Pub/sub Dissemination on the XRP Ledger

Flaviene Scheidt de Cristo*, Wazen M. Shbair*, Lucian Trestioreanu* and Radu State*
* University of Luxembourg, SnT, 29, Avenue J.F Kennedy, L-1855 Luxembourg
Email:{flaviene.scheidt, wazen.shbair, lucian.trestioreanu, radu.state}@uni.lu

***Abstract*—The XRP Ledger is one of the oldest and most well-established blockchains, using a particular type of consensus mechanism that differs from the typical Proof of Work and Proof of Stake. The underlying p2p network uses flooding to disseminate certain types of messages during the consensus rounds, leading to performance and scalability issues. In this work, we propose the use of publisher/subscriber dissemination on the XRPL using GossipSub to diminish the message overhead. We use Flexi-pipe, a tool that allows the integration between the XRPL validator and GossipSub, to evaluate the improvements brought.**

***Index Terms*—blockchain, unstructured p2p, gossipsub, XRPL**

I. Introduction

The XRP Ledger (XRPL) is the fourth most valuable blockchain in market capitalization[1], being one of the first to propose a novel consensus algorithm that largely differs from Bitcoin's *Proof-of-Work* (PoW). Instead, the XRPL relies upon a quorum-based Byzantine Fault-tolerant agreement protocol known as the XRP Ledger Consensus Protocol (XRP LCP) [1]. The XRP LCP is based on subnetworks that are collectively trusted to not collude to defraud, fork, or stall the ledger. Contrary to PoW and *Proof-of-Stake* (PoS), it does not rely on computational power or staking to select validators per round but uses the votes of trusted groups of validators to reach consensus.

While voting may seem like a slower strategy, in reality, it gives the XRPL an advantage regarding transaction speed. On average, the XRPL publishes a new ledger version every 3 seconds handling 18 transactions per second (TPS) [2], with a reported capacity of 1.5k TPS [3]. Whereas Bitcoin generates new blocks every 10 minutes with a maximum throughput of 7 TPS [4]. As for Ethereum, after the migration to PoS, the average block time is 12 seconds with 11 TPS [5], being the maximum throughput reported between 15 and 45 TPS [6].

In more detail, the XRP LCP employs *Unique Nodes Lists* (UNLs) to form the subnetworks that confabulate to reach consensus. Each UNL is trusted collectively to not collude to harm the ledger, meaning that no validator is trusted individually. Every node proclaims its trust in a UNL and listens to the positions of the peers in the list to cast their own votes.

The efficiency of the consensus process leans heavily on the performance of the communication between nodes. In most Blockchains, poor network performance may facilitate double-spend attacks [7] and even lead to forks. The XRPL, however, has mechanisms for fork prevention [8] and disagreements may stall the progress, thus violating the *liveness* of the ledger [1]. To ensure that all messages are delivered and no node gets eclipsed, the XRPL employs *flooding* to broadcast the votes and the final ledger position of each validator. The trade-off of flooding is that the message overhead created by the replication harms the performance of the network, causing latency and scalability issues [9].

In this work we propose the use of a *publisher/subscriber* (pub/sub) system to disseminate messages on the XRPL, taking advantage of the subnetworks system used in the consensus process. We integrate the XRPL with GossipSub [10], a state-of-the-art framework for pub/sub dissemination on blockchains. Using a tool called *Flexi-pipe* we identify the bottlenecks in message overhead created by the use of flooding in the consensus process and analyze the impact of using GossipSub to diminish or eliminate these bottlenecks.

II. Background

Several works analyze and discuss the characteristics of the XRP LCP. Schwartz et al. [8] present the first official whitepaper describing the XRP LCP algorithm and also introducing the concept of Unique Node Lists (UNL). The XRPL Foundation, however, recommends Chase and McBrough [1] as the current official whitepaper, presenting a more thoughtful analysis of safety and liveness, inferring that at least 90% of agreement between the validators is needed to ensure network safety. Additionally, the authors also suggest that the underlying message dissemination pattern may have the potential to leverage some safety concerns. McBrough also proposes later Cobalt [11], a novel atomic broadcast algorithm. Different from our work, Cobalt requires changes in the consensus fabric, while our proposal changes solely the way messages are broadcasted during two specific states of the consensus process.

