P2P Networking

Fusaka Interop @ Berlin ‘25

Raúl Kripalani & Marco Munizaga
Ethereum Foundation

Agenda

[10min] High-level overview.

[20min] Propagation layer planning.

- [10min] Gossipsub evolution.
- [10min] Next-gen propagation.

[5min] Transport layer planning.

[5min] Benchmark and Simulation.

[5min] Telemetry and Analysis.

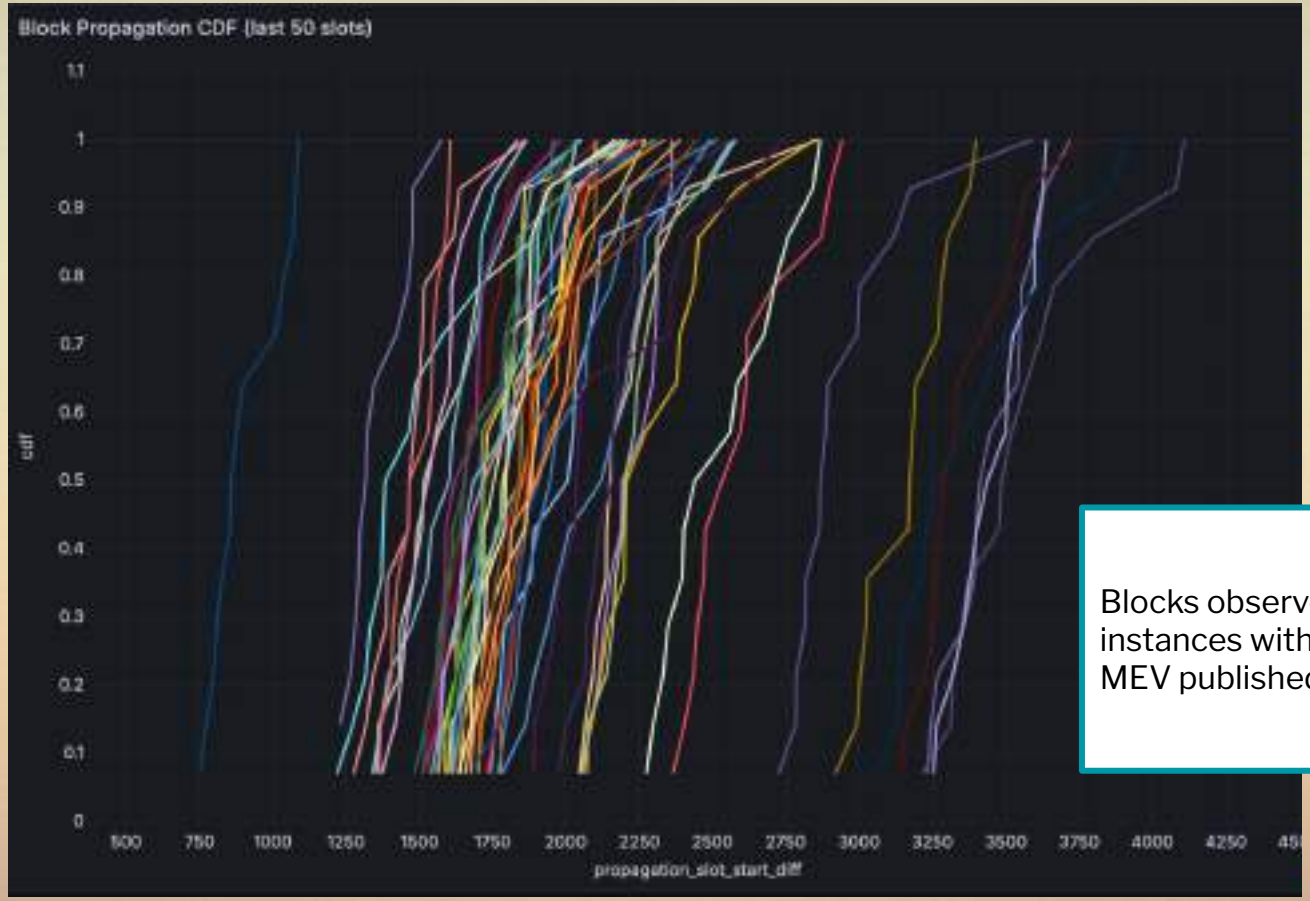
[10min] Paranoid mode on (Reliability, Security, Privacy, Anonymity).

[15min] First principles thinking network stack design.

[10min] Takeaways and conclusions.

Blocks

What's going on?



Blocks observed through Xatu instances within 500ms (likely MEV published)

