DEMYSTIFYING GOSSIPSUB

A scalable and extensible p2p gossip protocol

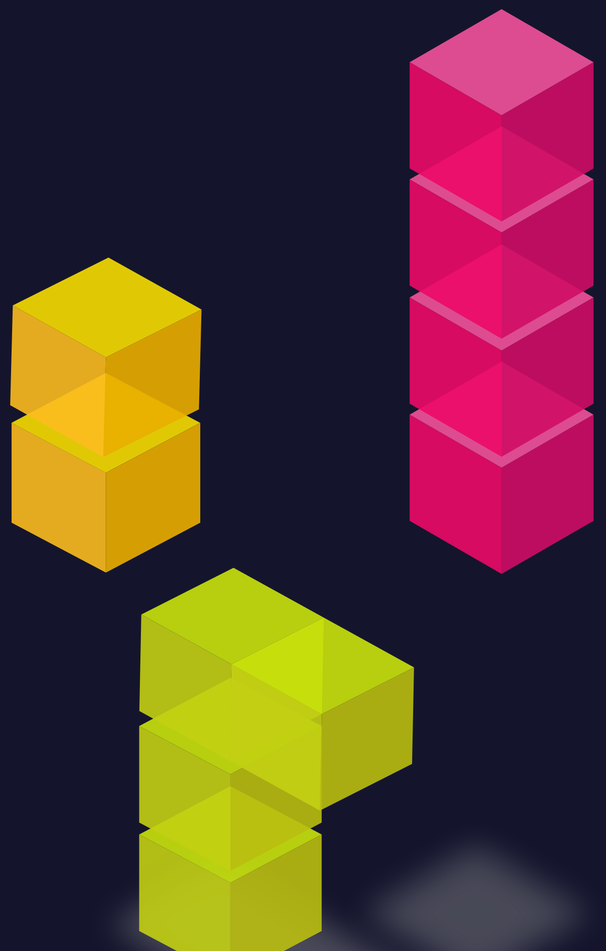
@ Devcon5 🖐

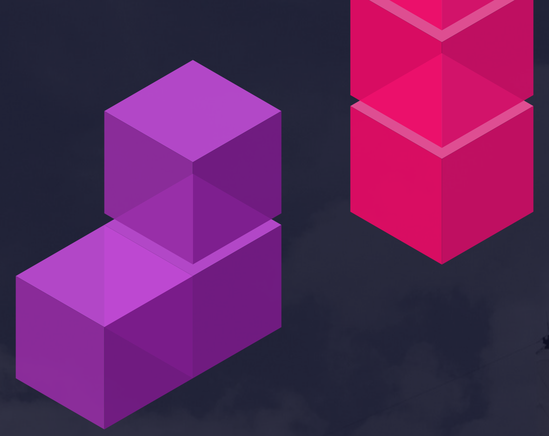
Raúl Kripalani

libp2p

today

AGENDA





- introduction to p2p pubsub
- what is gossipsub
- how gossipsub operates
- what’s next?

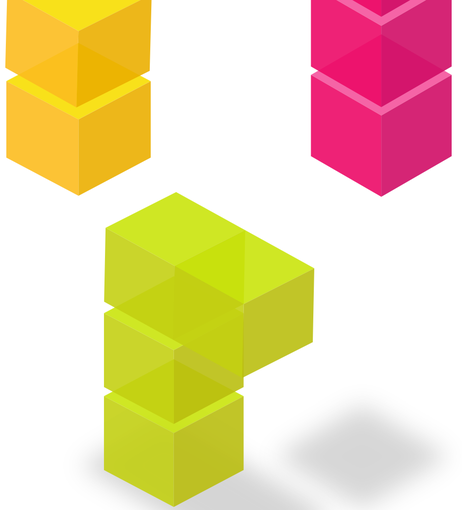
libp2p gossipsub @ devcon5

INTRO TO P2P PUBSUB

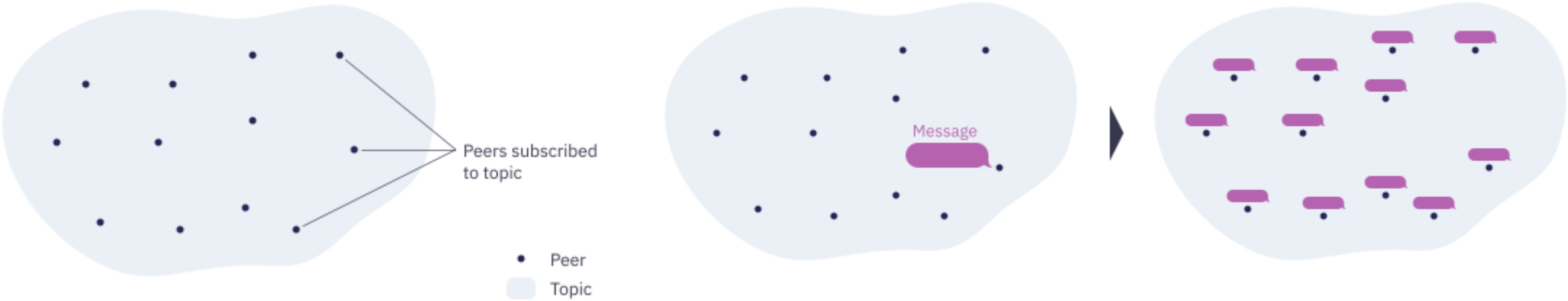
- **intro to p2p pubsub**
- what is gossipsub?
- how gossipsub works
- what's next?

introduction

WHAT IS PUBSUB?



- Publish/subscribe (pubsub) is a message-oriented communication pattern.
- The interaction model of pubsub is M:N. This is contrast to RPC or request/reply patterns which are 1:1.
- Pubsub communication is asynchronous and decoupled in nature.
- Processes act as publishers and consumers of messages, and they congregate around topics.
- Widely popular in enterprise software (Apache ActiveMQ, Apache Kafka, RabbitMQ). ***Centralized around brokers in private networks.*** **Not suitable for public message fabrics.**

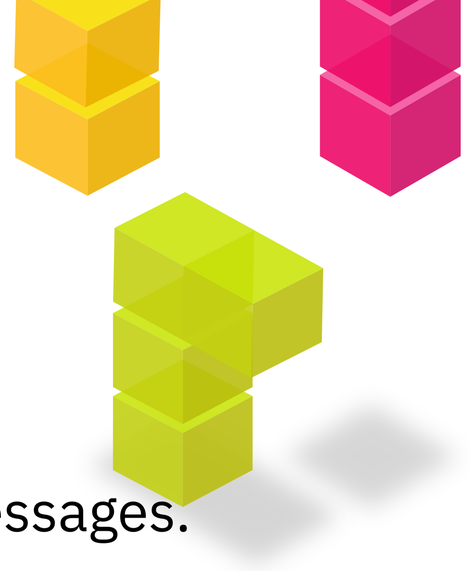


introduction

WHAT IS DECENTRALIZED P2P PUBSUB?