Subsequent works from Mauri et al. [12] and Amores-Sesar et al. [13] walk in the direction of more formal descriptions and bring new cases that may cause the network to break or to cease making forward progress, thus violating safety and liveness. Christodolou [14] tackles the UNL overlapping problem using an empirical approach, showing that the minimum UNL overlap can be relaxed when there are less than 20% of malicious nodes present. This work suggests space for optimizations on the minimum UNL overlap, suggesting also the necessity of a malicious node estimator. Ripple+ [15] jumps into this gap and brings a mechanism for selecting UNLs based on a structure of core and leaf nodes.

[1]As of 24/January/2023 on the coin ranking by market capital available at https://cryptoslate.com/coins/, excluding stablecoins and non-native tokens.

It is important to note that these works mainly address the problem of ensuring liveness and safety while keeping the conceptual trust structure intact. In this work, we focus on mitigating the message overhead caused by the use of flooding without compromising the safety and the liveness of the ledger. Our work also focuses on proposing a dissemination strategy that better suits the way the XRP LCP works, without altering the consensus fabric.

A. XRPL Consensus Protocol

XRP LCP uses voting to reach consensus; Every node in the network that self-proclaims itself as a validator has the right to cast votes. The XRP LCP works upon the idea of *subjective validators* instead of relying on established models for *Byzantine Agreement* and *Byzantine fault-tolerance* [13], meaning that each participant takes into account the position of its trusted peers to cast its votes.

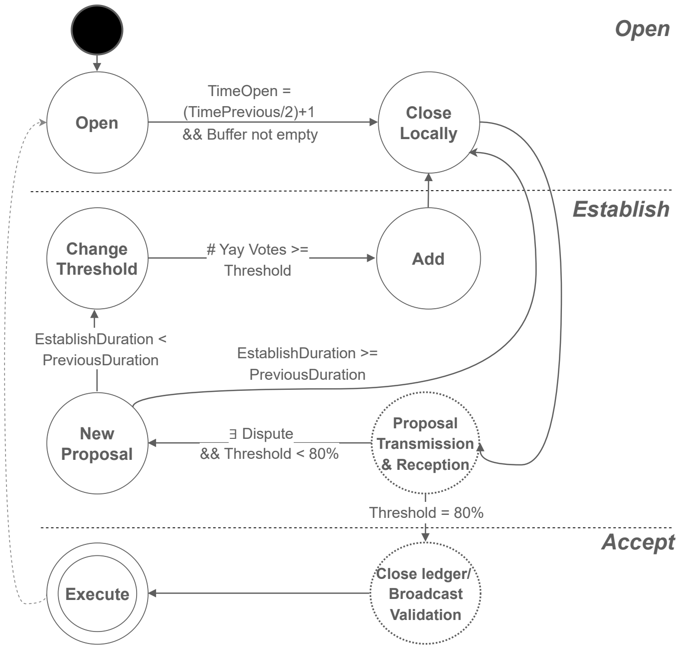
To better understand the proper functioning of the XRP LCP we need to take into account the following concepts:

- **rippled:** Official C++ implementation of the XRPL validator[2].
- **UNL:** Static list containing the validators a node trusts collectively not to collude.
- **Ledger:** The record of all the transactions on the network [8]. Formally, for the XRP LCP, the ledger also represents the *shared distributed state* [16].
- **Node:** A participant in the network. A machine running the *rippled* code as a *server*.
  - **Validator:** A node that casts votes.
- **Message:** Any message sent or received by a node.
  - **Transaction:** Any message that may cause a change in the shared-state.
  - **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* a node reaches after multiple consensus rounds.
- **Position:** The current proposal of a given node.
  - **Vote:** The position of a node on a given transaction.
  - **Dispute:** A transaction that is present in a node's position but not in the position of one or more of its peers - or vice versa.

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 shows a simplified view of each phase, with its internal states and transitions considering only the states related to the voting and ledger creation, abstracting all the states related to the syncing and ledger closing times since those are out of the scope of this paper.

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.

