IMPROVEMENTS IN CHARACTERISTICS OF CRYPTOCURRENCIES: RIPPLE

A Project Report

submitted by

MAYANK DEENBANDHU MUNDHRA

in partial fulfilment of the requirements
for the award of the dual degree of

BACHELOR OF TECHNOLOGY and MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **IMPROVEMENTS IN CHARACTERISTICS OF CRYPTOCURRENCIES: RIPPLE**, submitted by **Mayank Deenbandhu Mundhra**, to the Indian Institute of Technology, Madras, for the award of the dual degree of **Bachelor of Technology** and **Master of Technology**, is a bona fide record of the work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Chester Rebeiro
Dual Degree Project Guide
Assistant Professor
Dept. of Computer Science
IIT-Madras, 600 036

Prof. Ramkrishna Pasumarthy
Dual Degree Project Co-Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 14th June 2018

ACKNOWLEDGEMENTS

I would like to thank my guide Prof. Chester Rebeiro for his constant guidance and the patience and faith he put in me.

I would like to thank my co-guide Prof. Ramkrishna Pasumarthy for facilitating me to pursue this inter-disciplinary project.

I would like to thank the various teachers who have, over the course of time, taught me and helped shape who I am.

I would like to thank the Department of Computer Science, Department of Electrical Engineering and IIT Madras as a whole for providing me with the opportunity to follow my dreams, pursue my passion and grow tremendously.

Lastly, I would like to thank my parents, grandparents and sister whose constant motivation, support and guidance I am fortunate to have and am indeed indebted to them.

ABSTRACT

KEYWORDS: Cryptocurrency; Blockchain ; Byzantine Generals Problem; Real Time Gross Settlement ; Ripple ; Unique Node List ; Kelips ; Distributed Hash Table ; Network Overlay ; Hops ; Information Propagation.

A LaTeX class along with a simple template thesis are provided here. These can be used to easily write a thesis suitable for submission at IIT-Madras. The class provides options to format PhD, MS, M.Tech. and B.Tech. thesis. It also allows one to write a synopsis using the same class file. Also provided is a BIBTeX style file that formats all bibliography entries as per the IITM format.

The formatting is as (as far as the author is aware) per the current institute guidelines.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF TABLES v

LIST OF FIGURES vi

ABBREVIATIONS vii

1 INTRODUCTION 1

1.1 Cryptocurrency 1

1.1.1 The Double Spending Problem 1

1.1.2 The Byzantine Generals Problem 2

1.2 Bitcoin 2

1.3 Ripple 3

1.3.1 Comparison between Ripple and Bitcoin 3

2 LITERATURE SURVEY 4

2.1 Resources utilised and content surveyed 4

2.2 Select concepts from literature survey 5

2.2.1 Ripple Protocol Consensus Algorithm 5

2.2.2 Kelips 8

2.3 Introduction 10

3 OUR CONTRIBUTION 13

3.1 Fast Full Network Knowledge 13

3.1.1 Distributed payment sytems, their utility and value goals. . . 13

3.1.2 Towards preventing forks, speeding up consensus, dynamically updating UNLs and providing last mile connectivty using p2p inspired network topologies 15

3.1.3 Analysis - Information propagation, overlap and threshold. . 20
3.2 Simulation 25
3.2.1 Experimental setup 25

LIST OF TABLES

3.1 Results for SimC - The classic version used by Ripple for simulation 26

3.2 Results for UNL-A Link Latency Factor = 1 26

3.3 Results for UNL-A Link Latency Factor = 2 27

3.4 Results for UNL-A Link Latency Factor = 3 27

3.5 Total number of links in the network (Valid for all Link Latency Ratios) 27

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RPCA	Ripple Protocol Consensus Algorithm
DHT	Distributed Hash Table
UNL	Unique Node List
NML	Network Members List
P2P	Peer to Peer
RTGS	Real Time Gross Settlement

CHAPTER 1

INTRODUCTION

With this dual degree program, we embarked on a journey of learning, fun and research. We started with reading up relevant chapters of Andreas Antonopoulos' book Mastering Bitcoin. We then explored the relevance and possibility of implementing blockchain and bitcoin technology to make chit funds more secure and prevent frauds and finally zeroed down on bringing about improvements in Ripple from a security perspective and in general.

Over the course of the project, a huge amount of information was learnt, knowledge gained and we have attempted to create value by bringing about improvements in Ripple, and in general, so as to forward the field of Computer Science and salient areas of Electrical Engineering. The ideas generated as a result of this project and papers to be generated, will have applications in multiple areas of Computer Science and Electrical Engineering. It's applicability would range from computer systems, communication networks, consensus algorithms, smart grids, etc, and the scope of its application is limitless. In this thesis we provide insights into and a small glimpse of the various knowledge and many key concepts learnt, and ideas and systems generated.

1.1 Cryptocurrency

Cryptocurrency is a digital asset designed to work as a medium of exchange and as a virtual or alternate currency. Cryptocurrencies are able to satisfy and solve the Double Spending Problem and the Byzantine Generals Problem which are important challenges faced towards the implementation of Real Time Gross Settlement (RTGS) and Payment Systems and are thus able to function as the same, generally as distributed systems.