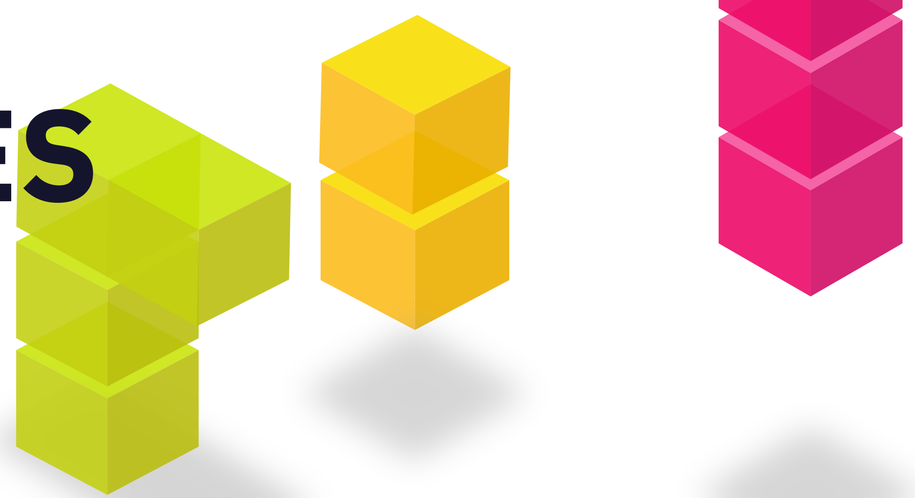
- Brokerless. No central authorities keeping track of subscribers, publishers, topics, etc.
- Self-regulating. No global knowledge. Network behaviour emerges from local decisions.
- Constructed as overlay networks, with peers collaborating to ensure message deliverability.
- Underlay is assumed to be well-connected (e.g. I can always find a path from A to B).

SOME TERMINOLOGY



- **Topic:** a subject around which peers congregate to emit and receive messages.
- **Subscription:** the act of expressing interest in a particular topic.
- **Publishing:** the act of creating a message and pushing it out to the network.
- **Forwarding:** the act of propagating a message to other peers in the topic.
- **Ambient discovery:** a discovery mechanism that is provided by the environment.

DESIRABLE PROPERTIES IN P2P PUBSUB



1. **Reliable:** all messages get delivered to all peers subscribed to the topic.
2. **Resilience:** peers can join and leave topics or the network without disruption under realistic churn scenarios, with **fast recovery** to mend the overlay to restore 100% hit.
3. **Efficient dissemination:** fast delivery, low write amplification, fair distribution of load.
4. **Altruism-free routing:** only nodes invested in a topic participate in its dissemination.
5. **Scalability:** enormous number of nodes, topics, messages, subscriptions, subscribers.
6. **Low overhead** of overlay maintenance: nodes track as little state as possible.
7. **Simplicity:** the system is simple to understand and implement.

Adapted from: PolderCast: Fast, Robust, and Scalable Architecture for P2P Topic-Based Pub/Sub

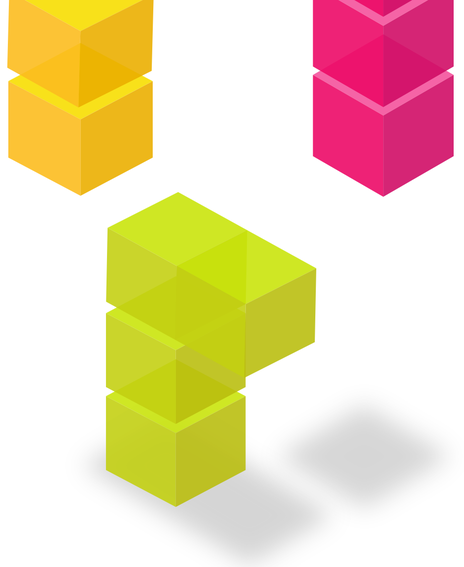
libp2p gossipsub @ devcon5

WHAT IS GOSSIPSUB?

intro to p2p pubsub
what is gossipsub?
how gossipsub works
what's next?

what is gossipsub?

KEY ELEMENTS



- Gossipsub is an scalable and extensible p2p pubsub protocol.
- Envisioned, specified and initially developed by @vyzo from the libp2p core team.
- **Provides base layer and primitives to build adaptive, self-optimizing routing algorithms capable of repair and rearrangement, *in the presence of churn, latency and failures.***
- Combines stable reciprocal meshes, with gossip about message flow metadata.
- Succeeds **floodsub** (flooding) — robust, resilient, minimum latency, simple, but huge amplification factor and bandwidth-inefficient.
- **Stepping stone towards the grand vision of *episub*,** concocting ideas from Plumtree (Epidemic Broadcast Trees), HyParView, GoCast.

what is gossipsub?

STATUS: ALPHA

https://github.com/libp2p/specs/tree/master/pubsub/gossipsub

gossipsub: An extensible baseline pubsub protocol

Lifecycle Stage	Maturity	Status	Latest Revision
3A	Recommendation	Active	r1, 2018-08-29

Authors: @vyzo

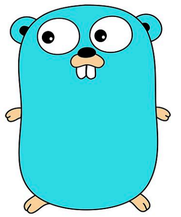
Interest Group: @yusefnapora, @raulk, @whyrusleeping, @Stebalien, @jamesray1, @vasco-santos

See the lifecycle document for context about maturity level and spec status.

This is the specification for an extensible baseline pubsub protocol, based on randomized topic meshes and gossip. It is a general purpose pubsub protocol with moderate amplification factors and good scaling properties. The protocol is designed to be extensible by more specialized routers, which may add protocol messages and gossip in order to provide behaviour optimized for specific application profiles.

Contents

implementations



libp2p/go-libp2p-pubsub (reference impl)

sigp’s rust-libp2p fork

ChainSafe/gossipsub-js

libp2p gossipsub @ devcon5

HOW GOSSIPSUB WORKS

- intro to p2p pubsub
- what is gossipsub?
- **how gossipsub works**
- what's next?

