A 9-dimensional Analysis of GossipSub over the XRP Ledger Consensus Protocol

Flaviene Scheidt de Cristo*, Jorge Augusto Meira*, Jean-Philippe Eisenbarth*, Radu State*
* University of Luxembourg, SnT, 29, Avenue J.F Kennedy, L-1855 Luxembourg
Email:{flaviene.scheidt, jorge.meira, philippe.eisenbarth, radu.state}@uni.lu

***Abstract*—This work shows a 9-dimensional analysis of the behavior and performance of Gossipsub according to the parameterization of the mesh. GossipSub is the state-of-the-art framework for safe and fast message dissemination on blockchains. This study focuses on its application in the quorum-based consensus of the XRP Ledger. Using data science techniques, we cluster sets of parameters, identify correlations between dimensions, and develop a decision tree for informed parameter selection. Contributions include demonstrating the adaptability of GossipSub for alternative consensus models, analyzing how parameters impact network performance and behavior, and providing a method for parameterization to meet specific network requirements.**

***Index Terms*—XRPL, unstructured p2p, pubsub, blockchain, GossipSub**

I. Introduction

One of the biggest challenges in scaling blockchains is how messages and blocks are disseminated in the underlying peer-to-peer (p2p) network. Bitcoin, for example, uses *flooding* to propagate events that update the ledger state in all nodes [1]. It is natural for message overhead and bandwidth requirements to increase with the number of nodes and transactions [2], creating a low performance environment and opening doors for possible double-spending attacks [3] and forks in the ledger.

GossipSub is the state-of-the-art in message and block dissemination in blockchains, using a *publisher/subscriber* (pubsub) model over an unstructured p2p network and a gossip overlay to reach distant nodes. The solution also employs a score function to identify Byzantine behavior and some mitigation strategies to alleviate threats to the network.

GossipSub has already been deployed in the Interplanetary File System (IPFS), as well as in blockchains such as Filecoin and Ethereum. To our knowledge, not much research has been done on tuning the mesh parameters of GossipSub to suit different types of consensus. In this work, we address this gap by studying how different parameters impact performance and behavior on a blockchain that uses an alternative type of consensus, different from the more mainstream *Proof-of-Work* (PoW) and *Proof-of-Stake* (PoS). We use data science techniques to group, categorize and analyze sets of parameters to determine the network configuration considering the topology and desired behavior and performance.

We connected GossipSub to the XRP Ledger (XRPL) using our prior work, Flexi-pipe [4]. The XRPL uses a quorum-based voting system to achieve consensus, employing lists of validators to determine trust. Each validator has a list of peers they trust not to collude to defraud the ledger. The underlying structure of those lists works as a pubsub system in itself and thus is ideal for our analysis. It also allows for different topic configurations, opening more opportunities to study the behavior of GossipSub according to its parameters.

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. We employ data gathered from successive executions on a private testnet, using 9 dimensions to understand the behavior of GossipSub according to different parameterizations. Three of these dimensions are measures of performance: *Bandwidth*, *PropagationTime* and *Message Overhead*, while the remaining represent events on the network, namely *MessageReceived*, *MessageDuplicated*, *graft*, *prune*, *iwant* and *ihave*.

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 contributions of this paper are the following:

1) How different sets of parameters influence the creation of the mesh and the network performance regarding *message overhead*, *bandwidth*, and *average propagation time* and behavior regarding the number of messages, number of messages duplicated, prune and graft events and iwant and ihave messages.
2) A study of the correlation between the dimensions discussed.
3) And finally, a tool to help parameterize GossipSub according to the desired performance and behavior of the network.

II. Related Work

The GossipSub protocol has been implemented and used in practice by popular distributed systems such as Ethereum, Filecoin, and the IPFS p2p network. GossipSub was first presented by Dimitris Vyzovitis and Yiannis Psaras from

Protocol Labs in 2019 [5]. This work establishes the basis for this new content-based pubsub protocol. The goal of GossipSub is to scale pubsub dissemination without excessive bandwidth or 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, the roles of which are explained in Section III.

Surprisingly, no academic work addresses the performance and scalability of GossipSub. Only one non-academic study [6], carried out by the Whiteblock company, proposes an evaluation of this problem. The details and source code can be found on GitHub [7]. It uncovered message losses and inconsistent delays due to implementation details in Golang. Furthermore, they tested the message dissemination time under 400ms latency delay and discovered that it is effective for Ethereum 2.0 deployment.