1.1.1 The Double Spending Problem

The Double Spending Problem refers to the promise satisfaction and resource shortfall problem arising when the same set of finite resources is pledged to multiple entities/

persons in lieu of goods or services rendered.
This problem is elegantly solved by blockchain, a chain of transaction blocks visible to all, easy to verify but difficult to tamper, which is thus utilised by cryptocurrencies. It is assumed that each participant can see the exact same consistent version of the blockchain at any given time (even after addition of new blocks).The assumption is difficult to satisfy in distributed settings owing to challenges such as latency, genuinely faulty nodes, malicious nodes

1.1.2 The Byzantine Generals Problem

The Byzantine Generals Problem refers to the challenge of achieving consensus in a group of generals via message passing. There may be possibility of the messages being delayed, not reaching, being forged, some of the generals being traitorous and preventing the group to reach at an appropriate consensus/ any consensus at all.
A system with genuine, faulty and malicious nodes connected directly or indirectly and with the network latency constraints can be considered as a representative system for the Byzantine Generals Problem in the blockchain and distributed systems domain. Cryptocurrencies attempt to solve the Byzantine Generals Problem by employing approaches such as Proof-of-Work, Proof-of-Stake and Consensus

1.2 Bitcoin

Bitcoin is a well-known and remarkable cryptocurrency which first introduced blockchain technology, bringing about a breakthrough in solving the Double Spending Problem and thus making it feasible for cryptocurrencies to serve as Real Time Gross Settlement (RTGS) and Payment Systems. Bitcoin was created/ invented by an unknown person or group of persons going by the name Satoshi Nakamoto. It is open source in nature.
Bitcoin is decentralised and works as a distributed peer to peer network, thus being able to work in the absence of a single centralised authority, bank or administrator and conduct transactions between users directly and in the absence of an intermediary.
It solves the Byzantine Generals Problem using Proof-of-Work, making it easy to verify new *blocks* (a group of newly broadcast transactions to be added to the blockchain) but difficult and time-consuming to produce. New blocks, which satisfy the Proof-of-Work

constraint, are generated by a process called mining. These new blocks are created in an expected block interval of 10 minutes. Mining helps incentivise the process of maintaining and building the blockchain by rewarding a certain number of bitcoins to the first miner who generates the accepted/ acceptable block.

1.3 Ripple

Ripple is a blockchain based distributed payments system and cryptocurrency by Ripple Labs, based on the white paper "The Ripple Protocol Consensus Algorithm" by David Schwartz, Noah Youngs, Arthur Britto.
Ripple solves the Double Spending Problem using distributed ledger and SHAMap to record the state of the system and solves the Byzantine Generals Problem using consensus via it's Ripple Protocol Consensus Algorithm (RPCA).
Ripple works on a network of servers running the Ripple Protocol Consensus Algorithm (RPCA). The state of the Ripple network and transactions is maintained in a distributed ledger to which new transactions are added post consensus and validation, once all servers agree on the transactions' validity. Each server relies on a Unique Node List (UNL), a list of trusted servers which it believes won't collectively defraud it. As per the server, the network has reached consensus on a transaction when a quorum (minimum percentage) of servers in its UNL agree upon the transaction's validity.
Ripple is currently being adopted by various financial institutions as it provides the benefits of cryptocurrencies and distributed decentralised systems while still having a party (Ripple Labs) which takes up necassary liabilities.

1.3.1 Comparison between Ripple and Bitcoin

Ripple uses consensus which is less intensive computationally and energy-wise compared to Bitcoin. A new block is added quicker in Ripple (4 seconds) as compared to Bitcoin (10 minutes), ensuring that transactions and economic activity proceed in near real-time and without any lag and, with similar security guarantees as that of Bitcoin.
Ripple is partly open-sourced and chaperoned by Ripple Labs, till an appropriate time when it can be taken up by a robust Ripple community, becoming fully open-sourced.
Bitcoin is fully open-sourced

CHAPTER 2

LITERATURE SURVEY

2.1 Resources utilised and content surveyed

Below is a list of some of the relevant resources utilised and content surveyed as part of the literature survey for this project

Books

- Mastering Bitcoin by Andreas Antonopoulos - Read select chapters

Papers and Reports

- The Ripple Protocol Consensus Algorithm by David Schwartz, Noah Youngs, Arthur Britto
- Kelips : Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead by Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse
- The Byzantine Generals Problem by Leslie Lamport, Robert Shostak, and Marshall Pease
- Ripple: Overview and Outlook by Frederik Armknecht, Ghassan O. Karame, Avikarsh Mandal , Franck Youssef, Erik Zenner
- Ripple Protocol Consensus Algorithm Review by Peter Todd - Read till Subsection 4.1 on Unique Node List
- *B*yzantine-*R*esilient R*a*ndom *M*embership *S*ampling (BRAHMS) by Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer - Utilised powerpoint presentation from FuDiCoIII Technical Session 6a: Perspectives on BFT 1

Courses

- Cryptography and Network Security, course by Prof. Chester Rebeiro (IIT Madras)
- Distributed Network Algorithms: Foundations and Future Directions, course by Prof. John Augustine (IIT Madras) and Prof. Gopal Pandurangan

- Cloud Computing Concepts - Part I, Coursera course by Prof. Indranil Gupta (UIUC) url: https://www.coursera.org/learn/cloud-computing
- Foundations of Memory and Consistency Models, course by Prof. Krishna Nandivada (IIT Madras) and Prof. Suresh Jagannathan

Tech Talks and Videos

- Ripple Tech Talk: Understanding Consensus (Mar 2015) by Ripple. url: https://youtu.be/7abKUs9tYZg
- Ripple Explained with David Schwartz, Chief Cryptographer of Ripple Labs by Naation. url: https://youtu.be/GyNXedeCyNg
- Epicenter EB92 âĂŞ Stefan Thomas: Understanding Ripple by Epicenter. url: https://youtu.be/KMydq_m9f_4
- How Ripple Works - The Consensus Process (ADVANCED) by Ripple. url: https://youtu.be/pj1QVb1vlC0

Ripple Codebase

- Ripple Github repository

Other knowledge sources

- Ripple Website
- Ripple Forum
- Ripple Developer Center

2.2 Select concepts from literature survey

starting with a brief description of Ripple's consensus algorithm, UNL overlaps, relevant work and a relevant analysis from our end

2.2.1 Ripple Protocol Consensus Algorithm

In this subsection we describe the following

1. Components of Ripple and it's consensus algorithm, as is in the RPCA white paper
2. Details of the Ripple Protocol Consensus Algorithm
3. Details on agreement and forking with relevance to the consensus algorithm

Relevant components

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 users 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.