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



Open
Open
TimeOpen = (TimePrevious/2)+1 && Buffer not empty
Close Locally
Establish
Change Threshold
# Yay Votes >= Threshold
Add
EstablishDuration < PreviousDuration
EstablishDuration >= PreviousDuration
New Proposal
∃ Dispute && Threshold < 80%
Proposal Transmission & Reception
Threshold = 80%
Accept
Execute
Close ledger/ Broadcast Validation

Fig. 1. State Machine of the XRP LCP, demonstrating the internal states of each phase with the respective transitions and its triggers.

The protocol now enters the *establish* phase with 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 going up until it reaches 80%.

During the *establish* phase, the proposal formed previously is transmitted to the entire network via *flooding*. At the same time, the node receives proposals from its peers; if there is any *dispute* and the threshold is lower than 80%, the validator starts to update its position by creating a new proposal. First, the threshold is recalculated based on the time left and then the voting starts: For each transaction on the disputed set, the node will consider the position of each one of its peers.

With the new proposal formed, the ledger is closed locally, returning to the *open* phase; if the threshold reaches 80% or there is no dispute, the *accept* phase starts by transforming the proposal into a validation and transmitting it to the network via *flooding*. The nodes must now agree upon which validation to apply. Considering 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 lost sync and must start the syncing process again.

B. The XRPL Network

Now that we conceptualized how the XRP LCP works, we can present the current structure of the XRPL mainnet. In this work, we focus on a level of abstraction called *trust overlay*. This overlay comprises the trust relations between nodes.

The UNLs dictate the topology of the *trust overlay*. As discussed previously, the XRPL network requires a minimum overlap between every two UNLs. In the production mainnet, the XRP Ledger Foundation recommends the use of one of three UNLs curated by Ripple Labs, Coil, or the XRP Ledger Foundation (XRPLF). It is important to note that this recommendation comes to keep the liveness and safety properties in the network, and not to enforce centralization.

As of February 2023, those 3 UNLs contained the same 34 validators. Considering that all the validators trust the same UNL, the center of the XRPL comprises a complete graph of 34 nodes using flooding to disseminate proposals and validations.

III. Problem Definition

The state machine presented in Figure 1 highlights the two states in which messages are propagated by flooding: *Proposal Transmission & Reception* and *Close ledger & Broadcast Validation*, being both of them represented by dashed lines. The idea is that transactions are spread via *gossiping* [17], having no need for a strong assurance of delivery since later they will be grouped into proposals. The use of flooding to disseminate proposals works as a reconciliation phase, guaranteeing that all nodes will see all the 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 network, the excessive traffic caused by the message overhead leads to higher latency and higher bandwidth usage of individual nodes.

Tsipenyuk et al. [9] first introduced this problem by presenting how the number of proposals and validations represents 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 from regarding a particular validator and tells the other nodes to "squelch" the connection for a given amount of time. This method was evaluated by the authors using a simulated graph structure showing a reduction of 76% in the total amount of messages in the network.

To further emphasize the problem, we present in Figure 2 the frequency of the reception of replicated validations on a single node. For this analysis, we used the tool presented in Section VI in a testnet of 24 nodes fully connected. This structure mimics the way the trust overlay of the XRPL Network is structured.

The x-axis presents every possible number of replicas a node may 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, considering that the node has 23 peers. The y-axis presents the number of single messages received, meaning around 6000 messages were received 23 times, as indicated by the red arrow.

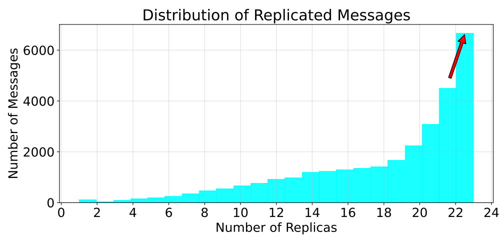


Fig. 2. Frequency of duplicated messages on a testnet with 24 nodes fully connected

What we take from this experiment is that flooding is a highly reliable method for disseminating messages, but causes a distribution of the number of replicas far from the ideal. There needs to be a tradeoff between the delivery assurance and the number of replicas to guarantee the scalability of the network. In the next Section, we present the state-of-the-art for message dissemination on Blockchains, giving strong assurance of delivery while maintaining a low message overhead.

