PhD-FSTM-2024-062
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 04/10/2024 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

FLAVIENE SCHEIDT DE CRISTO
Born on 22 November 1991 in Curitiba, Brazil

IMPROVING SCALABILITY AND OPTIMIZING MESSAGE DISSEMINATION ON THE XRP LEDGER

Dissertation defence committee

Dr Radu State, dissertation supervisor
Professor, Université du Luxembourg, Luxembourg

Dr Jacques Klein, Chairman
Professor, Université du Luxembourg

Dr. Eduardo Cunha de Almeida, Vice Chairman
Associate Professor, Universidade Federal do Paraná

Dr. Mats Brorsson
Research Scientist, Université du Luxembourg

Dr. Omar Cherkaoui
Associate Professor, Université du Québec à Montréal

”Knowing was a barrier which prevented learning.”

Frank Herbert in *Children of Dune*

Abstract

Blockchains have become crucial in numerous fields, notably within financial applications. The advent of Bitcoin coincided with the emergence of a global economy highly dependent on digital transactions, stimulating the development of numerous blockchains that aim to address different challenges. The XRP Ledger (XRPL) is an example of a blockchain designed to address cross-currency payments. Unlike its counterparts, the XRPL does not employ resource-intensive consensus mechanisms such as Proof of Work (PoW), employing instead a Federated Byzantine Agreement consensus protocol, in which subnets of validators vote to determine the next version of the ledger.

The XRPL network allows any node to act as a validator, but its consensus mechanism relies on a trusted group of validators called the Unique Nodes List(UNL). In its current implementation, the XRPL relies on a single UNL, curated by the XRP Ledger Foundation (XRPLF). The dependence on this centrally managed UNL raises issues regarding authority distribution and network scalability. Although the XRPL is decentralized in terms of communication, it does not feature dynamic trust overlay rearrangement, resulting in limited authority dispersion and possible performance problems as the network grows.

This thesis investigates ways to enhance the dispersion of authority and node-wise scalability of the XRPL. First, we explore a tool that has the potential of diversifying UNLs to achieve higher trust autonomy, we then propose the use of pubsub dissemination to reduce message overhead and improve scalability. The research also addresses key questions on the impact of pubsub mechanisms on the performance of the XRPL, aiming to provide insight into tuning parameters for optimal network efficiency. The ultimate goal is to determine how pubsub dissemination can enhance the scalability and performance of blockchains using FBA consensus, specifically within the context of the XRPL.

Acknowledgements

Throughout this journey, I was fortunate to meet and collaborate with many people who not only provided assistance and support, but also honored me by sharing their knowledge. I start by thanking Radu State, my supervisor, for the opportunities and guidance. Jorge Augusto Meira, for building the bridge that allowed me to be here and for all the work we did together. Also, thanks to my other coauthors, Jean-Philippe Eisenbarth, Lucian Trestioreanu, Wazen Shbair, and Arno Geimer for all we shared.

I am also immensely grateful for the UBRI sponsorship, without which I would not be able to gather so much knowledge around the world. Special thanks to Lauren Weymouth, Director of this amazing program, who always received us and our ideas with enthusiasm. Still from Ripple, I would like to thank Marco Neri for the fruitful discussions about the ledger decentralization and Aanchal Malhotra for all the guidance.

I also would like to thank XRPL Commons, especially David Bchiri, Mathilde Morineaux, Cassie Hirsh, Zsofi Borsi, and Vera Radeva for all the efforts to build a global XRPL community and the doors they opened for me.

To the members of the CET, Eduardo Almeida and Mats Brorsson for their time and guidance, and the other members of the jury, Jacques Klein and Omar Cherkaoui, for their valuable time and effort.

I am also grateful to my colleagues at the SEDAN group, former and current, starting with Farouk and Lujun for the crazy brainstorming conversations. But also to Christof, Antonio, Lama, Nino, Patricia, Oceane, Andres, Robert, Ramiro, Mary, Sean, Ilias, Grabriel, Jeovane, Tatiana, Bahareh, Beltran, Francisco, and Kyo. Also, a very special thanks to Valerie, for being there to help all the times (and there were many) I had bad luck during my trips and Jessica Giro for the prompt responses whenever I needed some help.

Special thanks to my friends Ricardo, Alexandre, Eveline, Raquel, Tom, and Doro. Also,

a thank you to my dance squad for making the studio my safe place, especially Sarah, Coralie, Steh, Carolina, the other Carolina, and Moni.

And finally, to Carolina and Monica for the professional psychological help, Francesco for the patience and for not letting me quit, and my family for the emotional and financial support. I cannot express how important you all were during this voyage.

Index

List of Figures

List of Tables

Acronymns and Abbreviations

AC Agglomerative Clustering. 65

CRPS Continuous Ranked Probability Score. 88, 89

DAG Direct Acyclic Graph. 9, 32, 33, 35, 78, 83, 85, 90, 99

DHT Distributed Hash Table. 15, 52

FBA Federated Byzantine Agreement. 2, 7, 10–12, 15, 98, 100

FTP File Transfer Protocol. 76

GCM Graphical Causal Model. 78, 79, 85, 88, 90, 93, 101

GML Graph Modeling Language. 78

IPFS The Interplanetary File System. 6, 23, 53, 61

KM K-Means. 65

LMC Local Markov Condition. 85, 86, 88

NDN Named Data Networks. 46, 47

NLAC Nested, Layered, Autonomous, Continuous. 98, 99

nUNL Negative UNL. 3, 38

p2p peer-to-peer. 1, 3, 6, 10, 14, 23, 34, 52, 53, 61, 73

PoS Proof of Stake. 2, 4, 5, 11, 12

PoW Proof of Work. 1–5, 11, 12, 31

pubsub publisher/subscriber. 6, 7, 9, 10, 23, 24, 29, 46, 52–55, 59, 62, 98–100, 102

RPC Remote Procedure Call. 46

SCM Structural Causal Model. 78, 79, 82, 85, 87, 90, 92, 93, 101

TCP Transmission Control Protocol. 76

TCR Token-Curated Registry. 9, 32–35, 38, 44, 99

TPS transactions per second. 5, 12, 45

UNL Unique Nodes List. 3–7, 9, 11–13, 15, 16, 20–23, 29, 31–38, 40–44, 48–50, 54–56, 59, 62, 64, 75, 93, 97–99, 102

XRP LCP XRP Ledger Consensus Protocol. 2, 4–7, 9, 11, 12, 29, 32, 43, 44, 46, 48, 51, 53–55, 59, 62–64, 66, 73, 75, 97–99

XRPLF XRP Ledger Foundation. 3, 9, 12, 13, 21, 22, 31, 33, 36, 45, 98

Introduction

Blockchains have become prominent in the development of solutions in multiple domains, predominantly in financial applications. Bitcoin[1] was introduced amid the rise of a more global and connected economy, in which digital payments became more and more common. Following the initial success of digital currencies, various blockchains started to appear, focusing on different sets of problems. The XRP Ledger (XRPL)[2] was a pioneer in this landscape, initially proposed to solve the issue of cross-currency payments in this fast growing global environment.

Blockchains are decentralized systems in which groups of machines collaborate to validate and record transactions in a permanent ledger structure. In this context, trust is given to a collective of computers, instead of being in the hands of institutions or human actors[3]. The process through which nodes in the networks collaborate to validate transactions is called *consensus*. The concept of consensus itself is not new to the domain of distributed systems, especially in the area of peer-to-peer (p2p) networks. Bitcoin introduced the idea of using Proof of Work (PoW)[4] to achieve consensus, ensuring the security and validity of the transactions recorded in the ledger, even in an environment where any device can join the network, thus establishing the concept of collective trust.

A ledger is a record of financial transactions performed by a set of accounts. A ledger can be as simple as a book summarizing debits and credits, ending with the monetary balance of each account involved in the transactions recorded. Blockchains store their financial transaction records in a permanent distributed ledger format. The data structure is said to be permanent because no modification can be done in previous entries, similar to how an append-only database works. The distributed nature comes from the fact that the record is replicated and shared among the network participants. Bitcoin updates its ledger by generating blocks of transactions that are validated using PoW and added to a temporal chain;

thus, the name *blockchain*. Each new block carries a hash of the previous one, as shown in Figure 1, ensuring that no prior block can be modified without spending the resources to recalculate and revalidate all subsequent blocks.

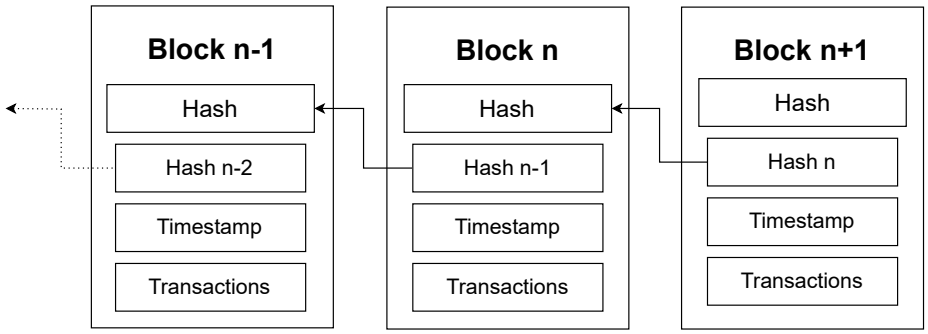

Figure 1: Simplified schematics of a Blockchain. Each block contains a hash of itself, a hash from the previous block, a timestamp, and transactions

To validate new blocks of transactions, Bitcoin applies a consensus algorithm called PoW, which is based on the idea of selecting a validator in each round who will decide the next block to be added to the chain. This idea was carried over to other types of consensus protocols, such as Proof of Stake (PoS)[5]. However, these strategies are expensive, either in terms of computational or financial resources. In the case of PoS, the validator is chosen from a list of stakers, while in PoW, the validator is chosen based on the amount of computing power they possess.

The XRPL, in turn, uses an alternative consensus protocol based on Federated Byzantine Agreement (FBA)[6]. Still carrying the idea of collective trust introduced by Bitcoin, the XRP Ledger Consensus Protocol (XRP LCP) uses the federation between subnets to reach consensus and validate the legitimacy of transactions. Rather than issuing new blocks during each consensus round, the XRP LCP generates new versions of the ledger. Although at first glance voting seems to be a more convoluted and slower process than selecting one validator per pound, in reality, the XRPL shows a better throughput than its counterparts.

Being a public blockchain, in the XRPL any node can join the network and become a validator, capable of casting votes and proposing new versions of the ledger. However, not all validators have their positions considered during the consensus process. Each node ac-

knowledges solely the positions of a set of entrusted validators to cast its own votes, trusting that this set will not collude to manipulate or break the ledger[7]; this set is called a Unique Nodes List (UNL). Although every validator must denote their trust set by selecting a UNL, certain overlap conditions between sets are necessary to avoid excessive disagreements during the consensus process[8]. A higher overlap guarantees faster and safer convergence to consensus, whereas a more relaxed overlap may cause delays and forks in the ledger. To avoid such cases the XRP Ledger Foundation (XRPLF) advises all validators to use the same UNL. This UNL is curated and maintained by the XRPLF and, while at first glance it may seem to bring improvements in performance and safety, it also raises concerns about how the authority is distributed in the XRPL network and how sustainable its growth is in terms of new nodes joining the network.

On the subject of the distribution of authority, we can start by looking at the works of Vergne et al.[9], which present a discussion on the concept of *dispersion of authority* by breaking it into two definitions: *decentralization* and *distribution*. While *decentralization* is defined as the dispersion of coordinated communications, *distribution* is characterized by the diffusion of the decision-making process among stakeholders. Applying those definitions to the proposed scenario, decentralization tells us how the network is structured and how the communication between nodes occurs, while distribution is about who has the power to make decisions.

The XRPL network is built upon an unstructured p2p system, duly considered *decentralized* since there is no central authority coordinating the way information is exchanged between participants. However, it does not have robust means to dynamically rearrange its trust overlay. The trust overlay comprises all trust relationships present in the XRPL, defined by the UNLs declared by all validators. The solution adopted in the event of a partial outage is the Negative UNL (nUNL)[10], which places trusted validators that are believed to be malfunctioning in a list of nodes that will be ignored during the consensus process. The nUNL is designed to handle limited disruptions, and the addition and removal of nodes from the nUNL still require manual intervention. This notion opposes what is commonly agreed on in terms of decentralization, derived from the work of Baran[11], which points out the ability of a

network to delegate routing decisions under adverse conditions. The trust overlay lacks the ability of self-arrangement, leaving the decision to reroute the trust in the hands of human actors.

There are two instances in which the XRPL network presents low degrees of distribution; first, in the manner that the trust overlay is currently structured in the XRPL network, having a single UNL consisting of 35 nodes[1]. This structure has been proposed by Chase and McBrough[8] to avoid stalls caused by excessive disagreement between the subset of validators. Those 35 nodes do not delegate decision-making and listen only to themselves. This closed system allows non-UNL nodes solely to transmit transactions and to relay proposals and validations generated by trusted validators. The second instance is in the curation of this main UNL, relying on a central authority to decide who are the trustable nodes and how many of them are necessary to keep the network alive and safe while maintaining good performance.

We can then argue that the trust overlay of the XRPL has a low dispersion and could benefit from higher decentralization. In addition to the overlapping issue, these characteristics also arise from the lack of appropriate tools and techniques that allow the automatic and safe generation and maintenance of trusted validator lists. Such techniques would support the implementation of a model closer to what is conceptually described in the XRP LCP specifications[7], providing a higher authority dispersion in the consensus process.

In addition, the nodes in the XRPL do not have incentives to be part of the main UNL and behave correctly. Unlike other consensus mechanisms, such as PoW and PoS[5], the XRPL does not issue rewards for validating ledgers. The only incentive a node has to become a validator is to help keep the network secure and the ledger progressing without forks. However, the requirements to become a trusted validator are expensive in terms of computational resources[12][13][2]. Therefore, even trusted nodes have no incentive to continue to behave in the best interest of the ledger.

The distribution of authority is not the only concern raised about the current implementa-

[1] As of July 2024, according to `https://livenet.xrpl.org/network/validators`

[2] The required resources to run an XRPL validator are still less than the ones of a Bitcoin miner, but the lack of benefits in running an XRPL validator turns the operation costly

tion of the XRPL network. As the adoption of the XRPL as a platform for financial solutions grows, the number of participants in the network also tends to scale up. Scalability is a crucial topic in blockchains, as increased adoption demands more resources to validate new blocks or new ledger versions. The concept of scalability can be defined in two different domains. First, we can think about the throughput; a blockchain needs to be able to scale with regard to the number of transactions it can validate per second. We define a blockchain as well-provisioned in terms of throughput when it is capable of handling an increase in the number of arriving transactions[14]. For example, Ethereum is well-provisioned for a throughput of 45 transactions per second (TPS). The second domain is the scalability with respect to the number of nodes, in which a blockchain is said to be well provisioned if the increase in the number of nodes in the network does not affect performance and safety. Blockchains that employ consensus based on PoW and PoS usually deal well with node-wise scalability, while they struggle to scale their throughput, resulting in transactions being queued and delayed.

It is desirable for a blockchain to be scalable both node-wise and throughput-wise, so it can guarantee healthy growth. The XRPL is not scalable node-wise, not because of its consensus algorithm, but because of the way certain messages are disseminated. Validators in the XRPL disseminate proposals and validations by flooding the network; i.e., a node will send their proposals and validations to all the nodes it is connected; the same way, a node will relay those messages to all of their peers. This mechanism guarantees that all nodes on the network will receive proposals and validations in a fast way, which is necessary for the consensus process, since delays and interruptions can cause excessive disagreement and stall the progress of the ledger[15]. However, this technique can cause a paradoxical effect as the number of nodes starts to grow.

In the XRP LCP, a node will only acknowledge proposals and validations from peers present in its UNL. However, due to the use of flood publishing, each node will receive messages from all nodes on the network. Even a validator that is not trusted by any other node will still have its messages broadcast, creating several ineffective replicas and increasing the message overhead. A higher message overhead means that more resources will be needed

to propagate and validate messages.

The logical takeaway is that an increase in the number of nodes will increase the bandwidth required to transmit messages, leading to higher latency, reducing performance, and increasing the risk of stalling or forking the ledger. Now, considering how validators only acknowledge the positions of peers they have in their UNLs, it is trivial to think that a solution lies in transmitting messages in a more targeted fashion. However, in the present implementation, the nodes do not know in which UNLs they are present (if they are).

To disseminate messages in a more targeted way, we can abstract the UNLs as pre-set topics. Nodes that are part of a UNL publish messages on the respective topic, while nodes that trust said UNL can subscribe to the topic to receive the messages. This model is called a publisher/subscriber (pubsub) system and serves to decrease the number of replicated messages transiting through the network, while guaranteeing that all nodes subscribed to a topic will receive the published messages.

Looking from this pubsub perspective, we can then say that validators *publish* their proposals and validations and the other validators *subscribe* to receive messages from the peers they trust. The UNLs are the mechanisms through which the nodes subscribe to receive proposals and validations from other nodes. With that in mind, we can explore pubsub mechanisms for disseminating messages looking at reducing message overhead to increase the node-wise scalability of the XRPL. In addition, we can research ways to tune those mechanisms for the particular characteristics of the XRP LCP by identifying how the configuration of the pubsub system can impact the general performance of the network.

However, the pubsub characteristics of XRP LCP are diminished by the current deployment of the main XRPL network (mainet). Returning to the problem of the distribution of authority, the presence of a single UNL also diminishes the pubsub characteristics of the XRP LCP, considering that in this case only one topic is present and every node in the network is subscribed to this topic. In this case, pubsub dissemination can still be employed in the network to reduce message overhead by abstracting message types as topics. Similar to how other p2p systems, such as Ethereum[16], FileCoin[17] and The Interplanetary File System (IPFS)[18] employ pubsub dissemination. However, the use of multiple UNLs is still

a more interesting scenario for exploring targeted message delivery.

In conclusion, the current implementation of the XRPL presents a low degree of authority dispersion and low node-wise scalability. Both are intertwined and can affect the performance, liveness, and safety of the ledger. The use of a single centralized UNL not only leads to a clustering of authority in the hands of some players, but also aggravates the message overhead issue and weakens the possibility of using strategies for targeted message dissemination. Currently, new ledger versions are validated by a cluster of 35 nodes that form a fully connected graph regarding the trust overlay. This structure reduces the pubsub characteristics of the XRPL, which could be better explored to increase the scalability.

The research questions that drive this thesis are derived from the issues presented. First, we ask (1) *if it is possible to have a higher dispersion of authority by building and maintaining more diversified UNLs, creating a space of trusted validators that is autonomous and automatic*. The diversification of UNLs helps us to also explore the pubsub characteristics of the XRP LCP, and so we can ask (2) *if the employment of a framework for message dissemination based on pubsub can decrease the message overhead and thus increase the scalability of the network*.

Having explored these two questions, we proceed to ask (3) *how can we exploit pubsub mechanisms for better performance using the XRPL as a foundational layer*. In other words, we seek to analyze a pubsub-based dissemination framework and understand how we can tune it to better suit the characteristics of the XRPL. For that, we ask the question of (4) *which are the parameters we can tune to better performance and how are they related to performance metrics*. Knowing which are the parameters, we can ask (5) *how those parameters impact the performance of the network considering the concrete case of the XRPL*.

Finally, we consider all those inquiries presented to derive the main question that this thesis seeks to answer: *How can pubsub dissemination increase the scalability and performance of blockchains based on FBA consensus?* Figure 2 shows a simplified diagram of how each question relates to the issues and causes cited, also pointing out which chapters discuss and answer the research questions presented, culminating in the main question,

which the answer is better discussed in the last chapter, *Discussion and Future Perspectives*.

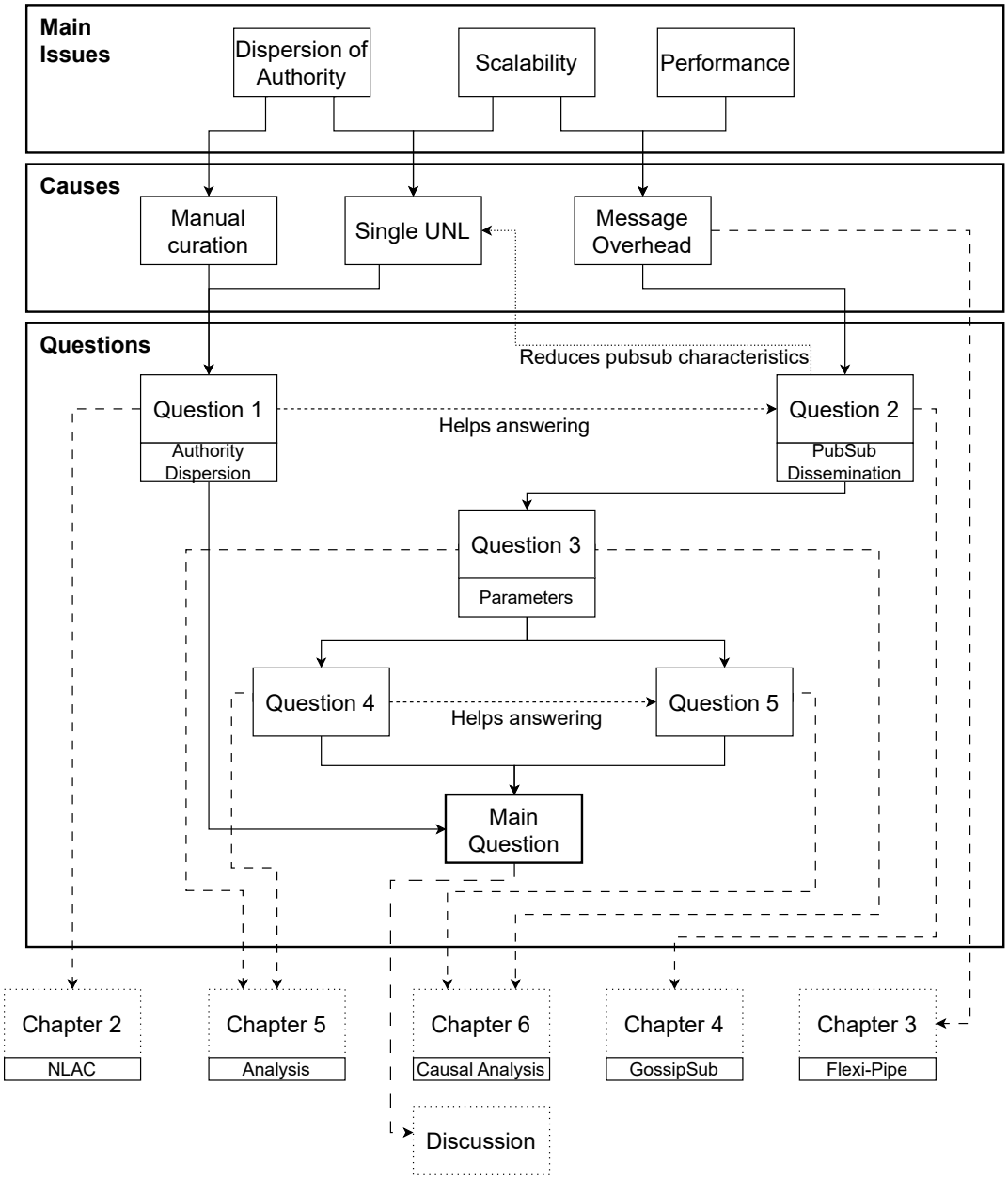

Figure 2: Simplified schematics of the thesis structures, with the main issues linked to the causes and the related questions answered

The structure of this thesis comprises three parts. First, we present the broad panorama of the main technologies we are tackling, starting with the XRPL and then GossipSub. The second part comprises the enhancements on message dissemination, made by means of enhancing the pubsub characteristics of the XRPL. Finally, the third part analyzes how the parameterization of the GossipSub mesh can impact performance and scalability, suggesting methods to better configure the GossipSub parameters for optimal functioning of the mesh.

We start the second part by answering question **1** in Chapter 2, proposing a novel framework to automatically create and maintain the XRPL trust overlay, employing mechanisms such as Token-Curated Registries (TCRs) and mathematical optimization to create a Direct Acyclic Graph (DAG) to represent the entire space of UNLs in the XRPL. We call our solution NLAC, meaning that it is nested, layered, autonomous, and continuous, and show how it can be used to create UNLs while maximizing the total trust of the system.

After proposing a method to enhance the pubsub characteristics of the XRPL, in Chapter 3 we present a quantitative analysis of the problem of excessive message overhead caused by flood publishing. To study the problem and propose solutions, we introduce a tool called *Flexi-pipe*. Flexi-pipe creates an overlay on an XRPL testnet in which we kidnap different types of message to study their behavior. This overlay also gives us the possibility to plug different types of message dissemination into the XRPL testnet without the need to modify the validator code, giving us flexibility in integrating and analyzing different message propagation mechanisms. We also look at a solution previously proposed by the XRPLF to diminish the message overhead.

After better introducing the problem and the tool used throughout this work, and having proposed a solution to enhance the pubsub characteristics of the XRP LCP, in Chapter 4 we address question **2** by integrating GossipSub into XRPL. We show that even without considering the strong pubsub characteristics of the XRPL, GossipSub can still diminish message overhead, using structures of topics similar to Ethereum and FileCoin, but also using the structure of UNLs created by NLAC, and structuring the topics in a more naive way, by abstracting every validator as a topic.

Going beyond just the use of GossipSub as a dissemination technique in the XRPL, in Chapter 5 we go further into the mechanism, analyzing its characteristics with the aim of better integration, enhancing not only scalability, but also performance to able to answer questions **3** and **4**. In this chapter, we dive into the GossipSub parameters using data science techniques to study the correlation between parameterization and some metrics of performance, such as average propagation time, bandwidth used, and message overhead. The aim of this analysis is to better understand how we can better utilize the GossipSub parameterization to enhance the XRPL performance and scalability and, as a by-product, to present a methodology to tune GossipSub for use in different unstructured p2p systems.

After analyzing and understanding which GossipSub parameters are related to performance metrics, in Chapter 6 we go a step further to understand the causal mechanisms that exist between these dimensions, tackling questions **3** and **5**. We start by employing causal discovery algorithms and the domain knowledge on GossipSub and the XRPL acquired in the previous chapters to model causal graphical models presenting the causal relationships between the GossipSub mesh parameters and message overhead. Having created a model that identifies causal relationships, we then quantified the causal influences of parameters on the target scalability metric and then simulated interventions in the causal model. The causal analysis helps us better understand how the modification of a parameter can impact the overall behavior of the system and helps us to study the system in possible scenarios that cannot or are hard to observe in real-life, providing us with the knowledge to better adapt the mesh parameters of GossipSub for different types of system. However, in general, it gives us the knowledge to exploit pubsub dissemination to achieve optimal message overhead on the XRPL, increasing its overall performance and scalability.

Finally, in the last chapter, we present a summary of the discoveries made throughout the thesis. This final chapter brings together all the answers and insights gained by tackling questions **1** to **5** to answer the main question proposed: *How can pubsub dissemination increase the scalability and performance of blockchains based on FBA consensus?*

Chapter 1

Background

1.1 The XRP Ledger

The foundation layer for the trustable functioning of the XRPL is the *consensus*; a process in which a subset of nodes participating in the network must agree upon a set of transactions that will form the next ledger version. We can formally describe it as "the state in which the nodes in a network reach the correct agreement"[2]. The concept is not only important in the context of a blockchain, but is also a topic of discussion on the matter of p2p networks in general, with a wide range of proposed solutions, such as Paxos[19], W-MSR[20] and PBFT[21]. However, traditional consensus algorithms do not usually consider permissionless environments, leading to the need to create new mechanisms or adapt existing ones.

When analyzing the top 10 crypto tokens in terms of market capital[1] in Table 1.1 we can observe that most of them uses either Proof of Work (PoW)[4] or variations of Proof of Stake (PoS)[5]. We then encounter an unusual element in the ranking, the XRPL, using a Federated Byzantine Agreement (FBA) type of consensus. FBA means that the XRPL does not select a unique validator per round; instead, it uses quorum-based voting to deliberate on the next version of the ledger. In this context, any node on the network can proclaim itself a validator, but not every node is included in the voting process.