libp2p gossipsub @ devcon5

OVERLAY TOPOLOGY

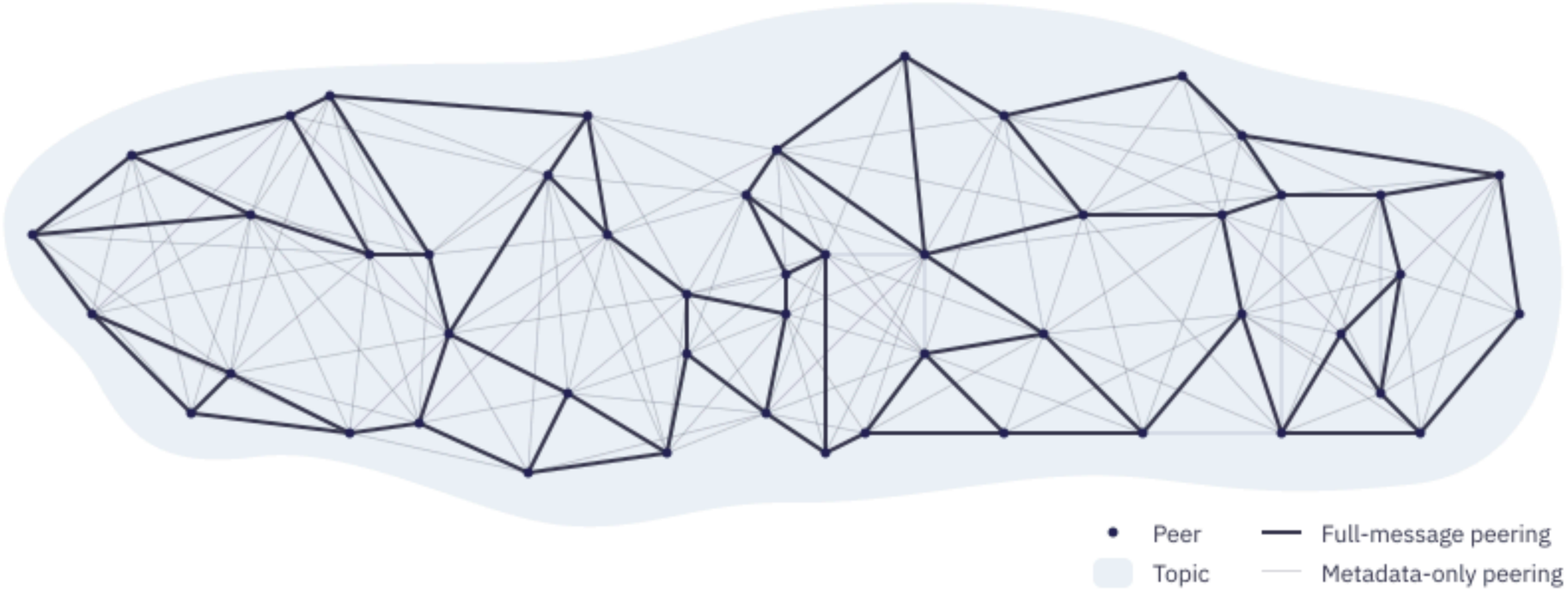
intro to p2p pubsub
what is gossipsub?
how gossipsub works
what's next?

overlay topology

A HYBRID OF TWO CIRCUITS

The gossipsub p2p pubsub overlay is made of two circuits:

- **full-message circuit,** formed by reciprocal links through which messages are forwarded.
- **metadata-only circuit,** carrying gossip about which messages peers have seen.

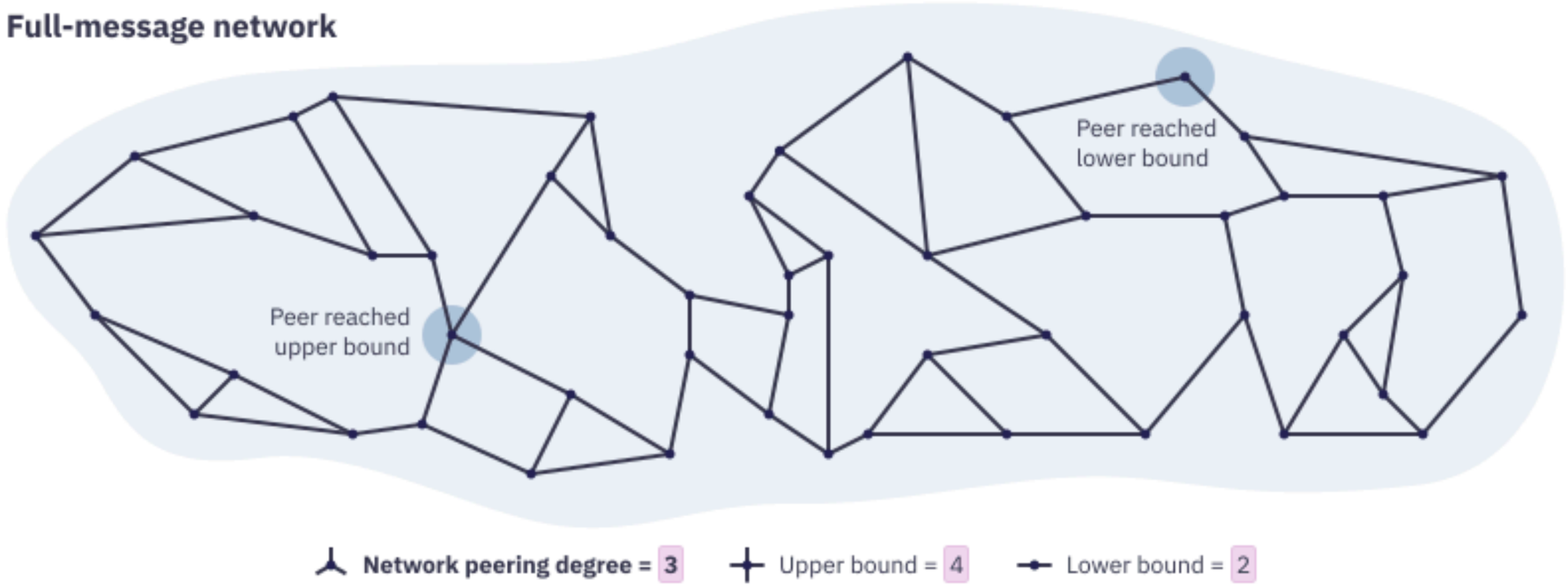


overlay topology

FULL-MESSAGE CIRCUIT (MESH)

Sparsely-connected circuit of reciprocal links between peers that are **actively consuming and forwarding** a topic. Construction is randomized, yet self-stabilizing. Trades latency for bandwidth efficiency.

Each peer strives to maintain a predefined amplification factor D (default: 6) links, taking compensating action at high/low watermarks (default: 4, 12, respectively) to grow or shrink its connectedness.

