GossipSub: A Secure PubSub Protocol for Unstructured, Decentralised P2P Overlays

Dimitris Vyzovitis
Protocol Labs
vyzo@protocol.ai

Yiannis Psaras
UCL, UK & Protocol Labs
yiannis@protocol.ai

ABSTRACT

This report is discussing the design choices behind *gossipsub*, the pubsub protocol in use today in the IPFS ecosystem and in particular as a message mechanism protocol for IPNS records. We are discussing the requirements of the protocol, related works in the area, as well as the specific parameters that influence its behaviour.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

IPFS, libp2p, pubsub, gossipsub

ACM Reference Format:
Dimitris Vyzovitis and Yiannis Psaras. 2019. GossipSub: A Secure PubSub Protocol for Unstructured, Decentralised P2P Overlays. In *Proceedings of Protocol Labs TechRep (PL-TechRep-gossipsub-v0.1-Dec19).* Protocol Labs, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Publish/Subscribe systems have traditionally been used to facilitate distribution of messages in an asynchronous manner between a set of publishers and subscribers. Senders (publishers) and receivers (subscribers) are not in direct communication, but instead communicate through the pub/sub system. Subscribers declare their topics of interest; publishers publish in one of the system's topics. The pub/sub system then matches the two and delivers new messages (or more commonly called events) to all subscribers under a topic. Pub/Sub systems have been extensively used by Internet applications (see Twitter, RSS Feeds, Facebook), but also by general purpose Peer-to-Peer (P2P) systems.

By and large, peer-to-peer (P2P) networks can be split in two categories: i) structured P2P overlays and ii) unstructured P2P overlays. In structured P2P overlays (or networks), the network has some structure, e.g., it is based on some topological or node hierarchy. In such cases, some nodes (often called Super Nodes) can be assigned more responsibilities than others, such as for example, relay published events to subscribed nodes. Those nodes are also assumed to be dedicated servers, hence, they can support routing

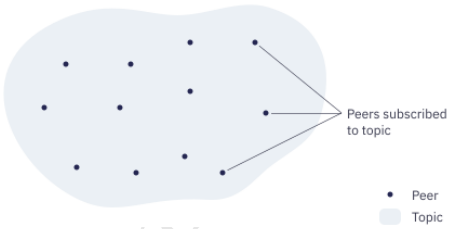

Figure 1: Topics & Peers

Figure 2: Message Delivered to Subscribers

of pub/sub messaging and other operations related to the system on a stable and continuous fashion.

Unstructured P2P overlay networks, on the other hand, do not assume any connectivity properties for any of its nodes. That is, in unstructured P2P networks, nodes can be of any type (i.e., from always-on rack servers, to ephemerally connected laptops and mobile devices) and thus, connect and disconnect at random times. These random connectivity patterns make it impossible to assign extra event routing or message caching responsibilities to any node in unstructured P2P networks. In turn, designing message propagation and guaranteeing reliability of message delivery (that is, that a message will reach all nodes in the network within a given amount of time) is very difficult.

For these reasons, unstructured P2P networks very often use pub/sub protocols that are closer to flooding, or random walks on the overlay to propagate event and membership information. However, naïve flooding introduces a lot of extra traffic in the network, while random walks may take extended amounts of time before reaching all nodes.

While previous work has addressed many different aspects and requirements of pub/sub design for structured P2P networks, little has been done for unstructured P2P networks. "Gossip" has been introduced as a way to limit the number of messages propagated between peers in pub/sub systems, as compared to flooding, where all published messages are forwarded to all subscribed peers. In gossip-based approaches, peers forward metadata related to messages they have "seen" without forwarding the messages themselves. There

PL-TechRep-gossipsub-v0.1-Dec19, Dec 2019, AoE

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

have been numerous studies that explored gossip-based systems, but the scalability of large-scale, unstructured pub/sub systems has not been addressed thoroughly.

In recent years, there has been significant momentum for design and deployment of decentralised Internet services and applications. Among others, such services include distributed and decentralised storage [8], [7], [12], [34], [2] but also computation systems [3]. The aim is to replace, or complement traditional centrally managed and operated cloud services. End-users are contributing part of their resources to the network and get rewarded according to contribution. These emerging systems are *distributed* in the sense of geographical spread and *decentralised* from the point of view of ownership, management and operation.

We are, therefore, witnessing a trend towards building P2P overlays, where, in most cases, unreliable and non-dedicated end-user devices are active contributors to the network. In the absence of central control, messaging in those systems is of utmost importance in order to communicate operational processes (*e.g.,* find file or execute function), but also propagate management events.

Pub/Sub has seen a surge in usage from distributed applications in the area of decentralised services, such as decentralised chat and social networks [12], [10], collaborative editing tools without a backend server [11], hosting of dynamic website content in unmanaged P2P networks [8], storage and synchronisation of evolving datasets, to name a few.