Orthogonally, Vyzovitis et al. [8] presented a detailed analysis of the resilience to Sybil and Eclipse attacks in distributed systems implementing GossipSub. They showed that under certain circumstances the performance decreases, but the protocol does not break. They proposed three mitigation strategies: *flood publishing*, *opportunistic grafting*, and *increased gossip propagation*. These strategies significantly enhance the performance of the protocol in the most adversarial environments. This work also explains each individual mesh parameter and provides some guidelines for choosing values based on the desired performance outcome. The default parameters are set to preserve low fan-out and reduce bandwidth requirements while still achieving an acceptable message propagation time. However, there is only a limited sensitivity analysis of how some of the mesh parameters may affect 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.

III. Background

A. 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, such as Bitcoin. While flooding ensures reliable message delivery in unstructured p2p networks, it also leads to redundancy and delays [9]. GossipSub addresses this challenge by introducing a solution rooted in 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* [10].

However, implementing pubsub systems on unstructured p2p networks presents a complex task. The GossipSub protocol is particularly prominent in this area. GossipSub introduces a gossip-driven pubsub protocol to enable efficient message dissemination in these p2p overlays [8] and is distributed as an extensible component within the libp2p library [11].

GossipSub is built on two main components, 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 the balance it brings between bandwidth consumption and performance [8]. The score function is used to rank peers according to their behavior on the network and flag malicious activities. All nodes in 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 operation against malicious activity.

TABLE I
GossipSub Mesh Parameters

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

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

The mesh is a global structure in which every node has only a local view. Nodes are connected to a certain number of peers, sending and relaying full messages solely to these peers (eager push). 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 connected, defining the average fanout of the network. *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 one is a new connection and occurs when the number of peers falls below *Dlo*. The node chooses 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, and the remaining are randomly selected. When a pruning event occurs, the node should retain at least *Dout* number of peers (greedy pushing).

To guarantee that a message will reach all its targets,

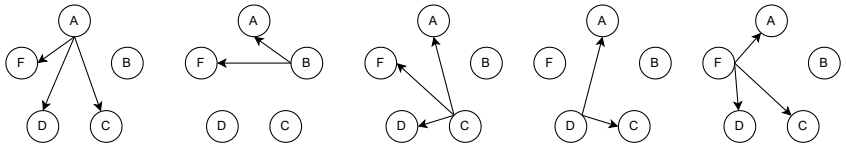

Fig. 1. Representation of the UNLs in a network with 5 validators. Arrows represent the directional trust relationship between two nodes.

GossipSub employs lazy pull. Nodes can communicate with other nodes outside their local view by gossiping. In this scenario, the nodes exchange announcements such as "I have message x" (*ihave*) and "I want message x" (*iwant*). The main parameters related to lazy pull are *Dlazy* and *GossipFactor*, both controlling the number of peers 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. Table I summarizes the parameters explained.

It is possible to visualize the GossipSub mesh as being built by two overlays, one that propagates full messages by eager push and the other that employs gossiping to reach distant nodes, using lazy pull. *D*, *Dlo*, *Dhi*, *Dscore*, and *Dout* are all linked to the first overlay, 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.

B. The XRP Ledger

The XRP Ledger was one of the first blockchains released [12], implementing a consensus mechanism that is significantly different from those employed by its more mainstream counterparts. Bitcoin and Ethereum, for example, employ PoW [13] and PoS[1] [14] respectively. Both mechanisms are based on the concept of selecting a validator in each round who will decide the next block to be added to the chain. 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. On the other hand, the XRP Ledger Consensus Protocol (XRP LCP) relies on groups of validators that come to a consensus at each round to decide the next version of the ledger.

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. Each validator has a list of trusted peers called *Unique Nodes List* (UNL) [15], which means that the vote of a validator is only taken into consideration if it is present in a UNL. Consider Figure 1, where we have the representation of a network with five validators: *A* to *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*. *A* receives transactions from different sources asynchronously and stores them in a buffer. At the start of a new consensus round, *A* aggregates the transactions in the buffer into a *proposal*, which is the initial proposition for the next version of the ledger. This proposal is broadcast to the XRPL network at the same time that *A* receives proposals from all other nodes. *A* then compares its own proposal with those received from *C*, *D* and *F*. If a discrepancy is found, 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 next proposal. After a new proposal is formed, it is broadcast to the network, and the process repeats. In the next interaction, the inclusion threshold increases. The process ends when either 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.

Now, consider how validators only acknowledge the positions of peers they have in their UNLs. Looking from the pubsub perspective, we can then say that validators *publish* their proposals and other validators *subscribe* to receive proposals from peers they trust. From this point of view, it is possible to say that the XRP LCP is inherently a pubsub system.

