Enter Gossipsub
A scalable, extensible & **hardened** P2P PubSub Router protocol
Agenda

1) Motivation for P2P PubSub
2) Evolution of libp2p PubSub: FloodSub -> RandomSub -> GossipSub
3) Hardening GossipSub
4) What’s next (or left !)
1) Motivation for P2P PubSub
2) Evolution of libp2p PubSub: FloodSub -> RandomSub -> GossipSub
3) 🌟 Hardening GossipSub
4) 🌟 What’s next (or left !)
Motivation for P2P PubSub
Enable **Real-Time** Applications on the IPFS/libp2p Network
To create the rich experience we got used with the Web 2.0
Without relying on any piece of centralised infrastructure
Orbit Distributed, Real Time Web3 Apps with IPFS and Ethereum

4,688 views • Oct 29, 2015

Ethereum Foundation
25.4K subscribers

Orbit Distributed, Real-time Web3 Application with IPFS and Ethereum

devcon two
Ethereum Developer Conference, 2018 September 19 - 21, Shanghai, China
PubSub key benefits

- Message oriented comms  —> Simple to understand
- Support for different Topologies —> Multiple types of interaction patterns
- Loose coupling  —> Separation of concerns
- Scalable  —> Adapts as your network grows
The challenges of building P2P PubSub System

- Permission-less network —> can’t control who joins/leaves
- Network topology is bottom up (e.g. broker-less)
- Network churn
- Optimizing for Latency, Bandwidth, Delivery Guarantees is about tradeoffs
- Malicious Actors 😈
Evolution of libp2p PubSub
• **libp2p-pubsub is an interface**
• **It enables libp2p to support multiple PubSub router implementations**
The first router - Floodsub
Floodsub

• Simplest possible protocol
• Ambient Peer Discovery (i.e. IPFS Main Network’s DHT)
• Routing is achieved by Flooding
Floodsub
Floodsub
Floodsub
Floodsub
Floodsub
Floodsub

• Pros
  ◦ Simple to implement
  ◦ Very robust (even with high network churn)
  ◦ Minimum latency

• Cons
  ◦ (Really) Bandwidth inefficient
  ◦ Unbounded degree flooding (wastes a lot of resources)
Floodsub

- 100 peers
- 5 msgs/s
- Run for 2s
- Time expansion 10x for viz
A stepping stone - Randomsub
Randomsub

• Experiment to see how the Network performed if we did message forwarding at random

• Pros
  ◦ Still simple to implement
  ◦ Less bandwidth intensive

• Cons
  ◦ Creates dark spots (nodes that don’t get the messages)

• Note, this implementation wasn’t spec’ed, it was done as an experiment
Current generation - Gossipsub 🌟
Gossipsub

- Trades latency for bandwidth efficiency
- Self-stabilising algorithm (nodes cooperate with each other)
- Support for protocol extensions
Gossipsub - Hybrid of 2 Networks

- Nodes construct meshes for message propagation pseudo-randomly.
- Nodes spread metadata to let other peers know of what we have seen (gossip!)
Reciprocal peering agreements between nodes

A node mesh is the nodes that it has peering agreements with

Notes connect to D (degree, default 6) nodes
• Dense network
• Nodes inform others of their local observed state through Gossip
• Nodes inform others of what they have received or what they are missing
Gossipsub - **Subscribing & Unsubscribing**

When a node joins:

- Nodes keep a partial view of the network
- Nodes announce their subscriptions on joining the network
- Subscriptions are updated through the lifetime of the node
Gossipsub - GRAFT’ing & PRUNE’ing

- **GRAFT’ing** - “I want to be part of your reciprocal full message”
  - If the peer rejects, the other node still knows that I’m interested in that topic and therefore will send me metadata.