In distributed storage systems and in the case of the InterPlanetary File System (IPFS) ecosystem in particular, pub/sub can be used for several purposes, including content routing, *i.e.,* one of the most central and vital functions of the system. IPFS is a content-addressable, distributed P2P storage network with hundreds of thousands of daily users. Users can participate in the network as unreliable nodes, e.g., using laptop devices and with frequent disconnections. The gossip-based pubsub protocol proposed here (acronymed "gossipsub") was developed with those system and environment requirements in mind (*i.e.,* unmanaged network and unreliable nodes) and is currently in use to push naming record updates to the decentralised naming system of IPFS (acronymed IPNS [9].

Pubsub and in particular the proposed gossipsub protocol will also be used in the very near future by emerging P2P systems, such as ETH2 [4] and Filecoin [5] as the main routing protocol for transaction blocks. Systems such as ETH2 and Filecoin are primarily financial systems and are expected to carry transaction messages worth millions in monetary value. That said, the security properties of gossipsub need to be investigated in detail, which we will do in subsequent versions of this report.

Previously proposed pubsub systems have proven scalability properties in managed and cloud-based environments. A few pubsub protocols have been proposed for unstructured P2P overlays and among those even fewer have been tested with more than 10,000 nodes and high rates of churn. IPFS already has hundreds of thousands of daily users and is expected to grow exponentially to multiple millions. ETH1.0 already has more than 16,000 users and when ETH2.0 arrives this number is expected to rise by several orders of magnitude.

That said, a thorough evaluation of the protocol is essential before its deployment in those systems. This is the goal that this paper intends to pursue. In its current version, we are starting by illustrating the protocol's main design choices, together with a conceptual comparison against related works in the area.

Gossipsub Design Summary: The pubsub protocol proposed here falls in the category of gossip-based pubsub protocols. Its main design features are a connected mesh, complemented by gossip functionality and lazy push. *Lazy push* is a technique used in pubsub systems, according to which only metadata and not the full message, is forwarded to subscribed nodes. In turn, if nodes are interested in the actual message whose metadata they have received, they request the message explicitly. As is the case with many unstructured P2P networks, the first version of the protocol was closer to flood-based pubsub. However, as the IPFS network grew over time, it was clear that flooding is not an efficient approach. Scalability requirements quickly surfaced and resulted in the design of *meshsub*, a connected mesh which is complemented by *gossip* and *lazy push* as core features of the protocol. The combination of those techniques, along with the ability to plug in custom routing heuristics, constitute the main novelty of gossipsub. Simplicity of implementation, as well as being reactive to dynamic network conditions have been the driving philosophies behind the design of gossipsub.

2 BACKGROUND & RELATED WORK

2.1 Tradeoffs in PubSub

General purpose pub/sub messaging systems can prove very useful from several different aspects (from management and operation to performance) in P2P networks and as such come with many tradeoffs [19], [25]. Due to the wide variety of applications building on top of pubsub systems, not all tradeoffs apply to all systems and many of them are contradictory to each other. Below, we discuss some of them and, where relevant, note how they influenced the design of gossipsub.

(1) **Reliable Delivery.** In case of no node downtime, all published messages should be delivered to all subscriber nodes. Pub/Sub systems should be *robust against node churn* and should still reach most nodes (*i.e.,* achieve high hit rate). Apart from robustness, fast recovery from churn is also necessary. Node churn and disconnected environments are tackled in gossipsub by using gossip messaging. This is a hard requirement for the protocol's use in the IPFS network, but also more generally as a feature in libp2p.
(2) **Load Balancing.** The event message relay load should be roughly equally split between nodes. Assuming a scaled up system where nodes might be subscribed to 5K events, relaying messages is becoming a heavy task and therefore, the more nodes a node is connected to (in terms of degree), the more the relay tasks it will have to carry out.
(3) **Scalability.** Given the growth of networked systems, both in a cloud environment, but also in decentralised, P2P environments, the system should be able to scale up to millions of nodes[1]. There have been very few (if any) systems that

[1]The Bittorent network had tens of millions of active daily nodes, while IPFS today has hundreds of thousands of nodes and the ETH2 network is expected to have millions of nodes.

achieved scalability of that order in unmanaged, P2P environments, while still achieving acceptable performance with today's standards (i.e., lookup in *ms* and delivery in less than 1*sec*).

(4) **Resilience & Resource-Efficiency.** It is generally common in pub/sub systems for a message to be delivered twice to subscriber nodes. Clearly, this increases load on individual relay nodes, but also the overall system's bandwidth requirements. On the other hand, redundant delivery can enhance the security and resilience properties of the system. Duplicate message delivery has to be kept within certain levels in order to both achieve resilience, but at the same time avoid having negative impact in terms of resource-efficiency.