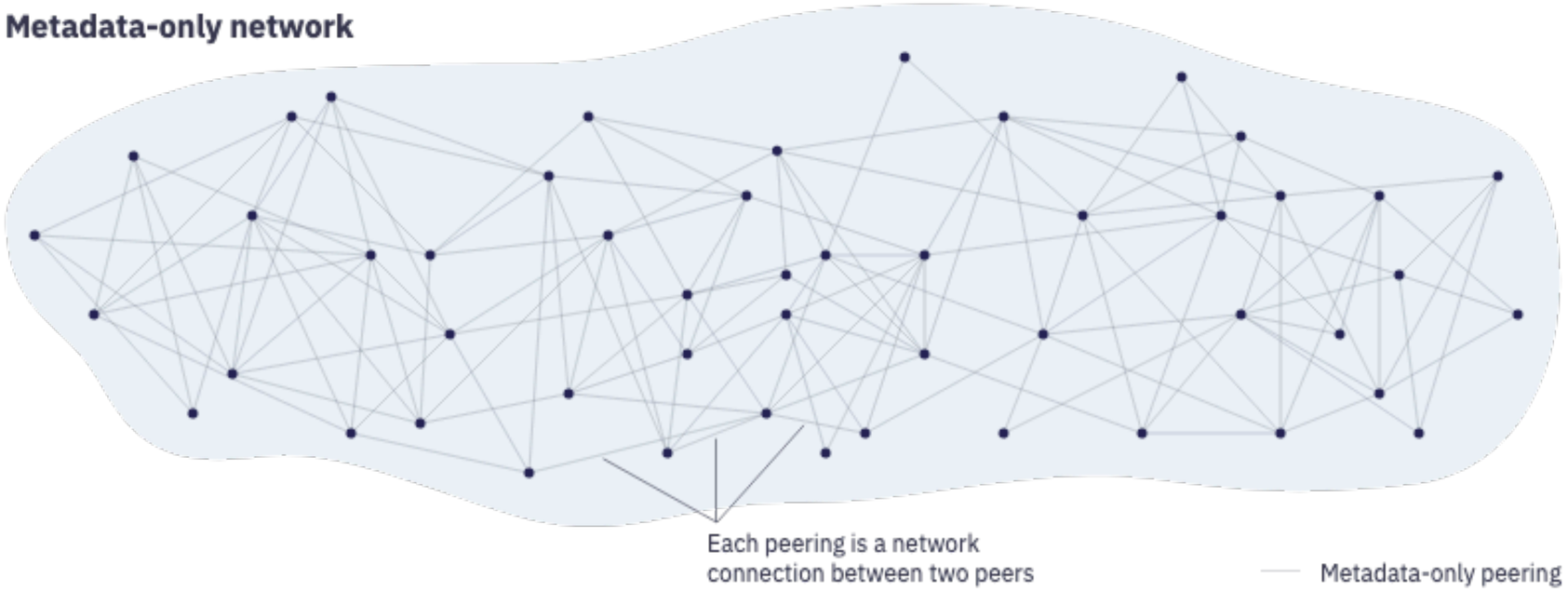


overlay topology

METADATA-ONLY CIRCUIT (GOSSIP)

Densely-connected circuit within a topic, augmenting the mesh with random gossip propagation about which messages are available, and when.

It enables routing decisions to be made (e.g. opportunistic hop jumping, constructing stable trees), and to repair overlay pathologies.



libp2p gossipsub @ devcon5

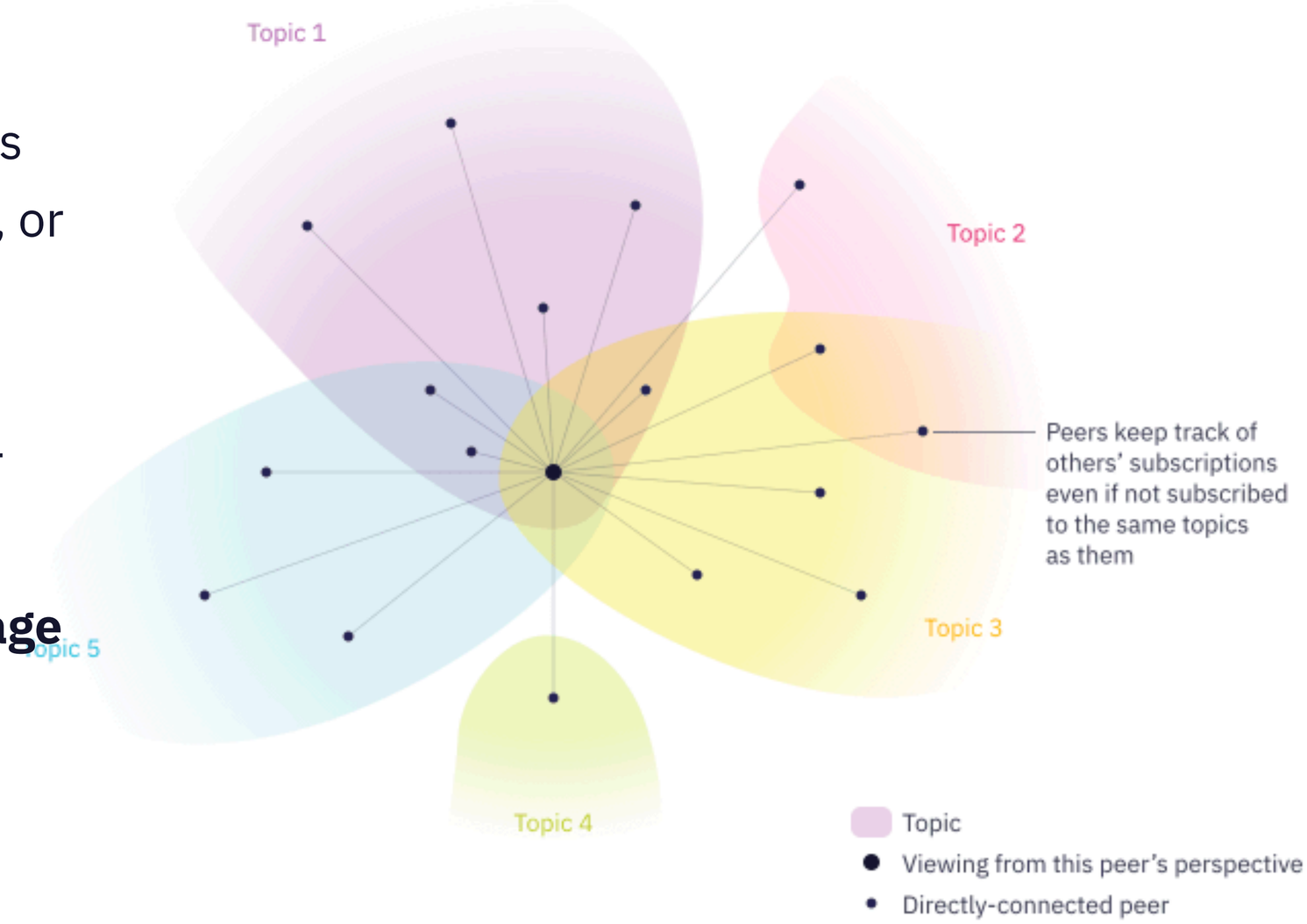
TOPIC MEMBERSHIP

- intro to p2p pubsub
- what is gossipsub?
- **how gossipsub works**
- what's next?

topic membership

OUR VIEW OF THE NETWORK

- Peers can participate in many topics at once. As subscribers, publishers, or both.
- Peers keep a view of all our neighbours' subscriptions, whether they overlap with ours or not.
- **Only subscribers engage in message propagation and gossip for pertinent topics.**



topic membership

SUBSCRIPTION BEACONING

On a new connection, peers exchange the topics they subscribe to:

Over time, they send updates to their peers:

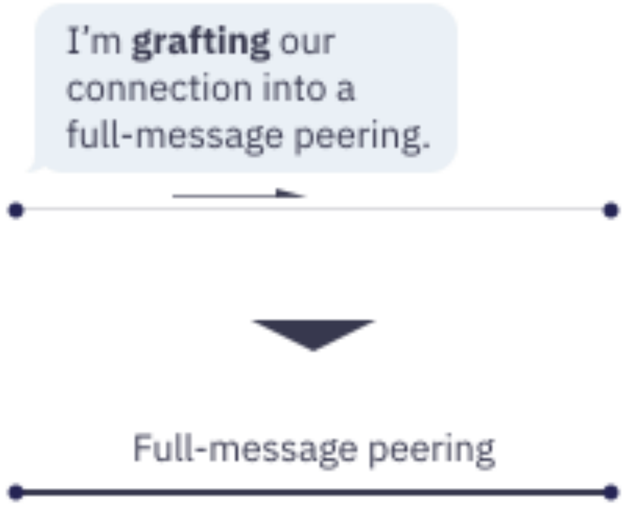
- Beaconing can piggyback on topic messages or control messages like **GRAFT** and **PRUNE**, to minimise signal-to-noise ratio.

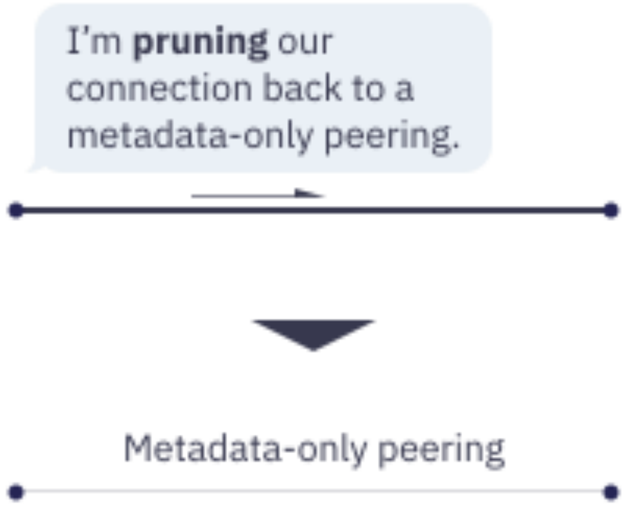
- Beaconing enables:
  - inbound metadata flows
  - ephemeral topic sending (fanout).

topic membership

GRAFTING AND PRUNING

- A ***GRAFT*** control message proposes a two-way full-message, stable link with a peer.
  - If accepted, the peer will respond with a ***GRAFT***. If rejected, we'll get a ***PRUNE*** but exchange gossip.
  - On joining a topic, we ***GRAFT*** a link with D peers, sourcing them from our local view.
- A ***PRUNE*** control message dissolves a two-way full-message, stable link, or rejects a ***GRAFT*** proposal.
  - On leaving a topic, we ***PRUNE*** away all our links on that topic.





topic membership

PEER DISCOVERY

- Before a peer can subscribe to a topic, it must find other peers and establish network connections with them.
- Gossipsub currently relies on **ambient peer discovery, e.g.:**
  - DHT.
  - Local network broadcasts.
  - Centralized trackers or rendezvous points.
- In the near future:
  - **Active peer discovery:** pubsub calls out to the discovery layer on the libp2p stack.
  - **Collaborative peer exchange**.

libp2p gossipsub @ devcon5

MESSAGE FLOW

intro to p2p pubsub
what is gossipsub?
how gossipsub works
what's next?

`message flow`

PUBLISHING MESSAGES

On topics that we subscribe to.

Message source

Creates a new message and broadcasts it to its **full-message peers**.

Peer creates new message of its own

Message sent to all other full-message peers

`message flow`

FORWARDING MESSAGES

On topics that we subscribe to.

Message routers

On receiving the message, they call the local consumer(s), store a copy in the message cache, and forward it to its full-message peers.

We break cycles by:

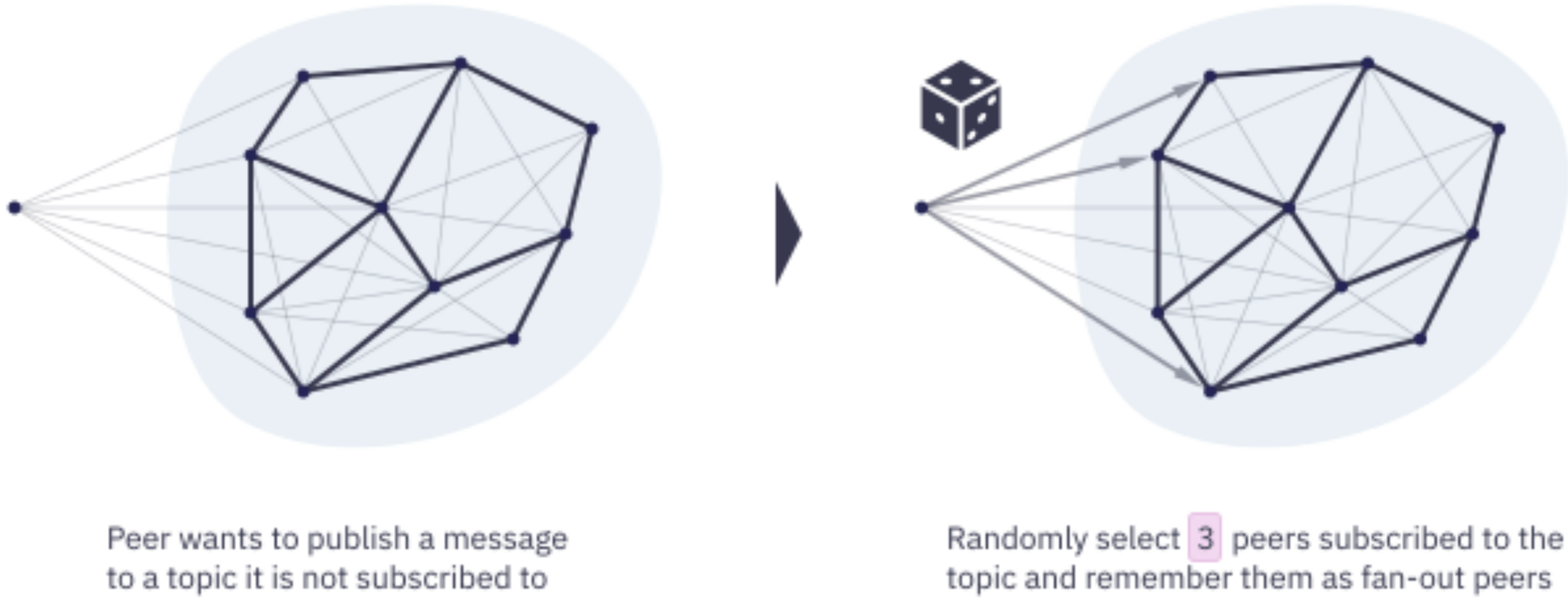
1. ignoring messages we've already seen in the last 2 minutes (default timed cache window).
2. suppressing propagation to the source.
3. suppressing back-propagation.

message flow

PUBLISHING MESSAGES

On topics that we do not subscribe to.

