Log inference on the Ripple Protocol: testing the system with an empirical approach

Marijn Roelvink[1] , **Mitchell Olsthoorn**[1] , **Annibale Panichela**[1]
[1]Delft University of Technology
m.a.t.roelvink@student.tudelft.nl, {a.panichella, m.j.g.olsthoorn}@tudelft.nl

Abstract

Ripple is a relatively new payments network that aims to improve the financial system by unifying its underlying infrastructure. Given its critical function, its system must be reliable and free of bugs. Therefore it should be tested extensively. One of the test methods that has not been used on it yet is log inference, a method that has a good potential for modelling complex communication protocols. Therefore, we have developed an empirical model of the Ripple Consensus Protocol by learning a Deterministic Finite Automaton (DFA) from the log files of two servers in the Ripple network. We have also developed a theoretical DFA of the Ripple Consensus Protocol and compared this to the empirical model to verify that the two systems function comparably. There has been found one notable difference between the two models, but whether this difference has a critical impact remains to be discussed.

1 Introduction

Ripple is an important up-and-coming payments network for making reliable global payments without too much delay. It accomplishes this by connecting our financial institutions in a unified infrastructure, making money flow as if it were data [17]. Through this approach, they have a big impact on our current society as they are improving the infrastructure that is already in place. 300+ financial institutions are already using Ripple to do their global payments [19]. As with all critical systems, it is of paramount importance that these systems work exactly as expected, e.g. no money gets lost or transactions get held up. The way Ripple ensures this is through their consensus protocol.

Debugging and verifying consensus protocols can be complex and different methods can be employed to track bugs and to verify that everything is working correctly. One method that has not yet been used for the Ripple protocol is performing log inference [12]. This method comprises three steps: gathering the log files of a validator in the Ripple network, learning an empirical model from these logs, and checking whether the implementation of Ripple matches the corresponding theory. This might be a missed opportunity as model learning can give a valuable insight into the actual behaviour of an algorithm and can process hundreds of trace instances of a routine automatically. Therefore, the purpose of this research is to develop a model using logs that are generated by two Ripple validators running in the live network, and to verify that this model is in line with the theoretical model of the Ripple Consensus Algorithm.

Model inference is the problem of deriving a minimal state machine from a series of inputs and outputs. This problem was first posed by Moore in 1956 [11]. Given that state machine inference from given data is an NP-hard problem [5], there have been many different tools and algorithms developed over the years that could generate near-optimal models in polynomial time ([10], [3], [9]). For this research, flexfringe is chosen as tool to apply log inference on the event logs [22]. It can infer different types of state machines and uses at its core a version of the evidence-driven state-merging algorithm that was first introduced by Rodney Price [7] and improved by Kevin Lang[8].

2 The Ripple network

Ripple functions as a distributed ledger [20]. A ledger is a collection of financial transaction records; it holds the information on all the accounts and a history of all the transactions that were applied to these accounts. The difference with a normal ledger though is that this ledger isn't maintained by one entity, but by multiple instances (nodes). These instances are often banks, financial institutions or governmental organizations [4]. Ripple works with ledger versions, each new ledger version is obtained by applying transactions to the previous ledger. A ledger version contains three elements: the current state of all the balances and objects, the set of transactions that were applied to its previous version to result in this one, and some metadata such as the current version number [16].

The nodes are connected through a peer-to-peer network. When a node shares information, it is accessible for all the peers in the network [18]. Each node in the network maintains its own copy of the current ledger. When a transaction is submitted to one of the nodes, this transaction

gets relayed through the network so that each node can add it to its ledger. Due to network failure or malicious participants, some transactions get delayed or manipulated, resulting in nodes maintaining different versions of the ledger. The main functionality of the consensus protocol is therefore to ensure that all nodes eventually have the same valid transactions applied to their copy of the ledger and that problems such as the double-spending problem cannot happen [14].