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 servers 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.

Agreement and forking

Agreement is the property that ensures that all the nodes in the distributed network agree to the same common version of the ledger and thus ensures absence of "forks" in the ledger where one of two or more different versions of ledger are accepted and exist as "ground truth" for certain subsets of nodes. The presence of fork results in the problem of "double-spending" resolving which is at the heart and crux of blockchain technologies and prevents defrauding of legitimate users by spending the same amount of money twice. It was proposed in the Ripple white paper that a minimum 20% overlap in the UNL suffices to prevent forks in the network when the consensus algorithm requires agreement of a threshold of at least 80% of a node's UNL. However, after a formal analysis of the forking criteria by [Ghassan,et al], they have come up with a mathematical formula **.....**to ensure that forks do not occur in the system. By this formula for ensuring no forks in the current Ripple system with a threshold of 80%, it is necessary that at least 40% of the UNLs overlap between any two nodes. This is the condition to prevent forks as mentioned by [Ghassan, et al].

2.2.2 Kelips

Kelips is a third generation peer-to-peer (p2p) Distributed Hash Table (DHT) system by Indranil Gupta, et. al. It achieves constant $O(1)$ time file look-up complexity by having increased memory and background overhead. It adapts quickly to churn and is tolerant to failures and faults through efficient query re-routing. This helps ensure most/all queries are serviced quickly.
One of the factors that makes using Kelips feasible in today's scenarios is the trade-off that memory is cheap and easily available while processing/ bandwidth is expensive. Bandwidth/ latency costs can be overcome by efficient query routing and appropriate selection of parameters for selecting affinity groups (eg. geographical distance, network speeds, etc.).

At the core of Kelips lies the concept of affinity groups. Consider a distributed system of N nodes. The nodes are divided into $\sqrt{N}$ buckets/affinity groups of size $\sqrt{N}$ each. The assignment of node to specific buckets is on the basis of hash functions and the parameter chosen for input for these hash functions and based on application.

Similarly files are stored in specific affinity groups selected on a similar basis of hashing.

Memory: Each node stores

- the file tuple entries (file name and IP address+port number pairs) for all files which are hashed to the node's affinity group.
- IP address and port number pairs for all nodes in the said node's affinity group.
- IP address and port number pairs for w nodes each in each of the other affinity groups (of which the node is not a part of).

Affinity groups: As previously mentioned each node is associated with a unique (node) affinity group determined on the basis of a hash function.

Similarly each file tuple entry (file detail) too is associated with a specific file affinity group(similar/same as node affinity group) determined on the basis of the above mentioned hash function. This entry is stored in all/most nodes associated with the said file affinity group.

Node join:

- the parameter gets hashed and the home affinity group is identified
- node contacts a well know introducer for that affinity group, which supplies
  - list of contacts/nodes in the affinity group.(Could be a complete or partial list)
  - list of members it has for each of the other affinity groups.(Could be a list of w nodes per affinity group or all known nodes)
- (optional) the node could also ask members of its own affinity groups and members of other affinity groups which it knows for relevant members' list
- the node intimates all nodes in its list about it's presence. The introducer too may intimate its contacts on the new node join.
- Apart from this there is a heart beat mechanism that intimates on a node's liveness and keeps the contact list fresh.

File look-up or File query. (Including multi hop)

There can be many variations and optimisations to the file search protocol. We'll just mention a simplified version of the same.

It can be divided into two main parts - 1. file look-up and file's home node identification 2. query servicing by the home node

1. File look-up and file's home node identification:

   When a query is initiated at a node, the node hashes the file name and checks as to which affinity group the file falls into. File affinity group is based on same hash function as the node affinity groups
   - If file falls in the same affinity group as the node, node checks if
     - it has the file
     - if not, checks file tuple entries for the group, and then forwards the query to the relevant IP address+port number pair; the home node for the file.
     - if the relevant file tuple entry/ IP address+port number pair is not found/ invalid, the query is forward to another node(s) in the same affinity group, selected based on specific criteria or randomly.
   - -If file does not belong to the same affinity group as the node and it's affinity group, the query is forwarded to the relevant contact(s) the node has associated with the file's affinity group. (Note: The contact(s) could be all or some associated with the relevant group. Choice of contact/node could be on the basis of parameters like latency, etc. or randomly)
   - Once the query is forwarded, the receiving node undergoes the above mentioned steps.

- If the query can not reach the relevant affinity group/node, it is sent to the next best node(s) in affinity groups associated with the current node or with the file tuple or other affinity groups other than the above mentioned ones.
  - The node receiving the query iterates thorugh the above steps.
    (Make a decision tree for this/ use kelips topology and demonstrate the choice via generic example)

2. Query servicing by the home node:

   Once the home node is identified and the node in question has checked that it has the file, it establishes a connection with the original source (client) of the query and the file transfer takes place.

File insertion: File insertion/storage happens at the node to which the file is uploaded, and once done, that node then gossips the file tuple to relevant affinity group and its contacts. The entry is propagated and kept alive via the gossip stream viz rate limited and has rations.