- **PRUNE’ing** - Dissolve a previous reciprocal full message agreement.
Gossipsub - Message Validators

- Gossipsub has support for custom message validators
- As messages are received, these can be validated before being forwarded
**Gossipsub - Gossip Messages**

- **Local state (Subscription changes, Activity and more)**
- **IHAVE - Share what we have + what we have seen**
- **IWANT - Used when peers request a message they learn about and that they haven’t received yet**
Gossipsub - Heartbeat

- Regular the mesh degree (GRAFT/PRUNE peers to hit the desired degree)
- Emit gossip for each topic
- Slice the message caches
- Heartbeat is done at every EPOCH and piggybacked on regular messages if possible
Gossipsub

- 100 peers
- 5 msgs/s
- Run for 2s
- Time expansion 10x for viz
Floodsub vs. Gossipsub

Generated using https://github.com/vyzo/gerbil-simsub/
Recap:
What libp2p Pubsub with Gossipsub gives us

- Real-Time updates
- Resiliency to network churn
- Adjustable tradeoffs (bandwidth vs. latency)
- Scales to large networks
Hardening Gossipsub
Hardening Gossipsub v1.1
Class of attacks
Class of attack - **Sybil Attack**

- When nodes join with the intent of causing harm
- Can squat areas of the network
- Can degrade the quality of the service
Class of attack - **Eclipse Attack**

- When a node controls what other nodes in the network see/get
Class of attack - **Spam Attack**

- Bombard a node with work until it chokes itself (~equivalent to the Denial of Service attacks)
Mitigation Strategies
The spec is a joy to read ✨
Mitigation - Peer Scoring

- Each Peer maintains a score for each other peer
- The score is the summation of each parameter $P$ multiplied by their weight $W$ to the topic weight $T$

$$\text{Score}(p) = \sum_{i} (w_1(t_i) \times P_1(t_i) + w_2(t_i) \times P_2(t_i) + w_3(t_i) \times P_3(t_i) + w_3b(t_i) \times P_3b(t_i) + w_4(t_i) \times P_4(t_i)) + w_5 \times P_5 + w_6 \times P_6$$

- Thresholds (all negative thresholds)
  - 0, the baseline threshold - peers with score below this threshold are pruned from mesh
  - gossip threshold - peers below this threshold don’t receive gossip
  - publish threshold - peers below this threshold don’t receive flood publishing (optional)
  - graylist threshold - peers below this threshold get their RPCs ignored

- Parameters
  - $P_1$: Time in Mesh for a topic.
  - $P_2$: First Message Deliveries for a topic.
  - $P_3$: (Expected) Mesh Message Delivery Rate for a topic.
  - $P_4$: Invalid Messages for a topic.
  - $P_5$: Application Specific score.
  - $P_6$: IP Colocation Factor.
Mitigation - Flood Publishing

• Flood publish to peers that are a certain publish threshold score

• This means that self publish messages are published to peers that beyond our mesh and instead, get sent to every peer we are connected and interested in topic and that are above the publish threshold score

• The publish threshold is used to decide
Mitigation - Peer Exchange on Prune

- When a node gets Prune’ed, it receives of other candidate peers
- This is a way to rely less on Ambient Peer Discovery and use the Gossipsub network to do the PeerDiscovery itself
Mitigation - Spam Protections

- GRAFT messages for unknown topics are ignored (vs. v1.0 would respond with PRUNE)
- IWANT responses are rate limited
- If spam is observed, it impacts the Score of the peer
Mitigation - Explicit Peering Agreements

• Establish direct connection agreements to keep connections always open

• Great between nodes that are federated or have any special reason to always be connected
We are not done just yet!
GossipSub Evaluation Plan

- **Test Plan - Benchmarking**
  - Test Case 0: Establishing the baseline of Gossipsub