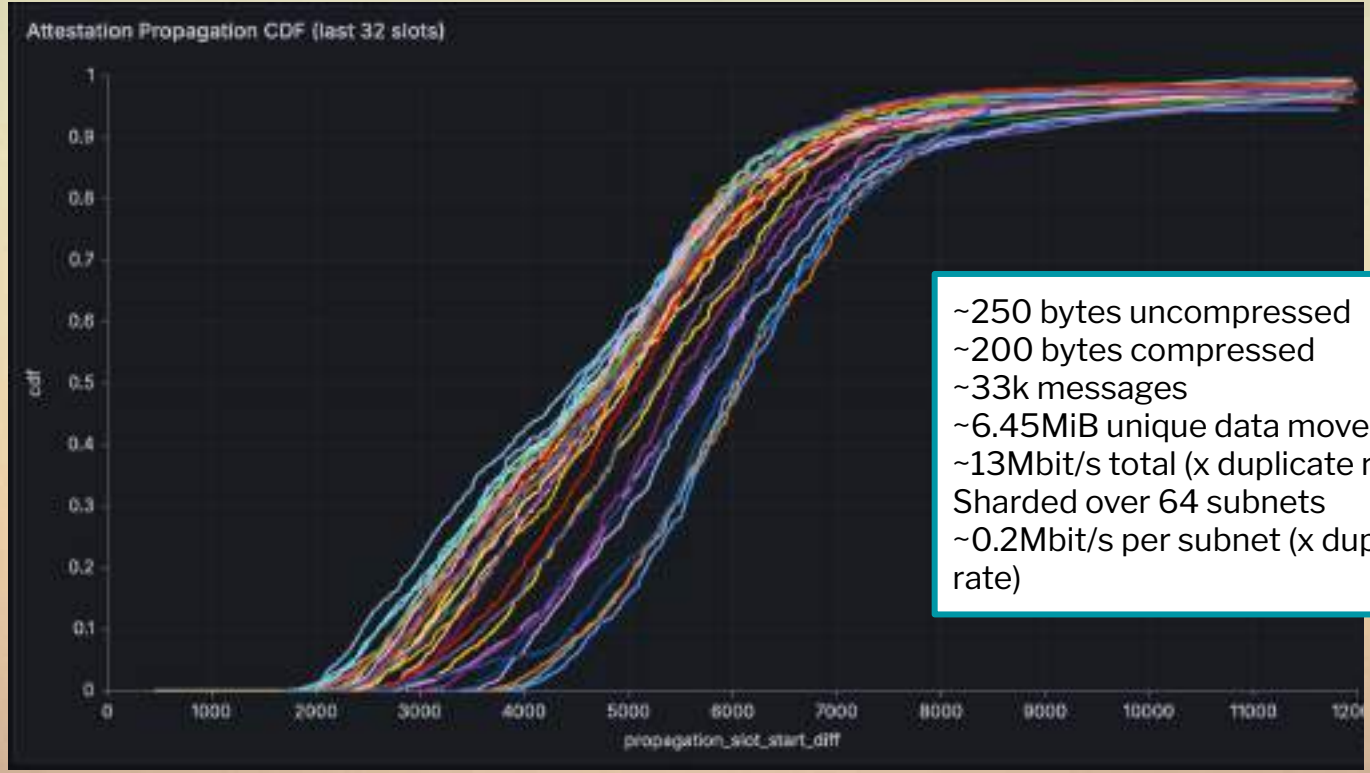
Block sizes

What's going on?

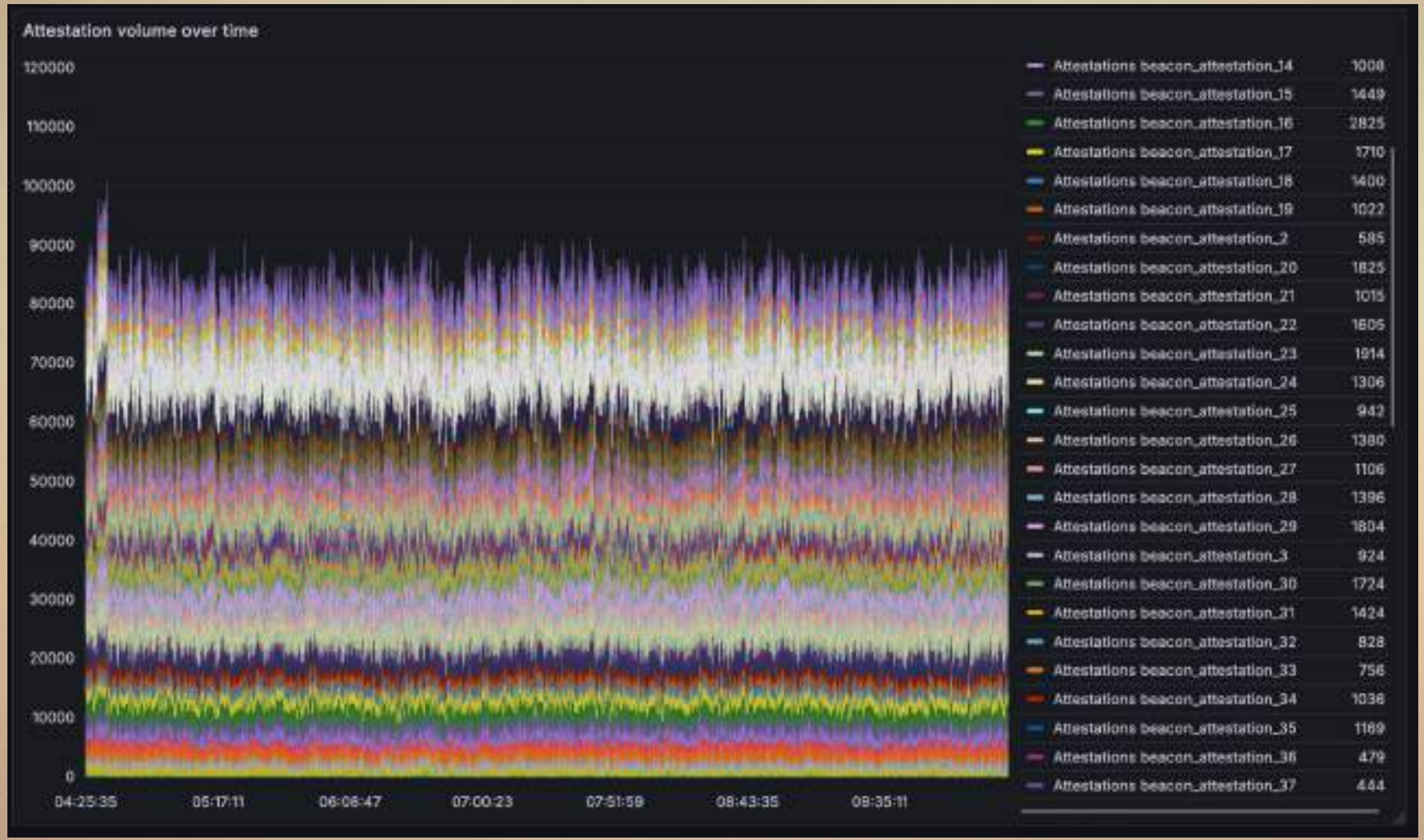
Attestations

NOTE: reviewing accuracy

What's going on?