(Note: acutal files are stored in the node to which file is uploaded. That node then broadcasts the file tuple entries (file name and it's IP address+port number pair). This data is then stored by nodes in the affinity group

2.3 Introduction

Ripple tries to solve the challenge of malicious nodes generating false/ faulty transactions by employing public key cryptography at a node level (enforcing trust and accountability) and account level (eliminating the vulnerability at the node level and moving it to the client level for appropriate and easier handling) and by letting nodes/ servers choose which nodes to trust on. However, there is scope for more work in this area.

The other challenge of network slow down and partitioning that is often faced is of interest in this paper.

resulting in negation of legitimate transactions and financial activity on the discarded fork(s) and also loss incurred due to the Double Spend attack

Safety net - While this ensures that the system is not defrauded and ensures security, it also means that no transactions go through and thus halting financial activity. Though relatively safer, it still is a loss making situation.

New transactions are added to the distributed ledger post agreement via the consensus

and validation processes which involve servers voting on the transaction's validity and accepting the same once passed or approved by a minimum threshold (quorum) of servers. From a server's view point, instead of the whole network a representative of the same, the Unique Node List (UNL), a list of trusted servers which the server in discussion believes won't collectively defraud it is chosen. Such a server running the RPCA believes that the network has reached consensus when the votes received from servers on its UNL crosses a certain minimum threshold required for consensus.
A server's UNL is manually chosen by its administrators and is expected to be chosen such that the servers in the UNL would be able to receive transactions from the various parts of the network and forward them to it, thus providing full coverage of the network.

However there is no guaranteed fool-proof way of ensuring this and there are no guidelines and mechanisms to ensure sufficient overlap of UNLs which ensures the correctness of the consensus and prevents forks. Thus there is scope for improvement in the way UNLs are chosen be it in a generic form of guidelines/suggested mechanisms which help achieve the desired security and correctness with sufficient overlap and network partition tolerance while improving the rate of achieving consensus. Utilising this, a way of automating the UNL updation can be come up with which helps improve ease of use. By setting a base network overlay structure for efficient message propagation at fast rates, configuring the UNL to leverage this and thus also increasing the overlap, one can further reduce the threshold of votes required to reach consensus, and increase the rate of arriving at a consensus thus significantly reducing the time it takes to achieve consensus. If properly designed, the overlay structure and message propagation can help achieve last mile connectivity. These approaches also increase the resilience to blockchain and network forks, reducing their risks. These benefits can potentially bring about significant improvements and utility benefits in Ripple, consensus algorithms, other approaches to solve the Byzantine Generals Problem (blockchain and other domains), thus making such systems more adoptable.

We leverage years of research done and more to come in the area of Distributed Hash Table(DHT) based P2P query and look-up systems. This ensures that as the research in these areas progresses further, the solution to Byzantine Generals Problem in blockchain, other domains and in general also progresses and improves. It also brings

back renewed focus and interest in DHT and P2P systems and helps improve newer areas as well. Getting inspired by the core mechanics of query, look-up, and routing mechanisms, and leveraging their base substrate/ structure, we come up with reverse routing mechanisms to help propagate information efficiently (speedily, surely and at lower costs). The network overlay of the DHT based systems is leveraged and set up for the propagation of information, the message passing being on the connected edges in the DHT and as per the reverse routing mechanism. The trusted nodes list (UNL) for consensus is designed to take full benefit of the underlying network overlay graph, and ensuring that we get sufficient overlap and receive all the information generated across the network easily, providing each node with full knowledge of the network in a manner of speaking and thereby tackling the Byzantine Generals Problem.
[Being inspired by and leveraging its base network overlay structure and coming up with a message and information propagation system by getting inspired by and leveraging its query routing system and creating a reverse routing mechanism, we are able to apply it to Ripple as a representative crypto-currency to help] improve transaction speed, reduce transaction time, reduce minimum thresholds for consensus, improve overlap for provable security, significantly reduce the chances of network and blockchain forks and provide last mile connectivity.

CHAPTER 3

OUR CONTRIBUTION

3.1 Fast Full Network Knowledge

We describe in detail our work, suggested algorithms, UNL structure, applying and leveraging the network topology and information propagation in Section 4, along with a mathematical analysis of the correctness and rate of convergence in Section 5. Section 6 deals with experimental work, set up, analysis and the interpretations, while Section 7 provides all the insights into benefits, general applicability followed by a brief conclusion to the paper.

3.1.1 Distributed payment sytems, their utility and value goals.

One of the goals of distributed payment systems and such systems in general is to improve utility by having faster and faster transactions with same/similar levels of certainty and security. At the same time, you wouldn't want to compromise on security and would prefer a system that is provably secure (especially in a tolerance threshold that you are comfortable with). Such a system should be resilient to and not prone to forks, which can lead to the double spending problem, the bane of such systems. The reason for avoiding the same is two fold - 1. if some one maliciously tries to spend the same money twice and thus try to defraud the system and 2. the fact that if the legitimate transactions on the other discarded fork of the blockchain are declared invalid, it will lead to a loss of value, as it will negate all the financial activity, trade and transactions happened post the fork (which is often detected later on). It may lead to trade happening only one sided, with the other party finally not receiving any value for the goods and services provided. Such challenges often undermine the trust and reliability of the system and bring into question the very veracity of such a system. Thus, undermining the value of the system and is one of the biggest concerns for their adoption. For such systems to be viable for use, it is thus imperative that they be

designed to not be vulnerable to such forks and be mindful of the various security considerations needed to operate them. Often there is a safety net designed in such systems and also Ripple, where it is said that in the eventuality of a fork, the system would not make forward progress and thus no malicious transaction will go through. While this ensures that the system is not defrauded and ensures security, it also means that no transactions go through and thus causing a halt in financial activity. This in itself is a loss making situation, though safer than the other alternative. Thus the work, engineering and research, going on in this area to ensure that such systems be fast, fault and partition tolerant, and secure is of importance to ensure the trust and adoption of such systems by a wider populace.