The XRP Ledger Consensus Protocol (XRP LCP) uses quorums of trusted subnets to authenticate the ledger. In this context, a validator node must declare a Unique Nodes List

[1] Generated over the coin ranking by market capital on 12/February/2024 available at `https://cryptoslate.com/coins/`, excluding stablecoins and non-native tokens

(UNL), comprising a list of peers trusted not to collude to defraud the ledger. Although every node has the power to choose its own trusted list, in reality, some scenarios may cause the ledger to fork or stall[7] and a minimum overlap between two or more lists is necessary to ensure *safety* and *liveness* of the network. By safety, we mean that nothing bad will happen, while liveness ensures that something good will eventually happen[7].

Table 1.1: Top 10 Blockchains by Market Capital

	Token	**Market Capital**	**Consensus Type**
Bitcoin	BTC	$939.82B	PoW
Ethereum	ETH	$298.71B	PoS
BinanceChain	BNB	$47.41B	PoS
Solana	SOL	$45.52B	dPoS
XRPL	XRP	$28.21B	FBA
Cardano	ADA	$18.92B	PoS
Avalanche	AVAX	$14.17B	PoS
Dogecoin	DOGE	$11.414B	PoW
TRON	TRX	$11.02B	dPoS
Polkadot	DOT	$8.96B	dPoS

At first glance, the strategy adopted by the XRP LCP does not seem to provide many improvements in transaction speed since it relies on quorum voting. However, the reality is that it presents advantages over schemes such as PoW and PoS in terms of transaction speed and resources consumption. On average, XRPL publishes a new version of the ledger every 3 seconds, handling 27 transactions per second (TPS)[22], with a reported capacity of 1.5k TPS[23]. Whereas Bitcoin generates new blocks every 10 minutes with a maximum throughput of 7 TPS[24]. As for Ethereum after the migration to PoS, the average block time is 12 seconds with 11 TPS[25], being the maximum throughput reported to be between 15 and 45 TPS[26].

However, the XRP LCP is not without flaws. There is an ongoing discussion on how to structure UNLs in ways that can prevent forking or stalling. This problem led to the XRP Ledger Foundation (XRPLF) to advise the use of a single UNL, curated and maintained by the entity. The next big issue is a direct product of that, putting in question how decentralized

and distributed the XRPL is in reality. In addition, the XRPL also suffers from a problem endemic to blockchains; The use of flooding to broadcast messages and blocks leads to an increase in message overhead, directly impacting the required bandwidth. This problem not only refers to performance and scalability, but can also be a safety issue for the entire system. The next section discusses some related works that present and attempt to find solutions to these and other challenges present on the XRPL.

1.1.1 The XRP Ledger in the Literature

Several works analyze and discuss the characteristics of the XRP LCP, as can be seen in the timeline presented on Figure 1.1. Schwartz et al.[2] present the first official whitepaper describing the XRP LCP algorithm. This work is considered outdated by the XRPLF[27], which recommends Chase and McBrough[8] as the official whitepaper, better formalizing the algorithm and presenting a deeper analysis of safety and liveness, inferring that at least 90% of agreement between the validators is needed to ensure network safety. The authors conclude that, in general, even a minor disagreement may cause the ledger to stop making forward progress. However, the analysis takes into account any two overlapping UNLs, which leaves open the question of the minimum overlap required when considering a more complex scenario with more than two UNLs of varying sizes. The authors also suggest that the underlying message dissemination pattern may have the potential to leverage some safety concerns. McBorugh also proposed later Cobalt[28], a novel atomic broadcast algorithm. Unlike our work, Cobalt requires changes in the consensus fabric, whereas our proposal changes only the way messages are broadcast during two specific states of the consensus process.

Subsequent works by Armknecht et al.[29], Mauri et al.[15] and Amores-Sesar et al.[7] walk in the direction of more formal descriptions and bring new cases that may cause the network to break or stop making forward progress, violating safety and liveness. Christodolou[30] brings a different approach to the UNL overlapping problem, with an empirical approach, showing that the minimum UNL overlap can be relaxed when there are less than 20% of malicious nodes present. The work suggests space for optimizations on the minimum UNL

overlap, also suggesting the necessity of a malicious node estimator.

Other works dive into different kinds of analysis of the XRPL and its validators. Tumas et al.[31] show a study of the topology of the underlying peer-to-peer (p2p) network that pow-

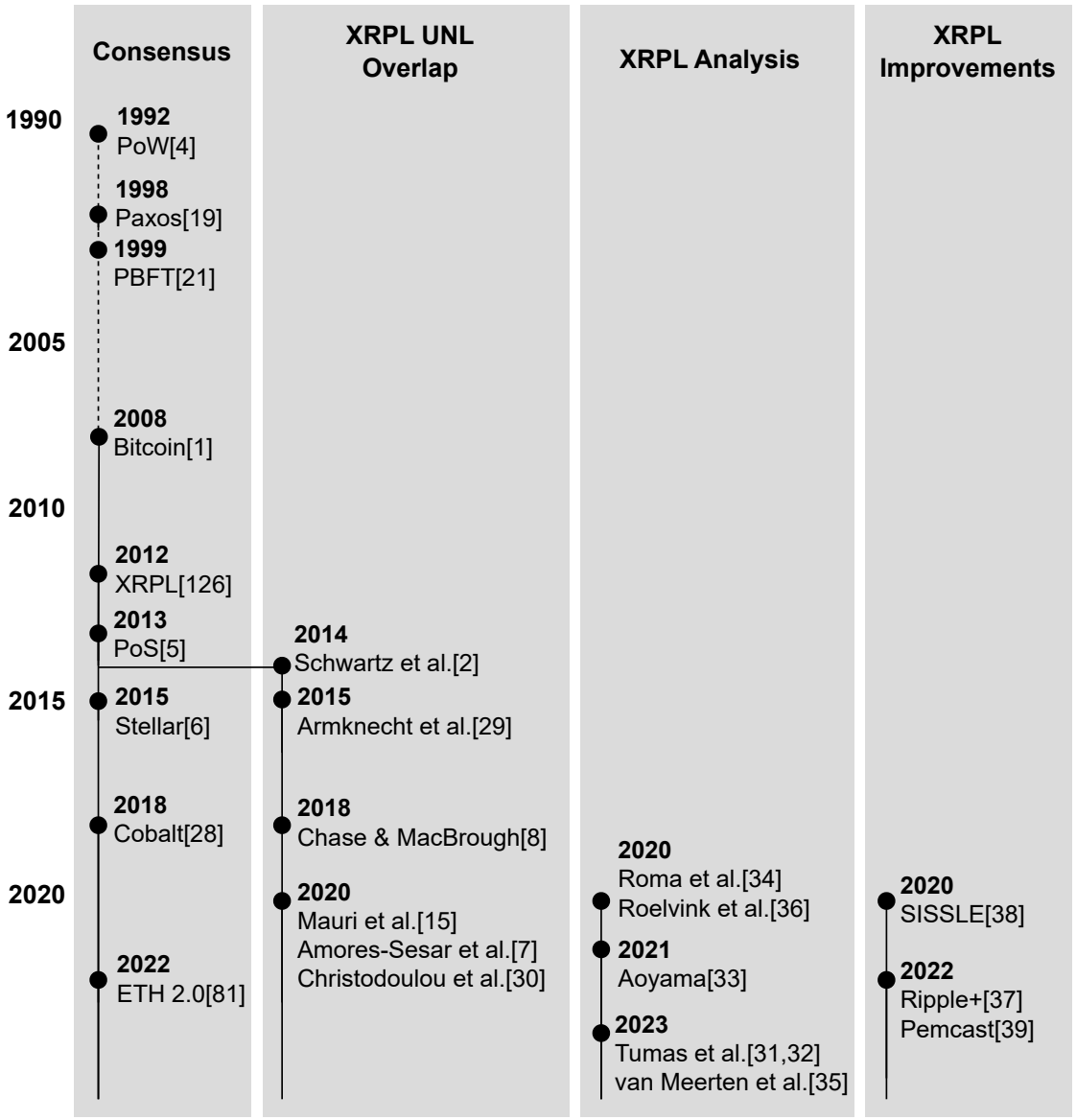

Figure 1.1: Timeline with the main works and technologies related to the XRPL

ers the XRPL, showing that there are some vulnerabilities still to be addressed. Later, the author also presents an analysis of the robustness of the FBA Protocol applied to the XRP LCP, introducing a new robustness metric and a mitigation strategy based on inbound and outbound connections between validators[32]. Aoyama[33] applies Flow Index to analyze the XRPL network from the point of view of the data flow of transactions on the ledger. In terms of the operation of single validators, Roma et al.[34] present an analysis of the energy consumption of the XRPL validators. On the other hand, van Meerten et al.[35] and Roelvink et al.[36], focus on validating the implementation of the *rippled* software using both log-inference and evolutionary approach for concurrency tests.

The works presented until now mainly address the problem of ensuring liveness and safety. There are two main studies that aim to provide the XRPL with more flexibility and decentralization targeting the UNL structure, targeting methods to relax the minimum overlap required. Ripple+[37] provides a mechanism for selecting UNLs based on a structure of core and leaf nodes, also proposing changes in the consensus process by allowing for weak synchronicity between nodes. On the other hand, SISSLE[38] addresses the problem by suggesting the use of Distributed Hash Tables (DHTs) to build UNLs, also presenting a study of the overlap problem as a function of information propagation and node reputation. However, both works suggest considerable modifications to the core of the consensus algorithm. In this work, we also address the UNL construction and maintenance, using a structure similar to Ripple+, but employing self-curated mechanisms to guarantee dispersion of authority, looking to keep core structures intact, providing tools for the XRPL to become more scalable and decentralized.

In this work, we also tackle the issue of mitigating the message overhead on the XRPL. To our knowledge, only one other work tries to solve this issue. Pemcast[39] uses probabilistic multicast routing to mitigate the number of duplicate messages that transit the network. However, the work does not consider important characteristics of the XRP LCP, and does not guarantee a 100% delivery rate for broadcast messages.

1.1.2 The XRP Ledger Consensus Protocol

The XRP LCP uses a federated consensus process in which nodes vote to reach consensus, working with the idea of *subjective validators* instead of relying on established models for *Byzantine Agreement* and *Byzantine fault-tolerance*[7], meaning that each participant takes into account the position of its trusted peers to cast its votes.

To better understand the proper functioning of the XRP LCP, we need to take into account the following concepts:

- **rippled:** Official C++ implementation of the XRPL validator[2].
- **Unique Nodes List (UNL):** Static list containing the validators that a node trusts will not collude. A validator trusts its UNL collectively, and not each validator separately.
- **Node:** A participant in the network. A machine that runs the *rippled* code as a *server*. Nodes can play different roles in the XRPL, in this work, we consider only one role: *validator*.
  - **Validator:** A node that votes. In the literature, a validator can also be called a *Proposer*[2].
  - **Peer:** A node to which a given node is connected, at any overlay level. For the XRP LCP we consider a peer to be any validator contained in the UNL of a given node.
- **Ledger:** The record of all transactions on the network[2]. Formally, for the XRP LCP, the ledger also represents the *shared distributed state*[40].
  - **Last-closed Ledger or Previous Ledger:** The last shared state recorded locally.
  - **Preferred ledger:** The last closed ledger agreed upon network-wise. While the *previous ledger* refers to the last state recorded locally by a node, the *preferred ledger* denotes the actual shared state.

[2] Available at `https://github.com/XRPLF/rippled`

  - **Open Ledger:** The ledger a given node is working upon on a given consensus round. None of the transactions contained in this ledger are final, as they did not pass through the consensus mechanism entirely.

- **Message:** Any message sent or received by a node.
  - **Transaction:** Any message that may cause a change in the shared state. Formally, a transaction is any instruction that causes changes in the ledger, such as payments, bidding, checks, and escrows.
  - **Proposal:** A proposal is a *set of transactions* that a given node proposes to apply to the *open ledger*.
  - **Validation:** The final set of transactions reached by a node after multiple consensus rounds.

- **Position:** The current proposal of a given node. In other words, the belief a given node has of which transactions should or should not be applied to the ledger.
  - **Vote:** The position of a node in a given transaction.
  - **Dispute:** A transaction that is present in the position of a node but not in the position of one or more of its peers - or vice versa.

- **Consensus round:** A *consensus round* comprises a full cycle of the state machine presented on Figure 1.2, from the *open* phase until reaching the *closed* phase.
  - **Proposal round:** An internal cycle of the distributed states machine, comprising the *establish* phase of the protocol. This is the phase in which ledger proposals are exchanged.

The Distributed States Machine

The XRP LCP works as a synchronous state machine. Transactions, however, are sent and received asynchronously and stored in a buffer until a new consensus round begins.

Figure 1.2 shows a simplified view of each phase, while Figure 1.3 shows a more detailed picture, with its internal states and transitions considering only the states related to voting and ledger creation, abstracting all the states related to synchronization and ledger closing times, as those are beyond the scope of this work.

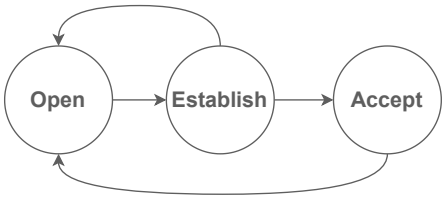

Figure 1.2: Simplified State Machine for the XRP LCP with the three main phases

The *open* phase is a period in which the new open ledger is created and the first position is generated. The node keeps adding transactions received during this time to a buffer. As soon as the time reaches half of the time spent on the previous *consensus round*, the ledger is closed locally and a proposal is formed.

The protocol then enters the *establish* phase incorporating a new component: the *threshold*. Given a dispute, the threshold is the minimum number of positions in which the disputed transaction must be present, so the node will cast a “yay” vote. The threshold changes during each *establish round*, starting at 50% and increasing until it reaches 80%.

During the *establish* phase, the proposal formed previously is transmitted to the entire network through *flooding*. At the same time, the node receives proposals from its peers; if there is a *dispute* and the threshold is lower than 80%, the validator starts to update its position by creating a new proposal. If the total time elapsed during the establish phase is lower than the time of the previous *consensus round* the node initiates a new proposal; if not, it closes the ledger and moves on to the next phase. First, the threshold is recalculated on the basis of the time left, and then the voting starts: For each transaction in the disputed set, the node will consider the position of each of its peers. If the percentage of peers that contain the dispute on its position (yay vote) is higher than the threshold, then the node adds the transaction to its proposal; if it is lower, the disputed transaction is ignored.

With the new proposal formed, the ledger is closed locally; If the threshold reaches 80%

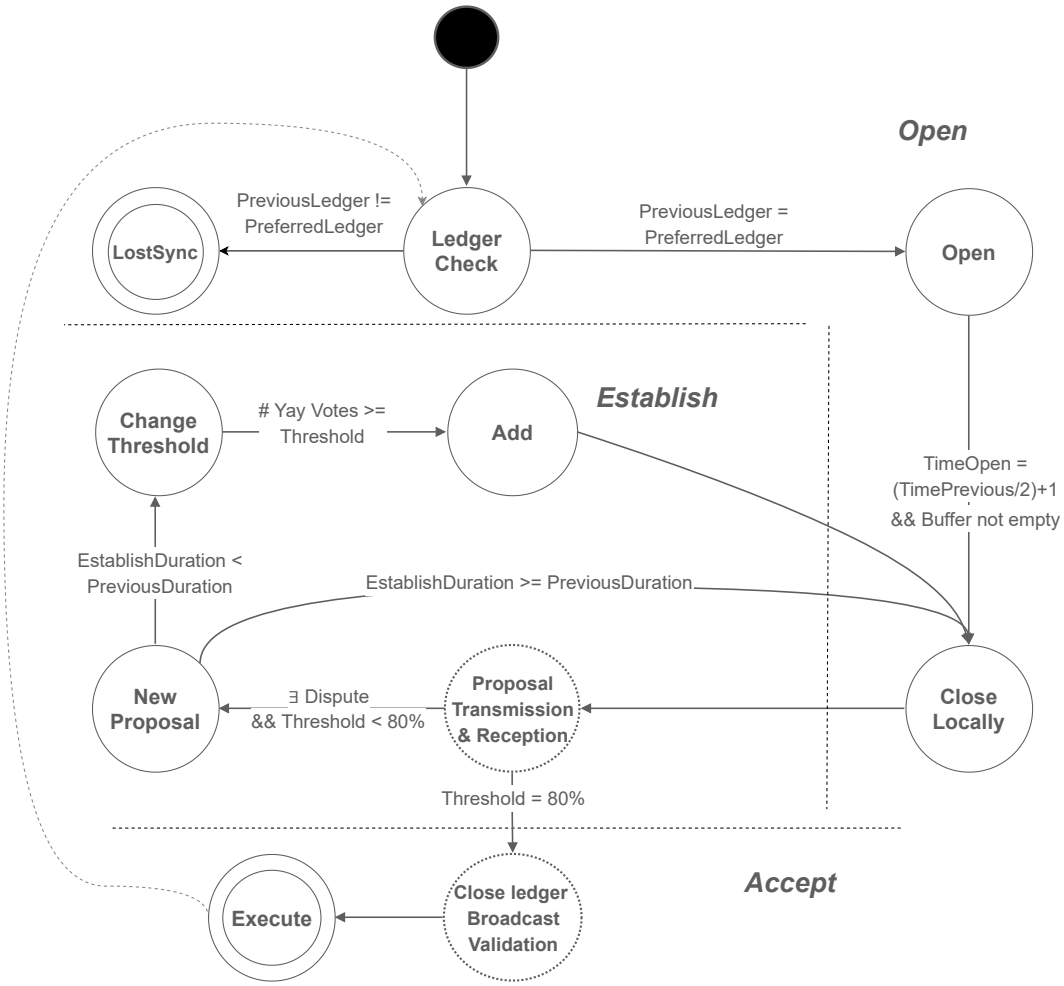

Figure 1.3: State Machine of the XRP LCP, demonstrating the internal states of each phase with the respective transitions and its triggers

or there is no dispute, the *accept* phase starts by transforming the proposal into a validation, then exchanging the validation by flood publishing. The nodes must now agree on which validation to apply. Taking into account a given node, if 80% of its peers have the same final position, the node applies the transactions contained in the validation to the ledger; if there is less than 80%, it means that the node is out of sync and must start the syncing process again.

1.1.3 The Trust Overlay

The idea behind the XRP LCP is that “a little trust goes a long way”[41]. This notion led to the creation of the concept of collectively trusted lists. Shifting the trust to these sets of validators, instead of trusting a unique or a set of participants selected based on a predefined criterion[3], means that the XRP LCP needs a tightly connected trust overlay to avoid forks, stalls, and possible malicious manipulation of the ledger.

We call *trust overlay* the entire space of trust relationships between nodes established by the declared UNLs. Certain overlap conditions must be met between trusted sets to guarantee the liveness, safety, and performance of the ledger. With a loosely connected trust overlay, validators would take longer to agree on a final ledger version due to clustering and the difficulty of transactions reaching distant nodes. A highly connected network, however, has the disadvantage of high message overhead, increasing latency, and, as a result, landing on the same problem of increasing the time needed to agree on closing a ledger or even increasing the possibility of a stall or fork[42].

1.1.4 Unique Nodes Lists

The XRPL uses quorum-based voting to deliberate on the next version of the ledger. In this context, any node on the network can proclaim itself a validator, but not every node is included in the voting process. Each validator has a list of trusted peers, the UNL[2], which means that the vote of a validator is only taken into consideration if it is present in a UNL. Consider Figure 1.4, where we have the representation of a network with five validators: *A*, *B*, *C*, *D* and *F*. Let U_A be the UNL of node *A*, so $U_A = \{C, D, F\}$, which means that the validator *A* trusts the votes of the nodes *C*, *D* and *F*. Following the example, $U_B = \{A, F\}$, $U_C = \{A, D, F\}$, $U_D = \{A, C\}$ and $U_F = \{A, C, D\}$.

Let us take a look inside the validator *A*, presenting the same process described in Section 1.1.2 from the point of view of the trust overlay. *A* receives transactions from different sources asynchronously and stores them in a buffer. At the start of a new consensus round,

[3]Computational proof in PoW and random selection on PoS

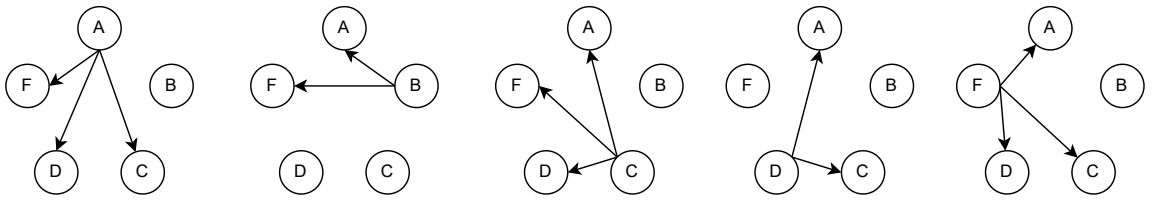

Figure 1.4: Representation of the UNLs in a network with 5 validators. Arrows represent the directional trust relationship between two nodes.

A forms its first *proposal*, which is transmitted to the network at the same time that *A* receives proposals from all other nodes. *A* then compares its own proposal to the ones received from *C*, *D* and *F*. If a discrepancy is encountered, a *dispute* is formed. For each dispute, *A* will count the number of trusted validators that included that transaction in their proposals if it is present in more than 50%, *A* will include the transaction in its new proposal. After forming a new proposal, it is transmitted to the network and the process repeats, increasing the inclusion threshold at each interaction until no dispute is found or the threshold reaches 80%.

In the scenario presented, all validators cast their votes by generating proposals and adjusting them according to the positions of their trusted peers. A well-trusted node, such as *A* has more influence on the network than nodes not so trusted, such as *C* and *D*. As for *B*, the node is casting its votes the same as the others; however, it does not actually participate in the consensus process, since it does not belong to any UNL; that is, it is not trusted by any of the other nodes and therefore does not have a voice in the voting process.

1.1.5 UNL Overlap

We presented the trust overlay and briefly discussed the density of trust connections within this layer, then showing how these relationships are defined and their impact on the consensus process. Taking into account that, the density is given by the intersections that exist between all UNLs declared in the network. The minimum overlap required is still an open question, and the current solution is to employ a unique UNL curated by the XRPLF.

As shown in Section 1.1.1, four primary works address the issue of the minimum inter-

section area between UNLs required to ensure safety and liveness. The first whitepaper published[2] presented the following formulation for the minimum overlap required ($w_{i,j}$) between two UNLs i and j:

$$|UNL_i \cap UNL_j| \geq w_{i,j} max(|UNL_i|, |UNL_j|) \, \forall i, j$$

This work concludes $w_{i,j} >= 0.2$, which means that the overlap should be greater than 20% of the size of the largest UNL, assuming $0.2 * n_{total}$ to be the minimum UNL size, being n_{total} the total number of nodes participating in the network. Later, an independent analysis by Armknecht et al.[29] showed that the 20% overlap may not be sufficient to reach consensus, increasing the minimum required overlap to 40%.

The current whitepaper[8] was released in 2018, stating that the intersection area between two UNLs should be superior to 90%, also pointing out a specific case where even with a 99% overlap, the ledger progress could stall. This work was the first to propose the use of a single UNL. Another independent analysis by Christodoulou et al.[30] tackles the UNL overlapping problem using an empirical approach, showing that the minimum UNL overlap can be relaxed when there are fewer than 20% malicious nodes present. This work suggests space for optimizations on the minimum UNL overlap, also suggesting the need for a malicious node estimator to relax the overlap.

1.1.6 The XRPL Main Network

Based on the studies presented, the XRPLF curates and maintains a main UNL, advising all participants to trust this list. Given these guidelines, as of July 2024, these main UNL contained 35 validators. Considering that all validators trust the same UNL, the center of the XRPL comprises a complete graph of 35 nodes that use flooding to disseminate proposals and validations. The centralized structure grows with the increase in the adoption of the XRPL, leading to latency issues. These issues may cause the validators to act at fault and increase the possibility of a stall on the ledger progress.

Trestioreanu et al.[14] shows a picture of how the underlying network that powers the

Table 1.2: The XRPL Network in 2021[14]

Nodes	Validators	UNL Size	Avg distance	Avg Degree	Diameter
892	152	35	2.37	20.62	5

XRPL was structured in 2021, Table 1.2 summarizes these numbers. Furthermore, this work points out one of the main issues caused by the use of flooding, concluding that the present implementation of the XRPL suffers from a lack of *node-wise* scalability. The 152 nodes that declare themselves validators flood the network with their proposals and validation in each consensus round, with the proposals being broadcast several times per round, as explained in Section 1.1.2. As the network grows, the problem may become more prominent, as it has already been identified on other blockchains, such as Bitcoin[43]. The next section presents the state-of-the-art in message dissemination on blockchains as a possible solution.

1.2 GossipSub

GossipSub is a modern communication system within blockchains, serving as a more efficient alternative to the less optimal flooding method observed in earlier blockchains. While flood publishing ensures reliable message delivery in unstructured p2p networks, it also leads to redundancy and delays[44]. GossipSub addresses this challenge by introducing a solution rooted in publisher/subscriber (pubsub) systems. These systems consist of publishers and subscribers, where subscribers express interest in specific topics and publishers share related messages, called *events*; the act of expressing interest in a topic is called *subscription*[45].

However, implementing pubsub systems in unstructured p2p networks presents a complex task. The GossipSub protocol is particularly prominent in this area, originally intended for integration with platforms such as The Interplanetary File System (IPFS)[18], FileCoin[17] and Ethereum[46]. GossipSub introduces a gossip-driven pubsub protocol to enable efficient message dissemination in p2p overlays[46] and is distributed as an extensible component within libp2p[47].

GossipSub is built by a *mesh* - employing *eager push*, *lazy pull* and a *score function* - interlaced with a set of mitigation strategies. The mesh is the component that makes the solution feasible for employment on blockchains given that its low fanout balances bandwidth consumption and performance, while the lazy pull employed guarantees that messages will be successfully propagated even to distant nodes[46]. The score function evaluates peers based on their network behavior and identifies any malicious actions. All nodes on the network observe the nodes to which they are connected and act accordingly in terms of routing messages. The mitigation strategies act upon these flagged events to protect the system against malicious activity.

Regarding the structure of GossipSub topics in blockchains, both Ethereum and FileCoin adopt similar strategies, although with nuanced differences[48]. FileCoin employs a dual-topic structure, with one topic for message dissemination and another for block propagation. On the other hand, Ethereum uses a more intricate arrangement, featuring five global topics, including two main and three secondary. The first main topic, known as the *beacon*, broadcasts newly *signed blocks* to the entire network. The second facilitates the dissemination of *aggregated attestations* to subscribing validators. Furthermore, Ethereum employs three secondary topics to propagate different types of information: *voluntary exit*, *proposer slashing*, and *attester slashing*.

1.2.1 GossipSub in the Literature

GossipSub is not the first gossip-based pubsub protocol to be proposed, as can be seen in the literature review timeline present in Figure 1.5. Other solutions such as Bayeux[49], Scribe[50], Meghdot[51], Tera[52], Rappel[53], Stan[54], Vitis[55] and Poldercast[56] have been previously implemented. However, none of those solutions was designed to be Sybil-resistant. Vyzovitis et al.[48] proposed a protocol called GossipSub as a replacement for the Ethereum message dissemination protocol, SQRT(N)[16], by closing the gap on Sybil-resistance left by previous works without sacrificing system performance.

