Replication: Security Analysis of Ripple Consensus

Jake Ginesin, Ryan Zhu

1 Introduction

Ripple consensus is the Byzantine distributed consensus algorithm underpinning the XRP cryptocurrency. As of December 2024, XRP has the third largest market cap at just over 150 billion dollars, and XRP actively employed by *Ripple Labs*, a financial services company primarily providing foreign exchange services to businesses and banks.

However, despite the prominence of XRP, its underlying consensus protocol has not seen much scrutiny, especially in comparison to Bitcoin and Ethereum the first and second largest cryptocurrencies respectively. There have been several papers that have studied safety properties of Ripple using hand-written proofs [2], and one study of an attacker scenario with respect to derived safety and liveness properties [1]. However, no previous works have closely examined the Byzantine security of Ripple consensus using formal methods tooling; all previous work done on the protocol was hand-written.

In this paper we make the following contributions:

- We mechanize the Ripple protocol in the Promela language and verify our model using the Spin model checker.
- We formalize four linear temporal logic properties for the Ripple Consensus algorithm, derived from previous work.
- We create a simple implementation of the Ripple protocol based upon our model.
- We replicate the attack shown in [1] with both our Ripple model and implementation.
- We discuss the implications of our and the current state of the Ripple network.

2 The Ripple Protocol

2.1 Overview

The Ripple consensus protocol is *fundamentally different* than other consensus algorithms employed by other cryptocurrencies. Ripple consensus does not use Proof-of-Work or Proof-of-Stake for agreeing on new transaction proposals; rather, Ripple consensus uses a modified Byzantine consensus algorithm that does *not* rely on reversing a computationally hard one-way function. Instead, Ripple consensus is conducted through a set of validators. To achieve consensus in a Byzantine setting, the Ripple consensus protocol has two main features:

- **Unit Node Lists (UNLs)**. Each Ripple consensus validator pre-declares a list of trusted validators; for a validator to accept a proposal, it only requires consensus between its trusted validators.
- **UNL overlap**. Different UNLs are *assumed* to have overlap. The existing literature on Ripple consensus has disagreed on the amount of UNL overlap required for consensus in the Byzantine setting.

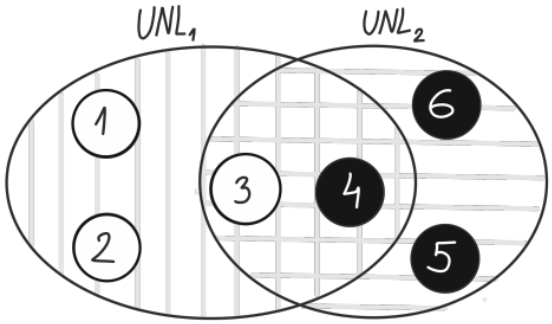

Figure 1: Visualization of a Ripple network configuration, taken from [2]. This network configuration features two UNLs: $\text{UNL}_1 = \{1, 2, 3, 4\}, \text{UNL}_2 = \{3, 4, 5, 6\}$. Notice, validators 3 and 4 have the most influence, as they are in the intersection of both UNLs.

2.2 Consensus Algorithm

The Ripple consensus protocol has three stages for agreeing on new proposals: *open*, *establish*, and *accepted*. The consensus phases are detailed as follows:

- **Phase Open**. All validators gossip transactions to each other, ensuring weak consistency. According to a cutoff timer determined by the length

of the previous consensus round, all validators construct a *proposal* with all recently received transaction requests. This proposal is then gossiped between the validators.

- **Phase Establish**. All validators agree on a proposal within their UNL via successive consensus rounds. The consensus rounds are bounded by heartbeat timers.
- **Phase Accepted**. Once a validator receives confirmation from a chosen majority (generally, 80% [2]) of validators within its UNL, it broadcasts an *accept* message to the validators within the UNL. Once each validator receives a majority-share of accept messages, the transaction proposal is committed. After this, the validator returns to *Phase Open*.

The timers that underscore *Phase Open* and *Phase Establish* essentially require the network is highly synchronous. Therefore we note Ripple consensus protocol is a highly *synchronous* consensus algorithm. For a full description of the Ripple protocol, we refer the reader to [1].