Topic non-members can publish messages to the topic by bootstrapping an temporary **fan-out group** from their local view of the network. No subscription is needed.

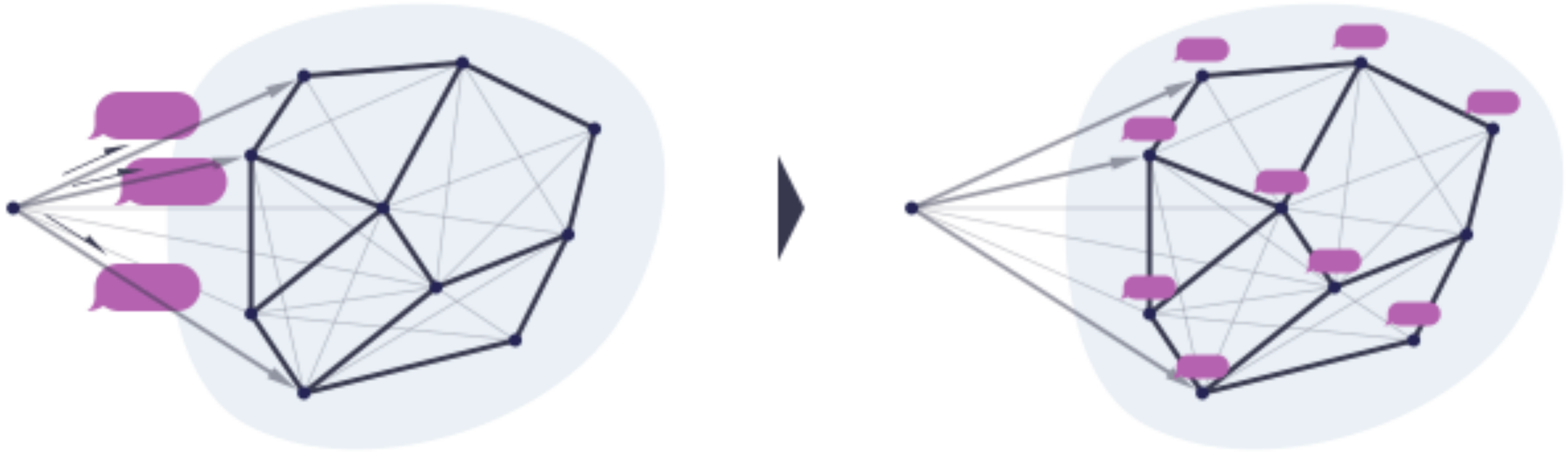


message flow

PUBLISHING MESSAGES

On topics that we do not subscribe to.

Messages get distributed inside the topic mesh once they arrive at a real subscriber.



New message sent to fan-out peers

Once inside the topic, the message is forwarded to all other subscribers as usual

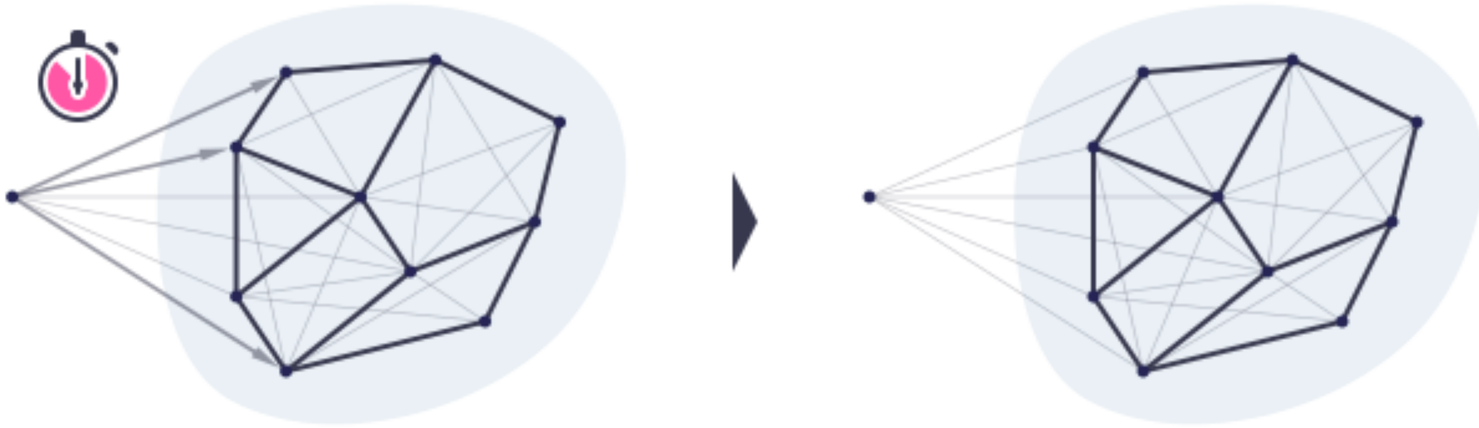
`message flow`

PUBLISHING MESSAGES

On topics that we do not subscribe to.

After an inactivity period (default: 2 minutes), we forget the temporary fanout group and remove it from our local state.

No need to send ***PRUNE*** messages, because we were never participating in the mesh for the topic.

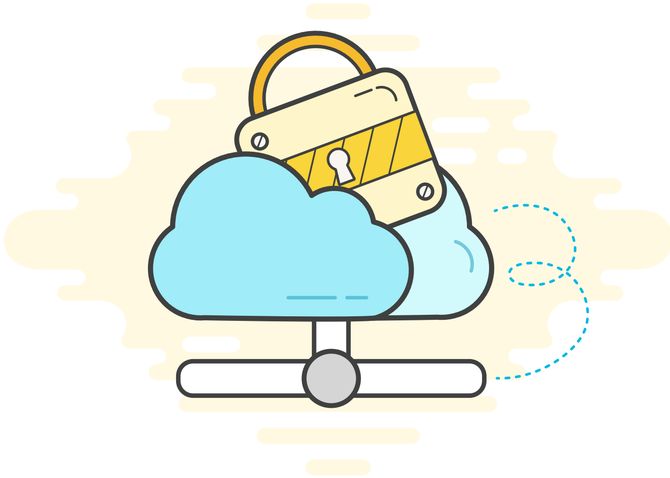


After not publishing any messages for 2 minutes...