Building broadcast (spanning) trees has been proposed before [22], [23] to deal with many of the tradeoffs listed above (*e.g.,* load-balancing and resource efficiency), but a tree can only scale linearly to its length and most importantly it is not robust against churn. Gossiping, on the other hand, is inherently more robust against node failures and churn, but might suffer from resource efficiency and load-balancing.

Striking the right balance, especially in an unmanaged, unstructured P2P overlay has not seen a solution to date. This is the gap that the proposed solution is filling, leveraging the simplicity of its design, but also a resource-efficient gossip approach.

The literature in this space is vast. The purpose of this section is not to survey the majority of the proposals, but instead to point to the most important contributions from which lessons can be learned and applied to an unstructured, unmanaged, P2P pub/sub overlay.

2.2 Scalable Topic-based Pub/Sub

Scalability in pub/sub systems was a key requirement in cloud environments, as cloud-based systems had to scale to accommodate demand. Amazon [1] and Google Cloud Messaging [6] both have pub/sub protocols in operation, although their operational details have not been widely revealed.

Scribe [18] was one of the very first pub/sub systems that proposed a decentralised multicast overlay, on top of the Pastry DHT. DHTs have been used in several pub/sub systems (see Meghdoot [21], Bayeux [36]), where in most cases the DHT is used to find where subscriptions are located and route to them [21], or as a rendezvous point [18] for a topic. Poldercast [31] is an interesting approach which uses a ring overlay (and additional links as optimisation). Subscribers to a topic are connected to this ring and messages propagate to all other subscribers. Clearly, the latency to inform all nodes is increasing linearly, unless direct links exist and can be exploited. Dynatops [35] was proposed as a self-configured topic-based pub/sub system which can deal with short-lived subscriptions. All these systems require a broker network that extends across a wide-area network to cover subscribers. Dynamoth [20] was proposed to reduce latency as a hybrid between the dissintermediated pub/sub and the directly-connected client-server models. It is, however, exclusively applicable to cloud-based systems and not to P2P overlays.

Closer to our work are systems such as Vitis [29], Tera [13], Rappel [27] and StAN [26], which use gossiping to propagate information and build on unstructured P2P overlays. Gossip-based protocols distinguish between two types of peering: *full message* and *metadata peering* (see Figs. 3 and 4). When two nodes are in full message peering, messages received by one node are forwarded to its full message peers. In contrast, when two nodes are metadata peers, they exchange only metadata for received messages. Metadata peers are "aware" of published messages, but do not actually have the published messages. Extending the concept of *meta-data peering*, gossipsub is implementing the previously proposed concept of *lazy push*, according to which peers in the network are notified of the existence of messages only (*e.g.,* through their message ID), but messages are only forwarded if a node explicitly requests the newly advertised message.

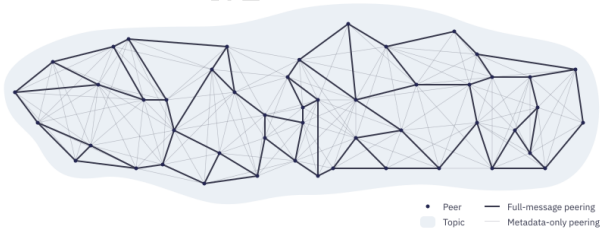

Figure 3: Full vs. Metadata Peering

In particular, the authors in [29] are arguing that most similar systems are building one separate overlay per topic, which results in nodes being members of an extensive number of overlays. This, in turn, according to Vitis [29] increases overhead for relay nodes, which might get disincentivised and leave the system. In contrast, Vitis is dealing with this problem by bounding the number of connections per node. This is done by using gossip messages to sample the topics that other nodes have subscribed to. Then grouping nodes into the same overlay where topics overlap reduces the number of overlays needed, while at the same time keeping nodes subscribed to the topics of their choice.

Rappel [27] is targeting low overhead and noise to subscriber nodes, that is, receiving messages for topics that the node is not subscribed to. Rappel achieves that by building a network of "friends overlay" building on interest locality. Rappel also targets fast dissemination of messages by taking into account network locality on top of interest locality.

While these have been very interesting approaches to gossip-based pub/sub for unstructured P2P overlays, none of them has been tested for scalability at massive scales (*e.g.,* millions of nodes), or significant node churn. Rappel [27] is claiming to be robust for up to 25% node churn, while Tera [13] leaves this part for future evaluation.

The scale required by systems such as name registry propagation in IPNS, or request routing in Filecoin and transaction routing in ETH2 can be orders of magnitude higher than the systems tested in the past for unstructured, unmanaged P2P overlays. Node churn can also be significant, but we expect that a 30% threshold should be acceptable.