IV. GossipSub

Flooding is a straightforward manner of disseminating messages reliably on unstructured p2p networks, such as used on blockchains, being a robust but not particularly scalable method, since messages are sent redundantly, increasing latency. A solution proposed uses pub/sub systems. At a high level, those systems are composed of, as the name suggests, *publishers* and *subscribers*, being the first role played by nodes that declare interest in a given topic and the second, nodes that publish messages on these topics. On pub/sub systems messages are also called *events*, and the declaration of interest on a topic, a *subscription* [18].

Pub/sub systems are, however, complicated to be implemented on unstructured p2p networks. The most notable work on this matter is *GossipSub*, originally proposed to be incorporated into FileCoin and Ethereum [19], and although both of these could be considered structured because of the employment of Distributed Hash Tables (DHTs), in reality, they behave as unstructured p2p networks, using DHTs solely as a peer discovery mechanism [20]. GossipSub is a gossip-based pub/sub protocol for message dissemination in p2p overlays [10] and is distributed as an extensible component inside libp2p [21].

Gossipsub has two main characteristics that make it highly suitable for the XRPL, as they guarantee fast and lighter dissemination while being attack-resistant.

1) **Mesh Construction:** The mesh construction employs *eager push*, keeping the delivery fan-out low to balance the bandwidth usage. But it also keeps 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 one, used for eager push, is a logic full-message network specified as a *local mesh*. Each node has its 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 [22].

2) **Scoring Function:** GossipSub employs a scoring function to help identify byzantine behavior. The scoring is local, meaning that every node keeps an internal score of its peers and makes routing and relaying decisions based on this scores [10]. The scoring function, however, is beyond the scope of this work.

More extensive analysis of the attack-resilience can be found in [10], and a performance benchmark in [23] and [24]. In the next section, we discuss how we mapped the XCP mechanism into GossipSub, explaining our design choices.

V. Proposal

GossipSub is the state-of-the-art for efficient message dissemination on blockchains. Besides being used by Filecoin, it has also been deployed on Ethereum in September of 2022 [25]. Both systems employ the framework similarly; Filecoin uses two topics: one to propagate messages, and another to propagate blocks. Ethereum structures the topics in a more sophisticated manner, with five global topics, two primary and three secondaries [26]. The first is a beacon that broadcasts newly *signed blocks* to the entire network, while the second propagates *aggregated attestations* to subscribed validators. The three secondary topics propagate respectively *voluntary exit*, *proposer slashing*, and *attester slashing*.

The employment of UNLs in the XRP LCP allows for some interesting forms to use pub/sub for disseminating messages. We present three setups based on how the network is structured in production and how it has been conceptualized. We propose the creation of an *pub/sub overlay* based on the existing *trust overlay*. And so we can keep the XRP LCP algorithm as it is, changing solely the abstract layer that creates trust relationships between nodes.

(a) First, we use GossipSub similarly to Filecoin and Ethereum, keeping the XRPL overlay characteristics. The idea is to have two topics according to the message types that need to be broadcasted to the entire network; The first topic refers to *proposals* and the second one to *validations*. Creating a parallel view with Ethereum, *validations* work similarly as 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 duplicated messages and thus making the structure more scalable.

(b) The XRP LCP, however, 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 pub/sub system. Meaning that we can understand a 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), there is not much sense in keeping a fully connected mesh of validators, for it would create a similar structure as seen in the 2-topics approach.

(c) The previous approach may cause some issues since the XRP LCP has a condition to keep liveness related to the minimum required UNLs overlap. Considering that all nodes on the network may choose their own UNL, but not all can agree to a minimum of 60% of common nodes. In this condition, the ledger would not be able to make any progress, as proved in the previous works discussed. We can reach another solution that will keep the liveness of the network while also maintaining the pub/sub characteristics of the XRP LCP. This solution is called *1 topic per UNL* (1-topic/UNL) and involves having several pre-defined UNLs that nodes can choose from and structuring the *pub/sub overlay* to abstract each UNL as a topic.

VI. Methodology

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

A. 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. Meaning that, from the point-of-view of the implementation, we have two layers: the *rippled overlay* and the *gossipsub overlay*, conceptually they represent the *trust overlay* and the *pub/sub overlay*. The *rippled overlay* is the existing layer in which the validator exchanges messages with its peers and the entire network. In contrast, the *gossipsub overlay* is the novelty we add.