All fan-out peerings revert to metadata-only peerings

message flow

TOPIC VALIDATORS AND MESSAGE SIGNATURES



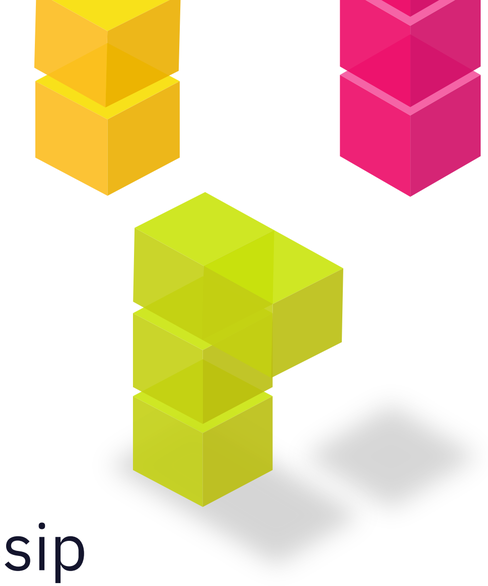
- **Users can attach validation predicates to topics.** Evaluated:
  - when a message is published locally.
  - before forwarding a message.
- **If validation fails, the message is dropped.**
  - Viable for well-formedness, cryptographic soundness, signatures, etc.
- **Async validators: use for heavier logic.**
  - Trade-offs! Careful with sliding message caches!
- Gossipsub can be configured to **sign outbound messages**, and to **strictly validate incoming signatures**.

libp2p gossipsub @ devcon5

HEARTBEAT, STATE, GOSSIP, WIRE

- intro to p2p pubsub
- what is gossipsub?
- **how gossipsub works**
- what's next?

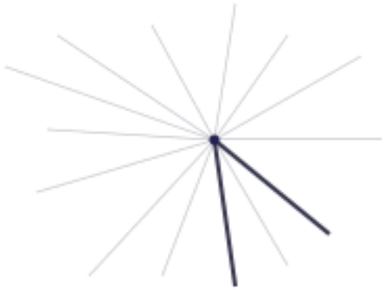
HEARTBEAT PROCEDURE



Every heartbeat interval (1s), gossipsub performs mesh and gossip maintenance functions:

1. Regulate mesh degree for every topic.
2. Emit IHAVE gossip for every topic.
3. Slide the message caches.

DEGREE REGULATION



Start with 2
full-content
peerings

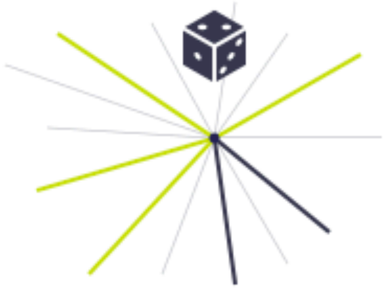
Ideal

Too few | Acceptable amount | Too many

Start with 14
full-content
peerings

Ideal

Too few | Acceptable amount | Too many



Select more peers to graft
to get to the ideal number

Ideal

Too few | Acceptable amount | Too many

Select peers to prune to
get to the ideal number

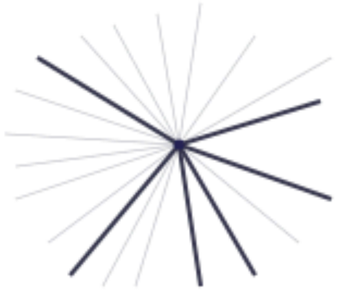
Ideal

Too few | Acceptable amount | Too many

Grafting complete

Ideal

Too few | Acceptable amount | Too many



Pruning complete

Ideal

Too few | Acceptable amount | Too many

LOCAL STATE

Peer state

Topics I subscribe to

Topic ID
Topic 1
Topic 4
Topic 5

Topics I recently sent a message to

(but don't subscribed to)

Topic ID	Time of last sent message
Topic 6	10 seconds ago
Topic 7	35 seconds ago

Peers currently connected to

Peer ID	Topics they subscribe to	Type of peering
Peer B	Topic 1	Full
Peer C	Topic 1	Full
Peer D	Topic 1	Metadata
	Topic 4	Full
Peer E	Topic 5	Full
	Topic 6	Fan-out
Peer F	Topic 7	Fan-out

Recently seen messages

Last 2 minutes / Last few seconds

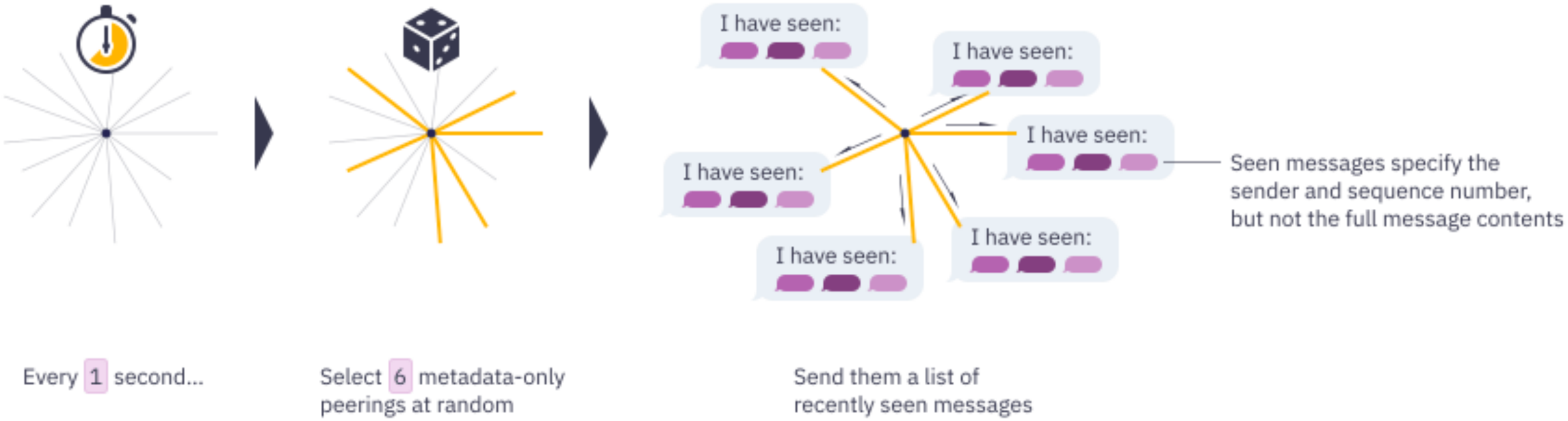
Sender (Peer ID)	Seq no.	Time first seen	Full message
Peer D	2	1 second ago	
Peer E	5	2 seconds ago	
Peer F	10	5 seconds ago	
Peer E	4	10 seconds ago	
Peer D	1	26 seconds ago	
Peer E	3	31 seconds ago	
Peer E	2	45 seconds ago	
Peer C	4	90 seconds ago	

gossip

IHAVE GOSSIP

On every heartbeat, for every topic we are invested in (subscribing or fanout), we push a list of *message IDs* in our message cache to a subset of the peers with which we hold metadata-only links.

We call this *IHAVE* gossip.