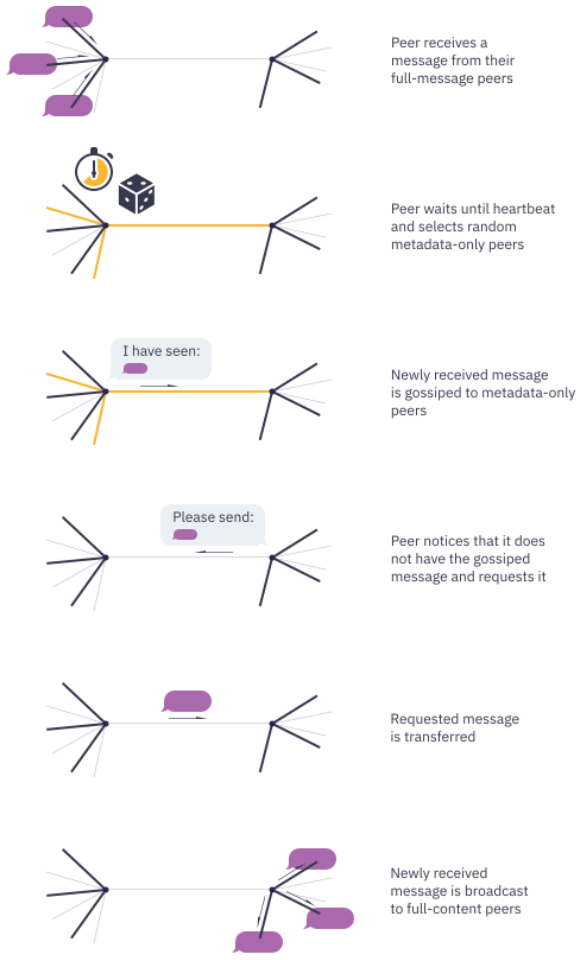

Figure 4: Gossip-based Message Delivery

Gossipsub is building on simplicity of both design and implementation, in order to be able to scale and address the size of network we require. Being responsive to changing network conditions, such as node churn in unreliable environments is another feature that is achieved by gossipsub's design.

2.3 Scalable Content-based Pub/Sub

Content-based pub/sub systems have been extensively studied in the past, *e.g.,* [17], but also more recently in the context of Information-Centric Networks, *e.g.,* [15], [16]. Generally speaking, content-based (or sometimes called attribute-based) pub/sub systems can provide finer granularity matching between publishers and subscribers, but in order to achieve this they require more compute resources. Gossipsub is is not building on content-based pub/sub, hence, in this subsection we discuss a few approaches for completeness.

BlueDove [24] is one of the well-known approaches in this space. It supports multi-dimensional attributes and is organising overlay servers in a scalable topology. E-StreamHub [14] is proposed as a middleware with the interesting feature that it adds and removes nodes based on their load. Both of these approaches are exclusively cloud-based and although they achieve scalability and low-latencies, they do not apply to unmanaged P2P networks. In fact, most content-based pub/sub systems either target cloud environments, or some broker-based infrastructure (see Elvin, Sienna [17], HERMES [28], Gryphon [33]). Privacy-preserving techniques to protect subscribers' information have also been investigated extensively in content-based pub/sub systems with some notable works, such as [30], [32].

3 GOSSIPSUB PROTOCOL DETAILS

3.1 FloodSub

The initial pubsub protocol in libp2p was floodsub. Floodsub implements pubsub in the most basic manner, with two defining aspects: i) ambient peer discovery; and ii) flooding, as the most basic routing and message propagation protocol

3.1.1 Ambient Peer Discovery. With ambient peer discovery, the peer discovery task is pushed outside the scope of the pubsub protocol. Instead, the mechanism for discovering peers is provided for by the environment. In practice, and in the case of IPFS, this can be embodied by DHT walks, rendezvous points or similar techniques. This level of decoupling is possible thanks to the modular design of libp2p. Floodsub relies on the ambient connection events produced by these discovery mechanisms. Whenever a new peer is connected, the protocol checks to see if the peer implements floodsub and/or gossipsub, and if so, it sends a 'hello' packet that announces the topics that it is currently subscribing to.

This allows the peer to maintain soft overlays for all topics of interest. The overlay is maintained by exchanging subscription control messages whenever there is a change in the topic list. The subscription messages are not propagated further, so each peer maintains a topic view of its direct peers only. Whenever a peer disconnects, it is removed from the overlay.

Ambient peer discovery can be driven by arbitrary external means, which allows orthogonal development and no external dependencies for the protocol implementation.

There are a couple of options we are exploring as canonical approaches for the discovery function:

- DHT rendezvous using provider records; peers in the topic announce a provider record named after the topic.
- Rendezvous through known or dynamically discovered rendezvous points.