The consensus protocol runs in a continuous sequence of consensus rounds. Within each consensus round all the nodes try to agree upon a batch of transactions to apply to their current ledger, resulting in a new ledger for the next consensus round. The consensus round itself can be split into three phases: the *open* phase, the *deliberation* phase and the *validation* phase.

In the *open* phase all the nodes collect transactions to apply to their ledger. These transactions have either been submitted to the nodes during the *open* phase or failed to be included in an earlier consensus round and are tried again. When there are transactions in the ledger and there has passed a specified amount of time, the nodes close their ledgers and move on to the *deliberation* phase. Transactions that they receive after this time are added to a waiting list for the next consensus rounds.

In the *deliberation* phase the nodes send proposals to each other about the transaction sets they have composed. Each node updates its own transaction set depending on how much support it receives for individual transactions from other nodes. Transactions are added or removed based on the number of nodes that support them. When a node has updated its transaction set, the node sends out a new proposal for it. It then repeats the cycle of updating its transaction set based on the new proposals of other nodes and proposing again. This process converges quickly to a transaction set that the majority of the nodes agree with. When this happens, the nodes move on to the *validation* phase.

In the *validation* phase, each node starts applying the previously agreed upon transactions to its ledger. When this process ends, the nodes calculate the hash of their resulting ledgers and send it to each other in a signed message. This message is called a validation. The nodes then proceed to decide on the next batch of transactions and start a new consensus round. However, it can happen that a node has built a defective ledger or has applied a different transaction set than its peers due to problems such as network failure or malicious participants. Therefore, a ledger is only fully validated when a node has received the same validation from a majority of nodes that it trusts.

During the consensus process, a validator can operate in four different modes [13]. The two normal modes are *proposing* and *observing*. In the *proposing* mode the validator functions as a full-fledged consensus participant. It participates in the consensus process and sends out proposals for transaction sets. In the *observing* mode the validator functions as a silent participant: it keeps track of the proposals of other nodes, but it doesn't send out proposals itself. When a new consensus round starts, a validator always starts in one of those two modes. However, during the consensus round a node may receive too many validations for a different ledger than the one it is working on and conclude that it has been working on the wrong ledger. When this happens, it switches from the *proposing* or *observing* mode to the *wrongLedger* mode and requests from the other nodes the correct ledger. It also sends a message to the other nodes indicating that its mode has changed and stops proposing transaction sets. The node starts practically functioning as if it were in *observing* mode. When the node receives the correct ledger, it switches to *switchedLedger* mode, and keeps functioning as if it were observing. Only in the next consensus round can it be in *proposing* mode again. This is chosen with the reasoning that when a validator has received the correct ledger, it probably still is behind its peers, and should defer to them to decide on the final transaction set.

3 Methodology

The purpose of this research is to compare an empirical model with the theoretical model of the Ripple consensus protocol. However, to make a good comparison between the two models, they had to be defined with a similar type of model. Therefore, the first step of the research was finding a good model type that could describe both systems. The second step was then modelling the theoretical system in the same model representation. Only after this part was finished was the empirical model developed. We did this to prevent that we would fit the theoretical model too much to our empirical model. In this section, we will elaborate on each of the steps of this process.

3.1 Choosing the model format

In order to model both systems with the same model type, this model type had to be supported by flexfringe. Flexfringe supports multiple state machine types such as probabilistic and non-probabilistic deterministic finite automatons (P/DFA) and deterministic finite-state transducers like Mealy and Moore machines. We have chosen to go with a non-probabilistic DFA. The reason for the non-probabilistic state machine was that the consensus protocol is in essence deterministic, so a probabilistic model is not applicable. The DFA was chosen because it provided the most straightforward way of modelling the consensus protocol.