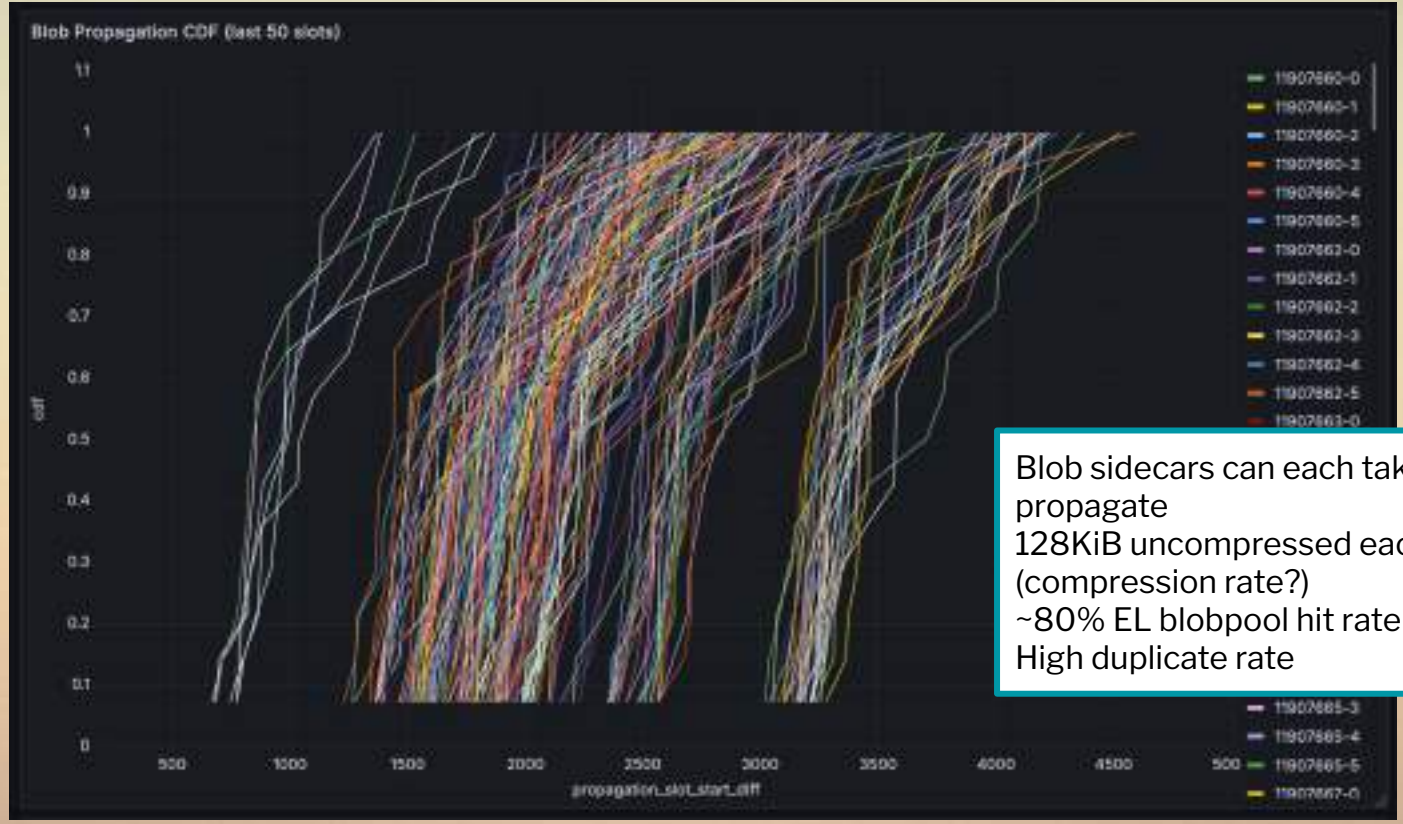
~250 bytes uncompressed
~200 bytes compressed
~33k messages
~6.45MiB unique data moved in ~4s
~13Mbit/s total (x duplicate rate)
Sharded over 64 subnets
~0.2Mbit/s per subnet (x duplicate rate)


Attestation volume over time
120000
110000
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
04:25:35
05:17:11
06:08:47
07:00:23
07:51:59
08:43:35
09:35:11
Attestations beacon_attestation_14 1008
Attestations beacon_attestation_15 1449
Attestations beacon_attestation_16 2825
Attestations beacon_attestation_17 1710
Attestations beacon_attestation_18 1400
Attestations beacon_attestation_19 1022
Attestations beacon_attestation_2 585
Attestations beacon_attestation_20 1825
Attestations beacon_attestation_21 1015
Attestations beacon_attestation_22 1605
Attestations beacon_attestation_23 1914
Attestations beacon_attestation_24 1306
Attestations beacon_attestation_25 942
Attestations beacon_attestation_26 1380
Attestations beacon_attestation_27 1106
Attestations beacon_attestation_28 1396
Attestations beacon_attestation_29 1804
Attestations beacon_attestation_3 924
Attestations beacon_attestation_30 1724
Attestations beacon_attestation_31 1424
Attestations beacon_attestation_32 828
Attestations beacon_attestation_33 756
Attestations beacon_attestation_34 1036
Attestations beacon_attestation_35 1169
Attestations beacon_attestation_36 479
Attestations beacon_attestation_37 444