The communication between both layers works by employing *remote procedure calls* (RPCs). Figure 3 demonstrates the schematics used to implement the Plug&Play Broadcast. Each big rectangle represents a node, inside of them we have the two layers: rippled and GossipSub.

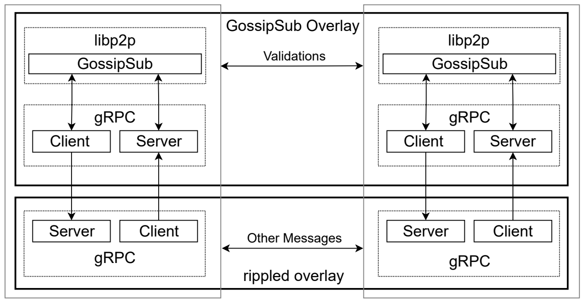


Fig. 3. Diagram of the communication between two nodes using GossipSub plugged through gRPC to disseminate validations.

The bottom layer is the rippled code version 1.7, written in C++, and comprises the *rippled overlay*. We changed the code to include a gRPC node containing a client and a server. We also deactivated the node's ability to send and relay messages of a certain type - in this work, validations - but kept all the other functions intact. So all other types of messages still transit at this level, also adding a new logging feature to the rippled code[3]. The goal of this feature is to provide meanings for better analyzing - and debugging - the integration of the two overlays.

The *gossipsub overlay*[4] is the top layer containing the broadcast mechanism connected to the validator. This layer contains a component controlling the creation and maintenance of the overlay - and the subscription to topics. Inside this component, we have another gRPC node with a server and a client. Upon receiving validations, the GossipSub nodes push them to rippled to be processed.

As shown in Figure 3, we used GossipSub as a module inside libp2p[5]. The reason is that GossipSub acts as a router for messages using pub/sub 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. This work, however, 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 used are the reference implementations, written in go without modifications. Those were the implementations chosen for they are better documented and reported as the most stable [27].

VII. Evaluation

A. Experimental setup

The experimental setup for this work 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 changed to only write logs for analytics - which we call vanilla - and another modified to work with Flexi-pipe. Both versions were based on the official 1.7 release.

We separated the experiments into two categories, according to the proposal described in Section V. First, we considered the 1-topic approach using a fully connected structure for the *trust overlay*. 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 *pub/sub overlay*. The *trust overlay* is a randomly generated graph with a median degree of 16 and a maximum degree of 18, so every node listens to at least 60% of the network. We built this structure in this form to keep 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.

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

[4]Available at https://github.com/FlavScheidt/gossipGoSnt

[5]https://github.com/libp2p/go-libp2p

B. Experiments and Results

In this work, we focused on minimizing the overhead generated by the message dissemination strategy used on the XRPL. To evaluate our proposal, we analyzed the number of duplicated validations received by one particular node.

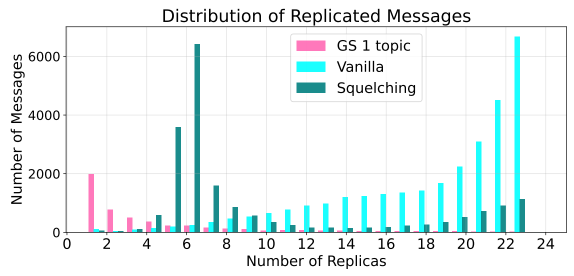


Fig. 4. Comparison of the frequency of replicated messages on vanilla rippled in a fully connected structure and GossipSub using a 2-topics setup

Figures 4 and 5 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 2.