Remarkably, the performance and scalability of GossipSub have not been addressed in academic research. A single non-academic investigation[57], conducted by the Whiteblock

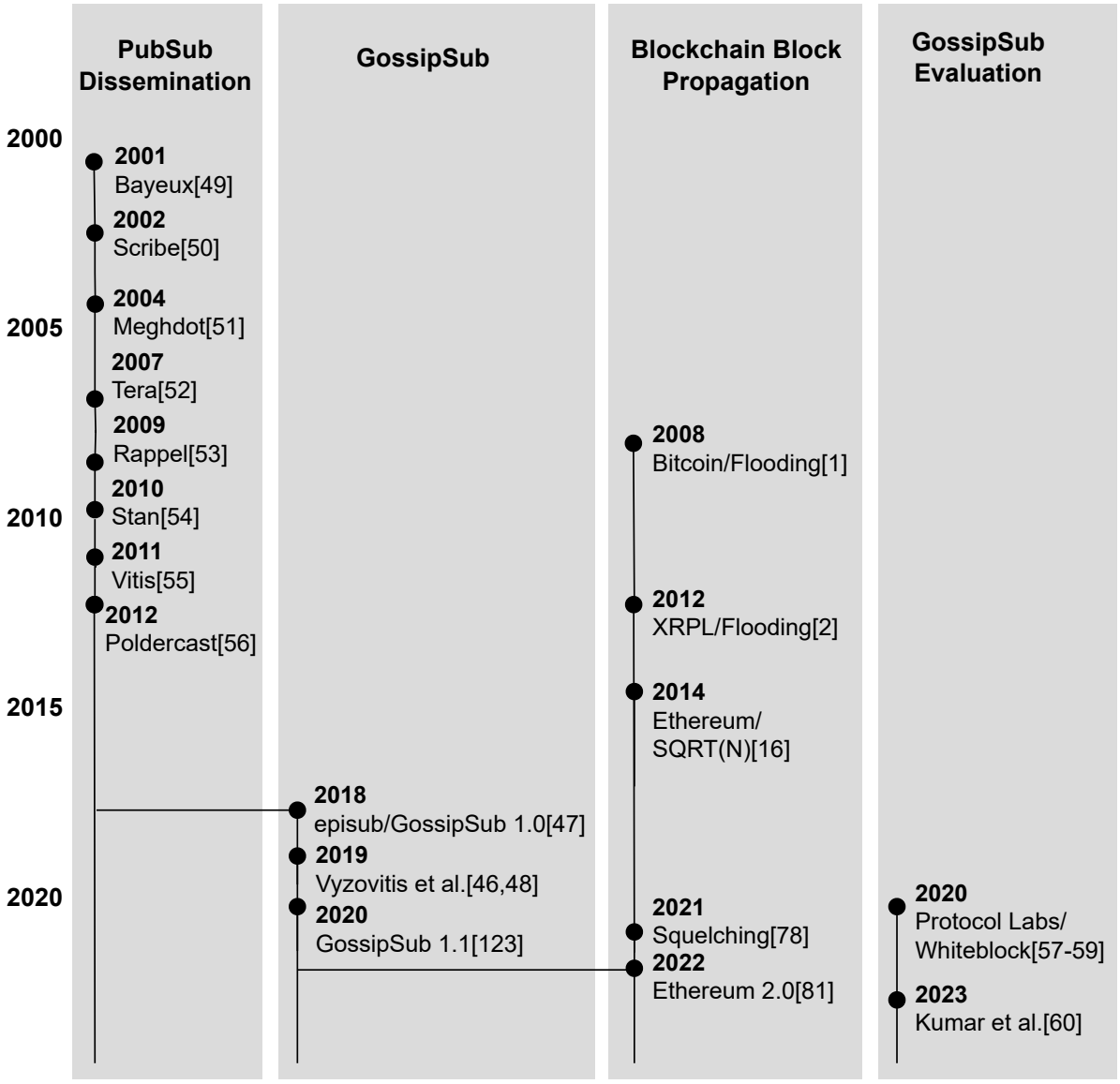

Figure 1.5: Simplified timeline of academic works and technologies related to GossipSub

company[4], offers an assessment of the performance and safety of GossipSub[58]. This study revealed lost messages and inconsistent delays, which were attributed to the specifics of the Golang implementation. Moreover, they evaluated the time it takes for messages to

[4]Which received a joint grant from the Ethereum Foundation and ConsenSys for this work

disseminate under a 400ms latency delay, concluding that it is suitable for Ethereum 2.0 deployment, despite the irregular intervals between message arrivals caused by the Golang implementation.

In a different perspective, the original GossipSub authors[46] also provided an in-depth examination of the resistance to Sybil and Eclipse attacks in distributed systems that employ GossipSub[59]. Their findings indicated that performance may decline under certain conditions, but the protocol remains functional. They suggested three countermeasures: *flood publishing*, *opportunistic grafting*, and *increased gossip propagation*. These methods markedly improve the performance of the protocol in the most adversarial environments. This research also elaborates on each mesh parameter and offers some recommendations for selecting values that align with the desired performance results. The default parameters are configured to maintain a low fan-out and minimize bandwidth usage while still ensuring an acceptable message propagation time. However, there is a limited sensitivity analysis of the potential impact of some mesh parameters on network performance. In this work, our aim is to close this gap by expanding the analysis with different sets of values, including also more parameters, and employing data science and causal analysis techniques. More recently, Kumar et al.[60] performed an analysis of the robustness of the protocol against attacks by misbehaving peers using the ACL2 prover, finding a particular case where peers are not flagged as malicious when misbehaving by not forwarding topic messages. The work suggests three concrete steps through which developers can harden GossipSub from specific attacks.

1.2.2 GossipSub Mesh Structure

We can abstract the GossipSub mesh as consisting of two layers, the first one being a *full-message overlay*, in which each node has a local view of the mesh. The local view of a node comprises the set of peers with which it is directly connected. Here, messages are sent directly to peers according to some parameters that control the fanout. The second layer can be called the *gossiping overlay*, employing lazy pull to guarantee that a message will reach all its targets. Nodes can communicate with other nodes outside their local view

by gossiping. In this scenario, in the gossiping overlay nodes exchange metadata as well as announcements such as ”I have message x” (*ihave*) and ”I want message x” (*iwant*) and full messages are then exchanged, working as a reconciling phase.

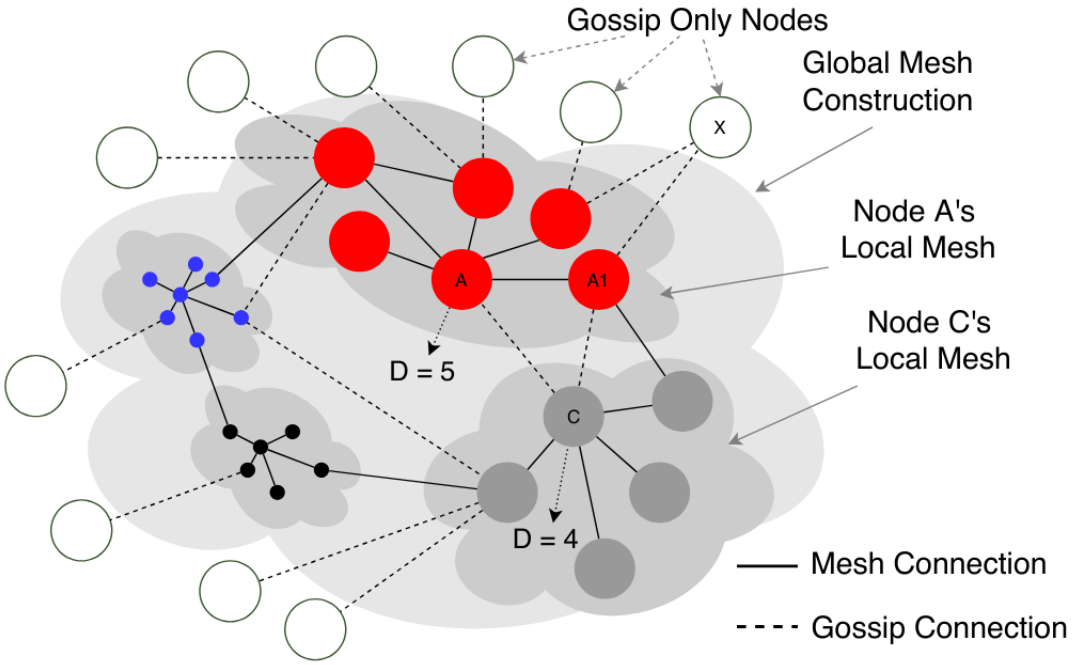

Figure 1.6: GossipSub mesh schematics. From the original GossipSub paper[46]

Figure 1.6 presents the original diagram published by Vyzovits et al.[46] showing the GossipSub mesh construction. We can see that each node has its local mesh, denoted by the full lines. The full message overlay is depicted as the gray area comprising all local meshes, using eager push to send messages through direct connections formed by the *local view of the mesh* of each node. The gossiping overlay is depicted by the dashed lines connections, this layer employs gossiping to reach distant nodes using lazy pull. The formation and maintenance of these two mesh components are controlled by a set of parameters presented in Table 1.3.

The number of peers to which a node is connected is given by three parameters: *D*, *Dlo*, and *Dhi*. *D* represents the desired number of peers to which a node should be directly

Table 1.3: GossipSub Mesh Parameters

Parameter	Concept
D	Desired number of peers[a]
Dlo	Minimum number of peers[a]
Dhi	Maximum number of peers[a]
Dscore	Number of high-scored peers to keep when prunning
Dout	Number of outbound connections to keep when prunning[a]
Gossip Factor	Factor (%) of how many peers to emit gossip[b]
Dlazy	Minimum number of peers to emit gossip[b]
Interval	Heartbeat for prune, graft, and gossiping events

[a]For each subscribed topic
[b]For connections outside of the meshes of the topics

connected, defining the average fanout of the network. In Figure 1.6, for example, the node *A* has the value of *D* set to 5, being connected to 5 peers, forming its local view of the mesh. *Dhi* and *Dlo* are parameters to relax the fanout, *Dhi* being the ceiling, and *Dlo* the floor. These parameters trigger two types of events within the local mesh: *prune* and *graft*. The first is a disconnection that occurs when the number of peers of a node grows above *Dhi* for each subscribed topic. Similarly, the second is a new connection and occurs when the number of peers falls below *Dlo*. The node chooses how many and to which peers to disconnect/connect based on two parameters: *Dscore* and *Dout*. *Dscore* sets the number of peers with the highest performance, according to the scoring function, to which a node must connect, the remaining being randomly selected. When a pruning event occurs, the node should retain at least *Dout* number of peers (greedy pushing).

The main parameters related to lazy pull are *Dlazy* and *GossipFactor*, both control-

ling the number of peers to which the node should emit gossip. The gossip is emitted to $GossipFactor * (\#non\_mesh\,peers)$ or *Dlazy*, whichever is bigger. The gossip emission occurs between time intervals, set by the *HeartbeatInterval* (or simply *interval*) parameter.

D, *Dlo*, *Dhi*, *Dscore*, and *Dout* are all linked to the full message overlay, being defined per topic, dictating how each node will construct its own view of the mesh, resulting in prune and graft events. *GossipFactor* and *DLazy* are used for the second overlay and control gossiping, which occurs through iwant and ihave messages. *HeartbeatInterval* defines the interval in which the four cited events may occur.

Chapters 5 and 6 show respectively a correlational and a causal analysis of how the parameters presented impact the overall performance of the underlying network, in our case, the XRPL. However, before diving into the integration between GossipSub and the XRPL, we first need to propose a way to enhance the pubsub characteristics of the XRP LCP so we can better explore different configurations for optimal pubsub dissemination. The next chapter presents NLAC, a tool to automatically generate and maintain UNL space, creating more diversified lists while seeking to maintain the maximum trust of the system.

Part I

Increasing Scalability on the XRP Ledger

Chapter 2

Building and Maintaining UNLs

2.1 Introduction

Since the release of Bitcoin[1] several blockchains have been proposed in different shapes and flavors. Being one of the first, the XRPL is now well established in this middle. Its unique consensus mechanism[2] also makes the XRPL a faster and less resource-intensive system, breaking away from the traditional Proof of Work (PoW). However, there is a discussion about how much *permissionless* and *distributed* the validation process is.

By definition, in the XRPL, each node in the network must declare trust in a Unique Nodes List (UNL), even those that do not proclaim themselves as validators. Given that, it is straightforward to think that certain conditions must be met on the trust overlay to avoid forks and stalls caused by the formation of clusters. Such conditions have been studied by some authors, and a quick discussion of them is presented in Section 1.1.5.

In practice, the XRP Ledger Foundation (XRPLF) curates and maintains a main UNL, advising all participants to put their trust in this list[61]. Given these guidelines, as of July 2024, the trust overlay of the XRPL network comprises a core of 35 validators connected in a complete graph. This structure and governance method leads to some considerations about the dispersion of authority in the current implementation of the XRPL.

In this chapter, we propose NLAC (*nested, layered, automatic, continuous*)[62], a framework not only for automatically generating single trusted UNLs, but also for creating, managing, and maintaining the entire UNL space. NLAC comprises three modules that manage the membership, classification and grouping of nodes into trusted lists without human inter-

vention by using layered Token-Curated Registries (TCRs) and mathematical optimization to achieve maximum trust in the UNL space while guaranteeing a better dispersion of authority.

2.2 Proposal

2.2.1 Properties

Nested

Since UNL overlap is still an open topic in XRP Ledger Consensus Protocol (XRP LCP) research, and a crucial requirement for the trust overlay, we propose a solution that allows **nesting**. When several lists of nodes are picked from a random graph, representing each set as a linear structure may not satisfy the overlapping requirement. In addition, it does not give control over the number of nodes at the intersection of such sets.

A nested structure as a tree is more suitable for this representation, as we may express every UNL as a tree and the set of all UNLs as a forest. Overlapping this forest gives us a Direct Acyclic Graph (DAG) as shown in Figure 2.1. This structure provides flexibility in choosing the size of the intersection between two or more UNLs and the general percentage of overlap of the entire space.

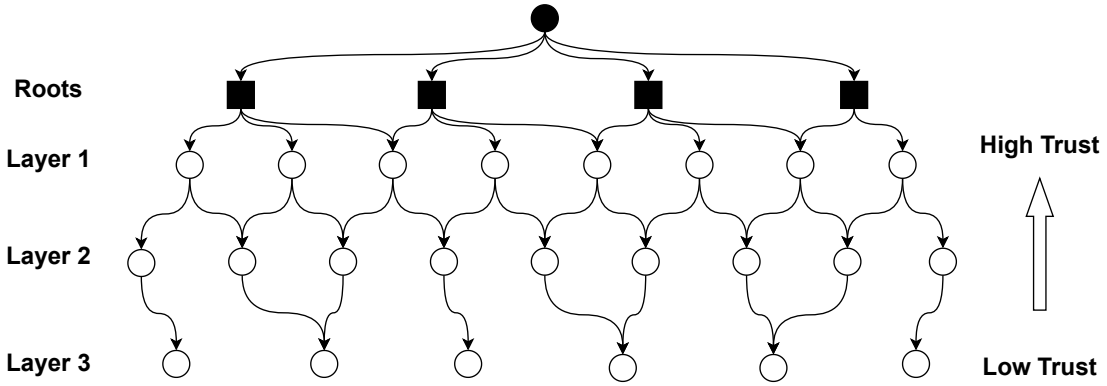

Figure 2.1: DAG proposed to represent the UNL space of the XRPL

Layered

The nodes in the XRPL network do not have direct incentives to become validators[63]. Consequently, there is no mechanism to incentivize validators to behave in the best interest of the network once they are added to the main UNL. We propose a reputation-based system comprising **layers** of trust. This structure is the foundation for developing an awarding system that gives monetary incentives to nodes to grow their good reputations.

Taking the DAG proposed previously, we have an inherently layered structure. New validators are added as leaves and may progressively rise in the tree as their good reputation increases. This model ensures that each list will have reputable nodes, being able even to control the number of nodes at each level to keep the current trust threshold at 80%.

Autonomous

The curation and maintenance of the main UNL are manual, with nodes sending their candidacy to the XRPLF for consideration. We need a solution able to self-curate to ensure more reliability and greater authority dispersion and autonomy. We then need a solution that can control the amount of overlap between the lists and the mobility between layers without human intervention. To tackle this issue, we propose the use of TCRs[64]. TCRs are lists curated in a decentralized way, using tokens to reward curators and incurring monetary penalties for low-quality content and bad curation.

TCRs allow us to have an automatic and transparent process for generating, curating, and maintaining UNLs. It also gives us tools to control layers, using some smarter implementations, such as Nested and Layered TCRs[65][66]. Another advantage is the possibility of implementing a rewarding system based not only on good behavior but also on good curation. However, some adaptations are needed to meet the requirements to generate UNLs, as described in this work.

Continuous

The XRPL is built on top of an unstructured peer-to-peer (p2p) network. As such, it comprises a dynamic system with nodes joining and leaving at any moment. Participants may also suffer from malicious intrusions, bad network conditions, and a variety of other byzantine behaviors. To account for this dynamic scenario, UNLs must adapt flexibly without compromising safety and liveness.

Nodes must be continuously observed and easily removed from trusted positions, providing the trust overlay the ability to self-arrange. We propose the use of continuous TCRs, giving token holders the ability to challenge validators that perform below the threshold of their layers[67]. This ability requires the use of a scoring system capable of detecting a change in the environment and in the attitude of the nodes. Integration between XRPL and GossipSub[68] - discussed in Chapter 4 - is the key to implementing this system, as the framework already provides a scoring system used for pruning and grafting connections. Being GossipSub a modular framework, the scoring system can be adjusted to better suit the characteristics of the network.

2.2.2 Architecture

To achieve the desired properties described above, we propose a solution based on the architecture shown in Figure 2.2. This section briefly describes each component and the interaction between them. We further expand on how they work and their concepts in the three following sections.

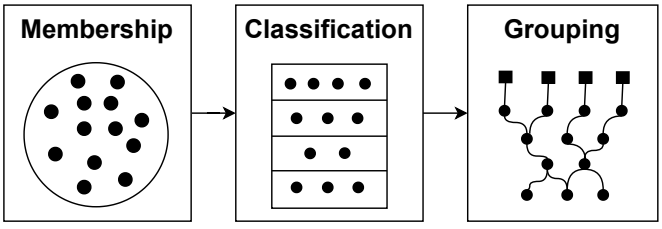

Figure 2.2: NLAC Modules Architecture

The first component, called *Membership*, comprises a layer in which the validators are

selected to be part of the *UNL space*. The UNL space encompasses all validators considered trustworthy by the network. Currently, the XRPL network already has the UNL space created, with 35 nodes manually selected to be part of the trusted set. In this work, we propose the use of TCRs[64] to create the UNL space without human interference.

After defining the membership in the set, we move on to the next component: *Classification*. The idea is to classify all validators present in the UNL space into trust levels according to a score obtained by their peers. Nodes at higher levels are said to be more reliable than the nodes at lower levels. These higher-level nodes have fewer chances to behave in a byzantine way.

Having the UNL space classified in layers of trust, we need to group the validators into lists to guarantee the dispersion of authority; hence the *Grouping* module. NLAC organizes lists as trees, taking advantage of the trust layers generated in the above component. These trees overlap each other, forming a DAG in the form of a forest, as shown in Figure 2.1, which means that some nodes will be part of two or more lists. To ensure reliability, highly trustable nodes are near the roots, and less trustable nodes stay in the leaves.

Consider the current structure of the XRPL trust overlay. Since there is already a space with 35 trusted nodes, we can consider that the first module already exists. Although not fulfilling the *autonomous* property, we will take this structure to explain how the components interact.

2.3 Membership

2.3.1 Token-Curated Registries

In its most naïve form, TCRs are simple publicly curated lists. Listees have no direct monetary reward, but indirect gains for figuring on high-quality lists. Take the example of a list of the best European universities in a certain field. Figuring on this list may ensure the universities a higher number of applications and higher investments in the particular field.

Each list usually has an associated token. To participate in the curation, one needs to possess a certain amount of tokens. Token holders can vote and challenge candidacies. To

apply to the list, a user [1] needs to stake a certain amount of tokens, which are held in a pool associated with the candidate and are untouched until the candidacy process is over.

During the candidacy period, any token holder can challenge the entry if they judge the candidate as not suitable for the list. To issue a challenge, the holder must stake tokens within the pool. All holders can vote to accept or reject the challenge. If the majority votes to accept the challenge, the challenger receives the staked tokens back, plus a reward for good curation. The candidate, for its part, loses its staked tokens and is not included in the list. However, if the holders vote to reject the challenge, the challenger is the one who loses the stake. In this situation, the conclusion is the same as if no challenge has been issued during the candidacy period: the candidate receives their tokens back and gets included in the list.

2.3.2 UNL Membership

Figure 2.3 shows the process to include a new node in the UNL space. To figure in the space of trusted validators, a node must stake a minimum amount of tokens within a token pool; then a candidacy period starts. During this period, the node is evaluated by its peers, who will construct the first reliability score. The proposals formed by this node are only observed for scoring and will not be taken into consideration until the node is included in the NLAC structure. The scoring comprises the same guidelines used by the XRPLF to evaluate candidates[12].

At the end of the evaluation period, the upper layer nodes can challenge the candidate if the guidelines[12] were not met or if the reliability score is lower than a certain threshold. All the nodes that were observing the candidate can vote to reject or accept the challenge. Note that it is not reasonable to have all validators in the trust overlay evaluating a candidate. This approach would require candidates to transmit their messages to the entire network, causing a high message overhead. The proposal for NLAC is that only a subset of validators observe and evaluate the candidate, choosing the evaluators using a peer-prediction mechanism similar to the one used in CitedTCR[69].

[1] Here “user” has the broad sense of a person, an entity, or a generic computational system.

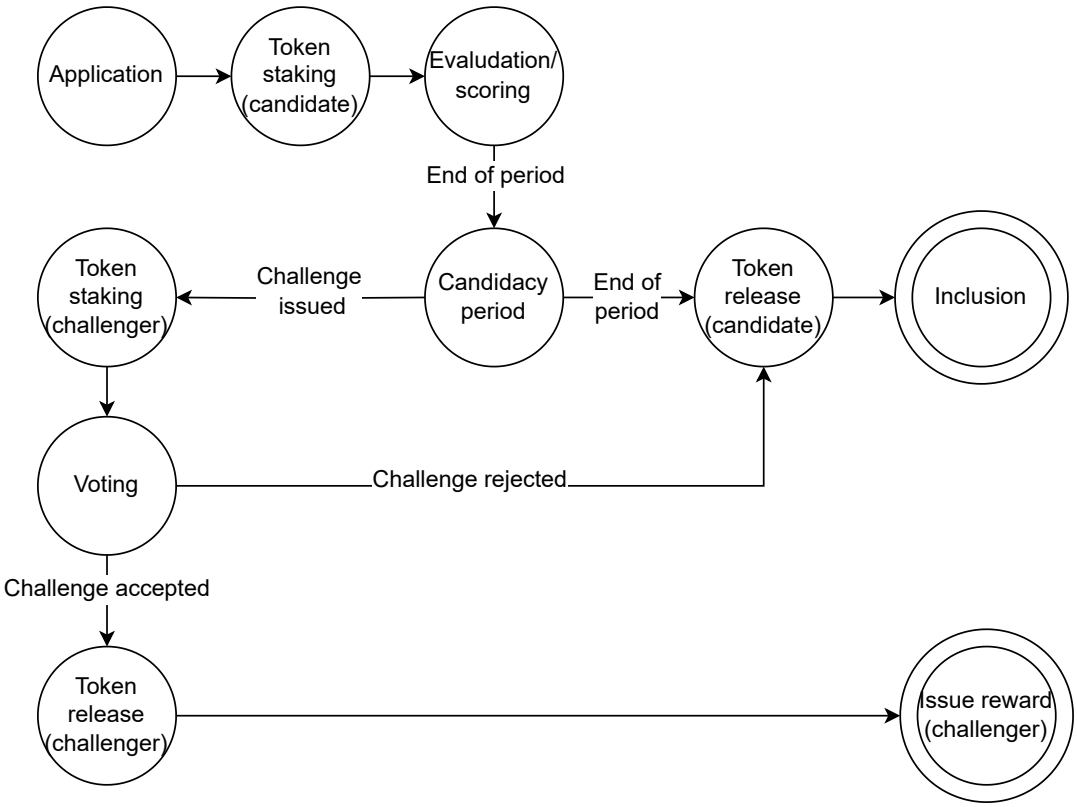

Figure 2.3: States machine of the membership assertion process

If not challenged, the candidate receives the staked tokens back and is included as a leaf, considering the balance of the tree, so UNLs have approximately the same size. If a challenge is issued, the appointed curators vote to determine the outcome. In case of dismissal, the challenger loses their tokens; the candidate receives their tokens back and is included as a leaf. In case of acceptance, the challenger receives their tokens plus a reward and the candidate is not included in any list, losing the stake.

After inclusion, the new listee will continue to be observed by the appointed peers. These peers may change as the tree adapts to the conditions of the network. If the validator starts presenting byzantine behavior, if the guidelines cease to be followed, or in case of complete disconnection, any appointed peer may issue a challenge. Acceptance of the challenge means complete removal of the validator from the DAG, which means that there is no downward movement. A node in a higher position has more to lose than a node present

at a lower level, guaranteeing that highly trusted validators will continue to act honestly on behalf of the ledger.

2.4 Classification

The Classification module serves not only as a necessary step for guaranteeing minimal levels of trust in each UNL but also as a mechanism to reward trustable nodes. The idea is to employ a Layered TCR[65] to incentivize the nodes to behave honestly to be promoted to higher levels, resulting in monetary rewards for curating lower levels.

In its current implementation, the XRPL network employs a *Reliability Measurement* mechanism to identify faulty nodes that should be added to the Negative UNL (nUNL). This measurement comprises the percentage of times a validator issued a validation that is agreed upon by the network to be true, considering the last 256 ledger versions. Each node computes the reliability score for each of its peers using the following equation[70]:

$$Reliability = V_a \div 256$$

Being V_a the total number of validations received from a given peer that a node considers true considering its ledger versions. When the reliability score of a peer falls below 50%, the node can submit a proposal to add the said peer to the nUNL. The network must reach a consensus on adding or removing validators from the nUNL.

Although this mechanism works well for temporary disruptions, it does not permanently remove or add validators to the UNL space; its sole purpose is to tell the network to temporarily ignore *proposals* and *validations* coming from faulty nodes. We propose an extension of the nUNL, in which we use a measurement similar to the reliability score to classify the nodes into layers of trust, with more trustable nodes placed in higher layers.

Movement between layers can only happen upward, to guarantee that nodes that become untrustworthy for a prolonged time are automatically removed from the UNL space, so nodes have an extra incentive to behave in the best interest of the ledger. The upward movement occurs in two instances, first when a node applies to be promoted to higher lev-

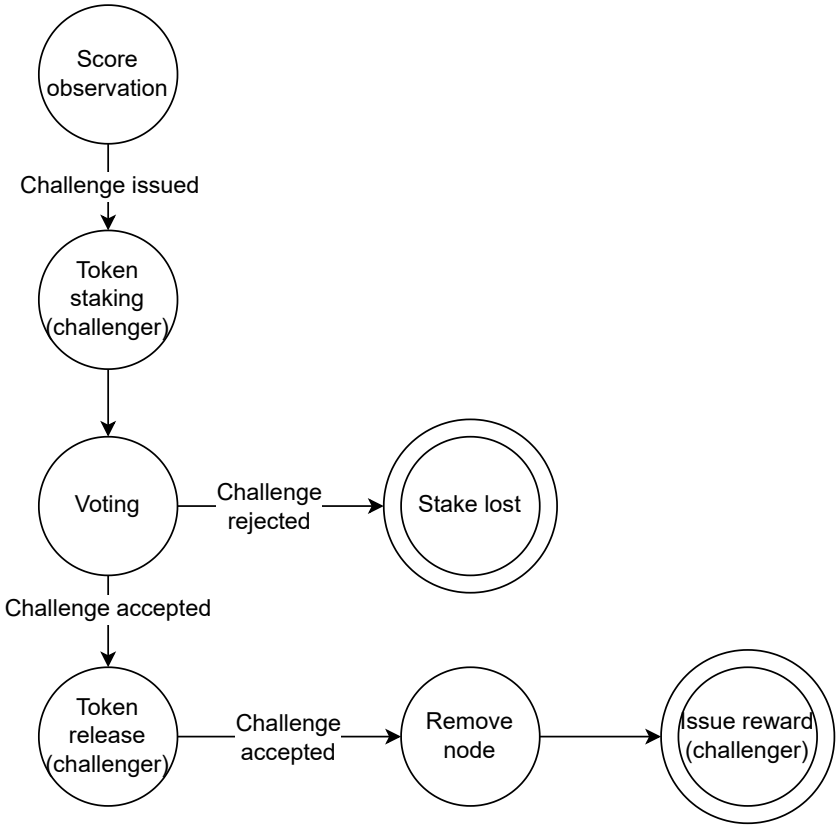

Figure 2.4: States machine of the classification process

els, and second when a high-level node is removed or promoted, and there is a node in the immediate level below that fulfills the minimum score required.

When a node applies for promotion, as shown in Figure 2.4, it must stake tokens within the pool and go through the same process described in the previous section. In the event that another node is removed or promoted, another node can propose a validator that it believes to be the best candidate for promotion. Again, by staking tokens within the pool and going through the curation process. In the same way, if the score of a node falls below the threshold set for the layer it sits on, a node on the levels above can propose its removal by staking tokens and going through the same process.