2.3 Properties of Ripple Consensus

As described in [1], Ripple consensus has four main properties:

- **Validity**: correct validators that propose correct transactions are eventually executed[1]
- **Agreement**: If a transaction is executed by a correct validator, then every other correct validator also eventually execute it
- **Integrity**: Correct validators do not execute a transaction more than once
- **Total Order**: All nodes execute transactions in the same order. That is, if validator A executes two transactions in a certain order, validator B will also execute those transactions in the same order

As further discussed in Section 3, *Validity* is best formulated as a liveness property, while *Agreement*, *Integrity*, and *Total Order* are safety properties.

2.4 Attacking Ripple Consensus

A recent work by Amores-Sesar et al. demonstrated that, contrary to previous proofs of Byzantine Agreement [2], a well-placed Byzantine node is able to violate

[1]We use "executed" here to mean that a validator commits a given proposal to its internal Ripple transaction ledger

safety and liveness [1]. To describe the attack simply, a Byzantine node that finds itself trusted by *multiple* UNLs violate safety and liveness of Ripple consensus by exhibiting split-brained behavior. That is, the Byzantine node agrees on one proposal with one UNL, and another proposal with the other UNL. An illustration is shown in Figure 2.

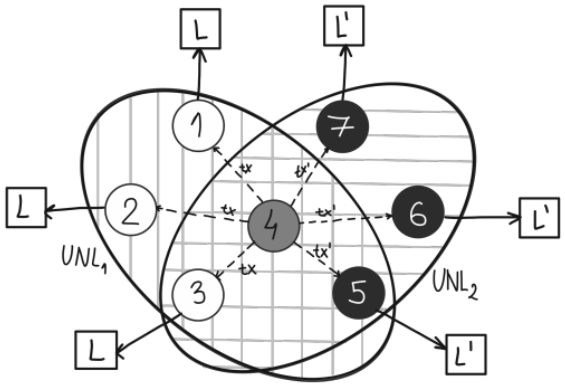

Figure 2: Visualization of an attack on the Ripple consensus protocol, taken from [1]. White nodes adopt one UNL, black nodes adopt another, and the grey node is Byzantine. The Byzantine node exhibits split-brained behavior, ensuring $\mathcal{UNL}_1$ and $\mathcal{UNL}_2$ agree on different proposals

3 Our Model

We construct a Promela model of the Ripple consensus protocol based upon the descriptions of the Ripple protocol in [1, 2], as well as the primary C++ Ripple implementation and the official Ripple specification. The most notable abstraction choice we make is

We formalize the four properties as described in Section 2.3 in Linear Temporal Logic. We formalize *Validity* as a liveness property, while *Agreement*, *Integrity*, and *Total Order* are safety properties. We provide two versions of our model: one with seven peers and two UNLs, and another with seven peers, two UNLs, and a single Byzantine validator.

The safety properties, *Agreement*, *Integrity*, and *Total Order*, pass on the non-Byzantine model, while the liveness property does not. The reason why we could not verify the liveness property with our Promela model of Ripple consensus is because Spin cannot model check probabalistic convergence of consensus. This is because in Spin, random variables are viewed *nondeterministically*; that is Spin

will explore the execution traces corresponding to every single possible value of the random variable. To property verify probabalistic convergence here, we would require a *probabalistic model checker* such as PRISM [4], or a constructive theorem proving methods.

For our Byzantine model, we find attack traces for *Agreement*, *Integrity*, and *Total Order*, effectively replicating the attack scenarios described in [1].