Blob sidecars

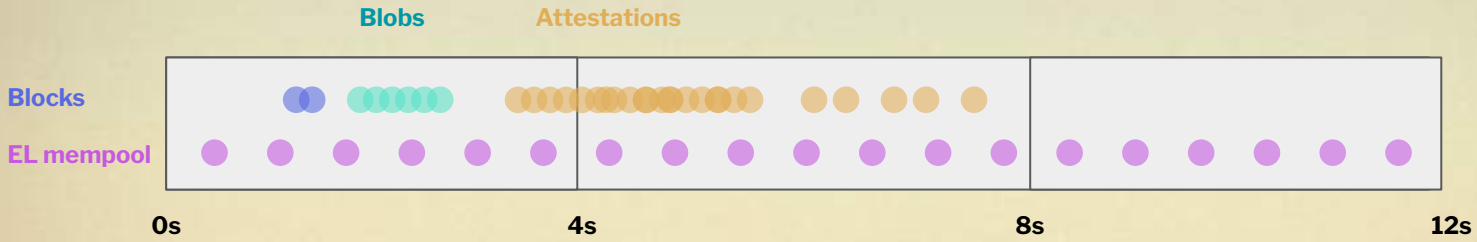
+ EL mempool traffic
+ Aggregates (marginal)

What's going on?



Blob sidecars can each take ~1s to propagate
128KiB uncompressed each (compression rate?)
~80% EL blobpool hit rate
High duplicate rate

Status quo



Pulsatile/bursty bandwidth utilization. Practically idle for over 5 seconds (barring mempool trickle).

Attestations can be directed to aggregators, but inefficiently bouncing around the gossip network.

Majority of blocks likely received very quickly as MEV relays have hyperconnectivity, less dependent on unstructured gossip (can we identify those peers, their scores should be very high?)

Besides reducing bandwidth wastage, can we leverage idle network time to do other work?

What are we designing *for*?

Requirements and new pressures:

- Slot restructuring => changes propagation patterns.
- Slot time compression => increases loads.
- 100M gas, towards 300M gas => larger payloads.
- 8x blobs PeerDAS => feasible with incremental gossip improvements.
- Towards 1000x blobs => requires major re-architecture.
- BAL => increases block sizes.
- ePBS => modifies slot architecture.
- FOCIL => introduces new network interactions.
- 3SF => introduces new network interactions.

Constraints

EIP-7870: Hardware and Bandwidth Recommendations

Node Type	Storage	Memory	CPU Cores / Threads	PassMark CPU Rating	Bandwidth Download / Upload
Full Node	4 TB NVMe	32 GB	4c / 8t	~1000 ST / 3000 MT	50 Mbps / 15 Mbps
Attester	4 TB NVMe	64 GB	8c / 16t	~3500 ST / 25000 MT	50 Mbps / 25 Mbps
Local Block Builder	4 TB NVMe	64 GB	8c / 16t	~3500 ST / 25000 MT	100 Mbps / 50 Mbps

Focus areas

Propagation
Gossipsub, network coding, structured topologies, backbones, etc.

Transport
QUICk pipes, low latencies, connectivity

Simulation & Benchmarking
A unified workbench

Telemetry & Analysis
Probes, traces, warehouse, fresh dashboards, notebooks

Security & reliability

First-principles thinking

Collaborations

Quick wins vs. larger changes

What can we focus on in the short-term to alleviate bottlenecks?

- Implementation deep profiling & hotspot finding
- Gossipsub quick wins (=> Marco)
- Transport quick wins enabling parameter changes (e.g. larger fanouts, e.g. D=32)
- Resolving implementation disparities

go-libp2p	QUIC ✅	Yamux ✅	TLS 1.3 ✅	Early muxer negotiation ✅
rust-libp2p	QUIC ✅	Yamux ✅	TLS 1.3 ✅	Early muxer negotiation ✅
nim-libp2p	QUIC ✅	Yamux ✅	TLS 1.3 ✅	Early muxer negotiation ✅
js-libp2p	QUIC ❌	Yamux ✅	TLS 1.3 ❌	Early muxer negotiation ❓

Propagation layer
Gossipsub evolution

The problem

- Gossipsub is designed for small messages (MTU sized or less).
- We're sending bigger messages with blobs
  - Duplicate messages are a problem
    - Wasted Bandwidth
    - Increased latency
  - Rewarding Duplicate messages is a problem (P_3 scoring)

General kinds of solution

- Send fewer duplicate messages
- Code messages differently. Fewer “duplicate” parts, and more useful parts.
  - RLNC
  - Cell level propagation with Erasure Coding

General kinds of solution

- **Send fewer duplicate messages <- Focus here for this section**
- Code messages differently. Fewer “duplicate” parts, and more useful parts.
  - RLNC
  - Cell level propagation with Erasure Coding

Send fewer duplicates: Terminology

Eager Push: A peer sends a full message directly to a peer right away. Current Behavior for mesh peers.
Lazy Push: A peer first sends an `IHAVE` and will only send the message if it is requested by the peer. Current behavior for gossip peers.

Send fewer duplicates: Overview

1. `**IANNOUNCE**`: Lazy push controlled by sender: https://github.com/libp2p/specs/pull/653
2. Message **Preamble** + **IAMRECEIVING**: https://github.com/libp2p/specs/pull/654
3. **Generalised Gossipsub**: https://github.com/libp2p/specs/pull/664
4. **PPPT**, lazy push ratio as a function of hop count: https://ethresear.ch/t/pppt-fighting-the-gossipsub-overhead-with-push-pull-phase-transition/22118
5. **Choke Extensions**: Lazy push controlled by receiver. https://github.com/libp2p/specs/pull/681

Send fewer duplicates

- We should improve current gossipsub performance for Fusaka
- What do we think we can ship?

