus process, showing at each stage the number of yes/no votes in the network as nodes adjust their proposals based upon the proposals of their UNL members. Once the 80% threshold is reached, consensus is achieved. We encourage the reader to experiment with different values of the constants defined at the beginning of "Sim.cpp", in order to become familiar with the consensus process under different conditions.

5. Discussion

We have described the RPCA, which satisfies the conditions of correctness, agreement, and utility which we have outlined above. The result is that the Ripple Protocol is able to process secure and reliable transactions in a matter of seconds: the length of time required for one round of consensus to complete. These transactions are provably secure up to the bounds outlined in section 3, which, while not the strongest available in the literature for Asynchronous Byzantine consensus, do

allow for rapid convergence and flexibility in network membership. When taken together, these qualities allow the Ripple Network to function as a fast and low-cost global payment network with well-understood security and reliability properties.

While we have shown that the Ripple Protocol is provably secure so long as the bounds described in equations 1 and 3 are met, it is worth noting that these are maximal bounds, and in practice the network may be secure under significantly less stringent conditions. It is also important to recognize, however, that satisfying these bounds is not inherent to the RPCA itself, but rather requires management of the UNLs of all users. The default UNL provided to all users is already sufficient, but should a user make changes to the UNL, it must be done with knowledge of the above bounds. In addition, some monitoring of the global network structure is required in order to ensure that the bound in equation 3 is met, and that agreement will always be satisfied.

We believe the RPCA represents a significant step forward for distributed payment systems, as the low-latency allows for many types of financial transactions previously made difficult or even impossible with other, higher latency consensus methods.

6. Acknowledgments

Ripple Labs would like to acknowledge all of the people involved in the development of the Ripple Protocol consensus algorithm. Specifically, Arthur Britto, for his work on transaction sets, Jed McCaleb, for the original Ripple Protocol consensus concept, and David Schwartz, for his work on the "failure to agree is agreement to defer" aspect of consensus. Ripple Labs would also like to acknowledge Noah Youngs for his efforts in preparing and reviewing this paper.

References

[1] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Consulted 1.2012 (2008): 28.

[2] Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982): 382-401.

[3] Attiya, C., D. Dolev, and J. Gill. "Asynchronous Byzantine Agreement." Proc. 3rd. Annual ACM Symposium on Principles of Distributed Computing. 1984.

[4] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." Journal of the ACM (JACM) 32.2 (1985): 374-382.

[5] Martin, J-P., and Lorenzo Alvisi. "Fast byzantine consensus." Dependable and Secure Computing, IEEE Transactions on 3.3 (2006): 202-215.

[6] Alchieri, Eduardo AP, et al. "Byzantine consensus with unknown participants." Principles of Distributed Systems. Springer Berlin Heidelberg, 2008. 22-40.