4 Our Implementation

We construct a simple Go implementation of the Ripple consensus protocol, based upon our Promela models. We replicate the attacker scenario as described in Section 2.4 with networked peers using docker-compose. We closely follow the pseudocode described in [1]. As of December 2024, Ripple is on version 2.3.0 as opposed to 1.6.0 in the original paper, but to the best of our knowledge the consensus protocol is not majorily updated since. We make many assumptions to ensure normal operation, namely:

- **Static Network**: We assume the network does not grow or shrink, meaning there are no crashes. We also assume the network is of size 7 for the attack to work properly.
- **Simplified Transactions**: A transaction typically can include a significant amount of information, but for our purposes we are only interested in consensus. Thus, transactions are modeled as single 32 bit integers.
- **Well-behaved**: All validators other than the byzantine node are well-behaved, and all messages are well formed
- **Fixed Round**: We run a single round as this is enough to showcase the security vulnerability within the network. This also means we have a fixed round time
- **Simplified Ledger**: Similar to the transactions, the ledger is interesting with respect to the currency. Since we are focused on the consensus, the ledger is largely ignored.

Instructions for running the code can be found in the README.md.

5 Discussion

As demonstrated in [1], and verified by our work, the Ripple consensus protocol is fundamentally flawed. Additionally, in recent years there has been ample progress on Byzantine agreement protocols with much stronger guarantees such as Hotstuff and Algorand [3]. We want to restate: XRP is the *third largest* cryptocurreny by market cap. So, we ask: why does XRP *still* use Ripple consensus? Additionally, why has there been *virtually no* discussion online regarding the Ripple consensus algorithm since [1] was published?

Primarily, we observe that in practice, Ripple consensus validators are *highly centralized*. The main validators for Ripple consensus are apart of just one UNL, "dUNL" (standing for, "default" UNL). Additionally, as of December 2024, Ripple consensus has *just 110* validators active on the network and *just 66* on the latest version, as shown in Figure 3. Just 110 validators for a cryptocurrency with a market cap of over 150 billion dollars is just insane.

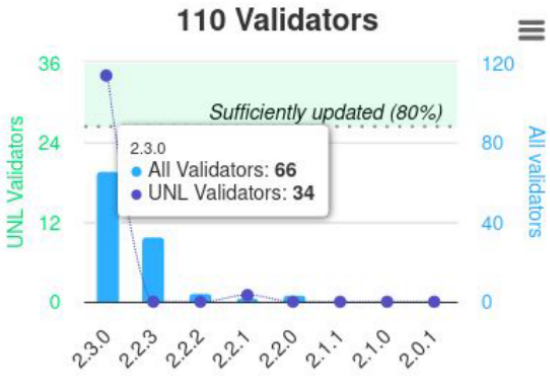

Figure 3: Screenshot from xrpscan.org, a website which tracks the current number of validators on the Ripple network.

It must be asked: why is the Ripple network in this state? The culprit, of course, is Ripple Labs, the company behind XRP and Ripple consensus. Ripple Labs currently owns approximately 50% of the XRP in circulation, and they reportedly receive a small fraction of their company's revenue from transaction fees from their financial products. And yet, Ripple Labs currently employs over a thousand people. Curiously, Ripple Labs makes most of their revenue essentially laundering XRP through an Asian subsidiary. Of course, this is essentially securities fraud, but they still get away with it. Ripple Labs spent over 100 million dollars from its

250 million in seed funding fighting the US Securities and Exchange Commission on the basis that XRP, a cryptocurrency, is not a security.

Needless to say, Ripple Labs isn't exactly invested in ensuring the Ripple consensus protocol is secure.

6 Conclusion

In this paper, we briefly describe the Ripple consensus protocol, formally model the protocol in Promela, verify our model using Spin, and create a small reference implementation. We use our model and implementation to replicate the attack on the Ripple protocol demonstrated in [1]. We conclude with a discussion on the state of the Ripple protocol.

References

[1] Ignacio Amores-Sesar, Christian Cachin, and Jovana Mićić. Security analysis of ripple consensus. (arXiv:2011.14816), November 2020. arXiv:2011.14816 [cs].

[2] Brad Chase and Ethan MacBrough. Analysis of the xrp ledger consensus protocol. (arXiv:1802.07242), February 2018. arXiv:1802.07242 [cs].

[3] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, page 51–68, Shanghai China, October 2017. ACM.

[4] Marta Kwiatkowska, Gethin Norman, and David Parker. *PRISM: Probabilistic Symbolic Model Checker*, volume 2324 of *Lecture Notes in Computer Science*, page 200–204. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.