It is important to note that although the XRP LCP was first formalized with the idea that each node would have its own UNL [15], posterior works [16] [17] [18] analyzed theoretically the UNL model and the overlap requirements to avoid stalls and forks. In 2018 Chase and McBrough [16] recommended the use of a unique UNL, since there is a case where, in the presence of malicious nodes, even with 99% overlap, the ledger may stop its progress. In the current implementation, the *XRP Ledger Foundation* curates and maintains a single UNL, advising the use of this UNL to avoid interruptions in the ledger progress.

[1]Ethereum changes its consensus from PoW to PoS in September of 2022

IV. Methodology

To evaluate how the GossipSub parameters affect the behavior of a system, we created a private testnet with 24 nodes, plugging GossipSub into the XRPL (rippled) validator [19] using Flexi-pipe [4], a tool that allows different dissemination techniques to be plugged agnostically into the rippled code. Flexi-pipe[2] works with the idea of creating an overlay network on top of an XRPL testnet through which certain types of messages transit to be analyzed for their behavior.

For this work, we plugged GossipSub version 1.1 written in Go using libp2p to create the overlay. 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 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.

Each parameter tested has been explained in Section III-A. For each set of parameters, we performed three tests of 30 minutes. We gathered data from the trace of events generated by GossipSub, 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 $0,15 * standard\,deviation$.

We tested 14 sets of parameters for each topology, totaling 28 sets of experiments. The reference set used the default values specified for Ethereum. Values were selected based on the configuration of 24 nodes and 8 UNLs, considering that each UNL has on average 16 nodes, 18 being the size of the longest list, so that every node listens to at least 60% of the network, and thus the networks can stay alive and progressing.

A. Data science methodology

To support the analysis of our experiments, we employ popular data science techniques to cluster the results and gain a deeper understanding of the impact of parameterization. Our methodology follows three sequential steps: feature engineering, unsupervised learning (clustering), and supervised learning (classification) applied for explainability.

Feature engineering is a critical step in data science to prepare experimental data for analysis. In our case, we first aggregate the corresponding data from each experiment and then we apply the standardization preprocessing. This preprocessing scales data values so that the mean is 0 and the standard deviation is 1.

Once the data is aggregated and standardized, we apply two clustering algorithms (i.e., K-Means (KM) and Agglomerative Clustering(AC)) to group data points with similar characteristics, thus revealing inherent patterns. These algorithms partition the data into clusters, giving a specific label to groups of data points that share common attributes in n dimensions. The choice of these two algorithms was based on the fact that they approach the clustering problem differently. While K-Means assumes that clusters are spherical and of equal size, Agglomerative Clustering is capable of dealing with non-spherical clusters of varying sizes. Thus, the final labels are given by a voting system between these two clustering algorithms.We empirically set the number of clusters to four for both algorithms, and, after the voting phase, five final clusters were defined.

After the clustering step, a decision tree is used to help interpret and extract valuable knowledge. For our analysis, we fit a decision tree using the configuration parameters of each experiment and the cluster labels given by the clustering algorithms. It is important to highlight that the decision tree does not take into account aggregated data from experiments, contrary to 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. Thus, the output of the decision tree is the correlation between the parameterization and the performance/behavior of each experiment.

V. Evaluation

Our experimental setup consists of a private testnet with a cluster of 24 virtualized nodes running Ubuntu 20.04.4 LTS with 31.25 GiB of RAM, 64 GB of disk, and 4 sockets with 2 cores each. We also use two control nodes with similar configurations running instances of the Flexi-Pipe maestro.

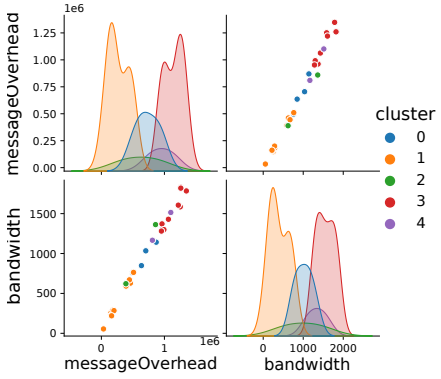

Fig. 2. Correlation between Bandwidth (in messages/second) and Message Overhead

A. The Dimensions

By applying the clustering algorithms, we found a configuration of 5 groups of sets of parameters[3]. Figures 2 and 4 to 9 show the distribution of each dimension for each cluster. *MessageOverhead* refers to how many duplicate messages traversed the network and was acquired by counting

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

[3]https://github.com/FlavScheidt/flexi-pipe/blob/main/data/DimensionalAnalysis/CorrelationAnalysis/votingCuster_KM-AC-B.xlsx

the number of times each message was received by each node on the network. *Bandwidth* is the consumed bandwidth 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 gossip messages. *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 across the network.