Send fewer duplicates: a case for Choke Extensions

- Two gossipsub protocol changes
  - Pair of new control messages: Choke/Unchoke.
    - Control whether a peer lazy pushes or eager pushes.
  - Mesh peers MAY send IHAVEs in lieu of pushing a message. (This is typically done in response to being Choked.)

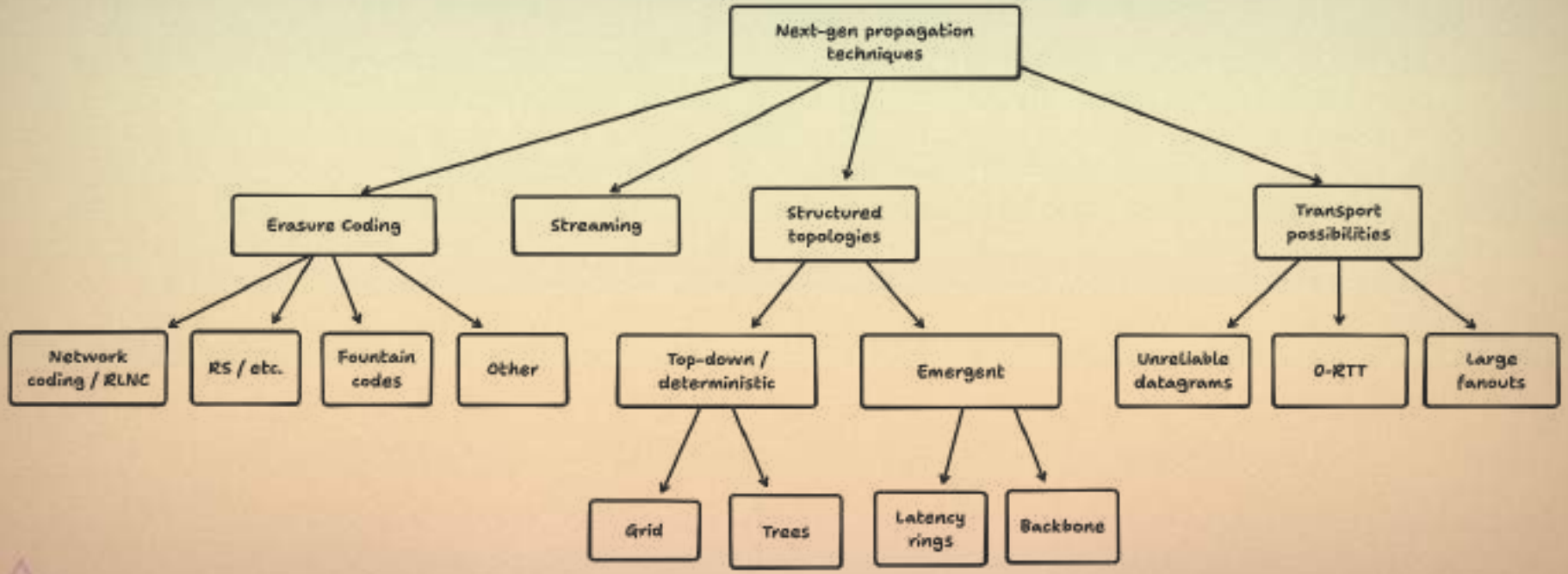
- Opens the protocol to implementation optimizations:
  - Enables receiver controlled lazy push (plumtree model)
  - Enables sender controlled lazy push (IANNOUNCE, PPPT)

- Minimal spec change
- Thoughts?

Propagation layer

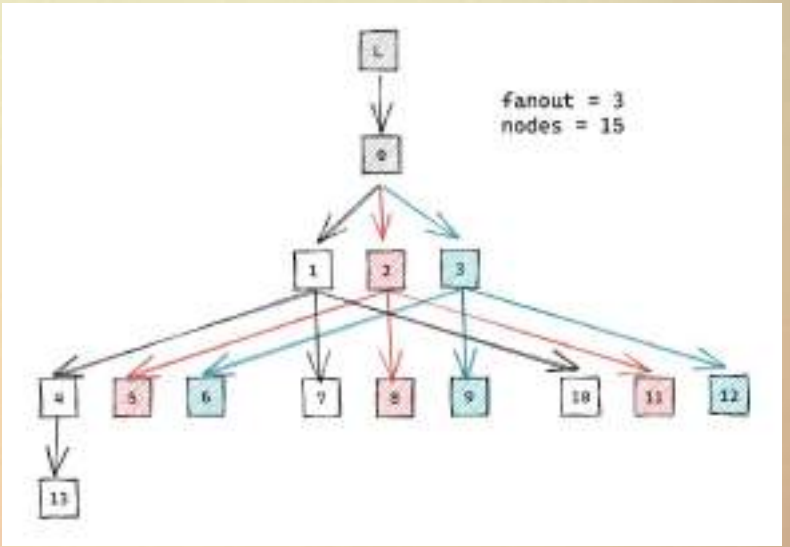
Where do we go next?

Where do we go next?