3.1.2 Flood Routing. With flooding, routing is almost trivial: for each incoming message, forward to all known peers in the topic. In our implementation, the router maintains a timed cache of previous messages, so that seen messages are not forwarded more than once. The protocol also never forwards a message back to the source or the peer that forwarded the message.

3.1.3 Retrospective. Evaluating floodsub as a viable pubsub protocol reveals the following highly desirable properties:

- it is straightforward to implement.
- it minimizes latency; messages are delivered across minimum latency paths, modulo overlay connectivity.

- it is highly robust; there is very little maintenance logic or state.

The problem, however, is that *messages do not just follow the minimum latency paths*; they follow all edges, thus creating a flood. The outbound degree of the network is unbounded, which further means that peers' bandwidth is saturated with redundant traffic. On the other hand, redundant traffic is increasing the resilience of the network and unbounded degree is naturally creating a resistance level against Sybil attacks. There are clear benefits that come with message redundancy that gossipsub attempts to exploit, but at the same time without being prohibitive to the peers' resources. This is essential in order to increase scalability and support decentralisation.

Similarly, the amplification factor is only bounded by the sum of degrees of all nodes in the overlay, which creates a scaling problem for densely connected overlays at large.

3.2 Controlling the flood

In order to scale pubsub without excessive bandwidth waste or peer overload, we need a router that bounds the degree of each peer and globally controls the amplification factor.

3.2.1 randomsub: A random message router. As a first step, we consider the simplest bounded floodsub variant, which we call *randomsub*. In this construction, the router is still stateless, apart from a list of known peers in the topic. But instead of forwarding messages to all peers, it forwards to a random subset of up to `D` peers, where `D` is the desired degree of the network.

The problem with this construction is that the message propagation patterns are non-deterministic. This results in extreme message route instability, manifesting as message reordering and varying timing patterns, which is an undesirable property for many applications.

3.2.2 meshsub: An overlay mesh router. *meshsub* is improving on the *randomsub* construction by limiting the number of messages in the pubsub network. This is essential in order to reduce bandwidth requirements, network and node/router load. However, instead of randomly selecting peers on a per message basis, we form an overlay mesh where each peer forwards to a subset of its peers on a stable basis. We construct a router in this fashion, dubbed *meshsub*.

Each peer maintains its own view of the mesh for each topic, which is a list of bidirectional, reciprocal links to other peers. That is, in steady state, whenever a peer A is in the mesh of peer B, then peer B is also in the mesh of peer A.

The overlay is initially constructed in a random fashion, based only on topics. Whenever a peer joins a topic, then it selects `D` peers (in the topic) at random and adds them to the mesh, notifying them with a control message. When it leaves the topic, it notifies its peers and forgets the mesh for the topic.

The mesh is maintained with the following periodic stabilization algorithm:

```
  at each peer:
loop:
if |peers| < D_low:
select D - |peers| non-mesh peers at random and add them to the mesh
if |peers| > D_high:
select |peers| - D mesh peers at random and remove them from the mesh
sleep t
```

The parameters of the *meshsub* algorithm are: `D`, which is the target degree, and two relaxed degree parameters `D_low` and `D_high`, representing admissible mesh degree bounds. Their goal is to introduce elasticity and curtail excessive flapping.

3.2.3 gossipsub: The gossiping mesh router. The *meshsub* router offers a baseline construction with good amplification control properties, which we augment with gossip about message flow. The gossip is emitted to random subsets of peers that are currently *not part of the mesh*. Using gossip messages we can propagate metadata about message flow throughout the network. The metadata can be arbitrary, but as a baseline we include the message IDs that the router emitting the gossip has seen in the last few seconds. The actual messages themselves are cached, so that peers receiving the gossip can request them for transmission with a control message.

The router can use this metadata to improve the mesh and create epidemic broadcast trees. Beyond that, the metadata can restart message transmission at different points in the overlay to rectify downstream message loss. Alternatively, it can simply jump hops opportunistically and accelerate message transmission for peers who are at some distance in the mesh.

Essentially, *gossipsub is a blend of meshsub for data and randomsub for mesh metadata, realised in the form of lazy push.* The combination of those techniques provides a powerful construction that meets requirements of disconnected mesh, "squatting peers", responsiveness to network conditions and most importantly scalability. Gossipsub provides bounded degree and amplification factor with the *meshsub* construction and augments it using gossip propagation of metadata with the *randomsub* technique.

3.3 The gossipsub Protocol Implementation

We proceed to provide a specification of the gossipsub protocol by sketching out the router implementation. The router is backwards compatible with floodsub, as it accepts floodsub peers and behaves like floodsub towards them.