3.2 Developing the theoretical model

There are three important sources for the construction of the theoretical model: (1) the Ripple whitepaper [2], which gives a more abstract overview of the principles of the consensus algorithm; (2) the Ripple website [20], which gives a more detailed insight into the function of the consensus algorithm within the Ripple network; and (3) the Github docs [13], describing more in detail how the algorithm practically works. While developing the theoretical model we had to consider which source to choose as leading example. Relying on the whitepaper more would ensure that the model adhered

more closely to the original theory and correctness proofs. However, choosing the Github docs would result in a model that would be closer to the implementation and in this way easier to compare to the empirical one. In the end, the Github docs have been chosen as leading source. This was to ensure that the model would be less open for interpretation and more precisely defined according to Ripple's specifications.

Another consideration was what entry point and final point to use for the theoretical DFA. A first option was to look at the lifecycle of a specific ledger version. The starting point for the DFA would then be the conception of the ledger, e.g. the moment the consensus round for that ledger starts, and the ending point would be the moment that the ledger is fully validated or identified as incorrect. However, there were two reasons not to use this as scope. The first reason is that the lifecycle of a specific ledger version can be very variable and would bring many edge cases with it if we tried to capture it in a DFA. The second reason is that all the three sources of documentation have the consensus round rather than the specific ledgers as main point of view for describing the process. Therefore, we decided to follow the documentation in this aspect as well, and chose for a DFA whose initial state indicates the beginning of a consensus round, and whose final states mark the ending of a consensus round.

After having settled on these considerations, we developed the theoretical DFA. We started with defining the main flow according to the three phases and their subroutines. Then we combined this with the different modes and created flows for when the wrong ledger is being used. After a round of feedback, we adapted the original model and reached a final version.

3.3 Developing the empirical model

As mentioned before, we have used flexfringe for generating the empirical model. Creating the model was done in three parts. The first step was processing the log files into a format that could be used by the tool. The second step was tuning the parameters and experimenting with different input sizes to build the optimal model with flexfringe. The last step was validating the obtained model by calculating its recall and specificity.

To make a model that gives a comprehensible description of the consensus protocol, the logs had to be filtered to some considerable extent. The first filtering operation was choosing which level of the logs to use. The logs had 5 levels of importance: fatal, error, warning, info, debug, and trace. The debug level has been chosen since it was the narrowest scope of the logs that still managed to provide a clear picture of the ongoing processes. The second and most influential filtering operation was removing messages from parts of the program that were not related to the consensus protocol. The Ripple system has of course a whole infrastructure around the consensus protocol that leaves traces as well. However, since these parts aren't considered in the theory of the main Ripple protocol it wasn't relevant to include them here. After this considerable downsizing, only a few other parts have been filtered out, such as the details of applying the transactions to the ledger. These processes have namely been abstracted in the theoretical model as well.

After the filtering, the data was transformed into the abbadingo format. This format stems from a competition in 1998 in state machine generation [7] and is used by flexfringe as input format as well. Each line in the format is a sequence of strings separated by white spaces. The sequence represents a route of edges taken through a DFA. Flexfringe builds with these sequences a prefix tree acceptor, and then repeatedly merges states in this DFA to make a more compact representation which it returns as result. A prefix tree acceptor is a DFA built by making states from all the prefixes occurring in the input sample and constructing a minimal DFA with them [6]. See Figure 1 for an example. For our model, the logs were split into traces describing each one consensus round. This was done by going chronologically through the logs and cutting them up each time a call for a new consensus round was encountered.

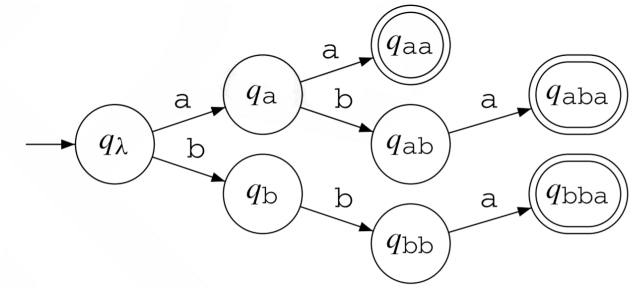

Figure 1: Example of a prefix tree based on input sample [aa, aba, bba], taken from De la Higuera [6]

One of the more challenging parts was developing the model with flexfringe. The tool has many parameters and different heuristics that it can use to derive a model from an abbadingo file. The final model was obtained via an experimental approach. Using a small input sample, we first developed DFA's using each of the heuristic types. After having found a good heuristic resulting in a generalized yet fitting DFA, the parameters were tuned again with a bit of trial and error and some pointers from the original developer of the tool. The input sizes and log sources have also been varied greatly, but they resulted in similar models.