Network coding (e.g. RLNC)

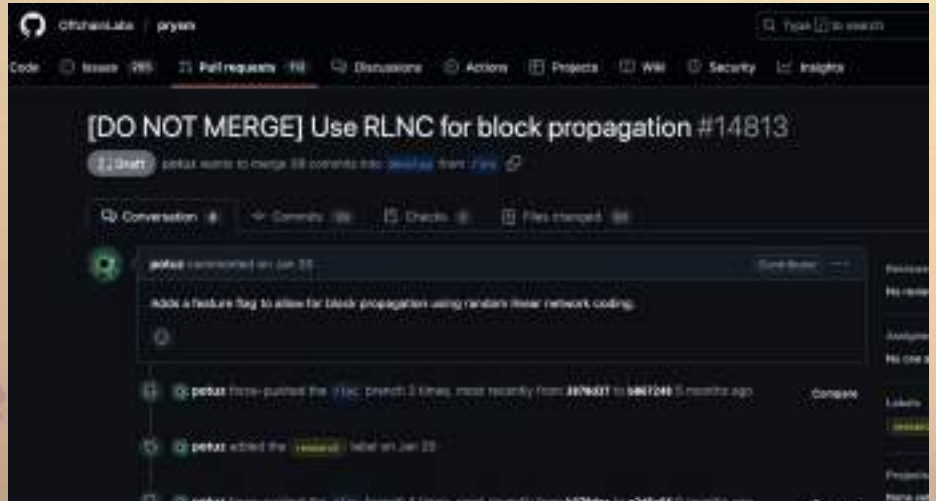
- Enables us to spread large payloads in parallel, through multiple network paths, saturating all links.
- Achieved via chunking + erasure coding + cryptography to enable secure forwarding/recombination, verifiability, and ease of reconstruction.
- Applicable to blocks, perhaps to blobs. Not applicable to attestations.
- Requires topological changes to reap the benefits: can explore randomized, tree-based, and others. Potentially increase the fanout massively.
- Mechanical sympathy: datagram orientation, MTU-sizing,
- Can lead to massive simplification of gossip layer 👍
- Solana Turbine uses EC, no RLNC. Caveat: it benefits from deterministically shuffled routing topology because of permissioned validator set and participation.

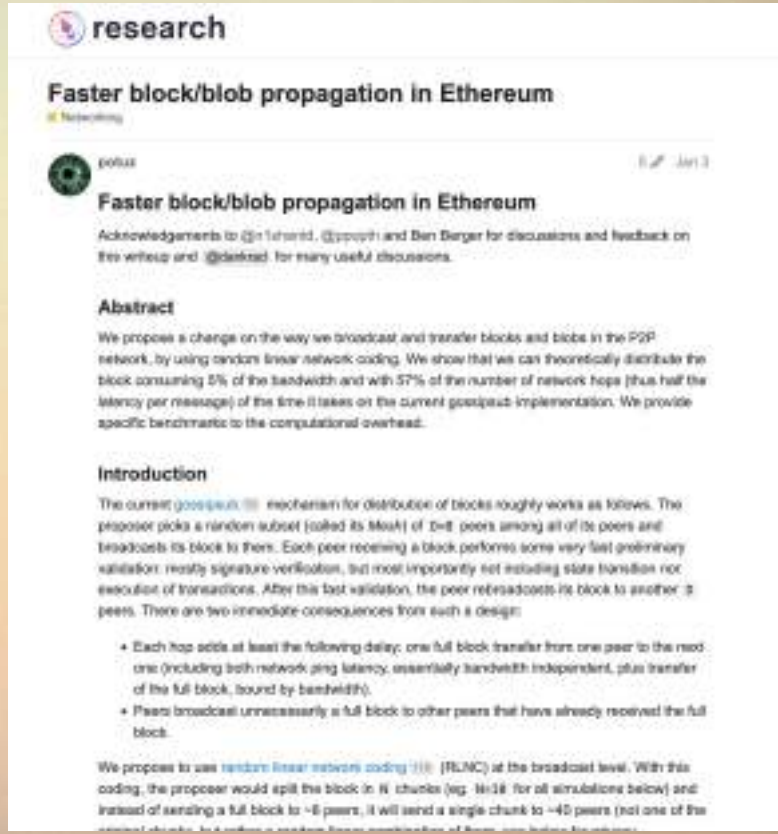


Network coding (e.g. RLNC)

Next planned steps:

- Land QUIC unreliable datagrams on libp2p
- Rebase existing prototype on top of it
- Simulate various topologies, strategies, and parameters.





Transport layer

[Marco] Transport layer

Transport layer planning: prioritizing QUIC, mplex deprecation, QUIC unreliable datagrams, 0-RTT + session resumption.

[Marco] Transport layer: mplex deprecation

Deprecation PR:
https://github.com/libp2p/specs/pull/661/files

[Marco] Transport layer: A QUIC introduction

- QUIC: A UDP-Based Multiplexed and Secure Transport
- Why QUIC:
  - Integrated connection establishment and encryption. 1-rtt vs 3-rtt* in tcp+{tls,noise}
  - Avoid TCP head-of-line blocking
  - Stream and Data Flow Control
  - Stream Error codes and stream resets
  - 0-RTT connections
  - Unreliable Datagrams
  - Connection Migration & Server Preferred Address
  - IETF standard
  - All in user space!

*sometimes 4, ask me later

[Marco] Transport layer: QUIC Datagrams

- Think UDP Packets but better
- Benefits over existing solutions (See also https://datatracker.ietf.org/doc/html/rfc9221#section-2)
  - Single authentication/encryption across reliable/unreliable data flows. Reduced latency.
  - Leverage QUIC's loss recovery during handshakes. Faster handshakes.
  - Single congestion controller across reliable/unreliable data flows. More effective
- Open up design space to datagrams