The reason why nodes are not automatically promoted as soon as they reach the minimum score required for a given layer is to promote rewards for nodes at higher levels to

curate the UNL space. Every proposal to modify the space in any way requires tokens to be staked, and, safe from unchallenged additions, rewards to be issued. This method rewards good curation and monetizes the participation of nodes in the UNL. It also provides incentives for nodes to behave correctly to maintain higher levels, where they can receive rewards for curating lower levels.

The number of levels and minimum score required, as well as a more in-depth study on the most suitable way to score nodes, is not the focus of this work. Further work on this topic is still necessary, considering the underlying characteristics of the XRPL network.

2.5 Grouping

The final module comprises the most important part of the NLAC framework. It is where trustable validators are combined into lists with the goal of better distributing the decision power while maintaining the maximum possible trust in the system. To achieve that, we employ optimization methods to maximize the total trust of the system, considering each layer of trust generated in the previous module.

Figure 2.1 shows the structure that we aim to achieve after optimizing the set. The black circle represents the root of the forest from which all lists derive. In the illustration, we show four lists, each of them rooted in what we represent as black squares. Each trusted validator is represented by a white circle. By traversing each tree, we obtain the UNLs we seek to represent. The entire forest shows how the lists overlap each other to create a tightly connected space while maintaining a certain degree of dispersion.

There are three layers of trust represented, with the higher levels closer to the root. The closer a node is to the leaves, the lower its reliability score. To avoid low-trust nodes being the majority in a list or being in too many lists at once, NLAC limits the maximum number of low-trust nodes in each UNL, meaning that the number of leaf nodes should also be limited, so we can avoid the scenario of low-trust nodes massively skewing the total trust of the system. It is important to note that we use this DAG to visualize how the NLAC space is constructed. This data structure is not directly implemented, being the lists represented by

adjacence matrices.

2.5.1 Mathematical Model

We generated a mathematical model for grouping the validators according to the trust layers fulfilling some conditions modeled as constraints while trying to achieve the maximum total trust in the system. We modeled the structure representing the UNL memberships as the matrix X with

$$x_{ij} = \begin{cases} 1 & \text{if } node_i \in UNL_j \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Thus, the row i represents the node i and the column j represents the UNL j, $i \in \{1, ..., N\}$ and $j \in \{1, ..., M\}$ with N the number of nodes in the UNL space and M the number of UNLs to be generated by the optimizer. Let K be the maximum number of layers generated in the previous module. Consider the trust vector y with $y_i \in \{1, ..., K\}$, where node i belongs to trust level k.

Maximizing the total trust of the system amounts to finding X_{opt} with

$$X_{\text{opt}} = \underset{X \in \{0,1\}^{N \times M}}{\arg\max} ||y^T X||_2 \tag{2.2}$$

To simplify the model, we do not employ the reliability score generated internally by each node for their peers because different nodes may have different scores for each validator. The average score of a node is already reflected in the level they occupy, which is agreed upon by the entire network.

The trivial solution for this objective function is X_{opt} being the matrix of ones. This, however, does not fulfill the dispersion requirement imposed previously. For this reason, we impose the following constraints:

First, we want to enforce a minimum size U for each UNL:

$$||Xe_j||_1 \geq U \qquad \forall j \in \{1, ..., M\} \tag{2.3}$$

with $||\ .\ ||_1$ being the L^1-norm and $e_j \in \mathbb{R}^M$ the j-th basis vector.
We want to guarantee that any two UNLs have a minimum overlap P:

$$(Xe_p)^T Xe_q = e_p^T X^T Xe_q \geq P \quad \forall\ p, q \in \{0, .., M\} \tag{2.4}$$

Considering that all nodes have the same voting power, to guarantee that low trust nodes will not be part of too many UNLs and skew the power balance, we limit the number of UNLs a node can be part of based on the trust level it is part of. Let L_k be the parameter that identifies the maximum number of UNLs a node at the trust level k can be part, with $k \in \{0..K\}$.

$$||e_{j_k}^T X||_1 \leq L_k, \qquad j_k \in \{1, ..., N\} \tag{2.5}$$

Another important limitation is the maximum number of nodes in a given layer a UNL can have. Let Q_k be the maximum number of nodes of trust level k the UNL j may have, then:

$$||v_k^T Xe_j||_1 \leq Q_k \qquad \forall k \in \{1, ..., K\} \tag{2.6}$$

where

$$(v_k)_i = \begin{cases} 1 & \text{if Node } i \text{ belongs to trust level } k \\ 0 & \text{otherwise} \end{cases}$$

2.6 Experimental Evaluation

We tested[2] the above formulation using Pyomo[71][72] with the cplex solver[73] on a Core™ i7-8665U CPU@1.90GHz × 8 machine with 16GB of RAM running Ubuntu 22.04.2 LTS. We fixed the values of $U = N * (1/3)$ and $K = 4$, also distributing the nodes randomly through the layers. The values for the vectors L and Q were $L = [16, 8, 4, 2]$ and $Q = [12, 8, 4, 2]$. These values were empirically obtained on a *testnet* of 24 nodes and 8 UNLs.

Table 2.1 shows the results of six combinations of M, N, and P, considering that these

[2]Code available at `https://github.com/FlavScheidt/NLACOptimization`

Table 2.1: Trust Optimization Evaluation

N	M	P	Total Trust	Constraints	Variables	Time
24	8	80%	280	121	193	0.05s
24	16	90%	280	193	385	0.05s
32	16	80%	436	209	513	0.39s
32	24	90%	436	281	769	4.38s
64	16	80%	712	273	1025	0.55s
64	32	90%	872	417	2049	18.47s

values can change depending on the state of the network. The goal of the optimization is to group N validators into M lists respecting the previous constraints while maximizing the trust of the system. The table also shows how many constraints and variables were generated by the model and the solution time.

The total trust of the system naturally grows as a function of N, however, it does not seem to be affected by M when $N < 64$. The total solution time is also not affected by M when $M < 24$. It is important to note that for $N = 64$ and $M = 32$ the solution time may be longer than is feasible for a distributed solution based on the XRP LCP. However, when $M > 2$ the complex scenario needs a deeper analysis. That being said, our tool provides means for maximizing trust, but some parameters require a more in-depth analysis.

The lower bound for U was kept constant, being experimentally verified to be enough to keep a *testnet* of 24 validators synchronized and alive when no byzantine behavior occurs. The value of P was also constant and determined based on the works presented in Section 1.1.3.

2.7 Discussion

The XRP LCP works on the assumption of collectively trusted sets of validators. The amount of overlap required between these sets to keep the ledger from forking or stalling is not yet a solved question. This characteristic, plus the lack of proper tools for curating and maintaining UNLs, led to the creation of the manually maintained main list. NLAC aims to

solve this issue by taking a broader view of the entire space of UNLs, proposing a framework for automatic curation and self-arrangement, and giving the trust overlay a mechanism for self-maintenance.

NLAC comprises three main modules, all with tunable parameters, leaving space for further improvements as the network evolves. These modules were constructed based on four desirable properties derived from the characteristics of the XRPL trust overlay: *nesting*, *layering*, *autonomy*, and *continuity*. The first module uses TCRs to establish membership in the UNL space, followed by the second module, which uses layered TCRs to create levels of trust. These levels are then used in the last module, where nodes are grouped into different lists, following guidelines to achieve maximum trust while guaranteeing a better dispersion of authority.

Overall, NLAC represents an important step forward in the development of the XRPL. By providing a framework for automatic curation and self-arrangement, NLAC enhances the unique features of the XRP LCP, while addressing key issues related to trust and dispersion of authority, and delineating a mechanism for implementing a rewarding system on the network. With ongoing research and development, NLAC has the potential to become a key tool for building secure, decentralized networks that can scale to meet the needs of a rapidly evolving digital economy.

Chapter 3

Measuring and Improving Message Overhead on the XRP Ledger

Consensus leans on how efficiently the nodes can communicate with each other. Given that poor network performance can facilitate double-spend attacks[42] and even lead to forks or stalls in ledger progress. Flood publishing is a straightforward strategy to propagate messages; however, it causes excessive message overhead, increasing the required bandwidth, and possibly clogging the network with redundant messages. Increased bandwidth can also affect the propagation time of blocks, negatively affecting the number of transactions per second (TPS) that a blockchain can perform. This is an issue that has already been reported in Bitcoin[24] and Ethereum[74].

In the XRPL different dissemination strategies are used for different types of messages. Flood publishing is used to propagate proposals and validations, with any node in the network being able to proclaim itself a validator, and thus flood the network with proposals and validations each consensus round.

To better understand how flooding impacts the scalability and performance of the XRPL network, we present *Flexi-Pipe*, a tool that allows the analysis of targeted message types on an XRPL testnet. Providing also the capability of plugging different dissemination techniques into the rippled validator code. We first used this tool to study the validation dissemination pattern, counting the number of duplicate messages that a single node may receive in a given time frame. We then discuss a mitigation strategy proposed by the XRP Ledger Foundation (XRPLF): squelching. In the following chapters, Flexi-pipe is also used as a tool and

methodology to study the use of publisher/subscriber (pubsub) dissemination in the XRPL, helping also to tune the state-of-the-art dissemination technique in blockchains for the specific requirements of the XRP Ledger Consensus Protocol (XRP LCP).

3.1 Flexi-Pipe

As part of our research, our aim is to isolate message types that create bottlenecks on the network to investigate dissemination techniques that might mitigate those specific points. To evaluate the performance of different dissemination techniques, we created a tool - *Flexi-Pipe*[1] - that allows us to plug different broadcasting mechanisms into the validator code.

In this work, Flexi-pipe is used to assess flood bottlenecks and assess Gossipsub performance, as well as in other studies to investigate different dissemination techniques, such as Named Data Networkss (NDNs)[75]. In this section, we focus on the identification of message exchange patterns, according to the distributed states machine presented in Section 1.1.2. The next chapters explore the use of a pubsub overlay to disseminate messages on the XRPL using the Flexi-Pipe architecture as a methodology for tests and evaluations.

3.1.1 Architecture

The idea behind Flexi-Pipe is to create an overlay where only targeted message types transit to identify dissemination patterns and evaluate different solutions. This means that from the point of view of implementation, we have two layers: the *rippled overlay* and the *dissemination overlay*. The *rippled overlay* is the existing layer in which the validator exchanges messages with its peers and the entire network. In contrast, the *dissemination overlay* is the novelty we add, showing only the targeted messages, such as *validations* and *proposals*.

The communication between both layers is done by Remote Procedure Calls (RPCs). Figure 3.1 shows the schematics used to implement Flexi-Pipe. Each big rectangle represents a node, inside of them we have the two layers: rippled and dissemination.

The bottom layer is the version 1.7 rippled code, written in C++, and comprises the

[1] Available at `https://github.com/FlavScheidt/flexi-pipe`

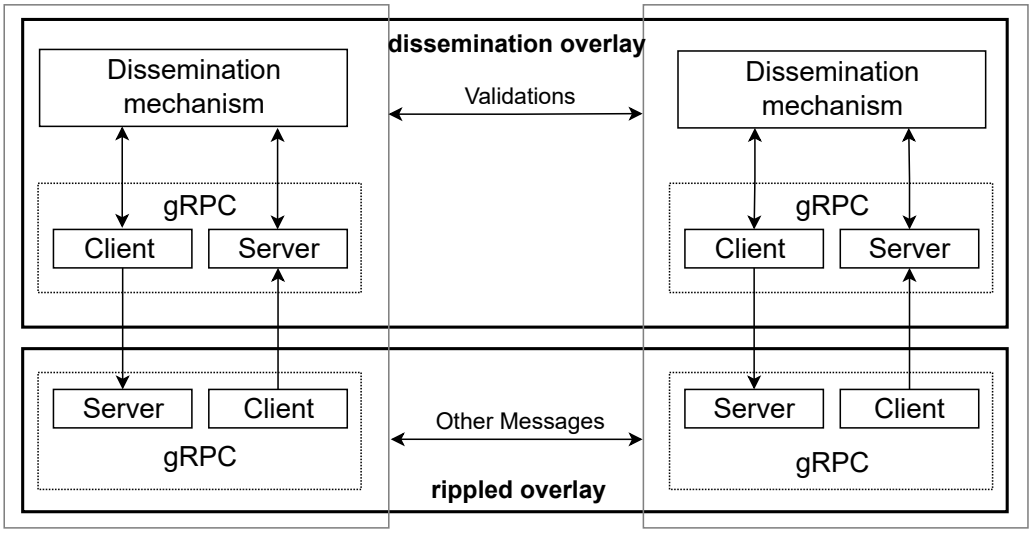

Figure 3.1: Architecture of Flexi-Pipe showing the two overlays: rippled and dissemination

rippled overlay. We changed the code to include a gRPC[76] node that contains a client and a server. We also deactivated the node's ability to send and relay messages of a certain type - namely, proposals and/or validations - but kept all other functions intact. Therefore, all other types of messages still transit at this level, adding also a new logging feature to the rippled code[2]. The goal of this feature is to provide meanings for better analyzing - and debugging - the integration of the two overlays.

The *dissemination overlay* is the top layer containing the broadcast mechanism connected to the validator. This layer contains a component that controls the creation and maintenance of the overlay. Flexi-pipe is agnostic, meaning that any mechanism can be plugged and serve as a dissemination overlay. Chapter 4 explores the integration between rippled and GossipSub, while other works[75] also use Flexi-pipe to integrate NDNs into the XRPL.

[2]Modified code available at `https://github.com/FlavScheidt/sntrippled`

3.2 Testbed

The experimental setup used throughout this thesis encompasses a cluster of 24 virtual machines with 31.25GiB of RAM, 64GB of disk, and 4 sockets with 2 cores each, running Ubuntu 20.04.4 LTS. We used two versions of rippled, one modified to only write logs for analytics - which we call vanilla - and another modified to work with Flexi-pipe. We also use two control nodes with similar configurations running instances of the Flexi-Pipe maestro.

Table 3.1: UNL structures

	UNLs	Avg UNL size	Mean UNL size	Min UNL size	Max UNL size
1	1	24	24	24	24
2	24	16,25	16	15	18
3	8	16,37	16	16	18

We considered three structures for the trust overlay which are summarized in Table 3.1. In the first one, we connected all nodes in a complete graph to emulate the current structure of the core of the XRPL network. The second category mirrors the way the XRP LCP was first conceptualized[2], with every node choosing its own Unique Nodes List (UNL). However, instead of relying on a minimum overlap, we created every UNL so that each node would listen to at least 60% of the network. This percentage was obtained experimentally and was shown to be enough to stabilize the network, considering the absence of malicious nodes. The third structure considers the UNL overlap issue and explores the pubsub characteristics, simulating a scenario where preset UNLs exist, created by applying a method similar to the one described in Chapter 2. This structure was created using NLAC and comprises 8 UNLs that were randomly assigned to the 24 nodes so that each UNL was assigned to at least 22 nodes, with the average overlap between every two nodes being 75%.

These three structures are used extensively in the next chapters to evaluate the XRPL behavior within different trust overlay structures. Partial disruptions are expected, but the assumptions do not include malicious activities. In this chapter, we analyze solely the first structure to better understand how the message overhead issue affects the current state

of the XRPL network. The previous chapter presented a tool that may enable the XRPL to relax its minimum overlap requirement by increasing the total trust of the system; Next chapters take advantage of this to analyze different scenarios that may improve the liveness and performance of the XRPL.

3.3 Measuring Message Overhead

The states machine presented in Section 1.1.2 highlights the two states in which messages are propagated by flooding: *Proposal Transmission & Reception* and *Close ledger & Broadcast Validation*, both of them represented by dashed lines. The idea is that transactions are spread through *gossiping*[77], without the need for a strong assurance of delivery, since they will later be grouped into proposals. The use of flooding to disseminate proposals works as a reconciliation phase, guaranteeing that all nodes will see all transactions at least once.

The problem is that not all nodes are actively considering the positions of all validators in the network, but all of them are receiving proposals and validations repeatedly from these nodes. A validator only considers the position of the nodes present on their UNL, but flooding dictates that a node must always send messages to all of their peers, those peers will also relay the messages through all of their connections, causing exponential growth on the number of messages with the growth of connections between nodes. In a highly connected network, such as the XRPL, excessive traffic caused by message overhead leads to higher latency and higher bandwidth usage of individual nodes.

Tsipenyuk et al.[78] first introduced this problem by presenting how the number of proposals and validations accounts for 72% of all messages. The proposed solution - called *Squelching* - focuses on diminishing the number of vertices in the dissemination graph. Each node selects a subset from which it chooses to listen regarding a particular validator and tells the other nodes to squelch the connection for a given amount of time.

Squelching works by allocating slots to manage the relaying of proposals and validations on the network. Each node maintains a list of validators from which it is relaying messages. When a second node, directly connected to the first one receives enough copies of a given

message, it tells the first node to squelch the relaying for the source, meaning that the relay should be disabled for a given period of time. In the default configuration, a node keeps 5 connections for each source.

The authors evaluated this method using a simulated graph structure that shows a reduction of 76% in the total number of messages on the network. Subsequent work[14] evaluated the impact of the technique on the connectivity of a single node on the mainnet, also measuring the bandwidth in messages per second. This study concludes that squeaking presented a 28.9% improvement in bandwidth over flood publishing. To date, there has been no work addressing the robustness and safety of the XRP LCP in the presence of squelching.

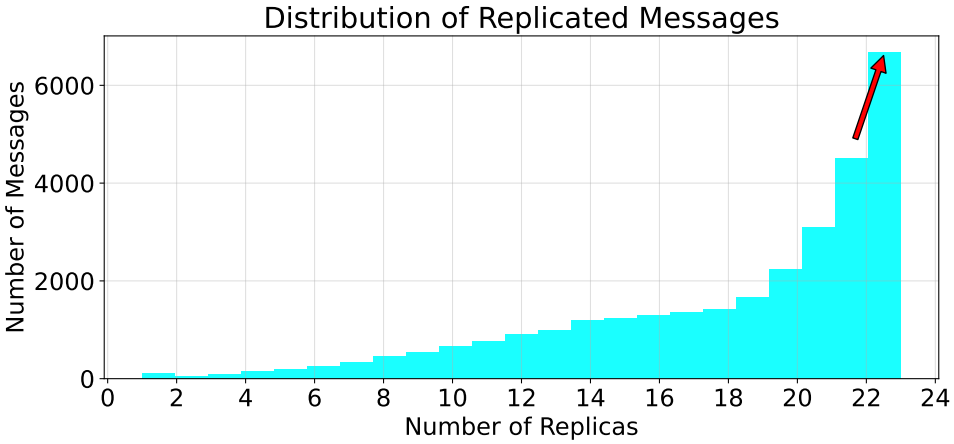

Figure 3.2: Frequency of duplicated messages on a testnet with 24 nodes fully connected without enabling Squelching

We used Flexi-pipe to measure message overhead from a different angle by counting the frequency of reception of replicated validations on a single node. We connected 24 validators using vanilla rippled without and with squelching enabled. The nodes form a complete graph to emulate the current state of the XRPL, in which 34 nodes comprise the core of the network. This arrangement helps us isolate the problem. However, it is important to note that in the mainnet the problem can be ever bigger, considering that several nodes declare themselves as validators but are not part of the main UNL. These nodes broadcast their proposals and validations even if they are not actively participating in the consensus process.

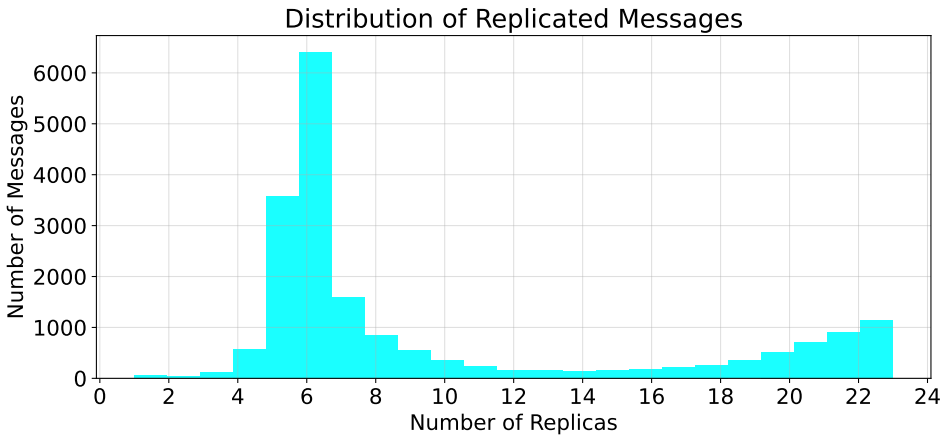

Figure 3.3: Frequency of duplicated messages on a testnet with 24 nodes fully connected with Squelching enabled

Figure 3.2 shows the results obtained without enabling squelching. The x-axis represents every possible number of replicas[3] that a node can receive for a single message. Since the node analyzed has 23 connections, the minimum number for a replica is 1 and the maximum is 23. The y-axis presents the number of single messages received, which means that around 6000 messages were received 23 times, as indicated by the red arrow. In terms of enabling squelching, Figure 3.3 shows a higher concentration of messages in the area between 4 and 8 replicas, which means that the node received most of the messages replicated between 4 and 8 times. This concentration is due to the default setting of squelching, which maintains at least five connections for each source.

What we take from this experiment is that flooding is a reliable method for disseminating messages but causes the distribution of the number of replicas to be far from ideal. There needs to be a trade-off between the delivery assurance and the number of replicas to guarantee the scalability of the network. Squelching presents a good solution for the dissemination of messages. However, more efficient methods have been studied[75][68], including GossipSub, which has been introduced in Section 1.2. The next Chapter explores the use of GossipSub to disseminate proposals in the XRPL network, exploring the pubsub characteristics of the XRP LCP to diminish the message overhead on the network.

[3]A replica is a message that is received more than once, relayed from different peers.

Chapter 4

PubSub Dissemination on the XRP Ledger

4.1 Introduction

Message propagation is a crucial aspect in a blockchain. Without fast and reliable propagation of messages and blocks, the entire network becomes vulnerable to several types of attacks. Flood publishing is the most straightforward method to guarantee the delivery of messages, relying on redundancy. It is used on the Bitcoin blockchain as well as on the XRPL[40]. However, it is not scalable, since the number of redundant messages grows exponentially when new nodes join the network.

In the previous chapter, we presented the message overhead issue, also showing the solution proposed by the XRPLF: squelching. Squelching seems to experimentally be a good solution for the problem; however, another answer could lie in the use of publisher/subscriber (pubsub) dissemination. At a high level, these systems are composed of, as the name suggests, *publishers* and *subscribers*, being the first role played by the nodes that declare interest in a given topic and the second, the nodes that publish messages on these topics. In pubsub systems, messages are also called *events*, and the declaration of interest on a topic a *subscription*[45].

Pubsub systems are difficult to implement in unstructured peer-to-peer (p2p) networks. The most notable work on this matter is *GossipSub*, originally proposed to be incorporated into FileCoin and Ethereum[48], and although both could be considered structured due to the use of Distributed Hash Tables (DHTs), in reality they behave as unstructured p2p networks, using DHTs solely as a peer discovery mechanism[79]. GossipSub is a gossip-based

pubsub protocol for message dissemination in p2p overlays[46] and is distributed as an extensible component within libp2p[47].

Gossipsub has two main characteristics that make it highly suitable for the XRPL, as they guarantee fast and lighter dissemination while being attack-resistant. First is the way it constructs the mesh in which messages transit and second is the employment of a scoring function to help identify byzantine behavior.

The mesh construction employs *eager push*, keeping the fan-out low to balance bandwidth usage, also maintaining strong assurance of delivery by employing *lazy pull*. For the two models to work properly, the protocol specifies the creation of two meshes; the first, used for eager push, is a full message logic network specified as a *local mesh*. Each node has its own local mesh formed by bidirectional links to nodes subscribed to the same topic. The second is the *global mesh*, where nodes exchange meta-data solely with nodes inside and outside their local mesh using gossiping[80].

GossipSub employs a scoring function to help identify byzantine behavior. The scoring is local, which means that every node maintains an internal score of its peers and makes routing and relay decisions based on these scores[46]. However, the scoring function is beyond the scope of this work.

More detailed analysis of attack resilience can be found in Vyzovitis et al.[46], and a performance benchmark in the Whiteblock study[58][57]. In the next section, we discuss how we mapped the XRP Ledger Consensus Protocol (XRP LCP) mechanism to GossipSub, explaining our design choices.

4.2 Proposal

GossipSub is the state-of-the-art solution for efficient message dissemination on blockchains. In addition to being used by The Interplanetary File System (IPFS), it is also used by Filecoin and was deployed on Ethereum in September 2022[81]. Both systems employ the framework similarly; Filecoin uses two topics: one to propagate messages and another to propagate blocks. Ethereum structure the topics in a more sophisticated manner, with five

global topics, two primary, and three secondary[82].

The use of Unique Nodes Lists (UNLs) in the XRP LCP allows some interesting forms to use pubsub to disseminate messages. We present three set-ups based on how the network is structured in production and how it has been conceptualized. We propose to look at the XRPL as a pubsub system, creating a *pubsub overlay* based on the existing *trust overlay*. Thus, we can keep the XRP LCP algorithm as is, changing only the abstract layer that creates trust relationships between nodes.

First, we use GossipSub with 2 topics, in a similar way to Filecoin and Ethereum, keeping the current XRPL network structure. The idea is to have two topics according to the types of messages that must be transmitted across the network; the first topic refers to *proposals* and the second to *validations*. Creating a parallel view with Ethereum, *validations* are similar to a *signed block*, while *proposals* can work as *aggregated attestations*. This approach keeps the trust overlay as it is, with a fully connected core of validators, while reducing the amount of replicated messages and thus making the structure more scalable.

However, the XRP LCP has some characteristics that allow for more sophisticated topic arrangements. In particular, the way nodes select sets of validators to trust works naturally as a pubsub system. That is, we can understand an UNL as a list of topics to which a node listens. Each validator is then abstracted directly as a topic, replicating the structure of the *trust overlay*. In this configuration, called *1 topic per validator* (1-topic/validator), it does not make much sense to keep a fully connected mesh of validators, as it would create a similar structure as seen in the 2-topic approach.

The previous approach may cause some problems, as the XRP LCP has a condition to keep liveness related to the minimum required UNLs overlap. Considering that all nodes on the network can choose their own UNL, not all can agree on a minimum number of common nodes. Under this condition, the ledger would not be able to make any progress, as proved in the previous works discussed in Section 1.1.5. We can find another solution that maintains the liveness of the network while also maintaining the pubsub characteristics of the XRP LCP. This solution is called *1 topic per UNL*(1-topic/UNL) and involves having several predefined UNLs from which nodes can choose and structuring the *pubsub overlay*

to abstract each UNL as a topic.

The 1-topic/UNL solution is a hybrid of the two setups presented above. While the 2-topic solution maintains liveness, it also keeps the pubsub characteristics of the XRP LCP from working. Similarly, the 1-topic/validator keeps the XRP LCP characteristics but is a weak solution that allows a break in liveness. By subscribing to predefined UNLs, the nodes ensure that there is a minimum overlap, allowing the ledger to progress. In our proposal, this structure appears to be the most suitable, blending the characteristics of the current XRPL *trust overlay* infrastructure with the concept of the XRP LCP.

4.3 Methodology

To validate our proposition, we plugged GossipSub into the rippled validator using Flexi-pipe, which was presented in Chapter 3. Similarly to the previous analysis, we chose to study how *validations* are exchanged on the trust overlay. We opted for this approach to simplify our analysis, considering that validations are exchanged only once per consensus round, while *proposals* are exchanged several times during the same round, thus generating a heavier workload for the pubsub overlay.

As shown in Figure 4.1, we use GossipSub as a module inside libp2p[1]. The reason is that GossipSub acts as a router for messages using pubsub and does not have the means to create and maintain an overlay. It is possible to use GossipSub directly inside the validators without the overhead of maintaining a second overlay. However, this work aims to evaluate how GossipSub can mitigate the message overhead and does not intend to change the rippled code for a production environment. The libp2p and GossipSub components[2] used are the reference implementations, written in go without modifications. These were the implementations chosen because they are better documented and reported as the most stable[83].