3.4 Evaluating the empirical model

The last step was evaluating the performance of the model. This was done using two metrics: recall and specificity. These two metrics have been defined as in the work of Mariani et al [10].

Recall measures the completeness of a model, e.g. how many correct traces it accepts as valid. Given a model M for a program P and a set of legal traces T obtained by executing

P, the formal definition for the recall of M was determined as:

$$Recall(M) = \frac{\textit{number of traces that M accepts}}{\textit{number of traces in T}} \quad (1)$$

The recall of the final model was calculated by transforming a part of the logs into abbadingo sequences and checking whether each sequence would be accepted by the model. For this we developed a tool that could parse a given DFA and automatically check what number of traces would be accepted.

Specificity is the ability of a model to reject illegal behaviour. Similarly, the specificity of a model M for a program P and a set of illegal traces I for P was defined as:

$$Specificity(M) = \frac{\textit{number of traces that M rejects}}{\textit{number of traces in I}} \quad (2)$$

To calculate the specificity, illegal traces had to be generated. These traces have been generated by permuting each of the traces of the positive evaluation set into new illegal traces. The mutations were made by performing one or more *del*, *swap* or *swap-r* operations on the positive traces. The *del* operation removes a random element from the sequence, the *swap* operation swaps two consecutive elements with each other, and the *swap-r* operation swaps two randomly located elements in the trace. This method had a probability that these mutations accidentally generated legal traces, but it turned out during measuring that this makes for a negligible difference.

4 Results

4.1 Experimental setup

The final empirical model has been developed using the logs from two different validators (validator 1 and 2) on two different dates. Both validators are hosted at the Electrical Engineering, Mathematics and Computer Science faculty at Delft University of Technology. To catch more irregular execution paths, logs were chosen on days that the validators didn't have a 100% agreement score, e.g. they did not always validate the same ledgers as their peers. The final abbadingo file contained 180 sequences from the logs of validator 1, and 180 sequences from validator 2. The logs from validator 1 were taken from the moment it was rebooted, resulting therefore in some irregular sequences as the validator needed to catch up with its peers. The logs from validator 2 came from a day when validator 2 was presumably performing less than 100%.

The heuristic that was chosen for developing the model is called count_driven, the choice for this was empirically motivated. There were many other parameters that could be tweaked as well, their final values are listed in Table 1.

Parameter	Value
heuristic-name	count_driven
data-name	count_data
state_count	-1
symbol_count	-1
sinkcount	0
satdfabound	2000
sataptabound	10
extend	0
extrapar	0.1
sinkson	0
finalred	1
lowerbound	20
blueblue	1
largestblue	1

Table 1: Parameter values used for the empirical model

4.2 Result

Figure 2 depicts the theoretical DFA from the viewpoint of a single node. When the node is working on the correct ledger and no other unexpected events happen, the node will follow the main flow from the *init* state to the *done* or *fully-validated* state in a straight line. The states above the main flow handle the case when a node has been working on the wrong ledger. The state below the main flow handles the premature ending of a round. This is not expected to happen but nonetheless included in the Github documentation of Ripple [13] as a possible event.

Figure 3 outlines the resulting empirical model. This model is an edited version of the model that was produced by the flexfringe program. The model in Figure 3 does not differ in its functioning from the original model, it has just been simplified and better elaborated to make for better readability. The edges indicate events and log messages, and the labels in the states indicate how often they have been traversed by the traces.