Also, in real life payment systems, it is essential to ensure last mile connectivity for wide spread adoption and the system to really succeed. Since you should be able to conduct transactions even in areas with relatively poor network connectivity. In our work below we present a method of implementing and automating updation of UNL (generic and randomised network topology) which ensures the same degree of certainty, trust and guarantee in transactions as with the original system (if not more) in a lowered threshold for consensus, hence reducing the time taken for consensus to be achieved. It also provides for the last mile connectivity requirements for such system. It also helps reduce chances of network partitions and forks, and helping make the system more secure.

The overlay network can be also be utilised to determine the network state, latency and connection detection.

It may be noted that since the network topology implemented is generic per se within the framework for UNL described, the UNL implementation too is generic overall.

3.1.2 Towards preventing forks, speeding up consensus, dynamically updating UNLs and providing last mile connectivty using p2p inspired network topologies

Network topology/overlay

We first set up the network topology, the same/similar as Kelips. The network is divided into $\sqrt{N}$ affinity groups of size $\sqrt{N}$ each. Membership to the affinity group can be determined on the basis of hash functions and various parameters as defined by constraints and conditions/the community. It could include parameters like hash of IP, relative netspeed and latencies(connections inside the affinity group being faster),etc.

Unique Node List(UNL): For the ease of understanding and application we divide the UNL into two parts - UNL-A and UNL-B. The UNL of a server would thus include

- **UNL-A:**$\sqrt{N} - 1$nodes (all the other nodes) from the server's own affinity group.
- **UNL-B:** c nodes from each of the other $\sqrt{N} - 1$ affinity groups(Ideally$c \geq 3$).

The specifics of UNL configuration and threshold related calculations could be based on various contraints which will be discussed later in the paper.

Network members list(NML): Is a list maintained by each server. It is a list of all live/ reasonably live servers the server has seen/been made aware of so far. It is a super set of the UNL with a list of

- known introducers for each affinity group, known as NML-C
- other servers in the servers' affinity group, known as NML-A
  $NML - A \equiv UNL - A$
- servers in affinity groups other than the server's, known as NML-B. Generally always,$(NML - B \supseteq UNL - B)$(Except near the boundary values of refresh time periods reached in both UNL and NML)

The servers in the list are classified/segregated by affinity group.

Server and introducer behaviour

Every server(other than introducers)

- pings its UNL
  - to servers in its UNL, periodically(t2).
  - to servers in its NML, over a longer time period(t3) than UNL pinging(not needed) (t3>>t2)
  - to requesting servers, on pull request.
- pings its NML
  - to requesting servers, on NML pull request. (NML pull request is rarely used, eg if all introducers are down)

Each round of consensus(many rounds result in consensus of a single ledger) takes time t1.t1<<t2<<t3

Introducers send NML on pull request. Every introducer is also a server and behaves like a server.

Non-faulty nodes under normal execution, participate in consensus and have liveness protocols running.

Various states of a non-faulty node under normal execution

A non-faulty node exists in three states(Is it needed? Has been put for the three headings)

- Node join: When it joins the network
- Node leave: When it decides to shut down/leave the network under normal circumstances
- Node live/normal execution: When the node functions normally and participates in the consensus process.

Node join: When a new node joins the network,

- the node details are first hashed and affinity group is identified at the node. The node if and when connects to other nodes, sends a header containing its details and affinity group. The affinity group and details can be cross verified at every node to which the node connects. It also starts building its UNL and NML.

- The node connects to known introducers (from its and other affinity groups) in the system (and sends an NML pull request with a node join tag: is this needed or is it understood).
- The introducers send their NML data[1]
- The node adds these to its NML and also connects to a total of the following number of new nodes of following specification it finds out about(make table for this)
  - in its own affinity group => all
  - in other affinity groups => $c \times b$ total for each other affinity group. Where $b > 1$ (Either mention this or all or once own affinity group is formed, from them and from introducers. The own affinity group awareness protocol will always be running)
- The nodes other than introducers on being connected to, provide only their UNL data to the requesting node.
- Condition for node to exit node join mode is as follows

```
if(server has contacted all the nodes per affinity group){
if(server has contacted all members of its own affinity group){
exit node join mode
}
}
```

  - Remember that the lists are updated every time a new node is found to be added to the appropriate group and condition rechecked.
- Once NML building is finished, as node join mode ends, UNL is made. It should be noted that NML-A and UNL-A are the same, and contain the same nodes. Hence UNL-A changes as NML-A changes across all modes.UNL-B is made by choosing c nodes for each affinity group at random from NML-B

Node leave: Send out a public key cryptographically signed leave message to each member of its UNL.

This is forwarded by the members in similar way as transactions.

Node liveness: Node is live is known by:

[1] Alternatively, introducers send data containing lists of
- introducers they are aware of
- nodes they are aware of in their affinity group
- nodes they are aware of in other affinity groups (these may be more than or less than the c nodes used in UNL)

Note: This is an optimzation and need not be done - since data received is a one time set-up cost. For high churn, however, the needful optimizations can be done and may make a difference.

*For knowing of liveness of other node:

- Receipt of info from/about any node
  - if it is not part of NML, update UNL and NML.

*For maintaining its own liveness with the network:

- Regular proposal push at t1 interval
- UNL push at t2 interval
- NML push at t3 interval

*On receipt of node leave message, delete the node immediately. Forward the node leave message.