We separated the experiments into two categories: First, we considered the 2-topic approach using a fully connected structure for the *trust overlay*. However, to simplify our anal-

[1]Available at `https://github.com/libp2p/go-libp2p`
[2]Module available at `https://github.com/FlavScheidt/gossipGoSnt`

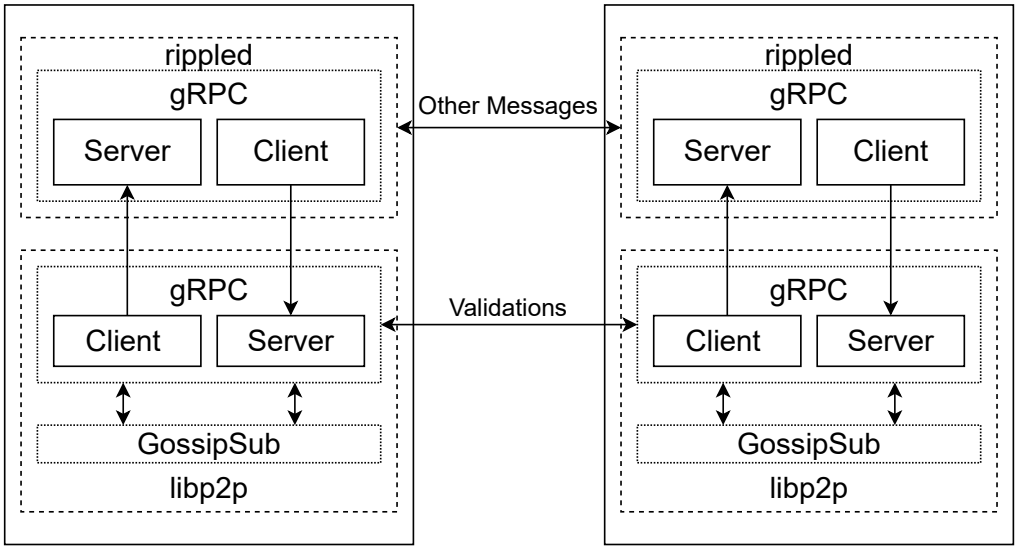

Figure 4.1: Diagram of the communication between two nodes using GossipSub plugged through gRPC to disseminate validations.

ysis we considered only validations, thus making it a *1-topic* approach. We compared the results against vanilla rippled with the same structure and also against vanilla with squelching enabled, using the default configuration.

The second category compares the 1-topic/validator and 1-topic/UNL against vanilla and squelching. In this setup, 1-topic/validator, vanilla, and squelching use the same *trust overlay* structure, with the first replicating this structure into its *pubsub overlay*. The *trust overlay* was generated based on the framework presented in Chapter 2 and comprises a randomly generated graph with a median degree of 16 and a maximum degree of 18 so that every node listens to at least 60% of the network. We built this structure in this form to maintain the liveness property. We kept the same characteristics for the 1-topic/UNL experiments, generating 8 UNLs randomly, each list containing at least 16 nodes.

4.4 Evaluation

In this chapter, we focus on minimizing the overhead generated by the message dissemination strategy used on the XRPL. To evaluate our proposal, we analyzed the number of

replicated validations received by one particular node.

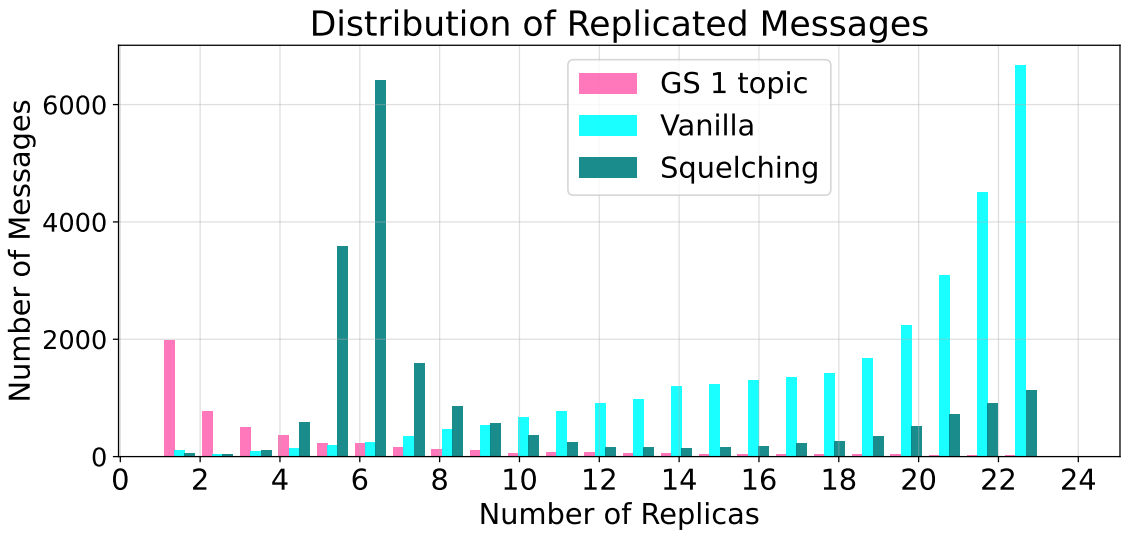

Figure 4.2: Comparison of the frequency of replicated messages on vanilla rippled in a fully connected structure and GossipSub using a 2-topics setup

Figures 4.2 and 4.3 present the amount of replicated messages received by a node during 30 minutes of synchronized execution. This amount of time was chosen to give the nodes enough time to run at least 600 consensus rounds. Both graphs should be read similarly to the one presented in Figure 3.2. The x-axis presents the number of duplicates, meaning the number of times the node received the same message. Meanwhile, the y-axis shows the number of times the node has received a message. And so the graphs say that the node received *y messages x times*. As an example, in Figure 4.2, in vanilla rippled, the node received 7000 different validations 24 times.

The optimal behavior regarding performance is to have no replication, with all messages being received only once. However, this pattern causes reliability issues in the presence of byzantine behavior or network failures, so we tolerate a certain amount of replication.

In both setups, vanilla presents a behavior that is suboptimal due to its conservative approach - especially in the fully connected structure of Figure 4.2. Vanilla with squelching enabled shows a better tendency, accumulating most of the distribution around 5 to 6 replicas, which is expected, considering that each node always keeps 5 connections for each

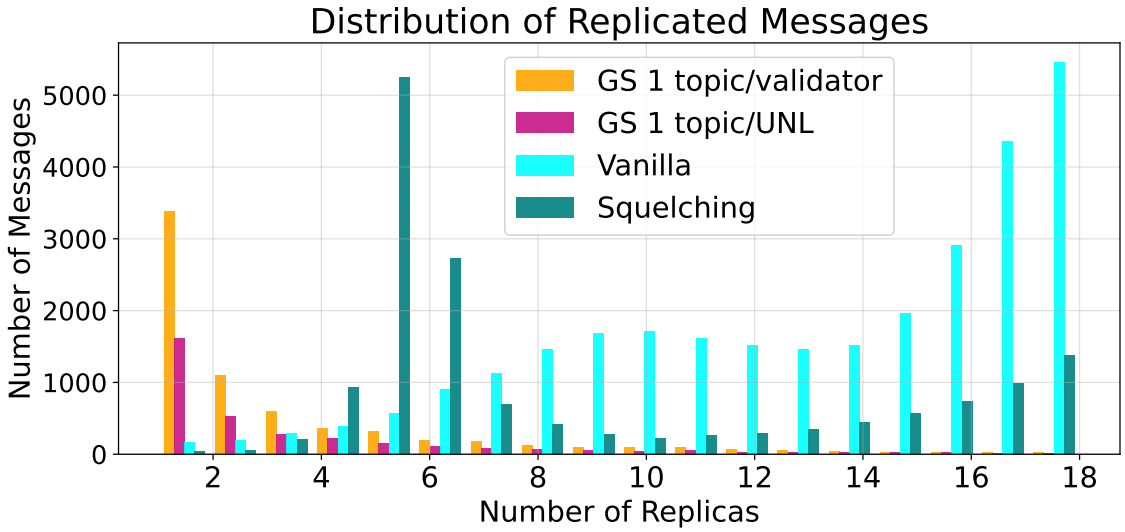

Figure 4.3: Comparison of the frequency of replicated messages on vanilla rippled in a 16-degree structure and GossipSub using a 1-topic/validator and 1-topic/UNL setups

node on the UNL at each squelching round. The behavior is better than pure vanilla, but it is still sub-optimal when compared to the results obtained with GossipSub.

The 1-topic approach suggests a tendency closer to the optimal, but the curve still has a more even distribution than the one presented by the 1-topic/validator setup in Figure 4.3, being this the curve that gets closer to the optimal. The 1-topic/UNL has a similar tendency, demonstrating a smaller number of total messages compared to its counterpart. This is a consequence not of the setup itself, but of how Flexi-pipe works. Due to the requirement of the rippled validator for bilateral connections and because not all trust relations in this scenario are bilateral, the node ends up connecting to peers it does not trust, but that subscribe to it. Flexi-pipe does not listen to most of those peers at the GossipSub level, so the consensus rounds take longer since the process considers the number of connected peers. This limitation would not exist with GossipSub being implemented directly inside of the validator.

We conclude that the use of GossipSub to disseminate validations on the XRPL network would be beneficial for the general performance. The overhead caused by flooding decreased, while the ledger continued to progress successfully. From the experiments, the most beneficial setup was the one in which the pubsub characteristics were enhanced by

the use of the structure generate by NLAC. The scenario that more closely simulates the actual structure of the XRPL network presented a good improvement, showing that GossipSub may be a good alternative for improving performance.

4.5 Discussion

Leveraging the inherent pubsub properties of the XRPL, we integrated GossipSub into the validator through Flexi-Pipe. GossipSub is the state-of-the-art for message dissemination on blockchains. The results presented by the experiments show that we correctly identified the bottlenecks and that GossipSub mitigated the message overhead in the network, thus improving the scalability of the XRP LCP.

However, this work does not take into account safety considerations, and further analysis of threats in the proposed scenarios is necessary. Considerations about UNL overlap in more complex scenarios are also of importance for the continuity of this work. The second proposed scenario is straightforwardly a pubsub network, presenting the best results regarding message overhead.

The next chapters seek to analyze how GossipSub can be parameterized to better suit the characteristics of the XRPL. Having successfully integrated GossipSub as an overlay for transmitting certain types of message, and having proved that GossipSub can improve message overhead, we can now analyze how validations are exchanged, and which GossipSub parameters can be better tuned for optimal performance and increased scalability.

Part II

Tunning GossipSub Parameters for Increased Scalability and Performance

Chapter 5

Dimensional Analysis of GossipSub over The XRP Ledger

5.1 Introduction

The previous chapter explored the use of GossipSub as a dissemination layer for proposals and validations in the XRPL. In this chapter, we dive deeper into the GossipSub mesh construction and management, presenting a dimensional analysis to understand the behavior of GossipSub according to different parameterizations. To support our analysis, we use data science techniques to group sets of parameters into clusters according to their behavior, find correlations between dimensions and, finally, generate a decision tree to support the identification of sets of parameters according to the desired behavior and performance of the network.

The clustering of sets of parameters, according to performance and behavior, and feature correlation heatmap were the tools used to draw several remarks about the impact of parameterization and the correlation between dimensions. These remarks, together with a decision tree based on parameterization, are crucial to better understand how GossipSub can be configured to different types of systems, with different requirements.

The GossipSub protocol has been implemented and used in practice by popular distributed systems such as Ethereum, Filecoin, and the The Interplanetary File System (IPFS) peer-to-peer (p2p) network. Although Ethereum and IPFS have been widely studied in general, to the best of our knowledge, the underlying GossipSub protocol has not been widely

addressed by academic research. GossipSub was first presented by Dimitris Vyzovitis and Yiannis Psaras from Protocol Labs in 2019[48]. This work establishes the basis for this new content-based publisher/subscriber (pubsub) protocol. GossipSub aims to enhance pubsub dissemination while minimizing bandwidth usage and avoiding peer overload. To achieve this, the authors propose a bounding between the degree of each peer and a global control of the amplification factor. *Graft* and *prune* messages are used to control a given mesh link according to a periodic stabilization algorithm using predefined parameters, whose functions are explained in Section 1.2.2.

5.2 Methodology

For this work, we considered two topologies for the XRPL trust overlay. The first topology comprises a fully connected set of nodes that mimics the current core of the main XRPL network. In this topology, we have a unique topic for all proposals, similar to how Ethereum exchanges blocks. The second involves a predetermined set of Unique Nodes Lists (UNLs) that each node can choose from. Each UNL is represented by a topic in the GossipSub overlay.

As discussed in Chapter 1, there are two instances in which the XRP Ledger Consensus Protocol (XRP LCP) uses flood publishing: to disseminate ledger proposals and to disseminate validations at the end of the consensus round. In this evaluation, we target proposals, since they may be exchanged several times during a round, whereas validations are exchanged solely at the end of the round. Each round of the XRP LCP takes on average 3 seconds, which means that a new version of the ledger is published on average every 3 seconds[84]. To simulate a heavier workload, Flexi-pipe can also shoot transactions every 2 seconds to each node in the network, performing simple random payments between two accounts. The XRPL mainnet performs an average of 50 transactions per ledger version[1].

We used two different structures of topics: one with 8 different UNLs (called *unl* on our analysis) and a core of validators connected as a complete graph (called *general*). The

[1] As of October 2023, according to `https://xrpscan.com/metrics`

first configuration represents the ideal structure for the XRPL network according to how the XRP LCP has been previously formalized and is also the instance in which the pubsub characteristics of the consensus protocol are more evident. The second reflects the current state of the XRPL mainnet and also mimics how blocks are disseminated on Ethereum.

Table 5.1: GossipSub Mesh Parameters with Default Values

Parameter	Concept	Default	Eth2
D	Desired number of peers[a]	6	8
Dlo	Minimum number of peers[a]	5	6
Dhi	Maximum number of peers[a]	12	12
Dscore	Number of high-scored peers to keep when prunning	4	4
Dout	Number of outbound connections to keep when prunning[a]	2	2
Gossip Factor	Factor (%) of how many peers to emit gossip[b]	0.25	0.25
Dlazy	Minimum number of peers to emit gossip[b]	6	8
Interval	Heartbeat for prune, graft, and gossiping events	1s	1s

[a]For each subscribed topic
[b]For connections outside of the meshes of the topics

Table 5.1 shows the main mesh parameters with which we experimented, also showing the default values and the values currently used by Ethereum. Each parameter has been explained in Section 1.2.2. For each set of parameters, we performed three tests of 30 minutes, so nodes would have sufficient time to sync and perform enough consensus rounds. We gathered data from the trace of events generated by GossipSub and the consensus report from the rippled logs, disregarding a 7.5-minute warm-up and a 7.5-minute cool-down period, and also ignoring faulty executions. We consider faulty executions the ones in which the number of events falls below a z-score of $0,15 * standard\,deviation$.

Table 5.2: Parameters testing ranges

parameter	minimum	maximum
D	6	20
Dlo	4	16
Dhi	8	24
Dscore	2	16
Dlazy	2	16
Dout	2	8
gossipFactor	0.25	0.5
Interval	0.25	3

We tested 14 sets of parameters for each topology, totaling 28 sets of experiments. The reference set used the default values specified for Ethereum. Eight sets had their values chosen by assumptions made considering the underlying characteristics of the XRP LCP. The remaining five had values randomly chosen, still observing the constraints set by the definition of each parameter[46]. The values were selected according to the configuration of 24 nodes and 8 UNLs and the ranges used are shown in Table 5.2.

5.2.1 Data science methodology

In order to facilitate the evaluation of our experiments, we employ data science methodologies to group the outcomes and dive deeper into the effects of parameterization. Our approach consists of three stages (Figure 5.1): feature engineering, unsupervised learning (clustering), and supervised learning (classification), which is used for explainability.

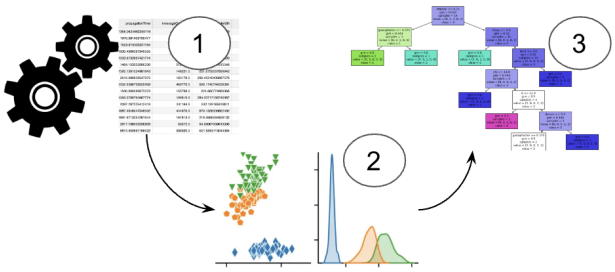

Figure 5.1: Step 1: Feature engineering; Step 2: Clustering; Step 3: Decision tree and explainability

Feature engineering prepares the experimental data for analysis. We first aggregate

the relevant data from each experiment (bandwidth, propagation time, overhead, number of events prune/graft/iwant/ihave/message received/message duplicate) and subsequently apply standardization preprocessing. This preprocessing adjusts the data values so that they have a mean of 0 and a standard deviation of 1. Standardization is especially important when employing clustering algorithms, as these algorithms are based on distance, and unprocessed data with extensive ranges can introduce unwanted distortions to the outcomes.

When aggregating the data, we use two clustering algorithms – namely K-Means (KM)[85] and Agglomerative Clustering (AC)[86] – to categorize data points with similar characteristics and uncover intrinsic patterns. These algorithms partition the data into groups, assigning a specific label to the elements of each cluster, which are fundamentally groups of data points that possess mutual attributes in *n* dimensions. In our evaluation, we consider nine dimensions linked to the aggregated data as previously stated. The selection of these two algorithms was based on their distinct approaches to the clustering problem. KM presumes that clusters are spherical and uniformly sized, whereas AC is equipped to handle non-spherical clusters of diverse sizes. The final clustering labels are determined by a voting mechanism among these two clustering algorithms. In summary, for this analysis, we empirically set the number of clusters to four for both algorithms and, after the voting phase (Table 5.3) we obtained five final clusters.

Table 5.3: Clusters voting phase

Final clusters	**0**	**1**	**2**	**3**	**4**
KM clusters	0	1	1	2	3
AC clusters	2	0	3	1	0

Following the clustering process, a decision tree is used to facilitate the interpretation and extraction of valuable knowledge. Decision trees create a hierarchy of decision rules based on data attributes and are beneficial for probing relationships between elements from distinct classes by offering easily interpretable rules. For our evaluation, we adjusted a decision tree using the configuration parameters of each experiment and the labels provided by the clustering algorithms. It is crucial to note that the decision tree does not consider

aggregated data from experiments, unlike the clustering algorithms. The idea behind this approach is to discover the relationships of the clusters created using the aggregated data and the parameters used in each experiment. Therefore, the output of the decision tree is the correlation between the parameterization and the performance/behavior of each experiment.

5.3 Evaluation

5.3.1 Impact on the Consensus

The first analysis we perform is to determine how the different sets of parameters can affect the convergence time for the consensus rounds. According to the closing times for each new ledger version, gathered from the rippled logs, all the sets show an average of 3 seconds for closing, with a standard deviation of 0.01. The average closing time for each new ledger version on the mainnet is also 3 seconds. Therefore, we conclude that the parameterization of the GossipSub mesh has no impact on the average convergence time of the XRP LCP.

5.3.2 The Dimensions

By applying the clustering algorithms, we found a configuration of 5 groups of sets of parameters[2]. Figures 5.2 to 5.8 show the distribution of each dimension for each cluster. *MessageOverhead* refers to how many duplicate messages traversed the network and was acquired by counting the number of times each message was received by each node on the network. *Bandwidth* is the bandwidth consumed measured in messages per second, obtained by counting the number of messages each node receives divided by the total execution time, considering full messages and gossiping. *PropagationTime* is expressed in milliseconds and measures the average time it takes for a message to reach the destination. *MessageReceived* accounts for the number of full messages received by all nodes. *Graft* and *Prune* represent the number of times these events occurred in the network. And finally, *iwant* and *ihave* are the number of ihave and iwant messages that transited through the

[2]Complete experiment data available at `https://github.com/FlavScheidt/flexi-pipe/blob/main/data/DimensionalAnalysis/CorrelationAnalysis/votingCuster_KM-AC-B.xlsx`

network.

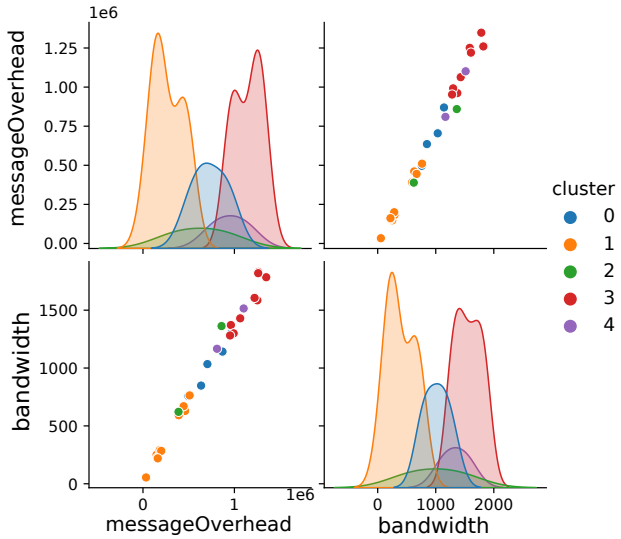

Figure 5.2: Correlation between Bandwidth (in messages/second) and Message Overhead

We suppressed from the graphs the dimension covering the number of messages received more than once (*DuplicatedMessage*). This metric is important because message duplication ensures delivery in adversary scenarios. This suppression is due to the fact that the correlation between *MessageReceived* and *DuplicatedMessage* is 1, as shown in Figure 5.9, which means that all messages on the network have at least one duplicate. This metric is not the same as the message overhead, since the *MessageOverhead* dimension accounts for how many replicas of messages transited on the network.

MessageOverhead and *Bandwidth* are represented together in Figure 5.2 because they also have a strong correlation between them, 0.99 according to the correlation heatmap on Figure 5.9. The correlation between the two dimensions was previously discussed by Vyzovitis et al.[46] and is used in this work as a metric of the accuracy of the methodology.

As stated above, some pairs of metrics have a strong correlation. Figure 5.9 shows the correlation matrix as a heatmap. It is interesting to see that *prune* and *graft* are also strongly correlated; this may be due to the fact that when pruning occurs, only *Dout* connections are kept and therefore a certain number of grafts may be necessary to maintain the number of connections between *Dhi* and *Dlo*. It is important to remember that we do not consider

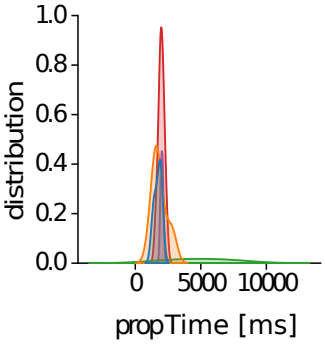

Figure 5.3: Propagation time in milliseconds

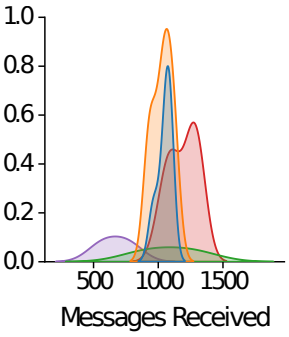

Figure 5.4: Number of messages received

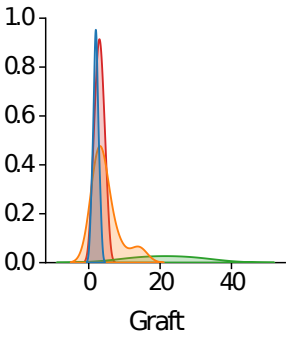

Figure 5.5: Number of graft events

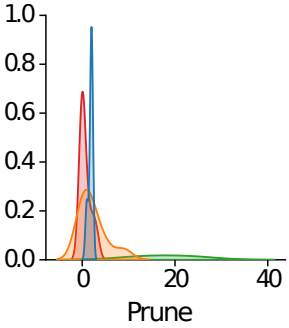

Figure 5.6: Number of Prune events

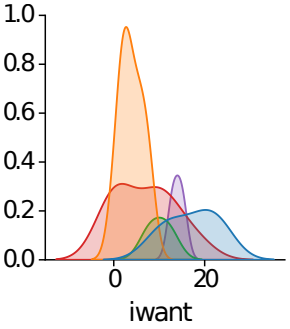

Figure 5.7: Number of iwant messages

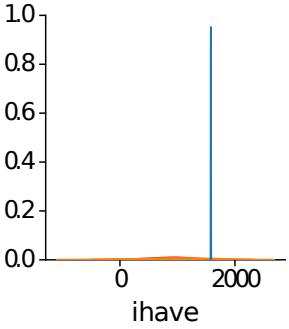

Figure 5.8: Number of ihave messages

the bootstrap of the network for our analysis, so all the initial grafting events are not taken into account. There is also a relatively high correlation between these two events and the *PropagationTime*, which means that the rearrangement of the mesh impacts this dimension but curiously has no impact on *Bandwidth* and *MessageOverhead*.

The dimensions that appear to be more correlated with *MessageOverhead* and *Bandwidth* are *iwant*, *DuplicatedMessage* and *MessageReceived*. Although the correlations are not strong, it is natural that the number of messages circulating in the network will affect the *Bandwidth*, but in the end, it can also depend on the mesh topology.

5.3.3 The Clusters

Using Figures 5.2 to 5.8 as references, we can analyze the characteristics of each group in relation to the nine dimensions discussed in Section 5.3.2. Each graph shows the distribution

of sets for each range of values, with the clusters being represented by colors, the legend of which can be seen in Figure 5.2 and in Table 5.4, which also shows the size of each cluster. We begin by analyzing the characteristics of each cluster and then present the decision tree for the parameters based on these clusters, which may be used as a tool to select suitable sets of parameters depending on the desired behavior of the network.

Table 5.4: Clusters Sizes and Labels

cluster	**0**	**1**	**2**	**3**	**4**
size	5	10	2	9	2
color	blue	orange	green	red	purple

Cluster 0 (blue) shows a decent *PropagationTime*, with a quantity of *MessagesReceived* and a *Bandwidth* in the middle range. It is interesting to see that it has by far the highest

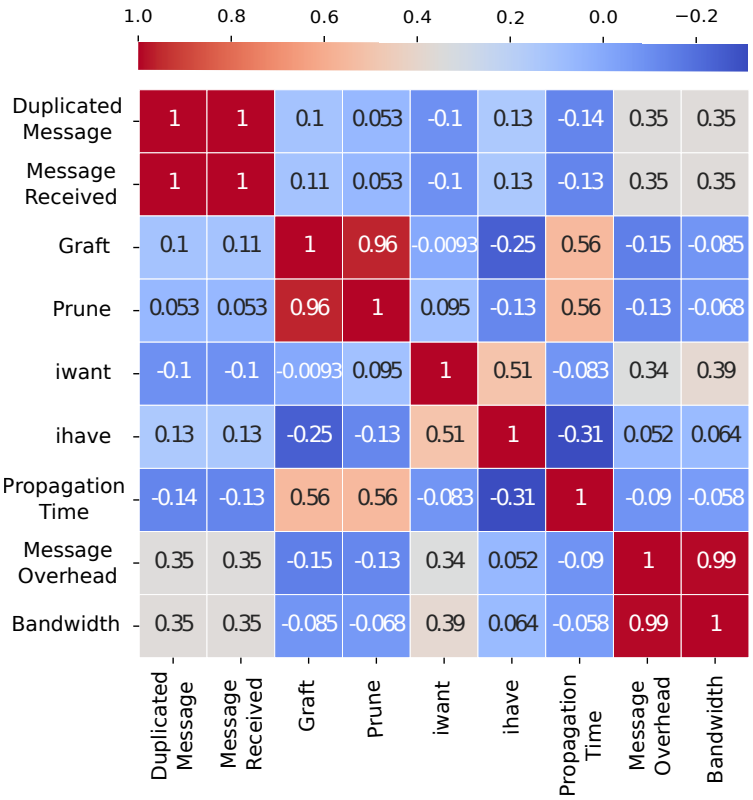

Figure 5.9: Heatmap of the correlation between every two dimensions

amount of *ihave* messages being exchanged, which means that it emits a high amount of gossip, with the amount of *iwant* messages also slightly higher than the average. The difference between the amount of *iwant* and *ihave* messages is, however, quite high, so the parameters related to gossiping could be better tuned for this particular group. All sets of this group were executed over the *general* topology, in which we have a smaller number of topics with all the nodes subscribed to them. By the number of *iwant* messages, we can infer that the full-message overlay does not reach the entire network, which is expected behavior for GossipSub and not a derogatory feature.

Cluster 1 (orange) has the lowest values for the *Bandwidth* and also the lowest quantity of *iwant* messages. In general, it seems to aggregate most of the sets executed on the *general* topology that did not fall on cluster 0, with some exceptions. The lower *Bandwidth* in this group is mainly due to the topology, considering that it is more densely connected, and so it needs fewer hops to reach the destination.