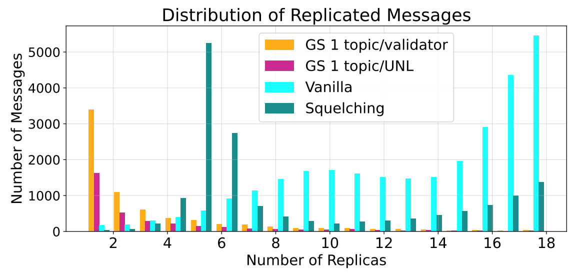


Fig. 5. Comparison of the frequency of replicated messages on vanilla rippled in a 16-degree structure and GossipSub using a 1-topic/validator and 1-topic/UNL setups

In both setups, vanilla presents a behavior that is suboptimal because of its conservative approach - especially on the fully connected structure of Figure 4. 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 node on the UNL at each squelching round. The behavior is better than pure vanilla, but is still suboptimal 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 5, 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. This limitation would not exist, being GossipSub 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 kept progressing successfully. From the experiments, 1-topic/validator was the most beneficial setup. 1-topic/UNL also had good results, but a limitation in the tool harmed the evaluation. The scenario that simulates more closely the actual structure of the XRPL network presented a good improvement, showing that GossipSub may be a good alternative for improving the performance.

VIII. Conclusion & Future Work

Our work conceptualized the XRP LCP. We presented and analyzed the state machine and pointed to the bottlenecks caused by the use of *flooding*. Notably, the points identified can be a cause of scalability issues by disseminating too many replicated messages.

Further, we introduced Flexi-Pipe, a tool for analyzing the message dissemination pattern of targeted message types. Flexi-Pipe also allowed us to plug a different dissemination technique into an XRPL testnet. Following the natural pub/sub characteristics of the XCP, we plugged GossipSub into the validator using our tool. The GossipSub framework is state-of-the-art for message dissemination in unstructured 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 XCP.

This work, however, makes no safety considerations, and further analysis of threats in the proposed scenarios is necessary. Considerations about UNL overlapping in more complex scenarios are also of importance for the continuity of this work. Being that the second scenario proposed is straightforwardly a pub/sub network, presenting the best results regarding message replication.

Acknowledgment

We thankfully acknowledge the support from the RIPPLE University Blockchain Research Initiative (UBRI) for our research.

References

[1] B. Chase and E. MacBrough, "Analysis of the xrp ledger consensus protocol," 2018. [Online]. Available: https://arxiv.org/abs/1802.07242

[2] "Xrpl charts," accessed: 2023-01-26. [Online]. Available: https://livenet.xrpl.org/

[3] "Xrp: Utility for the new global economy," accessed: 2023-01-26. [Online]. Available: https://ripple.com/xrp/

[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer *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.

[5] "Ethereum average block time chart," accessed: 2023-01-26. [Online]. Available: https://etherscan.io/chart/blocktime

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

[7] W. Hao, J. Zeng, X. Dai, J. Xiao, Q. S. Hua, H. Chen, K. C. Li, and H. Jin, "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, 6 2020.

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

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

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

[11] E. MacBrough, "Cobalt: Bft governance in open networks," 2 2018. [Online]. Available: http://arxiv.org/abs/1802.07240

[12] L. Mauri, S. Cimato, and E. Damiani, "A formal approach for the analysis of the xrp ledger consensus protocol." SciTePress, 2020, pp. 52–63.

[13] I. Amores-Sesar, C. Cachin, and J. Mićić, "Security analysis of ripple consensus," 2020. [Online]. Available: https://arxiv.org/abs/2011.14816

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

[15] 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, 2022.

[16] M. Ellery and I. Ashimine, "Consensus and validation," accessed: 2021-10-14. [Online]. Available: https://github.com/ripple/rippled/blob/develop/docs/consensus.md

[17] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 1–12.

[18] R. Baldoni, L. Querzoni, and A. Virgillito, *Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey*. Springer Berlin Heidelberg, 2009.

[19] 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

[20] 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.

[21] "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

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

[23] TrentonVanEpps, "Testing gossipsub with genesis," accessed: 2022-02-01. [Online]. Available: https://medium.com/whiteblock/testing-gossipsub-with-genesis-6f89e845b7c1

[24] "Eth2 - libp2p gossipsub testing," accessed: 2022-03-01. [Online]. Available: https://github.com/whiteblock/gossipsub-testing

[25] "The ethereum upgrades," accessed: 2023-01-26. [Online]. Available: https://ethereum.org/en/upgrades/

[26] "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

[27] "libp2p implementations," accessed: 2022-03-01. [Online]. Available: https://libp2p.io/implementations/