Nodes and servers also employ the following for maintaining the liveness of nodes in their NML and UNL:

- Failure detectors
- Tombstone node, if not heard from, after 2*time period.
- Delete node after 2*tombstone time period.
- Note: Time period is as per UNL and NML time period.
- If tombstoned nodes reupdate themselves and ping their details, before deletion/ their details with appropriate timestamp(timestamp after the tombstone) are got before deletion, they can be made live again.
- Tombstoned nodes are not an active part of the UNL(decide whether they will be a part of the consensus process and thresholds{mostly not}). They are not sent out as part of UNL/NML pushes.

Also UNLs and NMLs have a threshold and percentage of the amount of change they can tolerate and can happen over a certain time period.

Maintaining the UNL and NML:

The NML and UNL at a server are maintained by updating based on data received from UNL and NML push from other nodes and UNL/NML pull by the server. Nodes which are not there in the NML & UNL are added to respective categories of UNL and NML (This is for all cases excluding UNL-B).

UNL-B is created by randomly choosing c nodes at random from NML-B. If a node in UNL-B is tombstoned/deleted, a new node is selected by choosing it at random from NML-B (excluding nodes already in UNL-B). Thus UNL-B is maintained at c nodes per affinity group if NML-B is greater than or equal to c nodes for each affinity group. For smaller sizes of NML-B for specific affinity group, UNL-B mirrors NML-B for that specific affinity group. For sizes less than c for specific affinity group for UNL-B and/or less than c*b for specific affinity group for NML-B, the system actively tries to update the respective list and add nodes by querying other nodes it knows in the respective affinity group of the list and also other nodes from other affinity groups.

If size of UNL-B for each affinity group is less than 1, the system goes into a sceptical state(???) and actively tries to get a node for that particular affinity group.

Note: The concept of NML-B is different from randomly choosing nodes of other affinity groups from the list of known nodes of other affinity groups used in Kelips(The one used in Kelips is a one time thing, after that the updates in Kelips happens via push notifications. However 1. That is vulnerable to eclipse attacks and attacks detailed in BRAHMS paper 2. Not truly random and only done once and later network can converge to some nodes which keep pinging more, thus losing out on randomisation. The other approach, our resolves the above issues and implements the algorithm detailed in BRAHMS paper, thus improving security).

Modified consensus algorithm and dynamically updated UNL:

Here we propose slight modifications to the consensus algorithm in terms of thresholds, UNLs and some small points. For all other practical purposes, the algorithm is the same as the original Ripple Protocol Consensus Algorithm as mentioned in the ripple white paper and as quoted from the paper above in section.

For each round i at each server

Stage 1: Candidate set generation

- Transactions in the candidate set which could not pass the previous consensus round but are still live and valid are re-included in the candidate set.
- The server flushes it's transaction input queue which contains various transactions from clients and candidate set transactions of other servers, received previously. These can be received during the previous consensus round/

currently received(after the previous transaction flush)/pending from the previous queue flush

- Transactions the server feels are valid and potential candidates for consensus are added to the server's candidate set.
- The server then declares it's candidate set to other servers in it UNL.
- Note: The mandatory wait time for all servers to catch up and generate/share their candidate sets reduces due to the network topology. If some servers still are unable to catch up
  - their suggestions for potential transactions to candidate sets can be taken up the next round (If this is a large majority, the time can be tweaked, also the current consensus round wouldn't pass)
  - their suggestions and suggestions they receive get relayed indirectly through the multi-hop infrastructure.

This ensures that every server has visibility of most/all transactions

Stage 2: Consensus sub-rounds

- After the mandatory wait time, the servers send out their proposals for each sub-round of consensus
- The absolute threshold for mathematical certainty decreases after each sub-round as the number of hops increases and network overlap increases and servers get increasingly more visibility of the network.
- The sub round threshold increases after each sub-round, as is there in the ripple protocol. However, the threshold need not be more than the absolute threshold for mathematical certainity.

Note: The time for each sub-round is equal to the average time/median time it takes for 1 hop to complete in the network. It can change/adapt as per the network latencies.

Note: The servers don't need to send proposals for each sub-round; similar to RPCA. Refer video.

-== add this somewhere. == The proposals for sub-round can be sent for each round saying this sub round or send only once.

3.1.3 Analysis - Information propagation, overlap and threshold.

In this section, we describe how information is propagated, calculate the network overlap and determine the per sub-round thresholds for provable security and achieving mathematical ceratinty for the absence of forks.

Importance of Consensus, information propagation

Before we begin, we would like to briefly highlight the importance of the consensus algorithm, especially when block chain and public key cryptography resolve the byzantine generals problem to a very good extent. One can easily see that implementing public key cryptography prevents man in the middle attacks by ensuring that only the legitimate sender is able to send the message and the message is untampered with, thus eliminating one of the major concerns of the byzantine generals problem. However, a node can still behave maliciously by not forwarding the message and preventing the other nodes from making an appropriate choice. Also, while the nodes can not create faulty transactions, a malicious client side application/ user can try to double spend by issuing two transactions with the same transaction number to two different nodes. In the absence of an appropriate consensus process and the eventuality of a network partition/delay in information propagation, it may be possible for the two partitions to agree upon different sets of transactions. This makes the system open double spends, haemorrhaging the money and value of the system. While, it is possible to identify the malicious account(not necessarily user), the damage is already done. The money may not always be recoverable. Also, a fork in the block chain causes reduced trust in the system, causing decreased adoption, members leaving and thus causing a loss in the systems value. In light of the above mentioned points, it is important to have a consensus algorithm that prevents such vulnerabilities and also have an information propogation mechanism that ensures that not only are malicious nodes unable to prevent the network from making forward progress but also that they are not able to cause forks in the system.