Cluster 2 (green) shows a higher concentration of *prune* and *graft* messages, encompassing only sets executed over the *unl* topology, where we have more topics with fewer subscribed nodes. As stated previously, there is a strong correlation between *graft*, *prune* and *PropagationTime*. The distributions for this set demonstrate that this correlation is directly proportional and, in this case, seems to be derogatory to the performance of the network. The sets in this group are generally voted outliers by the clustering algorithms, having only two members, as can be seen in Table 5.4.

Cluster 3 (red) shows the highest *Bandwidths* of the entire set, as well as the highest amount of *MessagesReceived*. With few exceptions, the sets in this cluster were executed over the *unl* topology, which leads us to conclude that topologies with a higher amount of smaller topics have higher *Bandwidth* than topologies with fewer big topics.

Cluster 4 (purple) also concentrates sets executed over the *unl* topology. As in the previous group, the *Bandwidth* is also on the higher side, although the amount of *MessagesReceived* shows to be low. Interestingly, the number of *iwant* messages seems to be on the average higher side, while the number of *ihave* messages is consistent with clusters 0, 2, and 3.

With that in mind, we may conclude that topologies with smaller but more varied topics have a wider variation in behaviors and performance. Sets executed over the *general* topology tend to fall into two quite stable clusters with lower *Bandwidth*, while those over the *unl* topology show more variation, by being sorted into three highly variable clusters. We also corroborated some of the correlations presented in Section 5.3.2.

5.3.4 Decision Tree

Now that we have analyzed the clusters for their behavior and performance, we can classify the sets of parameters based on the desired behavior of the network. To do this, we use a decision tree, shown in Figure 5.10. As a decision tree performs a classification task, here we rename each cluster from the previous section as the corresponding class, for which the decision tree will find a set of rules that can identify the elements of that class based on the parameterization. Thus, the idea is to link the clusters created on the basis of the performance/behavior of each experiment run with its parameterization. We can immediately see that the topology is the first parameter considered for the classification, with 0 representing the *unl* and 1 representing the *general* topology. In both branches, the next parameter used is the *interval*, with the *Dlo* immediately separating two classes in one of the *unl* branches.

In all, the gossiping parameters seem to be the ones that have less impact on the classification task. *Dlazy* does not appear in the tree at all, while *GossipFactor* seems only to be a determinant factor for classification on the *unl* branch, and even then it is only present once near the leaves. *Dout* and *Dscore* appear to be the most determinant factors in the *general* branch, while the *unl* branch seems to be more tied to the three primary D parameters (*D*, *Dlo* and *Dhi*).

The decision tree itself may not provide many insights about how parameters and dimensions relate to each other, but presents a tool for informed parameter selection. While the clustering and the correlation heatmap give us an understanding of the behavior and performance of different groups of sets of parameters, the decision tree is the last piece that provides a way to facilitate the parameterization of GossipSub to meet specific network requirements.

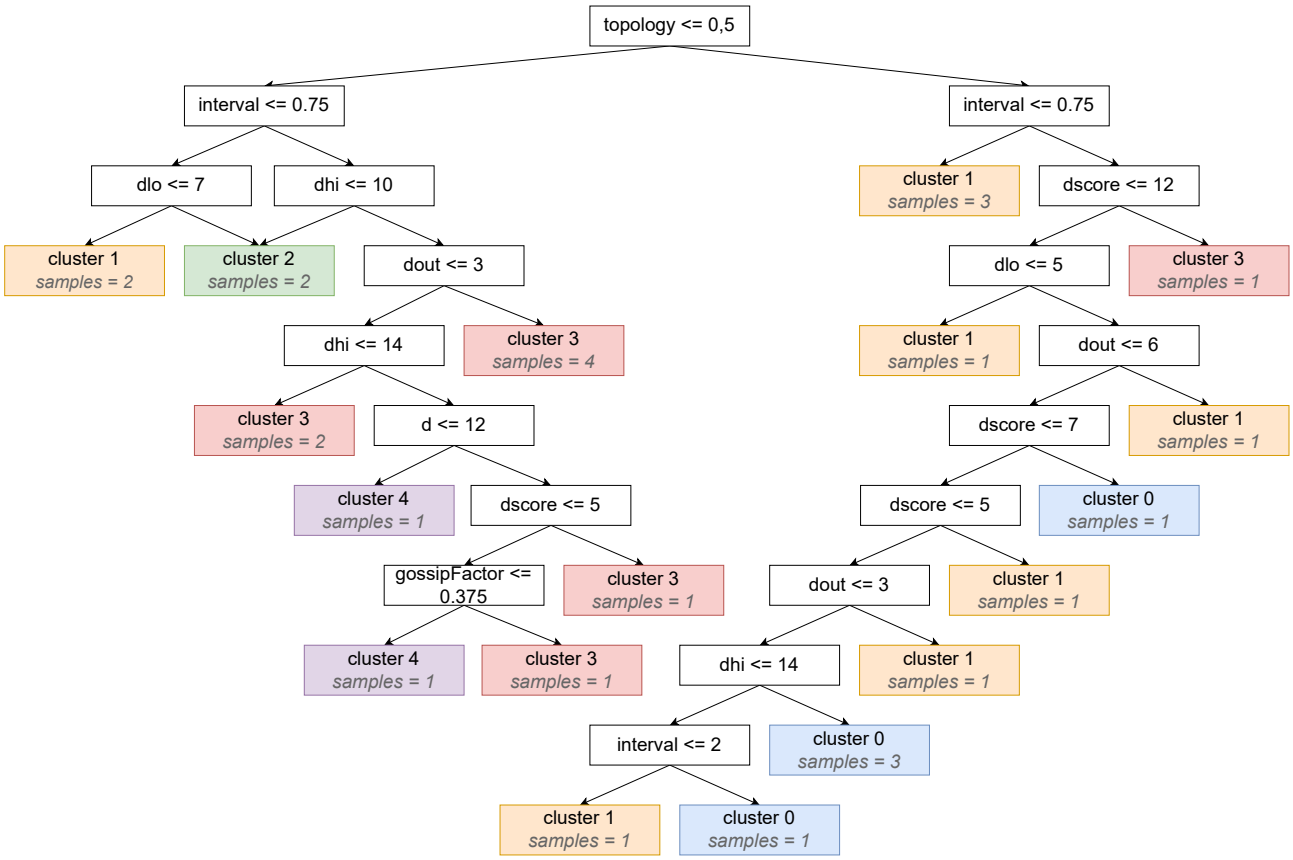

Figure 5.10: Decision Tree

5.4 Discussion

In this work, we presented a 9-dimensional analysis of the behavior and performance of GossipSub over an alternative consensus algorithm for blockchains. We plugged GossipSub into the XRPL and tested 13 sets of mesh parameters over two different topologies. We used data science tools to analyze nine dimensions of GossipSub and learn how it behaves and performs with different sets. These tools showed us how we can group different sets of parameters according to their behavior, how the dimensions correlate to each other, and finally, we provided a decision tree as a tool to select sets of parameters based on the behavior/performance of the clusters presented.

The nine dimensions studied were *MessageReceived*, *DuplicatedMessage*, *Bandwidth*,

MessageOverhead, *PropagationTime*, *graft*, *prune*, *iwant* and *ihave*, while the mesh parameters were *D*, *Dlo*, *Dhi*, *Dout* and *Dscore* for the eager push transmission, as well as *Dlazy* and *GossipFactor* for lazy pull, and finally, the *HeartBeatInterval*, that dictates the frequency of synchronous events on the network.

We used the clustered data and the correlation heatmap to draw remarks about the general behavior of GossipSub and also about how the dimensions relate to each other. We were able to trace a profile of the behavior of each cluster. Finally, we analyzed the decision tree generated over the parameterization. This tree was generated regarding the parameters, without considering the dimensions used for the clustering and the heatmap, and gave us insights into how to parameterize GossipSub according to the desired behavior and performance of the network as a whole.

We were also able to show that the underlying way messages are disseminated on the p2p network did not affect the convergence time of the XRP LCP. Thus, the GossipSub parameterization did not affect the consensus mechanism that lies upon the unstructured p2p network used by the XRPL blockchain.

In the next chapter, we go a step further and try to identify causal relationships between parameters and performance metrics. We use the findings of this chapter to select a target metric, identifying which variables are related to this measurement. The causal analysis helps us to gain more insight on how the alteration on the value of a parameter can impact on the behavior of the network, being able to even quantify the causal relationship.

Chapter 6

Causal Analysis for GossipSub Configuration

6.1 Introduction

As discussed in Chapters 1 and 5, GossipSub has tunable parameters related to its mesh construction and its score function, meaning that it is possible to configure it for better performance depending on the kind of system on which it is being used. Unfortunately, not much effort has been put into understanding how this configuration can affect the performance of the network. The configuration of Ethereum has been performed through extensive empirical tests on simulated networks[57]. However, in several scenarios, extensive empirical tests as the ones performed in Chapter 5 may not be feasible.

Considering a scenario where we have few means to extensively test and analyze the network according to its configuration, we propose the use of graphical causal tools to analyze the system from a causal point of view. Causal analysis can give us insight into how each parameter can affect a target performance measurement, also giving us tools to quantify those influences.

In this study, we focus on the parameters that GossipSub uses to generate its mesh. We base our methodology on the ideas presented by Judea Pearl[87] on the steps for causal inference, explained in Section 6.4. In Section 6.6 we find the causal relationships in the form of a graph, we then discover the distribution functions for each variable to form a causal model in Section 6.7. Finally, in Section 6.8, we quantify the relationships and simulate

interventions using the causal model created in the previous steps, seeking to answer the following questions:

- *Which parameters are more likely to affect the performance of the network?*
- *How these parameters affect the performance of the network?*

6.2 Limitations

As discussed in Chapter 1, the XRP Ledger Consensus Protocol (XRP LCP) works on the basis of quorum voting, with nodes casting votes according to the position of trusted peers. Each node maintains a Unique Nodes List (UNL), trusting that the peers on that list will not collude to defraud or stop the ledger. To converge into agreement, there needs to be a minimum overlap between all UNLs in the network, otherwise, the excessive disagreement may lead to a stall in the progress of the ledger[15][29][8][7], considering that the XRPL has fork prevention mechanisms.

The overlap required between the lists limits our evaluation when considering different combinations of number of topics and topic size. In most of the model-scenarios presented, there is an impossibility for the XRPL network to bootstrap, given that the UNLs will not have enough overlap between themselves for the consensus process to converge into a single ledger version. This characteristic of the XRPL limits our ability to directly evaluate the results obtained in the simulated interventions. Nevertheless, this work has the goal to better understand how the GossipSub mesh parameters impact the performance of the network, and not to directly find the ideal values for each parameter. This analysis seeks to understand the causal relationships between parameters and metrics, so we can better parameterize GossipSub for different scenarios than the ones it is already employed.

6.3 Related Work

Causal analysis is a tool used in several areas such as social sciences, biology, and medical sciences. It is not a new concept, but it has gained traction in recent years. It has also

been widely deployed in software engineering to study and solve problems related to fault localization, testing, and performance modeling[88].

To our knowledge, currently, there is no work that addresses causal analysis for studying the parameterization of networks; however, the use of causal analysis to study network performance is not completely new; Hours et al.[89] present the first study to make the case for the use of graphical causal methods to analyze network performance. The work focuses on data collected from an emulated Transmission Control Protocol (TCP) network, by measuring File Transfer Protocol (FTP) traffic, and on subsequent measurements obtained from a single FTP server connected to the Internet. In the first step, a causal graph is generated from the data collected in the emulated network, after causal inference methods are applied to get some predictions, which are validated based on the data collected by the real FTP server.

This study used the Tetrad[90] suite, concluding that constraint-based algorithms generated more informative graphs, based on knowledge of the TCP domain. The particular constraint-based algorithm used to subsequently predict network performance was the PC algorithm[91]. However, at the time of the study, Tetrad assumed either normality or linearity on the data. In our work, we considered four classes of discovery algorithms, including gradient-based ones that were not yet available at the time of the Hours et al. study. In addition, we used implementations that allowed us to make more assumptions about the distribution of the data.

In the domain of log analysis on distributed systems, Neves et al.[92] present Horus, a tool that uses graph databases to store causal graphs obtained from causally consistent aggregation of distributed systems. More specifically on the analysis of network events logs, we can cite Kobayashi et al.[93], which presents a methodology for the comparative analysis of two networks using causal graphs obtained from log data. The study focused on comparing logs from different sources by reconciling the generated causal graphs. In parallel, Jarry et al.[94] apply MixedLinGaM to the problem, generating weighted graphs with more causal relationships than the baseline given in the previous study. Regarding the use of causal analysis in network management, we can highlight Kim et al.[95], which

proposes an algorithm for Alarm Correlation based on root cause analysis without the need for domain knowledge.

6.4 Steps for the Causal Analysis

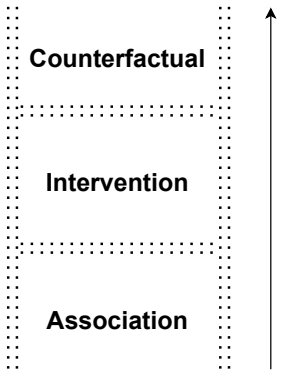

Figure 6.1: The causation ladder

In this section, we present the foundational theory on causality used to analyze the impact of the GossipSub mesh parameters on message overhead in the XRPL. This approach was pioneered by Judea Pearl[87] and uses a ladder to abstract the levels of relationships between variables. Figure 6.1 shows the *Ladder of Causation* with its three rugs, or steps. The higher we climb the ladder, the better our understanding of the causal relationships between the variables and the metrics studied.

In the first step, we have *Association*, in which we *observe* the system and try to understand how a change in an input can affect a possible output. Observations allow us to generate predictions that can be made from the perspective of *conditional probability*[96]. However, this mathematical formulation may not be able to fully express the causal relationships.

The association step may not give us information about causal associations, but gives us a good foundation to find correlations between variables. So we now climb to the second step, where we perform *interventions*. Interventions are made to observe how a modification in an element can impact an outcome. Mathematically, interventions can be expressed by

do-calculus.

Interventions help us find causal relationships between components of a system and how they impact certain outcomes; the next step of the ladder, counterfactuals, extrapolates the insights acquired to answer hypothetical questions. *Counterfactuals* can be seen as simulated or hypothetical interventions, where we try to emulate a state of the system that has never been observed in the real world.

In a broad view, we can consider *observation* the step where we start formulating a view of the system and identify correlations between variables and ask how those variables causally interact with each other. *Intervention* is where we can start answering the questions by means of experimentation. And finally, *counterfactuals* helps us answer hypothetical questions about the behavior of the modeled system.

6.5 Methodology

In this work, we opt for graphical tools to help us model and analyze causal relationships. The first tool used are Structural Causal Models (SCMs), which use Direct Acyclic Graphs (DAGs) to represent the causal relationships between variables. However, SCMs can only represent the causal relationships and their directions, without any information about the distribution of the data. To encode distributions, we need Graphical Causal Models (GCMs), which are sets consisting of a graph and a group of functions that can represent the distribution of the data for each variable in the model[97].

SCMs can be expressed as normal DAGs that can be represented using Graph Modeling Languages (GMLs) such as NetworkX[98] and GraphViz[99]. GCMs are more complex to represent and analyze, requiring more specific tools, such as DoWhy[100], a Python library for DAG-based causal inference. There is also the need to discover the causal relationships to obtain the SCM, to do that we employ both a manual domain knowledge-based approach and an algorithm for causal discovery, implemented in another Python library called gCastle[101].

Figure 6.2 presents a more detailed diagram of how we use the ladder of causation to

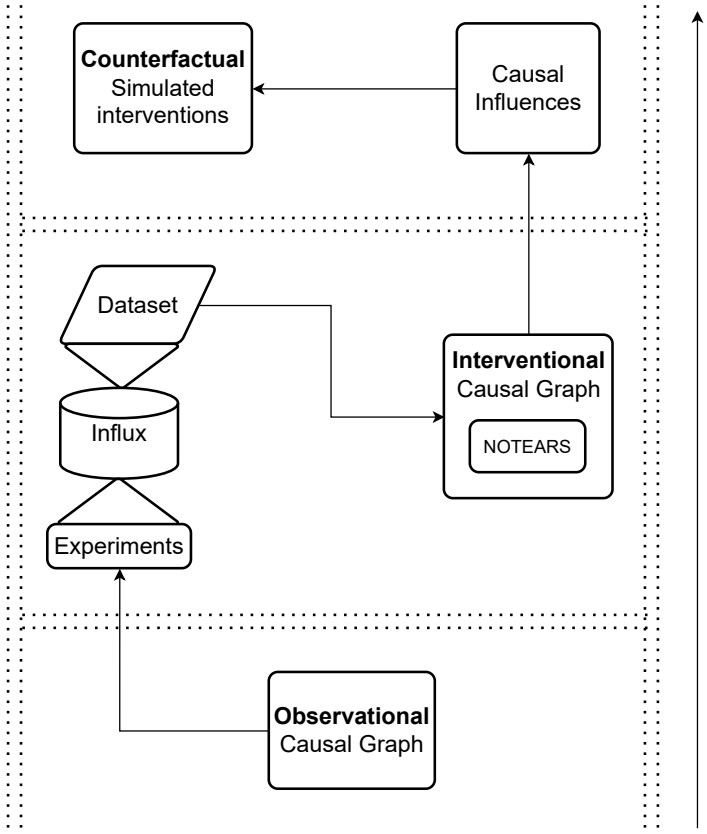

Figure 6.2: Steps for the Causal Analysis

discover and analyze causal relationships between the GossipSub mesh parameters and the target metric. We start by generating a SCM based on observational data and domain knowledge. The observational data were collected from the results presented in Chapters 4 and 5.

In the next step, we perform interventions on the GossipSub mesh parameters and use the data generated to create a second SCM using an algorithm for causal discovery. We then create a GCM using DoWhy to fit the distributions of the variables into the model. For this, we conducted randomized experiments using a testnet of 24 nodes running instances of *Flexi-pipe*[68]. We use GossipSub to propagate proposals, randomly selecting 44 sets of

parameters using three structures of topics. The previous chapters, used for the observational phase, did not consider how the topics were structured in quantitative terms, instead focusing on the structure of the trust overlay of the XRPL and how it can be mapped to GossipSub topics for optimal message overhead.

Table 6.1: Structures of Topics

ID	Topics	Subscriptions per node	Topics Size
1	1	1	24
2	24	$\mu = 16$	1
3	8	1	$\mu = 16$

Table 6.1 shows the three topic structures used in this phase from a quantitative point of view. Structure **1** uses one global topic to disseminate validations, with all nodes publishing on this topic and all nodes also subscribed to this topic. Structure **2** has 24 topics of size 1, with each node subscribed to 16 nodes on average, with a maximum of 18 and a minimum of 15 subscriptions, abstracting each node as a topic. The last structure, **3**, uses eight predefined topics, each node subscribed to 1 topic, the smallest topic having 16 nodes and the largest 18 nodes.

Structures **2** and **3** were previously used[102] to enhance the pubsub characteristics of the XRPL, while structure **1** is similar to how the XRPL *mainnet* is currently implemented, with a unique UNL to guarantee safety and liveness[8], also similar to how Ethereum disseminates blocks[46]. Here, the three of them serve a quantitative purpose, abstracted as two variables for the causal analysis: *topic size* and *number of topics*. The three structures also allow us to make focused interventions by varying either the topic size or the number of topics. Following this idea, each set of parameters was tested on each of the structures.

Having climbed the ladder, we get to the third step. We start by quantifying the strength of the causal relationships and determining which variables had the biggest causal influence on the chosen metric: message overhead. We then proceed to make interventions into some variables to analyze which set of parameters would create the desired behavior in the system.

This phase requires us to push the system to behave in ways that have not been previously observed. This is done by *hypothetical* or *simulated* interventions[96]. Those interventions are different from the ones described previously, in a way that they are not interventions done in the system, but in model-scenarios created in the two previous steps. First, we use causal graph tools to find quantitatively which are the variables that most influence the target measurement, we then proceed to use DoWhy to make simulated interventions to qualify these influences.

6.6 Observational Analysis

We used observational data and domain knowledge to generate an initial causal graph for GossipSub, considering parameters related to the mesh construction. Our initial goal was to model the entire system using the 9 dimensions presented in Chapter 5: *messageReceived*, *duplicatedMessage*, *prune*, *graft*, *ihave*, *iwant*, *bandwidth*, *propagationTime* and *messageOverhead* and the 8 parameters used to tune the mesh construction: *D*, *Dhi*, *Dlo*, *Dscore*, *Dout*, *Dlazy*, *GossipFactor*, *Heartbeat Interval*. However, this approach resulted in an overly complicated graph that hindered subsequent analysis of causation between variables.

Our next step was to simplify the graph for a more targeted analysis. For that, we chose a target measurement: *messageOverhead*. We then recreated the causal graph – Figure 6.3 – to express only the relationships that impact the target, namely most of the parameters related to the *full-message* overlay. Considering that most of the parameters that impact *messageOverhead* are defined per topic, we added two variables related to the number of topics and topic size, respectively *topics* and *topicSize*. In this scenario, the parameters are *D*, *Dlo*, *Dhi*, *Dout*, *Topics* and *TopicSize*.

The correlation heatmap presented in Chapter 5 helped us identify the variables to be used in our causal model. Considering the importance that *messageOverhead* has for the scalability and performance of the XRPL, we use this metric as a target in our model. We then identified which parameters and events showed the highest correlation with our target

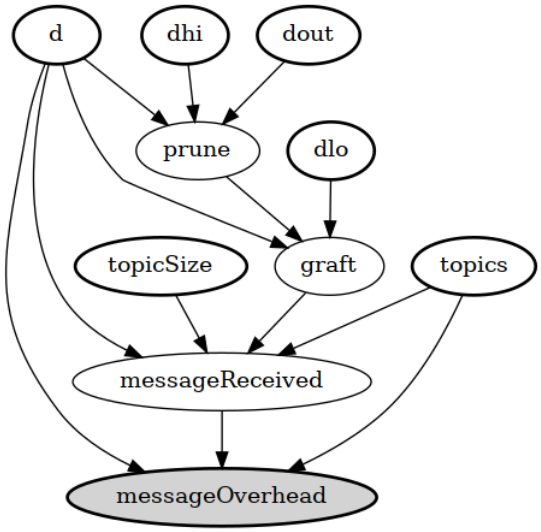

Figure 6.3: Causal Graph generated with observational data and domain knowledge

measurement and used domain knowledge to model the SCM shown in Figure 6.3.

D dictates the ideal number of connections per topic on the full message overlay and is more likely to affect the number of grafts and prunes. It may also cause the number of *messagesReceived* to increase and decrease; as for the *messageOverhead*, *D* influences the number of replicas a given node receives, forwarded by their direct peers. *Dhi* influences the *prune* of the connections, since it dictates the limit of the number of connections per topic. When a *prune* occurs, only *Dout* outbound connections of high-scoring peers are kept, so new connections must *graft*, therefore, *prune* events generally cause *graft* events, which are also bound to *Dlo*. The number of *topics* also affects *messageReceived* and affects *messageOverhead*, since each *d-value* is defined per topic. The *topicSize* affects the total number of *messageReceived*, directly affecting the *messageOverhead*.

6.7 Interventional Analysis

To gather interventional data, we set up an XRPL testnet with 24 nodes using GossipSub to disseminate proposals. We chose 43 sets of parameters and ran tests over three different structures of topics, each emulating different configurations for the XRPL trust overlay, as

shown in Table 6.1. Each set of parameters was used 3 times for 30 minutes in each of the structures, totaling 387 executions. We gathered data on the executions from the GossipSub tracer and loaded them into a timeseries database. The tracer provides time-stamped information about events on the network, both on full message and gossip overlays. After that, the data were aggregated per number of events in 5-second time chunks, excluding faulty executions from the dataset. The faulty executions are the ones in which the z-score falls below $0,15 * standard\,deviation$, considering the total number of data points recorded on the database for each execution.

We used the gCastle[101] library to perform causal discovery in the dataset obtained[1]. gCastle is a Python toolbox for learning causal structures, implementing 19 algorithms from different categories, and also providing evaluation metrics for the generated graphs. From the algorithms provided, we chose 4: PC[91][90], LiNGAM[103], GES[104][105] and NOTEARS[106]. We chose these algorithms to test the different categories of causal discovery: *constraint-based* (PC), *function-based* (LiNGAM), *score-based* (GES) and *gradient-based* (NOTEARS).

Experimentally, GES resulted in a convoluted graph that showed too many spurious causal relationships, as can be seen in Figure 6.4. We chose to exclude GES from further analysis because the graph would require pruning too many edges to make sense according to the domain knowledge; we also found cycles in the resulting graphs, which violates the concept of a DAG.

PC has the advantage of being the only algorithm implemented in gCastle that accepts the input of prior knowledge in the causal discovery in the form of required and forbidden edges. It works by forming a complete undirected graph and removing edges following a set of constraints that aim to discover independence[107]. We applied the three variants of PC available in gCastle: classical[108], parallel[109] and stable[110], inputting prior knowledge in the form of forbidden edges, preventing the algorithm from trying to find relationships between the parameters and in a reverse causal way, meaning that we prevent the creation

[1]Code and graphs available at `https://github.com/FlavScheidt/causalGossipSub/tree/main/1_Discovery`

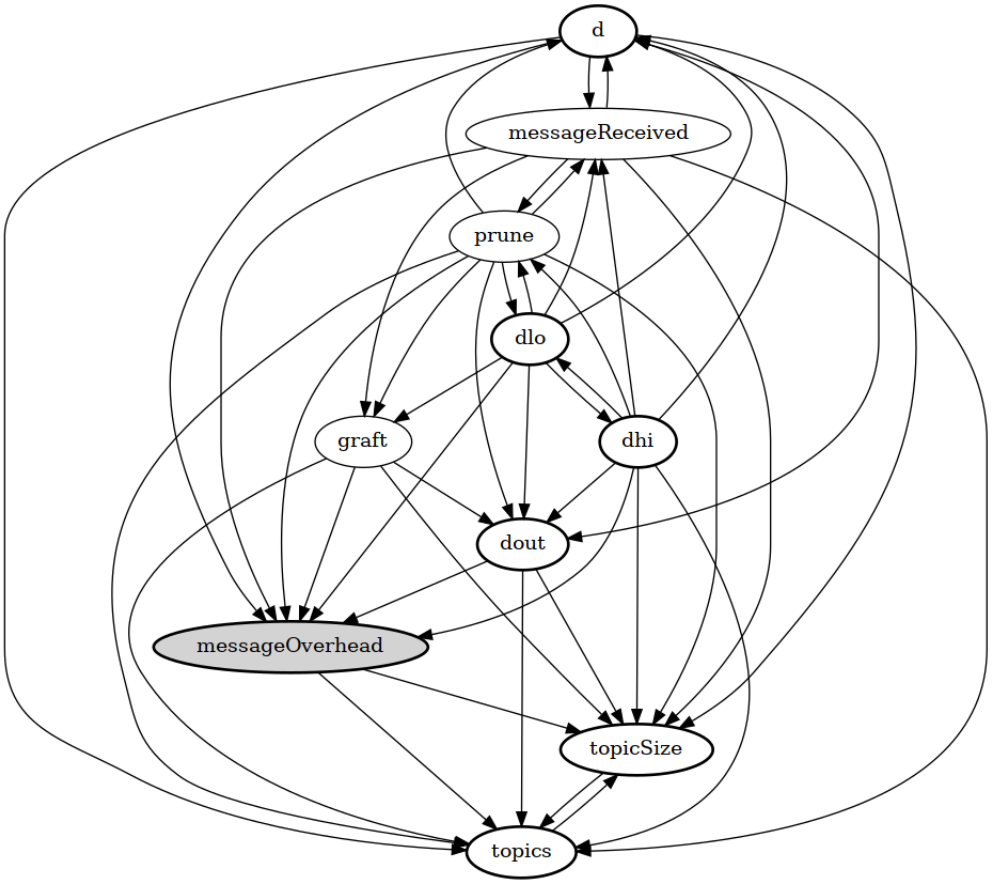

Figure 6.4: Structural Causal Graph generated by the GES algorithm

of edges from metrics resulting in parameters. We did not input any required edge so as to keep external interferences to a minimum.

LinGaM and NOTEARS do not accept prior knowledge; both found relationships between parameters that are true from a correlational point of view, considering that some *d-values* are constrained by the value of *D*. LinGaM also found spurious causal relationships between *topicSize* and *topics*. We pruned these edges for subsequent evaluations, as they are not truly causal relationships. The algorithms did not find any reverse causal connection; that is, in NOTEARS and LinGaM, the parameters always cause the metrics and not vice versa.