3.3.1 Control messages. The protocol defines four control messages:

- GRAFT: graft a mesh link; this notifies the peer that it has been added to the local mesh view.
- PRUNE: prune a mesh link; this notifies the peer that it has been removed from the local mesh view.
- IHAVE: gossip; this notifies the peer that the following messages were recently seen and are available on request.
- IWANT: request transmission of messages announced in an IHAVE message.

3.3.2 Heartbeat. The router periodically runs a heartbeat procedure, which is responsible for maintaining the mesh, emitting gossip, and shifting the message cache.

3.3.3 Router state. The router maintains the following state:

Figure 5: Graft New Peering Nodes

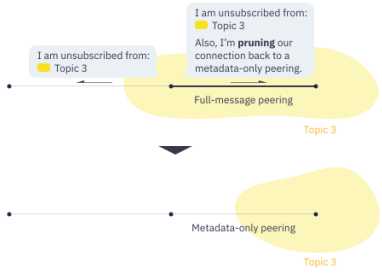

Figure 6: Prune Existing Nodes

- `peers`: a set of all known peers; `peers.gossipsub` denotes the gossipsub peers while `peers.floodsub` denotes the floodsub peers.
- `mesh`: the overlay meshes as a map of topics to lists of peers.
- `fanout`: the mesh peers to which we are publishing to without topic membership, as a map of topics to lists of peers.
- `seen`: this is the timed message ID cache, which tracks seen messages.
- `mcache`: a message cache that contains the messages for the last few heartbeat ticks.

The message cache is a data structure that stores windows of message IDs and the corresponding messages. It supports the following operations:

- `mcache.put(m)`: adds a message to the current window and the cache.
- `mcache.get(id)`: retrieves a message from the cache by its ID, if it is still present.
- `mcache.window()`: retrieves the message IDs for messages in the current history window.
- `mcache.shift()`: shifts the current window, discarding messages older than the history length of the cache.

The seen cache is the flow control mechanism. It tracks the message IDs of seen messages for the last two minutes. It is separate from `mcache` for implementation reasons in Go (the seen cache is inherited from the pubsub framework), but they can also be the same data structure. Note that the two minute cache interval is non-normative; a router could use a different value, chosen to approximate the propagation delay in the overlay with some healthy margin.

3.4 Topic Membership

Topic membership is controlled by two operations supported by the router, as part of the local pubsub SDK API:

- On `JOIN(topic)` the router joins the topic. In order to do so, if it already has `D` peers from the fanout peers of a topic, then it adds them to `mesh[topic]`, and notifies them with a `GRAFT(topic)` control message. Otherwise, if there are less than `D` peers (let this number be x) in the fanout for a topic (or the topic is not in the fanout), then it still adds them as above (if there are any). The algorithm also selects the remaining number of peers (`D-x`) from `peers.gossipsub[topic]`, and adds them to `mesh[topic]` notifying them with a `GRAFT(topic)` control message.
- On `LEAVE(topic)` the router leaves the topic. It notifies the peers in `mesh[topic]` with a `PRUNE(topic)` message and forgets `mesh[topic]`.

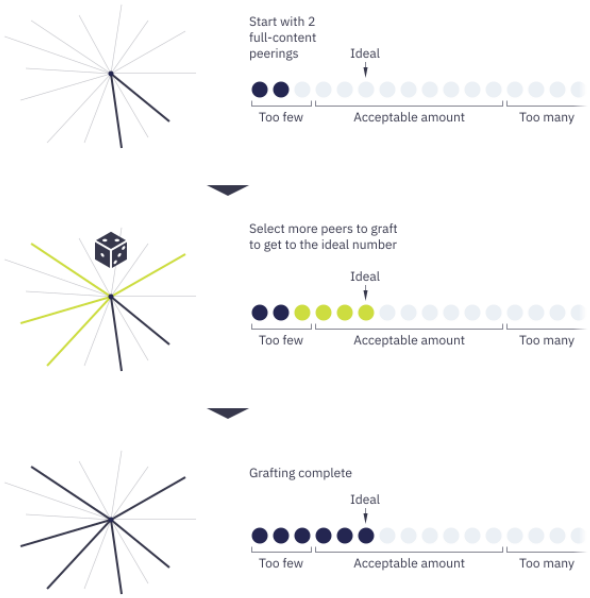

Figure 7: Graft Full Peering Nodes

Note that the router can publish messages without topic membership. In order to maintain stable routes in that case, it maintains a list of peers for each topic it has published in the fanout map. If the router does not publish any messages of a topic for some time, then the fanout peers for that topic are forgotten – in other words, this is soft state.

Also note that as part of the pubsub API, the peer emits SUBSCRIBE and UNSUBSCRIBE control messages to all its peers whenever it joins or leaves a topic. This is provided by the the ambient peer discovery mechanism and nominally not part of the router. A standalone implementation would have to implement those control messages.