The main flow of the theoretical and empirical model is similar. The open and deliberation phase are not distinguishable from each other in the empirical model because the logs do not give a clear signal when the validator has closed its ledger. However, the transition from the deliberation to the accepting phase is clearly marked by one edge. This edge with the "*peers have converged to consensus*" label is the only one leading from the *deliberation* phase to the *validating* phase. This edge also shows that the *validating* phase is only reached when the validator believes that the nodes have reached a consensus.

However, the different modes of the Ripple protocol have less impact on the general structure of the empirical model. It is clearly visible though that when the validator discovers it has the wrong ledger, it switches to *wrongLedger* mode and starts to retrieve the new ledger. Similarly, when the validator has retrieved the correct ledger it switches to *switchedLedger* mode. But regardless of whether the validator has the correct ledger, when its peers have reached consensus, it still applies the new transaction set and sends out a validation even if it is aware that it has the wrong ledger. This case was not

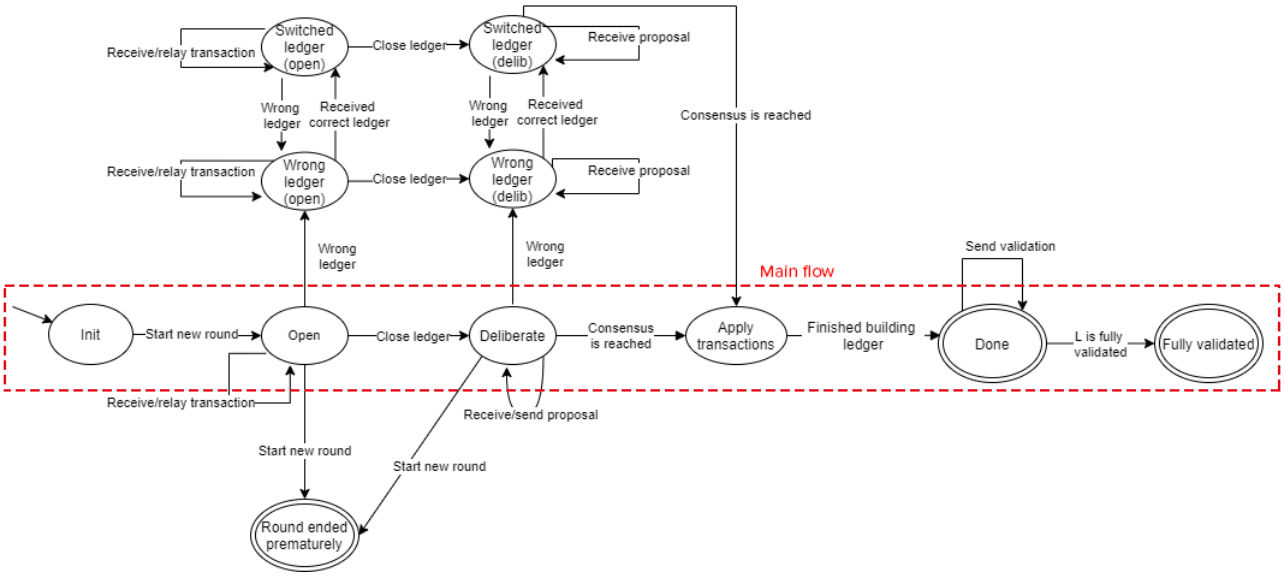

Figure 2: Theoretical DFA of the Ripple consensus protocol

described in the original white paper, due to the fact that that paper did not take into account that fetching the new ledger might take some time. This transition was also not included in our theoretical model. We assumed that applying transactions on an incorrect ledger would be futile and therefore not a valid execution path, but it presumably was the most straight forward way of keeping the validator up to date. This implementation choice probably also prevented the code from being too complex and generating too many edge cases. It is however a distinct difference with the theoretical model.

A process that does distinctly come forward in this model is the proposal round. Whenever the validator updates its position during the *establish* phase, it first indicates whether it has proposals from peers with one of the two "*timerentry*" edges. It then proceeds to update its transaction set with the new proposals, and then checks whether the peers have reached consensus.