LinGaM uses the non-Gaussianity assumption to break the symmetry between causal and anti-causal models[96]. We used two variations for our analysis: ICA[111] and Di-

rect[112]. NOTEARS works with the idea of continuous optimization, employing a differentiable function that ensures that the optimized graph is acyclic[96]. We used two variations of NOTEARS: pure NOTEARS LowRank[113] and GOLEM[114].

Having obtained a SCM in the form of a causal DAG, we now go one step further and assign causal mechanisms to each of the variables in the model to generate a GCM. This assignment is made by assuming a distribution for the variables based on a correlation matrix. DoWhy also gives the possibility of automatically assigning causal mechanisms. We chose the second approach, with the automatic assignment assuming *Discrete Addictive Noise* distributions for all the nonroot nodes of the graph.

6.7.1 Refuting Causal Graphs

Having generated SCMs using the algorithms supplied by gCastle, we now need to evaluate the models to determine which one provides a better representation of the real-world system. For this analysis, we first employ gCastle to measure the distance between the models generated by interventional data and the one extrapolated from observational data and domain knowledge. We then proceed to use DoWhy, which provides tools to analyze both the fitting and performance of the models according to the training dataset provided.

Table 6.2 shows the evaluation metrics obtained by gCastle for the graphs generated by each method. However, these evaluation metrics are generated on the basis of the underlying truth, which is represented by the causal graph generated during the observation phase (Figure 6.3). The metrics in Table 6.2 then give us the distance between the model generated with observational data using domain knowledge and the model generated by interventional data using discovery algorithms.

The metrics in Table 6.2 are only sufficient to evaluate the effectiveness of the discovery algorithms if we assume that the graph generated using observational data is true in representing the causal relationships of the system. At this point, we do not know whether the observational graph is consistent with the interventional data. This question cannot yet be answered directly. However, we have tools to refute a certain causal graph if it does not satisfy some conditional independence statements on its nodes called Local Markov Conditions

Table 6.2: gCastle Evaluation of Discovery Algorithms

	Precision	**Recall**	**F1**
PC Classical	0.42	0.69	0.52
PC Stable	0.42	0.69	0.52
PC Parallel	0.42	0.69	0.52
D LinGaM	0.31	0.38	0.34
ICA LinGaM	0.43	0.53	0.48
NOTEARS	0.41	0.53	0.46
NOTEARS LowRank	0.21	0.38	0.27
NOTEARS GOLEM	0.2	0.23	0.21

(LMCs)[115]. We use another Python package, DoWhy[100], to refute the generated graphs. DoWhy provides tools to abstract the causal reasoning process, making it more accessible to nonexperts. One of the key features of the package is the ability to evaluate models, providing an overview of different metrics that provide information on the performance of the causal model[116].

DoWhy can perform independence tests on separate sets of variables, but given the size and amount of graphs we wish to evaluate, we chose to use another tool from the package: graph falsification[2][117]. The falsification tool gives us the summary of two tests, represented by the p-value of each metric. The first compares the number of LMCs violated compared to randomly generated graphs, and the second, TPa, verifies whether a graph is falsifiable. A graph is falsifiable if there is a randomly generated graph with the same number of LMC violations[118].

Figure 6.5 shows the histogram with the falsification summary of the graph generated with the observational data. We do not refute the graph, instead using it as a baseline for the evaluation of the causal discovery methods in Section 6.7; the p-value for LMC being 0.1 and the p-value for TPa 0 meaning that the graph is better than 90% of the random graphs generated, and the graph is not falsifiable. To evaluate the performance and fitness of the causal models generated by observational and interventional data, we first generated

[2]Code and evaluation reports available at `https://github.com/FlavScheidt/causalGossipSub/tree/main/2_ModelEvaluation`

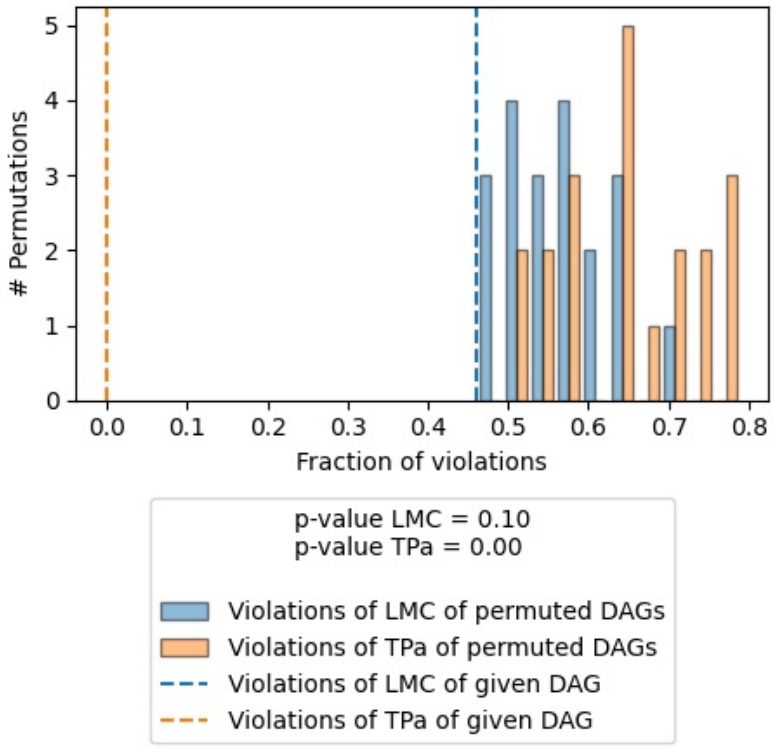

Figure 6.5: Falsification summary of the graph generated with observational data

the falsification summaries for the SCMs, shown in Table 6.3. We did not consider the TPa value, since all graphs showed to not be falsifiable.

Table 6.3: Falsification Summary of Discovery Algorithms

	LMC	LMC Violations	LMC Violations Rate
Observational	0.10	23/52	0.44
PC Classical	0.65	21/45	0.47
PC Stable	0.20	18/45	0.40
PC Parallel	0.35	22/45	0.49
D LinGaM	0.05	31/68	0.46
ICA LinGaM	0	29/64	0.45
NOTEARS	0.10	29/66	0.44
NOTEARS LowRank	0.05	37/64	0.42
NOTEARS GOLEM	0.05	42/78	0.54

DoWhy uses a threshold of 0.05 for the LMC p-value to refute graphs. By this estimation, only Direct LinGaM, ICA LinGaM, NOTEARS LowRank, and NOTEARS GOLEM would not be rejected. However, this threshold can be arbitrary in a real-world scenario, and so we do not assume any threshold to refute graphs, using the LMC p-value as a comparison metric, instead. We can see that ICA LinGaM has the ideal LMC p-value but still shows LMC violations, with PC Stable having the lowest rate of violations. We consider these violations to be present because of latent variables, that is, variables that cannot be observed but have a greater impact than some observed variable[89].

6.7.2 Performance Evaluation of the Graphical Causal Models

DoWhy can also evaluate GCMs to verify how well the models perform, if the assumption of the addictive noise model is correct, and how well the GCM captures the joint distribution of the observed data[116]. We chose two metrics in our evaluation, shown in Table 6.4. First, we look at the overall average KL-divergence (KL-Div in the table) between the generated and the observed distributions, the lower the KL-divergence, the better the model fits the observed data distribution. The second metric is obtained by nonroot node, evaluating the accuracy of the causal mechanisms expressed in the model, and is measured by the normalized Continuous Ranked Probability Score (CRPS); the closer this value is to zero, the better the precision of the causal mechanism for that given node.

Table 6.4: DoWhy Evaluation of Causal Discovery Algorithms

		CRPS			
	KL-Div	**graft**	**prune**	**messageReceived**	**messageOverhead**
Observational	4.88	0.14	0.88	0.35	0.18
PC Classical	5.06	0.15	0.06	0.11	0.35
PC Stable	5.07	0.15	0.06	0.11	0.35
PC Parallel	5.07	0.15	0.06	0.11	0.35
D LinGaM	3.78	0.25	-	0.35	0.13
ICA LinGaM	3.61	0.25	-	0.37	0.18
NOTEARS	3.62	0.25	-	0.37	0.13
NOTEARS LowRank	2.50	-	-	0.33	0.13
NOTEARS GOLEM	2.51	-	-	0.35	0.14

It is important to note that the results for LinGaM and NOTEARS – including all variations – had bigger KL-divergences before the pruning of the edges that represented relationships between the parameters. The values were around 7 before pruning and then decreased to around 3.7 to 2.5 afterward. We suppressed this analysis from this work for the sake of brevity, but it shows once again the importance of domain knowledge in the modeling of causal structures.

From the metrics represented in Table 6.4, we can see that NOTEARS LowRank and GOLEM have the smallest KL-divergence. However, both models ended up excluding *graft* and *prune* events, which may indicate that these events have little to no effect on *messageOverhead*. GOLEM has a high CRPS for *messageReceived*, making LowRank a better candidate. Nonetheless, we need to consider that both algorithms have low F1 scores (Table 6.2) when considering the domain knowledge model. So we turn our attention to the LinGaM variations and pure NOTEARS.

From Table 6.3, ICA LinGaM showed to better fit the data, also showed decent KL-divergence while having good or very good CRPS for the nonroot nodes, however showing a worst CRPS for *messageOverhead* compared Direct LinGaM and NOTEARS. Between those last two, NOTEARS showed a lower CRPS for the target measurement, keeping a KL-divergence close to ICA LinGaM and an acceptable F1 score. Therefore, using the evaluation tools and domain knowledge, NOTEARS generated a suitable causal model for the proposed scenario.

6.7.3 Selecting a Graphical Causal Model

Table 6.5: Distribution Functions of the GCM variables

	Distibution	**Function**
graft	Discrete	LinearRegression
messageReceived	Discrete	HistGradientBoostingRegressor
messageOverhead	Discrete	HistGradientBoostingRegressor

Domain knowledge is a crucial aspect of causal analysis, but it cannot find all the causal

relationships between variables. As a result, we analyze the resulting graphs for each category of discovery algorithms and select the SCM generated by pure NOTEARs that is most similar to the SCM generated with the observational data.

Having the SCM and the interventional dataset in hands, we can now generate a GCM by fitting distribution functions for each variable. For that, we use the DoWhy automatic assignment. The distribution functions attributed to each non-root node can be seen in Table 6.5.

6.8 Counterfactual Analysis

Counterfactuals can help us answer questions in the form of *what would have happened with variable x had I assigned the value y' to the variable y*. It helps us evaluate possible outcomes from states that were never observed or cannot be observed in the real world. But first, we must identify which variables are more likely to impact the outcome.

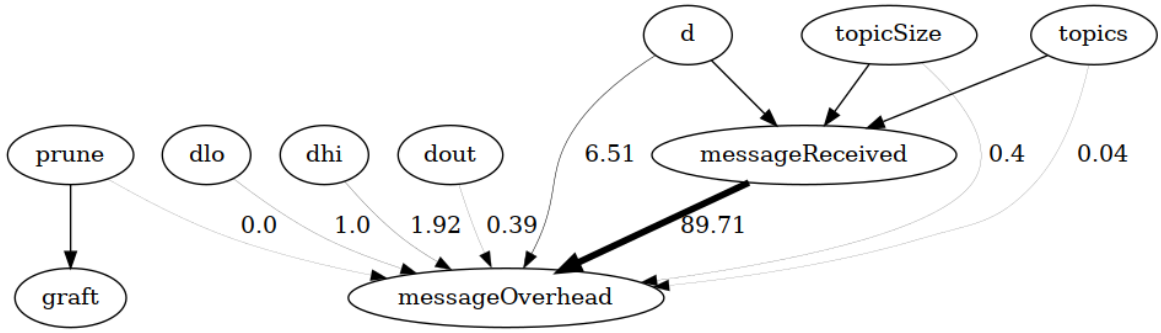

Figure 6.6: Causal strength of parameters over metrics, with messageOverhead as target

Figure 6.6 shows the causal strength of each variable in our target measurement, the values in the edges being the percentage of causal strength of each variable over the target. The graph was obtained again using the *direct strenght arrow algorithm*[119], provided by DoWhy. This algorithm tries to quantify causal relationships by abstracting the edges of the causal DAG as communication channels that can be corrupted by interventions. Those interventions cause new distributions on the data, the causal strength is then measured by the relative entropy distance between the two distributions.

We can see that the greatest influence on the *messageOverhead* variance comes from *messageReceived* and, on a minor scale, from *D*, *Dhi* and *Dout*. The causal relationship between *messageReceived* and *messageOverhead* is trivial, since more messages will always generate more duplicates.

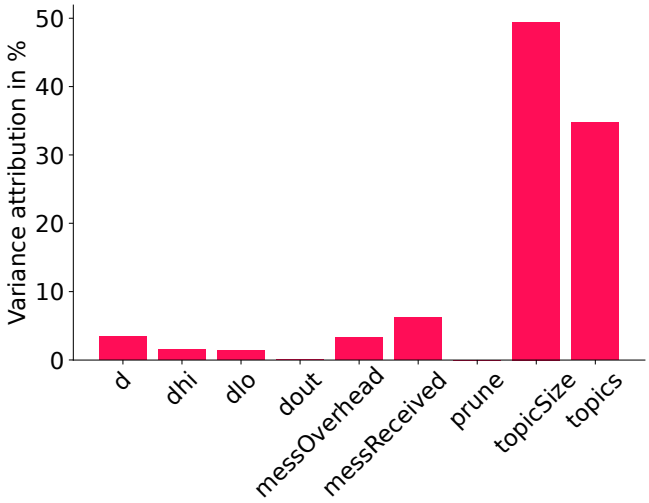

Figure 6.7: Causal Influence of parameters over metrics

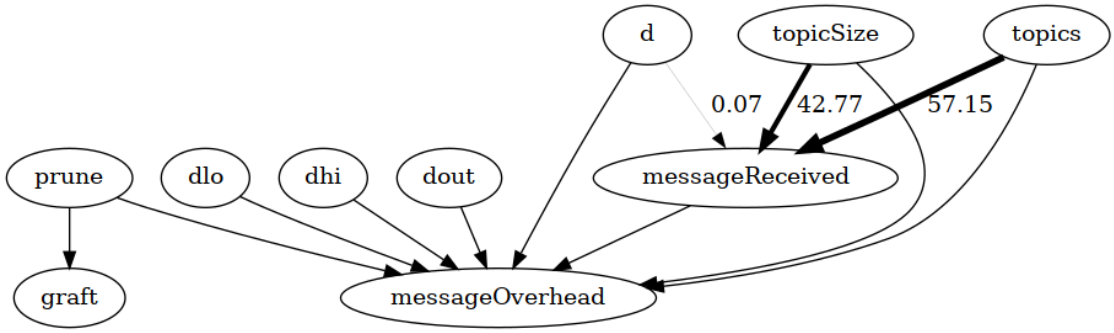

Figure 6.8: Causal strength of parameters over metrics, with messageReceived as target

We also ask what factors influence *messageOverhead* the most. This question can be answered by applying the *intrinsic causal contribution* method[120], which recursively rewrites each node as a function of its ancestors to make interventions that do not change the observed joint distribution. Figure 6.7 shows a bar graph that summarizes the results

obtained. We can now see that, in addition to *messageReceived*, *D* and *Dhi*, *TopicSize* and *Topics* also influence *messageOverhead*, in fact they have the highest influence.

Since we cannot capture much causal strength from *topicSize* directly in *messageOverhead*, we analyze the strength of causal relationships using *messageReceived* as the target measurement. Since the *intrinsic causal contribution* represents a node as a function of its ancestors, the previous results may imply that there are indirect causal effect that could not be captured by the *direct strength arrow algorithm*. Figure 6.8 shows the resulting SCM. *topicSize* and *topics* are indeed the variables with higher causal strength over *messageReceived*, confirming that the variables have an indirect causal effect over *messageOverhead*.

6.8.1 Simulated Interventions

After identifying the parameters that mostly affect our target measurement, we can now make interventions in the model-scenarios created in the previous steps of the ladder. Simulating interventions helps us better understand the impact that a certain value change in one variable can have on the target measurement. Here, we try to answer the questions in the form of *What happens to variable x if we change the value of variable y*.

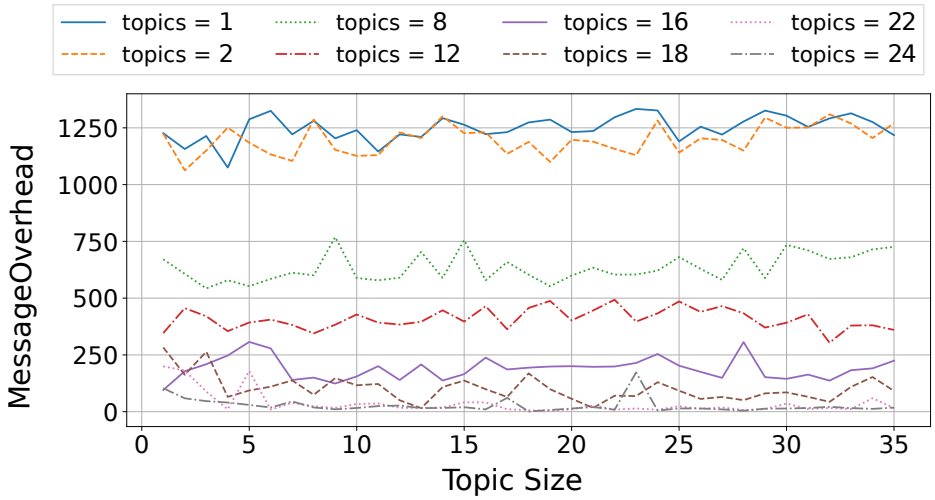

Figure 6.9: Simulated results for messageOverhead when intervening on topicSize and number of topics

We previously identified *topicSize* and *topics* as variables with a higher causal influence

over *messageReceived*, also causing a high indirect causal influence in *messageOverhead*. So, we selected these two variables for the first counterfactual analysis. It is hard to separate the two variables from each other, considering that the growth in the number of participants in the network may affect both dimensions.

In Ethereum and FileCoin, the number of topics remains constant no matter the size of the network, with only the size of the topics growing. The XRPL gives us the opportunity to try different arrangements for topics so that we can analyze how the network may behave in the case of a dynamic arrangement of topics. Consider the case in which UNLs can vary in number and sizes; Would this arrangement have any advantages over the approach where we mimic the structure used on Ethereum, with fixed topics always varying in size?

We try to answer this question by first simulating interventions on the number of topics to which each node is subscribed, with values ranging from 1 to 24. We use DoWhy to generate a dataset with simulated interventional samples, we then fit this dataset into a new GCM, again using the automatic distribution assignment capabilities, and the same SCM generated using NOTEARS. Since DoWhy uses the distribution functions previously assigned to the model to generate the simulated interventional samples, the same distribution functions were assigned to the GCM. We then used the GCM generated with the simulated interventional data for *topics* to simulate interventions in *topicSize*, with a range of 1 to 35.

Table 6.6: Default and Ethereum Configuration for GossipSub

	D	**Dlo**	**Dhi**	**Dout**
Ethereum	8	6	12	2
Default	6	5	12	2

Figure 6.9 shows a graphical representation of the dataset generated by the two subsequent simulated interventions in *topics* and *topicSize*. In order to isolate the two variables we are looking to analyze, we fixed the values of *D*, *Dhi*, *Dlo* and *Dout* using the Ethereum configuration, shown in Table 6.6. The graph shows that a smaller number of topics leads to a higher message overhead. When associated with the number of topics, *topicSize* does not appear to impact the target measurement as much as expected by the *intrinsic causal*

contribution of *topicsSize* in *messageOverhead*.

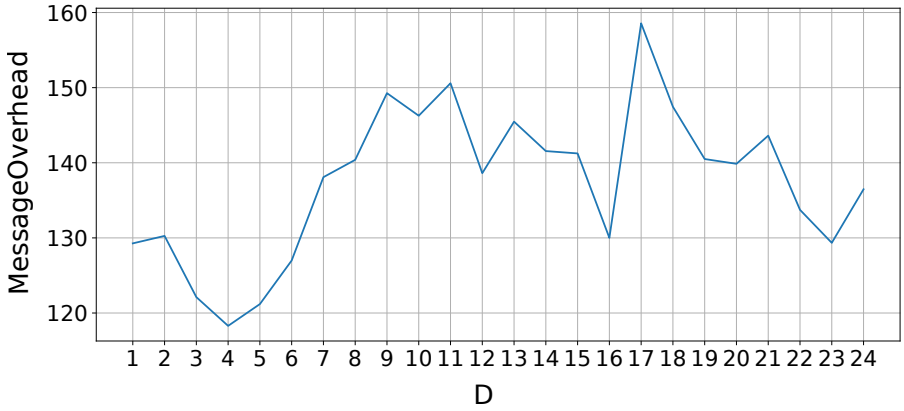

Figure 6.10: Simulated results for messageOverhead when intervening on D with 8 topics of size 16

What we can take from these results is that more varied topics can optimize message overhead. The size of the topics has less influence, and so a dynamic arrangement of topics according to the size of the network could be marginally beneficial for the performance of the system. It is important to note that in this study we do not consider the qualitative nature of topics, but we solely look at them from a quantitative point of view, not making any considerations about the difference between the nature of the messages published in each topic.

Going a step further, we simulate interventions in the next variable that shows to have the largest direct impact on *messageOverhead*: *D*. To isolate the variable being studied, we fixed the number of topics in 8 and the size of the topics in 16, using the dataset generated by performing simulated interventions in *topics* and *topicSize* according to the values indicated. We did not fix the values for *Dhi* and *Dlo* as they are dependent on *D*. We then simulated interventions in *D*, varying its values from 1 to 24.

Figure 6.10 represents the dataset generated by the simulated interventions in *D*. We can see that the smallest message overhead occurs in the region where *D* is between 3 and 4. Since the value of *D* can be relaxed from *Dlo* to *Dhi*, we consider it more suitable to refer to the ranges rather than to absolute values directly. After the optimal region, there is a natural increase in message overhead as the number of connections per topic increases,

except for a slight decrease in the region where *D* is equal to 16, which may be related to the size of the topics being also 16.

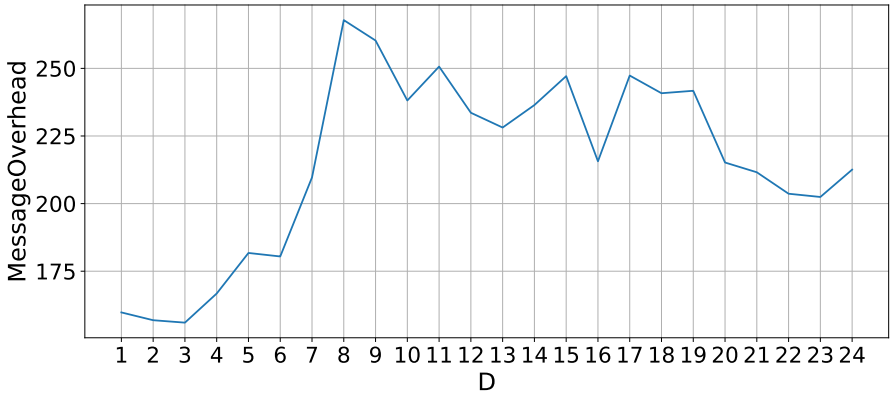

Figure 6.11: Simulated results for messageOverhead when intervening on D with 2 topics of size 24

It is important to note that the values of *D* are bound to the number of topics to which a node is subscribed. With that in mind, we repeat the analysis fixing the number of topics to 2 and the topic size to 24, similar to how we simulated the current structure of the XRPL. Figure 6.11 represents the resulting dataset, showing that, however smaller than in the previous simulation, there is still a valley where the values of *D* are optimal, in this case in the region between 2 and 3. There is again a slight decrease around 21 and 24, similarly to the previous analysis, and it may be due to the value of *D* being closer to the size of the topics.

This counterfactual analysis shows the importance of the structure of topics for optimal message overhead on GossipSub. First, *direct strength arrow algorithm* and the *intrinsic causal contribution* calculated using DoWhy showed a strong indirect influence of *topics* and *topicSize* on *messageOverhead*, and also the direct influence of *D* on the target measure. Then, the simulated interventions showed how *topics* and *topicSize* work together to influence the outcome and how important they are for the proper configuration of the *d-values*, which are bound to the value of *D*.

6.9 Evaluation

As stated in Section 6.2, the quorum-based nature of the XRPL hinders our ability to empirically evaluate our methodology. Most of the scenarios simulated cause excessive disagreement, which causes the network to not produce a valid ledger.

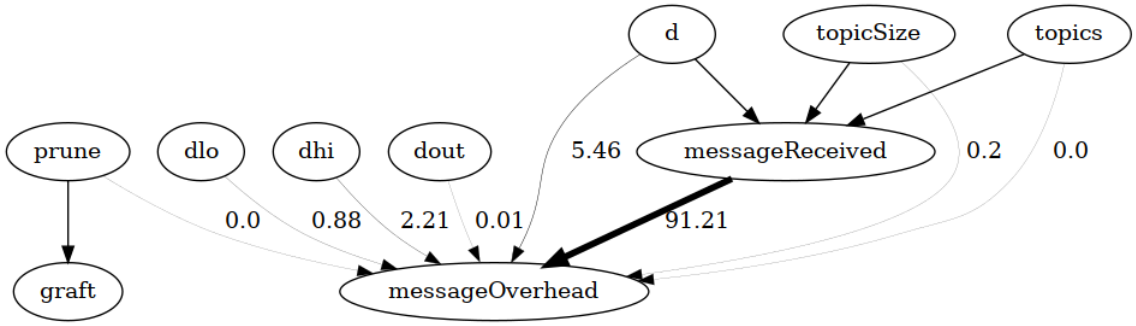

Figure 6.12: Causal strength of parameters over metrics, with messageOverhead as target. Generated using the simulated dataset

Instead of an empirical evaluation of the results obtained with the simulated interventions, we opted to evaluate our methodology by comparing the graph generated by the *direct strength arrow algorithm* in the interventional data (Figure 6.6) with the one generated by the simulated interventions (Figure 6.12). The two graphs are similar with regard to the strength of the causal influences calculated, from which we can say that the dataset generated by the simulated interventions is representative of the data distribution observed on the interventional dataset, and thus our methodology is consistent for the data distribution and to the causal relationships and strengths observed.

6.10 Discussion & Future Perspectives

In this Chapter, we presented a causal analysis of the GossipSub mesh parameters in relation to message overhead to understand how parameterization affects the performance of the dissemination system. We used graphical causal tools, establishing our methodology on the ideas presented by Judea Pearl[87] about how to understand causality.

In all, this study helped us to better understand the causal relationships between the parameterization of the network and its behavior. The visual causal tools and techniques helped us to reach conclusions that would not be possible simply by observing the system. However, domain knowledge played a huge role in making sense of the results given by graphical tools. In a more empirical sense, we could observe how strong the influence of the topic structure is, even impacting the optimal values for the third most influential variable: D.

However, we still have limited understanding of how the parameters related to the gossiping layer impact the overall performance of the system. Future work can move in the direction of a more complete analysis of the entire mesh of GossipSub, encompassing both the full message and gossip layers in the same model. This study was also limited by the necessity for XRP LCP to have overlapping sets of UNLs on a tightly connected trust overlay. A validation of our methodology in a broader scenario can also be achieved in the future.

Discussion and Future Perspectives

In this work, we explored enhancements in scalability and performance on blockchains by employing publisher/subscriber (pubsub) dissemination to reduce message overhead on Federated Byzantine Agreement (FBA)-based blockchains, instead of the more common proof-based consensus. The research was done using the concrete case of the XRPL, which uses subsets of validators to reach an agreement over multiple voting rounds.

Although the XRPL is a well-established and widely used blockchain, some scalability and performance issues may become more noticeable with growing adoption. The main problem is the lack of node-wise scalability, given the use of flood publishing, as well as the centralization of power in the hands of a few participants. In Chapter 3 we measure the problem of message overhead caused by message flooding, presenting a tool called Flexi-pipe, which we used throughout this work. Our analysis showed that the use of flood publishing is, in fact, a bottleneck that harms the scalability of the network in terms of the number of nodes.

We then made a case for treating the XRP LCP as a pubsub system, given its characteristic of using lists of trusted nodes to create what we call a *turst overlay*. Considering that every node only trusts a subset of the network, we abstract UNLs as topics since the trust put in each UNL is collective. However, the current implementation of the XRPL works with a single list, maintained and curated by the XRP Ledger Foundation (XRPLF). This structure is ideal to maintain network safety, but causes some problems concerning the distribution of power in the network, and also diminishing the pubsub characteristics of the XRPL.