3.4.1 Message Processing. Upon receiving a message, the router first processes the payload of the message. If it contains a valid

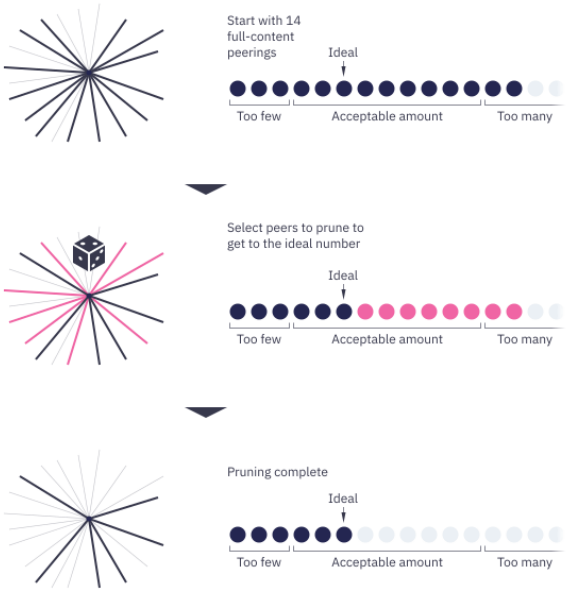

Figure 8: Prune Full Peering Nodes

message that has not been previously seen, then it publishes the message. In particular,

- It forwards the message to every peer in `peers.floodsub[topic]`, provided none of these peers are the source of the message.
- It forwards the message to every peer in `mesh[topic]`, provided none of these peers are the source of the message.

After processing the payload, it then processes the control messages in the envelope:

- On `GRAFT(topic)` it adds the peer to `mesh[topic]` if it is subscribed to the topic. If it is not subscribed, it responds with a `PRUNE(topic)` control message.
- On `PRUNE(topic)` it removes the peer from `mesh[topic]`.
- On `IHAVE(ids)` it checks the seen set and requests unknown messages with an IWANT message.
- On `IWANT(ids)` it forwards all request messages that are present in mcache to the requesting peer.

When the router publishes a message that originates from the router itself (at the application layer), then it proceeds similarly to the payload reaction:

- It forwards the message to every peer in `peers.floodsub[topic]`.
- If it is subscribed to the topic, then it must have a set of peers in `mesh[topic]`, to which the message is forwarded.
- If it is not subscribed to the topic, it then forwards the message to the peers in `fanout[topic]`. If this set is empty, it chooses `D` peers from `peers.gossipsub[topic]` to become the new `fanout[topic]` peers and forwards to them.

3.4.2 Control message piggybacking. Gossip and other control messages do not have to be transmitted on their own message. Instead, they can be coalesced and piggybacked on any other message in the regular flow, for any topic. This can lead to message rate reduction whenever there is some correlated flow between topics, and can be significant for densely connected peers. Although our choice of metadata (*i.e.,* message IDs) is used as a baseline (see Sec. 3.2.3), *piggybacking is a very elegant way to propagate this type of information without inserting extra traffic in the network and therefore, being able to scale to much larger network sizes.*

4 CONCLUDING REMARKS

We have presented the design choices of the pubsub protocol used in the IPFS ecosystem together with reasoning behind those choices. We have provided a brief survey of related works in the area of pubsub and have conceptually compared the design choices of gossipsub with those in related literature. This report should by no means be considered as a comprehensive literature survey. Instead, we have pointed to survey papers that paint the bigger picture.

With the imminent deployment of gossipsub as the main routing protocol for transaction messages/blocks in the Filecoin network, we are currently investigating security measures of gossipsub and will update this report once the initial security enhancements are designed.