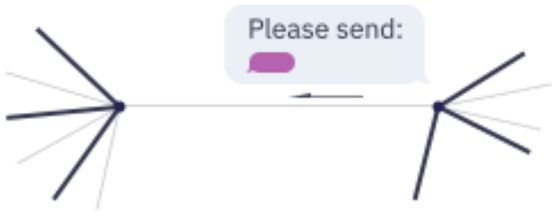


gossip

PULLING VIA *IWANT*

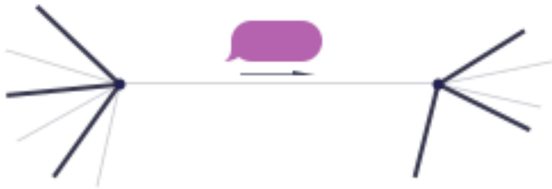
If a peer has not seen some of the messages we have, they request the full payload from us through an ***IWANT*** command.

Peer receives a message from their full-message peers

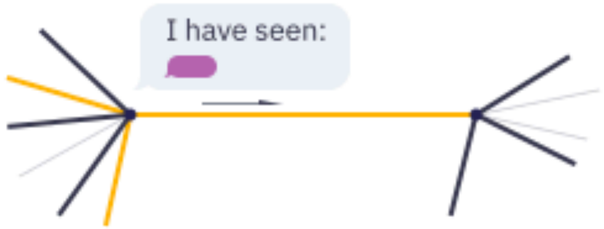


Peer notices that it does not have the gossiped message and requests it

Peer waits until heartbeat and selects random metadata-only peers



Requested message is transferred



Newly received message is gossiped to metadata-only peers

Newly received message is broadcast to full-content peers

ON THE WIRE

Network packet

Application messages

Application message

Sender	Seq no.
Peer A	5

Recipients (Topic ID's)

- Topic 1
- Topic 2
- Topic 3

Message body

```
The heaviest domestic cat on record is 21.297 kilograms.
```

Sender public key	Sender signature
133 179 70 43 69 11 130 124 170 …	111 138 217 17 106 181 66 98 59 …

Application message

I have seen these messages (IHAVE)

Recipient (Topic ID)	Sender	Seq no.
Topic 1	Peer D	4
Topic 4	Peer E	10
Topic 5	Peer F	6

Please send me these messages (IWANT)

	Sender	Seq no.
	Peer D	3
	Peer C	14
	Peer B	1

Subscription changes

For this topic...	I want to...	
Topic 1	☑ Subscribe	☐ Unsub
Topic 6	☐ Subscribe	☑ Unsub
Topic 7	☑ Subscribe	☐ Unsub

Grafting and pruning

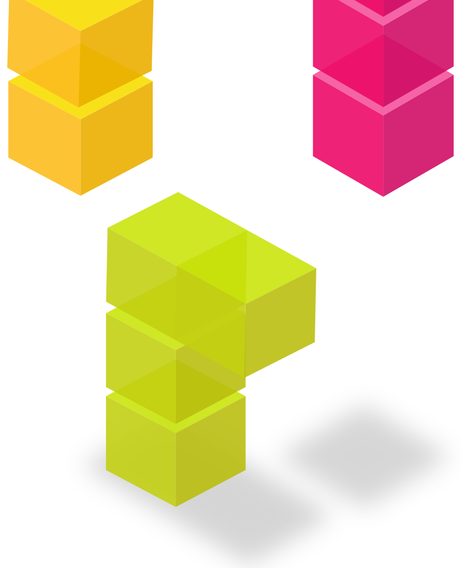
For this topic...	You have been...	
Topic 6	☑ Grafted	☐ Pruned
Topic 7	☑ Grafted	☐ Pruned
Topic 4	☐ Grafted	☑ Pruned

libp2p gossipsub @ devcon5

WHAT'S NEXT

intro to p2p pubsub
what is gossipsub?
how gossipsub works
what's next?

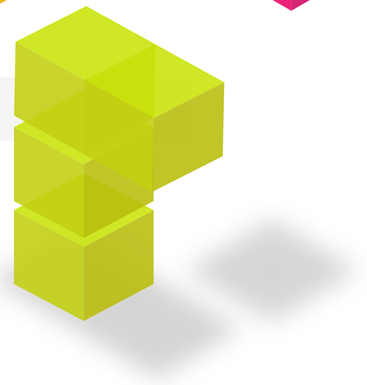
WHAT'S NEXT?



- Formalize Gossipsub into an academic paper.
- Harden the reference implementation (go). Refactor the codebase to enable pluggable routers.
- Integrate discovery mechanisms.
- Various benchmarking efforts underway.

- **Towards episub:**
  - Inspired by Plumtree (Epidemic Broadcast Trees), HyParView, GoCast.
  - Dynamic latency-optimized dissemination trees constructed from gossiped metadata.
  - Self-healing mesh.
  - Adaptive cache windows.
  - Collaborative active and passive membership views, and peer exchange.

https://docs.libp2p.io/concepts/publish-subscribe/



libp2p

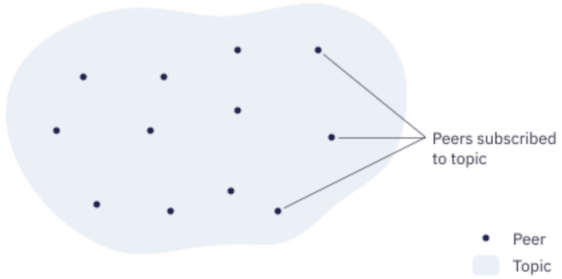
Search...

- Introduction
- Tutorials
- Examples
- Concepts
  - Transport
  - NAT Traversal
  - Secure Communication
  - Circuit Relay
  - Protocols
  - Peer Identity
  - Content Routing
  - Peer Routing
  - Addressing
  - Publish/Subscribe
  - Stream Multiplexing
- Reference
- Contributing
- Help & Discussion

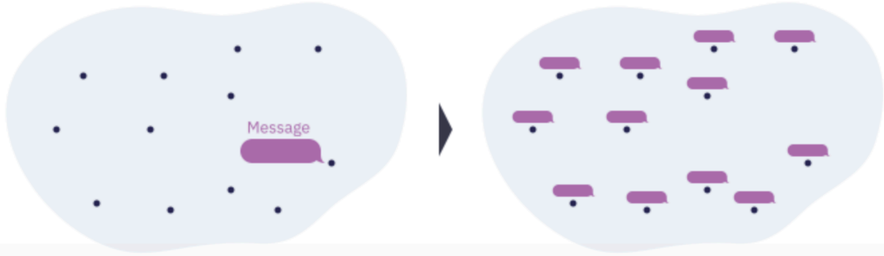
libp2p > Concepts > Publish/Subscribe

PUBLISH/SUBSCRIBE

Publish/Subscribe is a system where peers congregate around topics they are interested in. Peers interested in a topic are said to be subscribed to that topic:



Peers can send messages to topics. Each message gets delivered to all peers subscribed to the topic:



WE  YOU

get involved through GitHub!

shape the future

libp2p/specs

hack on meaty things

libp2p/devgrants

contribute

project repos

libp2p

wanna chat?

LET'S HACK ON THE FUTURE OF P2P NETWORKING TOGETHER.

RAÚL KRIPALANI

GITHUB
https://github.com/raulk

EMAIL
raul@protocol.ai

TWITTER
@raulvk