Information propagation and UNL overlap

In this section we discuss how information is propagated per-hop in a p2p(Kelips)-inspired topology (in the context of Ripple) and how this affects UNL overlap between two nodes. It should be noted that there is a subtle difference between candidate set propagation and proposal propagation in the context of UNL overlap. For UNL overlap, it is proposal propagation which should be considered. This is owing to the way Ripple's consensus protocol is currently configured. We explain this via an example.

We now describe how the information propogation takes place and analyse the UNL overlap. Apart from some differences, the overall propagation(via hops) is similar to Kelips' file querying and file insertion mechanisms, as the substrate/network topology used is Kelips' . The value lies in identifying the usage of p2p-inspired network toplogies, in the implementation in the context, dynamically updating UNLs and getting reduced thresholds for same security, preventing forks, and also other applications and benefits.

In the context of Ripple and applying p2p(Kelips)-inspired topology, each round has two halves equalling a total of 2 hops. The first half is the propagation of the candidate transaction while the second, propagation of proposals. We can for the purpose of utility configure the system such that it treats consensus proposals also as possible candidate transactions. Thus a proposal when received is checked in not just into the proposal queue but also the candidate set queue(could do check for candidate set on the proposal queue instead as a memory optimization). This may possibly help reduce messages sent(message cost and thus dependency on latency) and thus reduce the time it takes to reach consensus for a transaction's life time and also for a consensus ledger. Thus, all proposals are potential candidate transactions(the ones before entering candidate set) but not all candidate set transactions are proposals. Thus, here we discuss for each hop, considering proposal propogation.

Ai receives transactions from client and verifies validity and propagates it. Overlap between Ai and Aj belonging to same affinity group

Synchronous system in absence of failures

Hop 1.

No/minimal overlap (Proposal stand point)

Hop 2.

$UNLOverlap \geq 50\%$(own affinity group).

Hop 3

100% overlap

Synchronous system in the presence of f failures. Link Latency ratio (UNL-A/UNL-B) = 1

f1 failures internal to affinity group A , f2 failures for nodes connected to Ai, f3 failures for nodes connected to Aj

Hop 1

No/minimal overlap (Proposal stand point)

Hop 2

$UNLOverlap \geq 1 - \frac{f_1}{\sqrt{N}-1} \times' a'$

'a' is the weightage of UNL-A in the consensus process.

'b' is the weightage of UNL-B in the consensus process.

Hop 3

Hop 4

$\left(1 - \frac{f_1}{\sqrt{N}-1} \times' a'\right) + \left(1 - \frac{f-f_1}{c\times\left(\sqrt{N}-1\right)} \times' b'\right) \leq UNLOverlap \leq \left(1 - \frac{f_1}{\sqrt{N}-1} \times' a'\right) +' b'$

Consensus sub-round thresholds

As highlighted in section 3.2 of the paper [Citation Ghassan O'Karame paper] for a % threshold, % overlap in UNL is needed. By utilising the formula derived, it suffices to say that if there is an increase in possible overlap between each UNL, the min threshold needed for arriving at consensus can be further reduced, thus decreasing the time taken to reach consensus. It can be noted that for a 90% overlap in UNL, we can work with as little as 55% threshold. (Note the minimum 66% threshold is valid in the classical byzantine generals problem. In the case of implementing public key cryptography to the problem, we can go much lower than the 66% threshold)

Round 1: x%

Round 2

The other benefits of this is that even if a server is able to connect to 1 other server/ one non-faulty server, it's message gets relayed and the whole system will reach consensus in normal number of rounds+1. The server can then receive the info that consensus passed after a certain number of rounds from that link and thus the server

while not receiving proposals acts as a relay of transactions and also receives latest updates on the last/latest validated ledger. This way, we can ensure last mile connectivity. The same previously was not guaranteed to happen. It also protects the system/ server when all but 1 other servers in its UNL are non faulty/non-byzantine. Only one good node connected takes care of the outreach and preventing eclipse attacks. This apart from the NML-B/UNL-B choosing structure/process which incorporates BRAHMS. Thus improving security. Since the server's success and other statistics are maintained by the community, extremely malicious nodes can be weeded out. Due to public key crypto, fake transactions are very very difficult to create. So a node can only be malicious by witholding info(solved due to last mile connectivity) {and by declaring wrong ledger(reputational cost)}.

Message complexity(normal and compared to ripple standard)

Time complexity(normal and compared to ripple standard)

Memory storage complexity

Benefits

- Reduced threshold, same security but less time
- Improved security, UNL more systematized
- Last mile connectivity
- Network diagnostics
- Connection detection
- Network state
- Network latency
- Load balancing
- 3 degrees of separation
- Resilience to Byzantine attacks, Sybil and Eclipse attacks; BRAHMS
- Consensus is energy efficient compared to Proof of Work and also resilient to Sybil

3.2 Simulation

3.2.1 Experimental setup

For the purpose of experimental analysis and benchmarking, we utilise the simulation code 'Sim.cpp' mentioned in the Ripple White Paper. We compare 3 different versions of the code with number of nodes and malicious nodes scaled appropriately for comparison purpose. A brief description of the different versions is mentioned below

1. SimC - The original version of Sim.cpp by Ripple. Number of links, link latencies, connections remain same as the original. We call this version Sim-Classic.
2. SimRM - A modified version of Sim.cpp with the number of links and link latencies similar to SimK, described below. The UNL and network topology is randomised as is in SimC. We call this version Sim-Ripple Mid (as it has number of links scaled to the number in SimK but employing the randomised topology as is in the SimC)
3. SimK - A modified version of Sim.cpp where the network topology is inspired by the p2p system, Kelips. The connections, UNL, number of links are appropriately set up in a manner similar to the approach described in the paper above. The ratio of link latencies (links within affinity group to links outside) is varied, as described in the results below. We call this version Sim-Kelips.