4.3 Evaluation

In table 2 the values for the recall and specificity are given. For calculating the recall, we used an evaluation set comprised of 560 traces. In this set, 280 traces originated from validator 1, and 280 traces originated from validator 2. We generated these traces from the log files with the same script that was used to generate the abbadingo sequences for the model. All of the traces in the evaluation set have not been used for developing the model.

To calculate the specificity, we have generated three new sets from the aforementioned evaluation set by permuting each sequence in the evaluation set. For each new set, we increased the number of permutations we applied to each sequence. This resulted in one set with one permutation per sequence, one set with two permutations per sequence, and one set with three permutations per sequence. For each permutation it was randomly determined whether to apply a *del*, *swap* or *r-swap* operation and at which location it was to be applied. As one can see in Table 2, the specificity quickly converged to one when the number of permutations was increased.

Metric (N=560)	**Score**
Recall	0.9982
Specificity	
#permutations=1	0.8580
#permutations=2	0.9736
#permutations=3	0.9948

Table 2: Recall and specificity of the empirical model, calculated respectively with equation 1 en 2. Each value has been calculated from a different set with 560 traces.

5 Discussion

The purpose of this paper is to assess to what extent the empirical model matches the theoretical model. We can conclude that this is mostly true. The important transitions from the *deliberation* phase to the applying of transactions to the validating are all included in the empirical model in a straightforward and absolute way. The edges in the theoretical model that indicated a premature ending of a round were also not encountered in the empirical model. This matches the GitHub documentation with the idea that this should not happen. This shows that even when the validators are functioning less than perfectly, they still follow the same main flow. In fact, in each consensus round the validators have ended in the validation phase. The only striking difference between the empirical and theoretical