We started by proposing a framework to give the XRPL tools to create and maintain more diversified UNLs to better disperse the authority and take the curation out of the hands of human actors. Our proposal, called Nested, Layered, Autonomous, Continuous (NLAC), was presented in Chapter 2 and encompasses three modules that can be used together or

separately. The first module uses Token-Curated Registries (TCRs) to determine membership in the space of trusted nodes; The second creates layers of trust using a variation of TCRs to rank nodes according to their reliability, giving nodes in higher layers more power to determine who can be part of the trusted space, using game theory to diminish the probability of malicious behavior. Finally, the third module uses optimization to separate the nodes in the network into balanced lists that overlap each other while maximizing the total trust of the system.

By applying the three of them in sequence, we could achieve a forest in the form of a DAG in which we can guarantee a minimum overlap between each list, keeping the trust overlay tightly connected. NLAC not only provides means to better disperse the authority between the network participants but also gives us the ability to enhance the pubsub characteristics of the XRPL. This enhancement allows us to better explore the use of pubsub dissemination to address another issue present in the current implementation of the XRPL: the high message overhead caused by the use of flood publishing.

After improving the pubsub characteristics of the XRPL, we could test our hypothesis that pubsub dissemination could reduce the message overhead caused by the use of flood publishing. We again used Flexi-pipe to create an overlay over the XRPL to isolate the dissemination of proposals, also using the tool to plug GossipSub into the XRPL.

We tested three ways of mapping GossipSub to the XRPL, the first using the same structure as Ethereum, which also complies with the current implementation of the XRPL, with different topics for different types of messages. We then took advantage of the pubsub characteristics of the XRPL and tried two scenarios, the first in which each node publishes proposals and validations on its own topics and the nodes subscribe to those topics, mapping the UNLs into lists of subscribed topics. The second takes advantage of the structure created previously by NLAC, with predefined sets of UNLs, mapping each UNL as a topic, with each node subscribing to a unique topic.

GossipSub improved the message overhead in the three proposed schemes. However, the best results were obtained in the scenarios in which we treated the XRP LCP as a pubsub system, mapping its trust overlay into topics. The results corroborated our proposition

of treating the XRPL as a pubsub system. After this, we jumped into the second part of this work, which focuses on deepening our understanding of how the GossipSub parameterization works, facilitating the portability of the dissemination system into other platforms than the ones it was originally proposed for (ie. IPFS and Ethereum), specially for FBA consensus blockchains.

To start understanding how to configure GossipSub for use in different platforms, we performed an analysis of the tunable parameters used for the mesh construction. Since there was no previous academic research focused on parameter tuning, we turned to empirical studies made to configure GossipSub for the particular demands of Ethereum. From these studies and from our understanding of how GossipSub constructs the mesh, we selected eight parameters regarding both the full message and the gossip overlay. We then chose three performance metrics to study: *bandwidth*, *message overhead*, and *propagation time*, also identifying some network events that can affect these metrics: *graft*, *prune*, *iwant*, and *ihave*.

With the three metrics and the four types of events accounting for nine dimensions, we then used data science techniques to find relationships between the dimensions and the parameters. Once more we used Flexi-pipe to perform experiments, collecting metrics directly from the GossipSub tracer. We started by using unsupervised learning to cluster the metrics gathered from the executions, looking to identify patterns between them. We then generated a correlation heatmap between the dimensions to understand which events may impact the most on the performance metrics. Lastly, we used supervised learning to generate a decision tree based on the parameters values of each cluster, to help in the decision of sets of parameters according to the desired behavior of the network.

The dimensional analysis using data science tools gave us insight on how parameters, network events, and metrics relate to each other, and how parameters can affect network performance. However, we still lacked a deeper understanding of how different parameters caused different behaviors. We then took a step further and used the acquired knowledge to perform a causal analysis, seeking to understand causal relationships between parameters and metrics, and also to quantify said relationships to aid in the parameterization of the

GossipSub mesh.

We started our analysis by observing the behavior of GossipSub based on the findings of previous chapters and applying domain knowledge. This phase resulted in the selection of the parameters used for our analysis, as well as the selection of the target measurement. The final product was an SCM, which laid the foundation for the subsequent steps.

We then moved to the second phase, which involves making interventions in the system and analyzing the results to answer questions about what happens to a certain variable when we change the values of another variable. We made interventions and measured results on an XRPL testnet, using the XRPL as a concrete case for the usage of GossipSub to disseminate messages on blockchains. From the data gathered on these interventional experiments, we generated a new SCM, this time using a causal discovery algorithm, NOTEARS in three variations. To choose which graph would better represent the model, we compared the results of the three variants with the SCM obtained with observational data and domain knowledge and selected the one with the lowest distance. We then used DoWhy to fit the distribution of each variable into distribution functions. The outcome of this phase was a more refined understanding of the causal relationships present in the system, as well as insights on the data distribution of the dataset obtained by making interventions in the real-world system, culminating in a model represented by a GCM.

The final step is the one that gave us the most insight into how the parameters relate causally to the target measurement. To start the analysis, we asked which parameters had the greatest influence on message overhead, by applying two causal influence algorithms. We discovered that the number of topics, the size of the topics, and D were the parameters with a higher causal influence on the message overhead. We then proceeded to perform simulated interventions in the model to be able to visually study the impact of the simulations on message overhead.

With the causal analysis we learned that the arrangement of topics is the factor that causes the most variation in the message overhead. D also played a role in influencing the target measurement. Events such as *graft* and *prune* showed to have little to no causal influence. We also discovered that values of D close to the number of topics showed a lower

message overhead. However, our analysis was limited by the impossibility of the XRPL to form a trust overlay and produce ledger versions for certain configurations, which made it harder evaluate our methodology.

In all, this work made the case for treating the XRPL as a pubsub system, proposing tools to enable the creation and maintenance of UNLs automatically, presenting GossipSub as an alternative to flood publishing and showing a deep analysis of the GossipSub configuration for improved scalability and performance. We showed how our tool for creating and maintaining UNLs can increase the dispersion of authority and enhance the pubsub characteristics of the XRPL. We also showed that pubsub dissemination is a viable option to decrease message overhead without harming the ledger, even when we maintain the current single UNL strategy.

Therefore, we were able to answer the general question presented, providing concrete results for the use of pubsub techniques, which showed to decrease the message overhead in different scenarios. We were also able to answer the more specific question of how the mesh parameters related to pubsub dissemination correlate to the performance and scalability metrics proposed, by employing data science and graphical causal tools to find inferential and causal relationships between the parameters and the metrics. However, this work does not make any considerations about the safety and resilience against byzantine faults in the network. We considered the previous studies into the GossipSub and XRPL sturdiness to be enough, but further studies may be necessary in this area. We also had limitations in validating the causal models used to study the causal relationships; studies in more diverse scenarios may be required in this sense.

Main Publications

[62] F. Scheidt de Cristo, A. Geimer, and R. State, “Nlac: A self-maintained trust overlay for the xrp ledger,” in *2023 IEEE Latin-American Conference on Communications (LATINCOM)*, IEEE, 2023.

[68] F. Scheidt de Cristo, W. M. Shbair, L. Trestioreanu, and R. State, “Pub/sub dissemination on the xrp ledger,” in *2023 IEEE Latin-American Conference on Communications (LATINCOM)*, Best Paper Award, IEEE, 2023.

[102] F. Scheidt de Cristo, J.-P. Eisenbarth, J. A. Meira, and R. State, “A 9-dimensional analysis of gossipsub over the xrp ledger consensus protocol,” in *NOMS 2024 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2024.

Publications as 2nd Author

[14] L. A. Trestioreanu, F. Scheidt de Cristo, W. Shbair, J. Francois, D. Magoni, *et al.*, "To squelch or not to squelch: Enabling improved message dissemination on the xrp ledger," in *NOMS 2024 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2024.

[75] L. Trestioreanu, W. M. Shbair, F. Scheidt de Cristo, and R. State, "Xrp-ndn overlay: Improving the communication efficiency of consensus-validation based blockchains with an ndn overlay," in *NOMS 2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2023.

Non Related Publications

[124] F. Scheidt de Cristo, W. M. Shbair, L. Trestioreanu, R. State, and A. Malhotra, “Self-sovereign identity for the financial sector: A case study of paystring service,” in *2021 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2021, pp. 213–220.

[125] F. Scheidt de Cristo, W. M. Shbair, L. Trestioreanu, A. Malhotra, and R. State, “Privacy-preserving paystring service,” in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2021.

References

[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008. [Online]. Available: www.bitcoin.org.

[2] D. Schwartz, N. Youngs, and A. Britto, *The ripple protocol consensus algorithm*, 2014. [Online]. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[3] M. Becker and B. Bodó, "Trust in blockchain-based systems," *Internet Policy Review*, vol. 10, 2 2021, ISSN: 21976775. DOI: 10.14763/2021.2.1555.

[4] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual international cryptology conference*, Springer, 1992, pp. 139–147.

[5] V. Buterin, "What proof of stake is and why it matters," *Bitcoin Magazine*, vol. 26, 2013.

[6] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, vol. 32, pp. 1–45, 2015.

[7] I. Amores-Sesar, C. Cachin, and J. Mićić, "Security analysis of ripple consensus," in *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, ser. LIPIcs, vol. 184, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, ISBN: 978-3-95977-176-4. [Online]. Available: https://www.dagstuhl.de/dagpub/978-3-95977-176-4.

[8] B. Chase and E. MacBrough, *Analysis of the xrp ledger consensus protocol*, 2018. DOI: 10.48550/ARXIV.1802.07242. [Online]. Available: https://arxiv.org/abs/1802.07242.

[9] J.-P. Vergne, “Decentralized vs. distributed organization: Blockchain, machine learning and the future of the digital platform,” *Organization Theory*, vol. 1, 4 Oct. 2020, ISSN: 2631-7877. DOI: 10.1177/2631787720977052.

[10] *Xrpl negative unl*, Accessed: 2023-02-14. [Online]. Available: https://xrpl.org/negative-unl.html.

[11] P. Baran, “On distributed communications networks,” *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[12] *Xrpl - the foundation unique node list*, Accessed: 2023-03-31. [Online]. Available: https://foundation.xrpl.org/unl/.

[13] *Xrpl - system requirements*, Accessed: 2023-03-31. [Online]. Available: https://xrpl.org/system-requirements.html.

[15] L. Mauri, S. Cimato, and E. Damiani, “A formal approach for the analysis of the xrp ledger consensus protocol,” SciTePress, 2020, pp. 52–63, ISBN: 9789897583995. DOI: 10.5220/0008954200520063.

[16] V. Buterin *et al.*, “Ethereum white paper,” *GitHub repository*, vol. 1, pp. 22–23, 2013.

[17] *Message propagation — filecoin spec*, Accessed: 2022-07-19, 2020. [Online]. Available: https://spec.filecoin.io/systems/filecoin_blockchain/message_pool/message_syncer.

[18] *Libp2p - publish/subscribe*, Accessed: 2023-09-12. [Online]. Available: https://docs.ipfs.tech/concepts/libp2p/#publish-subscribe.

[19] L. Lamport, “The part-time parliament,” pp. 277–317, 2019, ACM SIGOPS Hall of Fame Award in 2012.

[20] H. J. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram, “Resilient asymptotic consensus in robust networks,” *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 4, pp. 766–781, 2013.

[21] M. Castro, B. Liskov, *et al.*, “Practical byzantine fault tolerance,” in *OsDI*, vol. 99, 1999, pp. 173–186.

[22] *Xrpl charts*, Accessed: 2023-01-26. [Online]. Available: https://livenet.xrpl.org/.

[23] *Xrp: Utility for the new global economy*, Accessed: 2023-01-26. [Online]. Available: https://ripple.com/xrp/.

[24] K. Croman, C. Decker, I. Eyal, *et al.*, "On scaling decentralized blockchains: (a position paper)," in *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*, Springer, 2016, pp. 106–125.

[25] *Ethereum average block time chart*, Accessed: 2023-01-26. [Online]. Available: https://etherscan.io/chart/blocktime.

[26] *The ethereum vision: Understanding the ethereum vision*, Accessed: 2023-01-26. [Online]. Available: https://ethereum.org/en/upgrades/vision/.

[27] *Xrpl consensus research*, Accessed: 2024-01-18. [Online]. Available: https://xrpl.org/consensus-research.html.

[28] E. MacBrough, *Cobalt: Bft governance in open networks*, Feb. 2018. [Online]. Available: http://arxiv.org/abs/1802.07240.

[29] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," vol. 9229, Springer Verlag, 2015, pp. 163–180, ISBN: 9783319228457. DOI: 10.1007/978-3-319-22846-4_10.

[30] K. Christodoulou, E. Iosif, A. Inglezakis, and M. Themistocleous, "Consensus crash testing: Exploring ripple's decentralization degree in adversarial environments," *Future Internet*, 2020. DOI: 10.3390/fi12030053. [Online]. Available: www.mdpi.com/journal/futureinternet.

[31] V. Tumas, S. Rivera, D. Magoni, and R. State, "Topology analysis of the xrp ledger," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 1277–1284.

[32] V. Tumas, S. Rivera, D. Magoni, and R. State, “Federated byzantine agreement protocol robustness to targeted network attacks,” in *2023 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2023, pp. 443–449.

[33] H. Aoyama, “Xrp network and proposal of flow index,” in *Proceedings of Blockchain in Kyoto 2021 (BCK21)*, 2021.

[34] C. A. Roma and M. A. Hasan, “Energy consumption analysis of xrp validator,” in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2020, pp. 1–3.

[35] M. van Meerten, B. K. Ozkan, and A. Panichella, “Evolutionary approach for concurrency testing of ripple blockchain consensus algorithm,” in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2023, pp. 36–47.

[36] M. Roelvink, M. Olsthoorn, and A. Panichella, “Log inference on the ripple protocol: Testing the system with an empirical approach,” 2020.

[37] C. Ma, Y. Zhang, B. Fang, H. Zhang, Y. Jin, and D. Zhou, “Ripple+: An improved scheme of ripple consensus protocol in deployability, liveness and timing assumption,” *CMES - Computer Modeling in Engineering and Sciences*, vol. 130, pp. 463–481, 1 2022, ISSN: 15261506. DOI: 10.32604/cmes.2022.016838.

[38] M. Mundhra and C. Rebeiro, “Sissle in consensus-based ripple: Some improvements in speed, security, last mile connectivity and ease of use,” *CoRR*, 2020.

[39] V. Tumas, S. Rivera, D. Magoni, and R. State, “Probabilistic edge multicast routing for the xrp network,” in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 5129–5134.

[40] M. Ellery and I. Ashimine, *Xrpl - consensus and validation*, Accessed: 2021-10-14. [Online]. Available: https://github.com/ripple/rippled/blob/develop/docs/consensus.md.

[41] *Xrpl introduction to consensus - trust-based validation*, Accessed: 2022-09-13. [Online]. Available: https://xrpl.org/intro-to-consensus.html#trust-based-validation.

[42] W. Hao, J. Zeng, X. Dai, *et al.*, "Towards a trust-enhanced blockchain p2p topology for enabling fast and reliable broadcast," *IEEE Transactions on Network and Service Management*, vol. 17, pp. 904–917, 2 Jun. 2020, ISSN: 19324537. DOI: 10.1109/TNSM.2020.2980303.

[43] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, IEEE, 2013, pp. 1–10.

[44] H. Barjini, M. Othman, H. Ibrahim, and N. I. Udzir, "Shortcoming, problems and analytical comparison for flooding-based search techniques in unstructured p2p networks," *Peer-to-Peer Networking and Applications*, vol. 5, pp. 1–13, 2012.

[45] R. Baldoni, L. Querzoni, and A. Virgillito, "Distributed event routing in publish/subscribe communication systems: A survey," in H. Miranda, L. Rodriguez, and B. Garbinato, Eds. Springer Berlin Heidelberg, 2009.

[46] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and ETH2.0 networks," 2020. [Online]. Available: https://arxiv.org/abs/2007.02754.

[47] *Gossipsub v1.0: An extensible baseline pubsub protocol*, Accessed: 2022-03-01. [Online]. Available: https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md.

[48] D. Vyzovitis and Y. Psaras, *Gossipsub: A secure pubsub protocol for unstructured, decentralised p2p overlays*, 2019. [Online]. Available: https://research.protocol.ai/blog/2019/a-new-lab-for-resilient-networks-research/PL-TechRep-gossipsub-v0.1-Dec30.pdf.

[49] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Pro-*

ceedings of the 11th international workshop on Network and operating systems support for digital audio and video, 2001, pp. 11–20.

[50] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 1489–1499, 8 Oct. 2002, ISSN: 07338716. DOI: 10.1109/JSAC.2002.803069.

[51] A. Gupta, O. D. Sahin, D. Agrawal, and A. El Abbadi, "Meghdoot: Content-based publish/subscribe over p2p networks," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, 2004, pp. 254–273.

[52] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "Tera: Topic-based event routing for peer-to-peer architectures," in *ACM International Conference Proceeding Series*, vol. 233, 2007, pp. 2–13, ISBN: 1595936653. DOI: 10.1145/1266894.1266898.

[53] J. A. Patel, É. Rivière, I. Gupta, and A. M. Kermarrec, "Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems," *Computer Networks*, vol. 53, pp. 2304–2320, 13 Aug. 2009, ISSN: 13891286. DOI: 10.1016/j.comnet.2009.03.018.

[54] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "Stan: Exploiting shared interests without disclosing them in gossip-based publish/subscribe.," in *IPTPS*, 2010, p. 9.

[55] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *2011 IEEE International Parallel & Distributed Processing Symposium*, IEEE, 2011, pp. 746–757.

[56] V. Setty, M. Van Steen, R. Vitenberg, and S. Voulgaris, "Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub," in *Middleware 2012: ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings 13*, Springer, 2012, pp. 271–291.

[57] T. Van Epps, *Testing gossipsub with genesis*, Accessed: 2022-02-01. [Online]. Available: https://medium.com/whiteblock/testing-gossipsub-with-genesis-6f89e845b7c1.

[58] *Eth2 - libp2p gossipsub testing*, Accessed: 2022-03-01. [Online]. Available: https://github.com/whiteblock/gossipsub-testing.

[59] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, *Gossipsub-v1.1 evaluation report*, 2022.

[60] A. Kumar, M. von Hippel, P. Manolios, and C. Nita-Rotaru, "Formal model-driven analysis of resilience of gossipsub to attacks from misbehaving peers," in *2024 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, 2024, pp. 21–21. DOI: 10.1109/SP54263.2024.00017. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00017.

[61] *Run rippled as a validator*, Accessed: 2022-07-28. [Online]. Available: https://xrpl.org/run-rippled-as-a-validator.html.

[63] *Xrpl - introduction to consensus*, Accessed: 2022-09-13. [Online]. Available: https://xrpl.org/intro-to-consensus.html.

[64] M. Goldin, *Token-curated registries 1.0*, Accessed: 2022-09-13, 2017. [Online]. Available: https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7.

[65] M. Lockyer, *Token curated registry (tcr) design patterns*, Accessed: 2022-10-19, 2018. [Online]. Available: https://hackernoon.com/token-curated-registry-tcr-design-patterns-4de6d18efa15.

[66] D. de Jonghe and F. Gong, *The layered tcr*, Accessed: 2022-10-19, 2018. [Online]. Available: https://blog.oceanprotocol.com/the-layered-tcr-56cc5b4cdc45.

[67] S. de la Rouviere, *Continuous token-curated registries: The infinity of lists*, Accessed: 2023-02-15, 2017. [Online]. Available: https://shorturl.at/bAISZ.

[69] K. Ito and H. Tanaka, “Token-curated registry with citation graph,” *arXiv preprint arXiv:1906.03300*, 2019.

[70] *Negative unl - reliability measurement*, Accessed: 2023-02-14. [Online]. Available: https://xrpl.org/negative-unl.html#reliability-measurement.

[71] M. L. Bynum, G. A. Hackebeil, W. E. Hart, *et al.*, *Pyomo–optimization modeling in python*, Third. Springer Science & Business Media, 2021, vol. 67.

[72] W. E. Hart, J.-P. Watson, and D. L. Woodruff, “Pyomo: Modeling and solving mathematical programs in python,” *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[73] I. I. Cplex, “V12. 1: User’s manual for cplex,” *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[74] C. Zhao, T. Wang, and S. Zhang, “Lightblock: Reducing bandwidth required to synchronize blocks in ethereum network,” in *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, IEEE, 2021, pp. 868–874.

[76] *Introduction to grpc*, Accessed: 2022-09-13. [Online]. Available: https://grpc.io/docs/what-is-grpc/introduction/.

[77] A. Demers, D. Greene, C. Hauser, *et al.*, “Epidemic algorithms for replicated database maintenance,” in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 1–12.

[78] G. Tsipenyuk and N. D. Bougalis, *Message routing optimizations, pt. 1: Proposal & validation relaying*, Accessed: 2023-01-26, Mar. 2021. [Online]. Available: https://xrpl.org/blog/2021/message-routing-optimizations-pt-1-proposal-validation-relaying.html.

[79] J.-P. Eisenbarth, T. Cholez, and O. Perrin, “Ethereum’s peer-to-peer network monitoring and sybil attack prevention,” *Journal of Network and Systems Management*, vol. 30, no. 4, p. 65, 2022.

[80] A. de la Rocha, *Playing with gossipsub*, Accessed: 2022-02-01. [Online]. Available: https://adlrocha.substack.com/p/adlrocha-playing-with-gossipsub.

[81] *Ethereum upgrades*, Accessed: 2023-01-26. [Online]. Available: https://ethereum.org/en/upgrades/.

[82] *Ethereum 2.0 networking specification*, Accessed: 2022-07-19. [Online]. Available: https://github.com/goerli/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md.

[83] *Libp2p implementations*, Accessed: 2022-03-01. [Online]. Available: https://libp2p.io/implementations/.

[84] *What is xrp? - overview*, Accessed: 2023-10-05. [Online]. Available: https://xrpl.org/xrp-overview.html.

[85] J. Hartigan, "Asymptotic distributions for clustering criteria," *The Annals of Statistics*, pp. 117–131, 1978.

[86] P. H. Sneath, "Numerical taxonomy," in *Bergey's manual® of systematic bacteriology*, Springer, 2005, pp. 39–42.

[87] J. Pearl and D. Mackenzie, *The book of why: the new science of cause and effect*. Basic books, 2018.

[88] J. Siebert, "Applications of statistical causal inference in software engineering," *Information and Software Technology*, p. 107 198, 2023.

[89] H. Hours, E. Biersack, and P. Loiseau, "A causal approach to the study of tcp performance," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 7, no. 2, pp. 1–25, 2015.

[90] P. Spirtes, C. N. Glymour, and R. Scheines, *Causation, prediction, and search*. MIT press, 2000.

[91] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.

[92] F. Neves, N. Machado, R. Vilaça, and J. Pereira, “Horus: Non-intrusive causal analysis of distributed systems logs,” in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2021, pp. 212–223.

[93] S. Kobayashi, K. Shima, K. Cho, O. Akashi, and K. Fukuda, “Comparative causal analysis of network log data in two large isps,” in *NOMS 2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2022.

[94] R. Jarry, S. Kobayashi, and K. Fukuda, “A quantitative causal analysis for network log data,” in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, 2021, pp. 1437–1442.

[95] D. S. Kim, H. Shinbo, and H. Yokota, “An alarm correlation algorithm for network management based on root cause analysis,” in *13th International Conference on Advanced Communication Technology (ICACT2011)*, IEEE, 2011, pp. 1233–1238.

[96] A. Molak, *Causal Inference and Discovery in Python: Unlock the secrets of modern causal machine learning with DoWhy, EconML, PyTorch and more*. Packt Publishing Ltd, 2023.

[97] J. Peters, D. Janzing, and B. Schölkopf, *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[98] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using networkx,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[99] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, Springer, 2002, pp. 483–484.

[100] A. Sharma and E. Kiciman, “Dowhy: An end-to-end library for causal inference,” *arXiv preprint arXiv:2011.04216*, 2020.

[101] K. Zhang, S. Zhu, M. Kalander, *et al.*, *Gcastle: A python toolbox for causal discovery*, 2021. arXiv: 2111.15155 [cs.LG].

[103] P. O. Hoyer and A. Hyttinen, "Bayesian discovery of linear acyclic causal models," *ArXiv*, vol. abs/1205.2641, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:11717717.

[104] D. M. Chickering, "Optimal structure identification with greedy search," *Journal of machine learning research*, vol. 3, no. Nov, pp. 507–554, 2002.

[105] D. Chickering, "Learning equivalence classes of bayesian-network structures," *The Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.

[106] X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing, "Dags with no tears: Continuous optimization for structure learning," *Advances in neural information processing systems*, vol. 31, 2018.

[107] C. Glymour and K. Zhang, "Review of causal discovery methods based on graphical models," *Frontiers in genetics*, vol. 10, p. 418 407, 2019.

[108] M. Kalisch and P. Bühlman, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm.," *Journal of Machine Learning Research*, vol. 8, no. 3, 2007.

[109] T. D. Le, T. Hoang, J. Li, L. Liu, H. Liu, and S. Hu, "A fast pc algorithm for high dimensional causal discovery with multi-core pcs," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 5, pp. 1483–1495, 2016.

[110] D. Colombo, M. H. Maathuis, *et al.*, "Order-independent constraint-based causal structure learning.," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3741–3782, 2014.

[111] S. Shimizu, P. O. Hoyer, A. Hyvärinen, A. Kerminen, and M. Jordan, "A linear non-gaussian acyclic model for causal discovery.," *Journal of Machine Learning Research*, vol. 7, no. 10, 2006.

[112] S. Shimizu, T. Inazumi, Y. Sogawa, *et al.*, "Directlingam: A direct method for learning a linear non-gaussian structural equation model," *Journal of Machine Learning Research-JMLR*, vol. 12, no. Apr, pp. 1225–1248, 2011.

[113] Z. Fang, S. Zhu, J. Zhang, Y. Liu, Z. Chen, and Y. He, "On low-rank directed acyclic graphs and causal structure learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[114] I. Ng, A. Ghassami, and K. Zhang, "On the role of sparsity and dag constraints for learning linear dags," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 943–17 954, 2020.

[115] D. Janzing and B. Schölkopf, "Causal inference using the algorithmic markov condition," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 5168–5194, 2010.

[116] *Dowhy: Evaluate a gcm*, Accessed: 2024-03-20. [Online]. Available: `https://www.pywhy.org/dowhy/v0.11.1/user_guide/modeling_gcm/model_evaluation.html`.

[117] E. Eulig, A. A. Mastakouri, P. Blöbaum, M. Hardt, and D. Janzing, "Toward falsifying causal graphs using a permutation-based test," *arXiv preprint arXiv:2305.09565*, 2023.

[118] *Dowhy: Graph refutations*, Accessed: 2024-03-20. [Online]. Available: `https://www.pywhy.org/dowhy/v0.11.1/user_guide/modeling_causal_relations/refuting_causal_graph/refute_causal_structure.html`.

[119] D. Janzing, D. Balduzzi, M. Grosse-Wentrup, and B. Schölkopf, "Quantifying causal influences," *The Annals of Statistics*, vol. 41, no. 5, Oct. 2013, ISSN: 0090-5364. DOI: 10.1214/13-aos1145. [Online]. Available: http://dx.doi.org/10.1214/13-AOS1145.

[120] D. Janzing, P. Blöbaum, A. A. Mastakouri, P. M. Faller, L. Minorics, and K. Budhathoki, *Quantifying intrinsic causal contributions via structure preserving interventions*, 2024. arXiv: 2007.00714 [cs.AI].

[121] X. Zheng, C. Dan, B. Aragam, P. Ravikumar, and E. Xing, "Learning sparse nonparametric dags," in *International Conference on Artificial Intelligence and Statistics*, Pmlr, 2020, pp. 3414–3425.

[122] *Dowhy: Modeling graphical causal models (gcms)*, Accessed: 2024-04-23. [Online]. Available: https://www.pywhy.org/dowhy/v0.11.1/user_guide/modeling_gcm/index.html.

[123] *Gossipsub v1.1: Security extensions to improve on attack resilience and bootstrapping*, Accessed: 2024-07-25. [Online]. Available: https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md.

[126] J. McCaleb, *Bitcoin without mining*, Accessed: 2024-03-11, 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=10193.0.