[Marco] Transport layer: QUIC 0-RTT connections

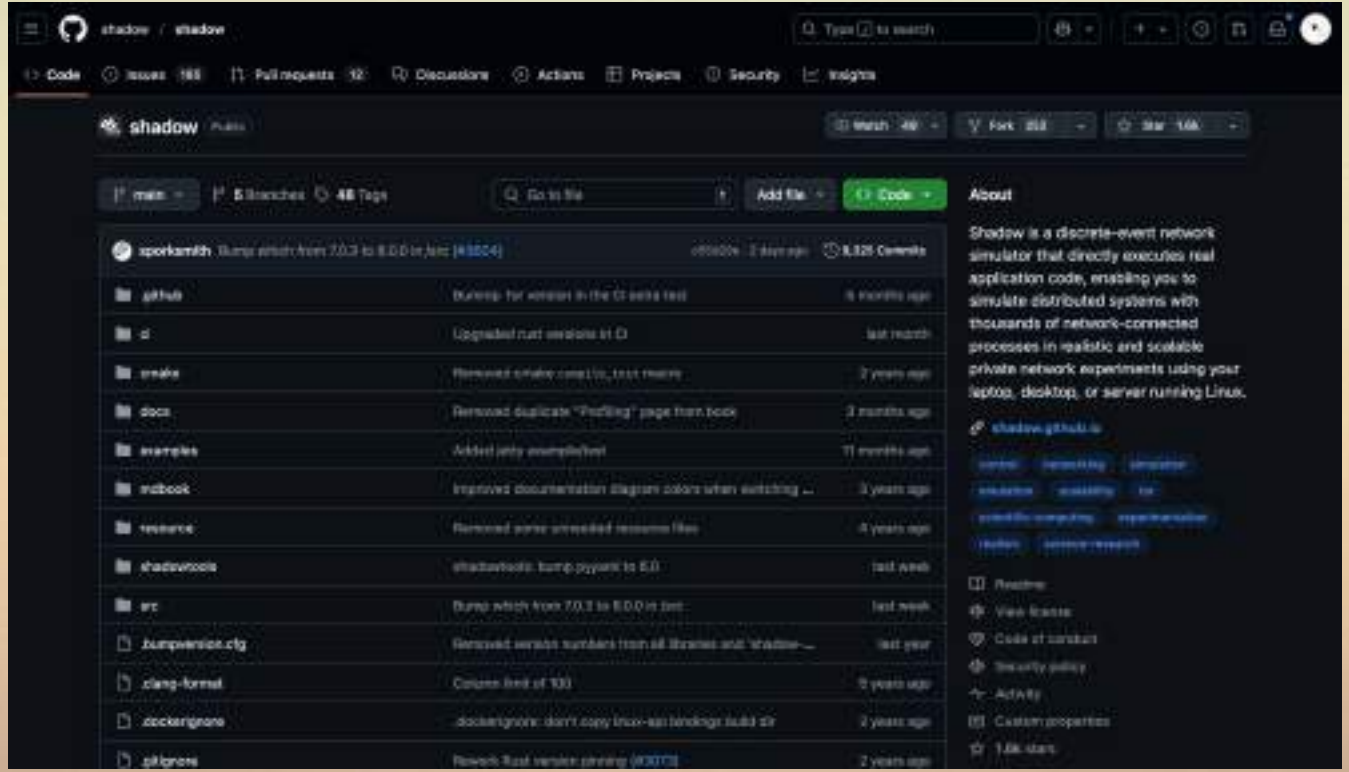
- TLS Session Resumption.
  - After initial connecion. Future connections can be 0-RTT
- “Lightweight” connections

[Marco] Transport layer: Summary, why QUIC

- Integrated connection establishment and encryption. **1-rtt** vs 3-rtt* in tcp+{tls,noise}
- **Avoid TCP head-of-line blocking**
- Stream and Data **Flow Control**
- Stream Error codes and stream resets
- **0-RTT** connections
- Unreliable **Datagrams**
- Connection Migration & Server Preferred Address
- IETF standard
- All in user space!

Benchmark and simulation

Simulation and Protocol Benchmarking



Simulation and Protocol Benchmarking

- Shadow
  - Discrete event simulator with simulated time.
  - Efficient way to run protocol simulations.
  - Thousands of nodes on one machine without having process contention skew timing results.
  - Implementation agnostic. Runs linux binaries

Simulation and Protocol Benchmarking

- Shadow
  - Discrete event simulator with simulated time.
  - Efficient way to run protocol simulations.
  - Thousands of nodes on one machine without having process contention skew timing results.
  - Implementation agnostic. Runs linux binaries
- Gossipsub Interop tester: https://github.com/libp2p/test-plans/pull/648
  - Cross implementation interoperability tester
  - Protocol benchmarking
  - Uses shadow

Gossipsub Interop Tester

- Instructions
  - The tester defines a set of instructions that implementations should handle
  - Examples:
    - Connect to peer X
    - Join topic T
    - Publish Message M
- Scenarios
  - A set of instructions to test a certain scenario
  - Examples:
    - Every 12s a random node in the network will publish a message.
    - A batch of messages are published together.
    - A network partition partially splits the network, can some nodes bridge the split?
    - The network has a large number of malicious or misbehaving nodes.

Gossipsub Interop Tester

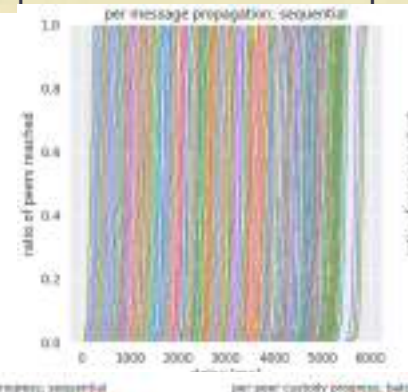
- Once an implementation is written to handle instructions, it can be run on any scenario.
- The simulated network can be composed of various implementations of different concentrations.
    - Example:
        - A network of only go-libp2p nodes
        - A network of only rust-libp2p nodes
        - A network of 50/50 go/rust nodes.
        - Etc.