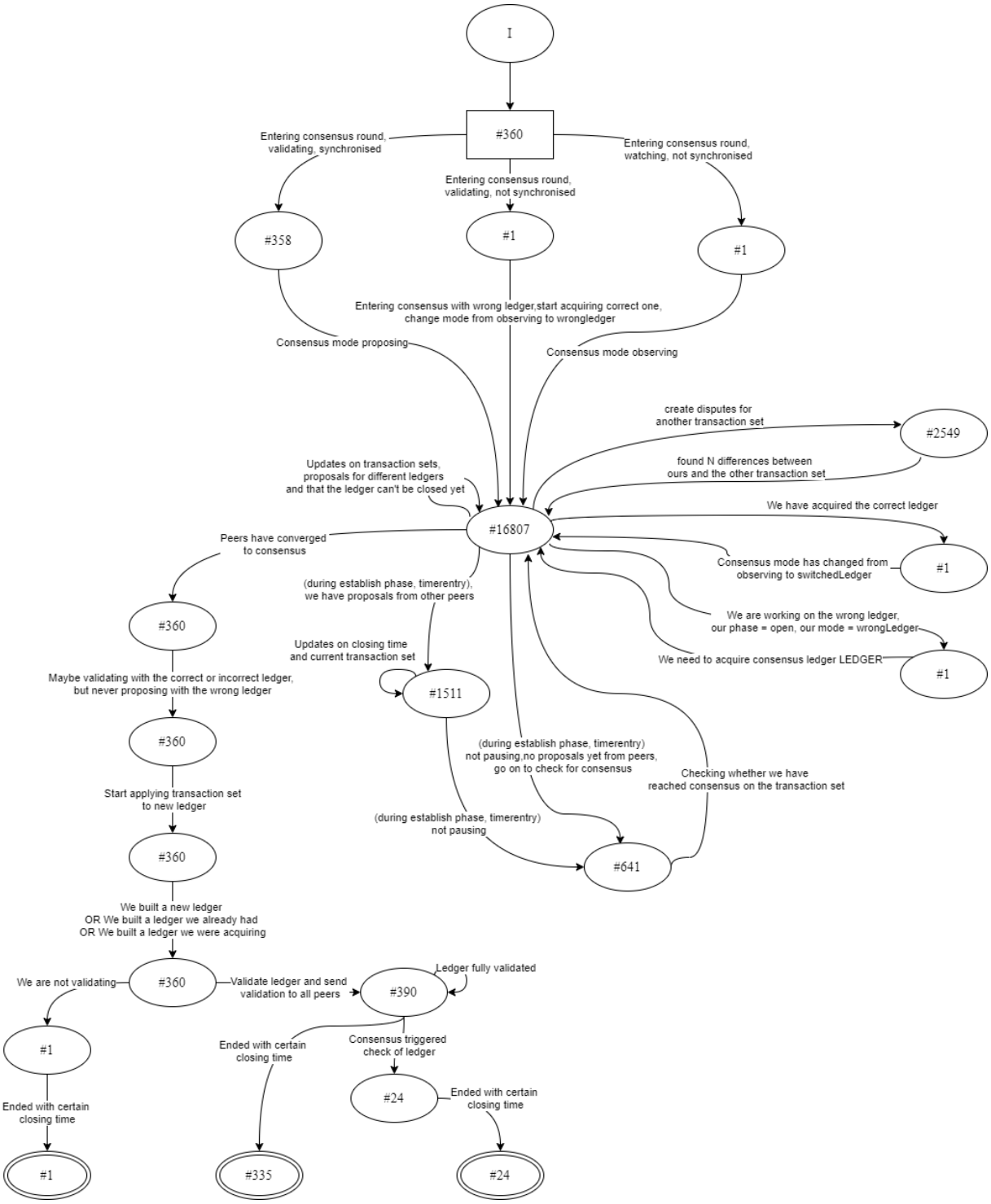

Figure 3: Final empirical model, edited for clarity

model is that even with the wrong ledger the validator still applies the transactions and builds a new ledger. However, this should probably not make a big difference in the general functioning of the Ripple consensus protocol.

Of course, this way of modelling a system is quite limited in its details. Other tests should be, and are performed by Ripple and other independent institutes to verify that the whole system functions as it was specified [15]. Our model does not indicate for example what Boolean conditions need

to be satisfied to realistically enter certain states. The model can describe what the order of events should be in the system, but not what specific details should be causing the transition from one state to another. A possible solution for this would be to develop a guarded DFA, like the one described by Mariani et al. [10]. Another limitation of this model is the range of data it covers. Now there is only data from two validators originating from two different days due to the time span of this research. To catch more edge cases the validators should be monitored over a longer period of time.

6 Responsible research

According to [1], responsible research covers a range of practices considered as instrumental for building an innovative, open and sustainable society. A few examples of these practices are considering the implications and consequences of the research, performing the research in an ethical way, making results that are beneficial to a wider part of the community, and being transparent and openly accessible with the methods and results.

Considering ethical conduct, this research is in a safe zone since no human interaction or human data was needed to analyse the Ripple algorithm. As the main purpose of the research was testing a system, the only place where there was further possible room for morally ambiguous behaviour was when critical bugs would be discovered and not properly reported. Luckily, this has not been the case.

One area where one could wonder whether this research is responsible is whether assisting in developing such a cryptocurrency system is beneficial for the wider public in the short-term or long-term run, or only for a small privileged few. Ripple aims to revolutionize the financial system from within, by making transactions between each of the financial institutions go in a unified and seamless way. One could argue that such an improvement of efficiency mostly helps the bankers and corporations make more money. However, Ripple also makes the global financial system available to people and entrepreneurs over the whole world, regardless of the currency in their countries and how developed their banks are. By dramatically reducing the time and cost of cross-country transactions even small entrepreneurs are enabled to take a part in the globalisation process [23]. Next to that, XRP (Ripple's currency) is also one of the most sustainable currencies currently on the market [21]. Of course, there can be held a lengthier debate about the ethical implications of cryptocurrencies, but we conclude for now that developing this system has added value for the society as a whole.