- **Test Plan - Resilience in Adversarial environments**
  - Test Case 1: DoS & Quiet GRAFT Attack
  - Test Case 2: Sybil Attack - Network Degradation & Squatting
  - Test Case 3: Sybil Attack - Sybils are generated in a small pool of IPs
  - Test Case 4: Eclipse Attack against a Target - Distributed GRAFT Attack
  - Test Case 5: Eclipse Attack - Eclipse the entire Network
  - Test Case 6: Cold Boot Attack
  - Test Case 7: DoS (Denial of Service) Attack - Target a set of nodes and get them to halt operation by exhausting their resources
  - Test Case 8: Covert flash Attack
  - Test Case 9: Consensus & Timing Attack
  - Test Case 10: SPAM Attack - IWANT Spam
  - Test Case 11: SPAM Attack - GRAFT Spam
  - Test Case 12: SPAM Attack - Invalid Message Spam
**Testground**

**What is Testground**

Testground's goal is to provide a set of tools for testing next generation distributed systems.

**Table of Contents**

- Background
- How to use Testground
- Team
- Contributing
- License

**Background**

You may have noticed a few test efforts with similar names underway! We are exploring and experimenting a lot, until we land on an end-to-end solution.

- Interplanetary Testbed (IPTB): https://github.com/ipfs/iptb
  - a simple utility to manage local clusters/aggregates of IPFS in a local network
- libp2p testlab: https://github.com/libp2p/testlab
  - a Nomad deployer for libp2p nodes with a DSL for test scenarios
- And others such as https://github.com/ipfs/interop and https://github.com/ipfs/interop-interop-cli

Testground aims to leverage the learnings and tooling resulting from the above efforts for the execution of various types of tests and benchmarks, written in a way that allows the specific commit of IPFS and/or libp2p, and quantifying its characteristics, such as utilisation, stability, Interoperability, etc., when compared to other commits.

Testground aims to be tightly integrated with the software engineering process.

**How to use Testground**

- Installation:
  - `npm install -g testground`
  - `testground --help`

**So... what is testground?**

“A platform to answer **quantitative questions** about the **performance, quality and properties** of **distributed/p2p software**, at various scales.”

Interested? Onboarding session coming soon, stay tuned.
Jupyter Integration into Testground

1) Open Test Plan

2) Adjust the parameters

3) Run and visualise the results!

We are producing a Gossipsub report to be available soon™
Wrapping up 🎁
You can try it today!

This repo contains the canonical pubsub implementation for libp2p. We currently provide three message router options:

- Floodsub, which is the baseline flooding protocol.
- Randomsub, which is a simple probabilistic router that propagates to random subsets of peers.
- Gossipsub, which is a more advanced router with mesh formation and gossip propagation. See spec and implement for more details.

PSA: The Hardening Extensions for Gossipsub (Gossipsub V.1.1) can be found under development at https://github.com/libp2p/go-libp2p-pubsub/pull/263

Repo Lead Maintainer

@vyzo
This repo follows the Repo Lead Maintainer Protocol

Table of Contents

- Install
- Usage
- Implementations
- Documentation
- Tracing
- Contribute
- License

Implementation status

Legend: ✓ = complete, ✂ = in progress, ! = not started yet

<table>
<thead>
<tr>
<th>Name</th>
<th>v1.0</th>
<th>v1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>go-libp2p-pubsub (Golang)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>gossipsub-js (JavaScript)</td>
<td>✓</td>
<td>!</td>
</tr>
<tr>
<td>rust-libp2p (Rust)</td>
<td>✂</td>
<td>!</td>
</tr>
<tr>
<td>py-libp2p (Python)</td>
<td>✓</td>
<td>!</td>
</tr>
<tr>
<td>jvm-libp2p (Java/Kotlin)</td>
<td>✓</td>
<td>!</td>
</tr>
</tbody>
</table>
And many others that have contributed by reviewing the spec, opening issues and testing the code ❤️
Thank you!

Join the discussion

- Chat at #libp2p in freenode (with a matrix.org bridge!)
- Forum at https://discuss.libp2p.io/