The system initially starts in perfect disagreement (similar to the original simulation) with half of the nodes proposing "yes" and the other half proposing no. The initial state of the nodes too is randomised and changes for each test case. The nodes change their stance to the stance taken by a majority of nodes in their UNL(greater than 50%). Note: This threshold is the same as is in the original ripple simulation Sim.cpp. It is technically sound owing to the fact that in the case of implementing public key cryptography, the minimum permissible threshold is 50%. However, the network and simulation reach consensus when 80% of the network agrees to a certain result.

Both Sim.cpp and the versions mentioned above are generic compared to the actual real life implementation of Ripple. This is so that it be applicable in a number of different settings and not just the specific case. However as everything else is the same, it is possible to compare the differences occuring due to introduction of a network topology inspired by Kelips.

Table 3.1: Results for SimC - The classic version used by Ripple for simulation

	SimC		
	Consensus Time (ms)	Number of Messages	Total Links
Average	585.60	19019.82	5120
Median	577	19028	5120
Mode	552	19073	5120

Table 3.2: Results for UNL-A Link Latency Factor = 1

	SimK		SimRM	
	Consensus Time (ms)	Number of Messages	Consensus Time (ms)	Number of Messa
	SimK(1,1)		SimRM(1,1)	
Average	198.20	69535.13	311.69	85715.72
Median	180	70570.5	289	86671.5
Mode	122	72896	284	89418
	SimK(1,2)		SimRM(1,2)	
Average	237.17	64618.22	416.85	84880.89
Median	212.5	64281	398	86148.5
Mode	166	72375	352	87556
	SimK(1,3)		SimRM(1,3)	
Average	242.48	60507.18	440.64	81840.16
Median	213	58422.5	420	82280
Mode	163	52464	384	82837

It might be noted that in SimK the threshold for reaching consensus is set at 80% similar to the other two cases, and does not decrease as the rounds proceed. In the eventuality where the thresholds for changing stance (in simulation 50%; in actual implementation, accepted if a transaction is a possible candidate) and also for reaching consensus decrease as the number of rounds proceed in SimK, there is expected to be a speed up in the consensus process compared to SimC and SimRM where the thresholds remain constant with time. At the same time as demonstrated above and in [Citation Ghassan O'Karame paper], the security and reliability guarantees are maintained similar to the original Ripple Protocol.

Table 3.3: Results for UNL-A Link Latency Factor = 2

	SimK		SimRM	
	Consensus Time (ms)	Number of Messages	Consensus Time (ms)	Number of Messa
	SimK(2,1)		SimRM(2,1)	
Average	258.83	71812.23	350.98	84779.98
Median	231.5	72074.5	327	85946
Mode	164	81605	307	81133
	SimK(2,2)		SimRM(2,2)	
Average	396.21	69603.78	583.12	85055.69
Median	368.5	71083.5	552	85745.5
Mode	406	72004	501	89719
	SimK(2,3)		SimRM(2,3)	
Average	460.23	66929.68	702.52	84447.96
Median	412.5	67962.5	665.5	85656
Mode	285	75661	645	86181

Table 3.4: Results for UNL-A Link Latency Factor = 3

	SimK		SimRM	
	Consensus Time (ms)	Number of Messages	Consensus Time (ms)	Number of Messa
	SimK(3,1)		SimRM(3,1)	
Average	273.76	70496.5	355.24	82563.23
Median	245	69783	332	83794
Mode	222	77565	315	84769
	SimK(3,2)		SimRM(3,2)	
Average	477.29	71502.03	639.88	84685.34
Median	433.5	72530.5	597	85861
Mode	312	74275	501	87854
	SimK(3,3)		SimRM(3,3)	
Average	603.99	69484.87	819.78	83811.76
Median	534.5	70103	784	84565
Mode	360	71069	806	89394

Table 3.5: Total number of links in the network (Valid for all Link Latency Ratios)

	Total Links	
	SimK	SimRM
Average	20969.3	23040
Median	20970	23040
Mode	20960	23040

REFERENCES

1. **Lamport, L.**, *LaTeX: A document preparation system*. Addision-Wesley, 1986.

2. **Ramachandran, P.**, MayaVi: A free tool for CFD data visualization. *In 4th Annual CFD Symposium*. Aeronautical Society of India, 2001. Software available at: http://mayavi.sf.net.

3. **Ramachandran, P.** (2004). *LaTeX class for dissertations submitted to IIT-M*. Ph.D. thesis, Department of Aerospace Engineering, IIT-Madras, Chennai – 600036.

4. **Ramachandran, P.**, **S. C. Rajan**, and **M. Ramakrishna** (2003). A fast, two-dimensional panel method. *SIAM Journal on Scientific Computing*, **24**(6), 1864–1878.

5. **van Rossum, G.** *et al.* (1991–). The Python programming language. URL `http://www.python.org/`.

LIST OF PAPERS PLANNED

1. Description: Conference Paper:
   Tentative Title: "Fast Full Network Knowledge".
   Keywords: DHT, Kelips, Ripple, UNL, Threshold, Network Overlay

2. Description: Journal Paper:
   Untitled
   Keywords: DHT, Kelips, Pastry, Chord, Ripple, UNL, Threshold, Network Overlay