The last consideration on responsibility is whether our methods and results are transparent and openly accessible. We have tried as much as was relevant to explain all the steps in the process of this research. The main results will be disclosed in the appendix and the repositories with the code and all the other results will also be made available. With all these practices we hope to conclude that the research was done responsibly.

7 Conclusion

Though this research does not cover all the aspects of the Ripple system, we can conclude that the Ripple consensus protocol implementation seems to follow the specifications of Ripple, given the data that we had. Only one significant difference has been found between the theoretical and empirical model, and the obtained empirical model appears to be coherent.

For future work, other types of models such as guarded DFA's could be inferred, and more data could be gathered to achieve a better representation of the Ripple consensus protocol. Model inference could also be applied to other elements of the Ripple infrastructure, to provide a more complete overview of the functioning of the Ripple system.

References

[1] Caixa. What does rri mean? https://www.rri-tools.eu/research-community, 2020. Accessed: 2020-05-30.

[2] Brad Chase and Ethan MacBrough. Analysis of the xrp ledger consensus protocol, 02 2018.

[3] Weidong Cui, Jayanthkumar Kannan, and Helen Wang. Discoverer: Automatic protocol reverse engineering from network traces. 01 2007.

[4] David Schwartz Dave Cohen and Arthur Britto. The xrp ledger protocol – consensus and validation. https://xrpl.org/consensus.html#the-xrp-ledger-protocol-consensus-and-validation, 2019. Accessed: 2020-06-17.

[5] Mark E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[6] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[7] Kevin Lang, Barak Pearlmutter, and Rodney Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. volume 1433, pages 1–12, 01 1998.

[8] Kevin J. Lang. Evidence driven state merging with search, 1998.

[9] David Lo and Siau-Cheng Khoo. Smartic: Towards building an accurate, robust and scalable specification miner. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, page 265–275, New York, NY, USA, 2006. Association for Computing Machinery.

[10] Leonardo Mariani, Mauro Pezze, and Mauro Santoro. Gk-tail+ an efficient approach to learn software models. *IEEE Transactions on Software Engineering*, 43(8):715–738, 2017.

[11] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, december 1956.

[12] Dr. A. Panichella. Ripple: Improving a major decentralized money transfer network. https://projectforum.tudelft.nl/course_editions/23/projects/845, 2020. Accessed: 2020-04-22.

[13] Ripple. Consensus and validation. https://github.com/ripple/rippled/blob/develop/docs/consensus.md, 2018. Accessed: 2020-05-05.

[14] Ripple. Consensus principles and rules. https://xrpl.org/consensus-principles-and-rules.html, 2019. Accessed: 2020-05-03.

[15] Ripple. Consensus protections against attacks and failure modes. https://xrpl.org/consensus-protections.html#software-vulnerabilities, 2019. Accessed: 2020-06-09.

[16] Ripple. Ledger history. https://xrpl.org/intro-to-consensus.html#ledger-history, 2019. Accessed: 2020-06-09.

[17] Ripple. Our story. https://ripple.com/company, 2019. Accessed: 2020-06-17.

[18] Ripple. Software ecosystem. https://xrpl.org/software-ecosystem.html#rippled-the-core-server, 2019. Accessed: 2020-06-17.

[19] Ripple. Who runs on ripple. https://ripple.com/customers, 2019. Accessed: 2020-04-22.

[20] Ripple. Xrp ledger overview. https://xrpl.org/xrp-ledger-overview.html, 2019. Accessed: 2020-05-16.

[21] Team Ripple. 50 years of earth day: A look at the sustainability of currency. https://ripple.com/insights/50-years-of-earth-day-a-look-at-the-sustainability-of-currency/, Apr 2020. Accessed: 2020-05-31.

[22] Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017.

[23] Emi Yoshikawa. Block stars: How digital assets will help create a sustainable global economy. https://ripple.com/insights/block-stars-how-digital-assets-will-help-create-a-sustainable-global-economy/, May 2020. Accessed: 2020-05-31.