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 3, 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 2 because they also have a strong correlation between them, 0.99 according to the correlation heatmap on Figure 3. The correlation between the two dimensions is trivial and was previously discussed by Vyzovitis et al. [8] 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 3 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

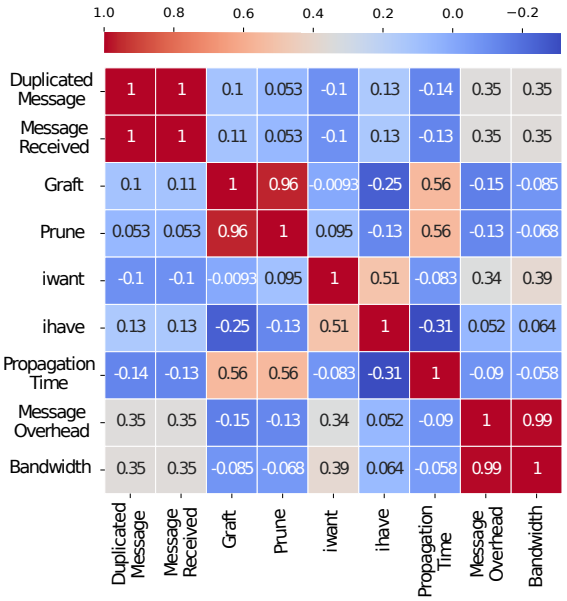

Fig. 3. Heatmap of the correlation between every two dimensions

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 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, which is discussed in the following subsection (V-B).

B. The Clusters

Using Figures 2 and 4 to 9 as references, we can analyze the characteristics of each group in relation to the nine dimensions discussed in Section V-A. 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 2. 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.

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 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

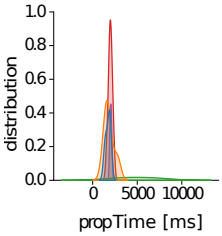

Fig. 4. Propagation time in milliseconds

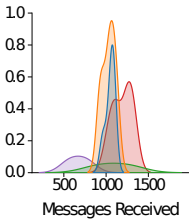

Fig. 5. Number of messages received

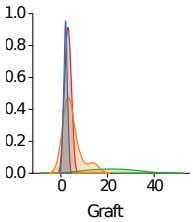

Fig. 6. Number of graft events

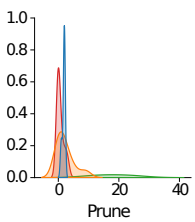

Fig. 7. Number of Prune events

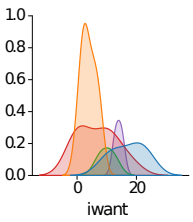

Fig. 8. Number of iwant messages

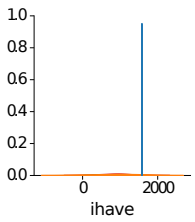

Fig. 9. Number of ihave messages

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.

Cluster 3 (red) shows the highest *Bandwidths* of the entire set, as well as the highest value 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 V-A.

C. 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[4] to find sets 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.

In the decision tree, 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.

[4]https://github.com/FlavScheidt/flexi-pipe/blob/main/data/figures/Dtree.pdf

VI. Conclusion & Future Work

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 XRP Ledger and tested 14 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.

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 parameterization. This tree was generated regarding the parameters, without considering the dimensions used for the clustering and the heatmap, and provided us insights into how to parameterize GossipSub according to the desired behavior and performance of the network as a whole.

Future work in this area can be mainly focused on a comprehensive analysis of how parameters may impact the safety and liveness of the network, as well as a more profound understanding of ways GossipSub can affect the consensus process.

Acknowledgment

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

REFERENCES

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

[2] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 817–831.

[3] M. Essaid, H. W. Kim, W. G. Park, K. Y. Lee, S. J. Park, and H. T. Ju, "Network usage of bitcoin full node," in *2018 International conference on information and communication technology convergence (ICTC)*. IEEE, 2018, pp. 1286–1291.

[4] F. S. de Cristo, L. Trestioreanu, W. Shbair, and R. State, "Pub/sub dissemination on the xrp ledger," in *Proceedings of the 15th Latin-American Conference on Communications*. IEEE, 2023.

[5] 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

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

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

[8] 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

[9] 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.

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

[11] "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

[12] "What is xrp," accessed: 2023-09-12. [Online]. Available: https://xrpl.org/what-is-xrp.html#xrp-is-cryptocurrency

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

[14] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, no. 1, 2012.

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

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

[17] L. Mauri, S. Cimato, and E. Damiani, "A formal approach for the analysis of the xrp ledger consensus protocol," in *ICISSP 2020 - Proceedings of the 6th International Conference on Information Systems Security and Privacy*. SciTePress, 2020, pp. 52–63.

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

[19] "rippled validator source code," accessed: 2022-07-28. [Online]. Available: https://github.com/XRPLF/rippled