REFERENCES

[1] Amazon SND. https://aws.amazon.com/pub-sub-messaging/.
[2] DAT Protocol Foundation. https://dat.foundation.
[3] Dfinity: The Internet Computer. https://dfinity.org.
[4] Ethereum 2.0. https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/eth-2.0-phases/.
[5] filecoin: A decentralised market for storage.
[6] Google/Firebase Cloud Messaging. https://firebase.google.com/docs/cloud-messaging/.
[7] IPFS - Content Addressed, Versioned, P2P File System. https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.
[8] IPFS: InterPlanetary File System. https://ipfs.io.
[9] Ipns: Interplanetary naming system. https://docs.ipfs.io/guides/concepts/ipns/.
[10] Mastodon Social Network. https://joinmastodon.org.
[11] PeerPad. https://peerpad.net.
[12] Secure Scuttlebutt. https://scuttlebutt.nz/.
[13] Baldoni, R., Beraldi, R., Quema, V., Querzoni, L., and Tucci-Piergiovanni, S. Tera: Topic-based event routing for peer-to-peer architectures. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems* (New York, NY, USA, 2007), DEBS '07, ACM, pp. 2–13.
[14] Barazzutti, R., Felber, P., Fetzer, C., Onica, E., Pineau, J.-F., Pasin, M., Rivière, E., and Weigert, S. Streamhub: A massively parallel architecture for high-performance content-based publish/subscribe. pp. 63–74.
[15] Carzaniga, A., Khazaei, K., Papalini, M., and Wolf, A. L. Is information-centric multi-tree routing feasible? *SIGCOMM Comput. Commun. Rev. 43*, 4 (Aug. 2013), 3–8.
[16] Carzaniga, A., Papalini, M., and Wolf, A. L. Content-based publish/subscribe networking and information-centric networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking* (New York, NY, USA, 2011), ICN '11, ACM, pp. 56–61.
[17] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems 19*, 3 (Aug. 2001), 332–383.
[18] Castro, M., Druschel, P., Kermarrec, A. ., and Rowstron, A. I. T. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications 20*, 8 (Oct 2002), 1489–1499.
[19] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. The many faces of publish/subscribe. *ACM Comput. Surv. 35*, 2 (June 2003), 114–131.
[20] Gascon-Samson, J., Garcia, F.-P., Kemme, B., and Kienzle, J. Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. *Proceedings - International Conference on Distributed Computing Systems 2015* (07 2015), 486–496.
[21] Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. Meghdoot: Content-based publish/subscribe over p2p networks. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware* (Berlin, Heidelberg, 2004), Middleware '04, Springer-Verlag, pp. 254–273.
[22] Leitao, J., Pereira, J., and Rodrigues, L. Epidemic broadcast trees. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)* (Oct 2007), pp. 301–310.

[23] Leitao, J., Pereira, J., and Rodrigues, L. Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (Washington, DC, USA, 2007), DSN '07, IEEE Computer Society, pp. 419–429.

[24] Li, M., Ye, F., Kim, M., Chen, H., and Lei, H. Bluedove: A scalable and elastic publish/subscribe service. In *IEEE IPDPS 2011* (05 2011), pp. 1254–1265.

[25] Malekpour, A., Carzaniga, A., Pedone, F., and Toffetti Carughi, G. End-to-end reliability for best-effort content-based publish/subscribe networks. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System* (New York, NY, USA, 2011), DEBS '11, ACM, pp. 207–218.

[26] Matos, M., Nunes, A., Oliveira, R., and Pereira, J. Stan: Exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *Proceedings of the 9th International Conference on Peer-to-peer Systems* (Berkeley, CA, USA, 2010), IPTPS'10, USENIX Association, pp. 9–9.

[27] Patel, J., Riviere, E., Gupta, I., and Kermarrec, A.-M. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks 53* (08 2009), 2304–2320.

[28] Pietzuch, P. R., and Bacon, J. M. Hermes: a distributed event-based middleware architecture. In *Proceedings 22nd International Conference on Distributed Computing Systems Workshops* (July 2002), pp. 611–618.

[29] Rahimian, F., Girdzijauskas, S., Payberah, A. H., and Haridi, S. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium* (Washington, DC, USA, 2011), IPDPS '11, IEEE Computer Society, pp. 746–757.

[30] Rao, W., Chen, L., and Tarkoma, S. Toward efficient filter privacy-aware content-based pub/sub systems. *IEEE Transactions on Knowledge and Data Engineering 25*, 11 (Nov 2013), 2644–2657.

[31] Setty, V., van Steen, M., Vitenberg, R., and Voulgaris, S. Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub. In *Proceedings of the 13th International Middleware Conference* (New York, NY, USA, 2012), Middleware '12, Springer-Verlag New York, Inc., pp. 271–291.

[32] Shikfa, A., Önen, M., and Molva, R. Privacy-preserving content-based publish/subscribe networks. vol. 297, pp. 270–282.

[33] Strom, R., Banavar, G., Chandra, T., Kaplan, M., Miller, K., Mukherjee, B., Sturman, D., and Ward, M. Gryphon: An information flow based approach to message brokering. *CoRR cs.DC/9810019* (10 1998).

[34] Tarr, D., Lavoie, E., Meyer, A., and Tschudin, C. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2019), ICN '19, ACM, pp. 1–11.

[35] Zhao, Y., Kim, K., and Venkatasubramanian, N. Dynatops: A dynamic topic-based publish/subscribe architecture. pp. 75–86.

[36] Zhuang, S. Q., Zhao, B. Y., Joseph, A. D., Katz, R. H., and Kubiatowicz, J. D. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (New York, NY, USA, 2001), NOSSDAV '01, ACM, pp. 11–20.