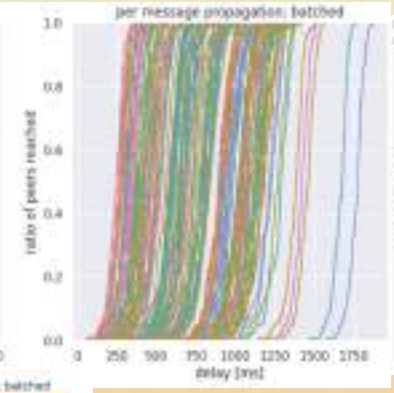
Gossipsub Interop Tester, as a protocol benchmark tool

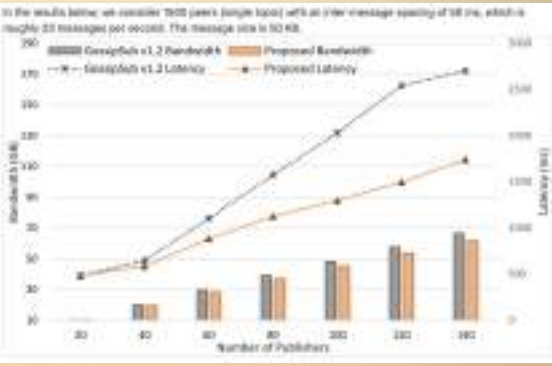
- We have many proposals to improve Gossipsub. Each proposal comes with graphs to show improvements. The graphs demonstrate a specific scenario that is improved.

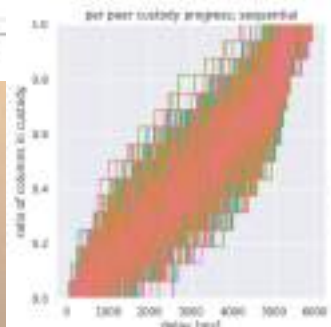
Gossipsub Interop Tester, as a protocol benchmark tool

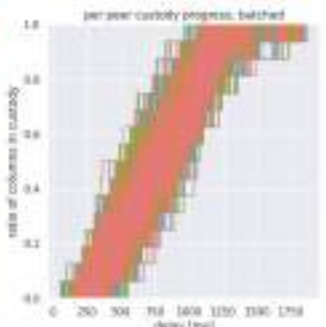
- We have many proposals to improve Gossipsub. Each proposal comes with graphs to show improvements. The graphs demonstrate a specific scenario that is improved.





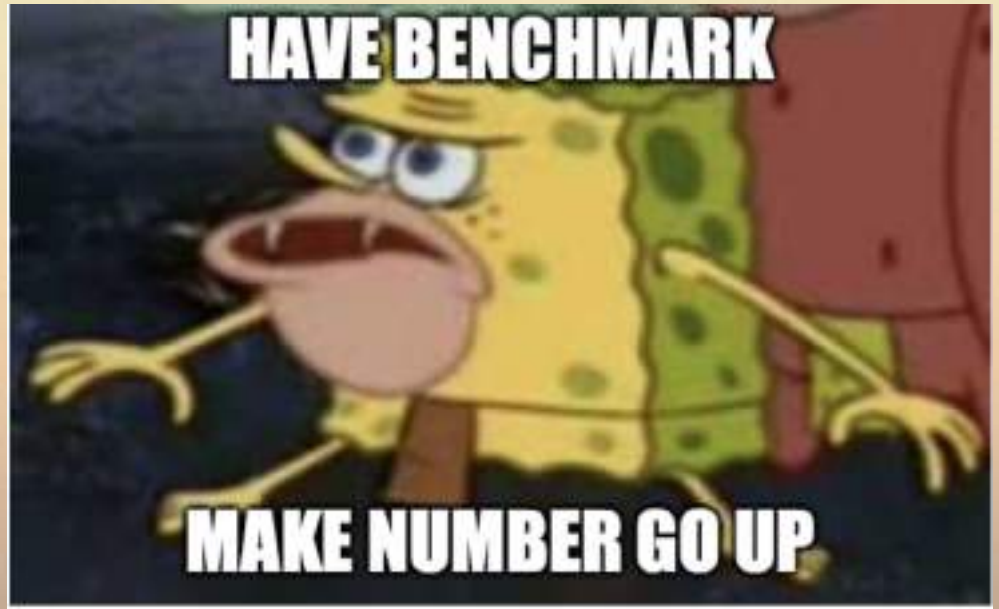






Gossipsub Interop Tester, as a protocol benchmark tool

- We haven't agreed on a well defined scenario we want to improve.
- Scenario should be representative of the workloads.
- Once we agree on the scenario, we just make number go up (or down depending on the metric!)



Gossipsub Interop Tester, as a protocol benchmark tool

- Let's us run **future proposals on all scenarios**, not just the ones we think will be improved.
- Let's us test that all implementations see expected improvements. And in combination! (interop)

- Real life example: Nagle's algorithm & Delayed Acks
  - Both improve bandwidth utilization
  - But, incompatible with each other!

Gossipsub Interop Tester, as an accurate simulation

- Feedback from telemetry/metrics, more accurate simulations

Telemetry and Analysis

Status quo

- Multiple teams, including academics, are independently researching DAS, propagation, EL mempool, Engine API, connectivity, and anonymity.
    - Insights often overlap, offering confirmation and added value.

- Client forks are common for instrumentation, but typically used for short-term analysis, whose insights become stale quickly.
    - What if they ran continuously?

- Many groups operate scanners and tracers, but data is often siloed and hard to correlate. Some produce sliding-window reports, but continuous time series would be valuable.

- Over time, EthPandaOps has built a rock-solid observability stack we could lean on.

Our plan to level up

Converge

- Centralize all our traces and measurements under EthPandaOps' Clickhouse (e.g. make Nebula stream into Clickhouse).
- Adopt schemas that enable easy correlation with other events / metrics.

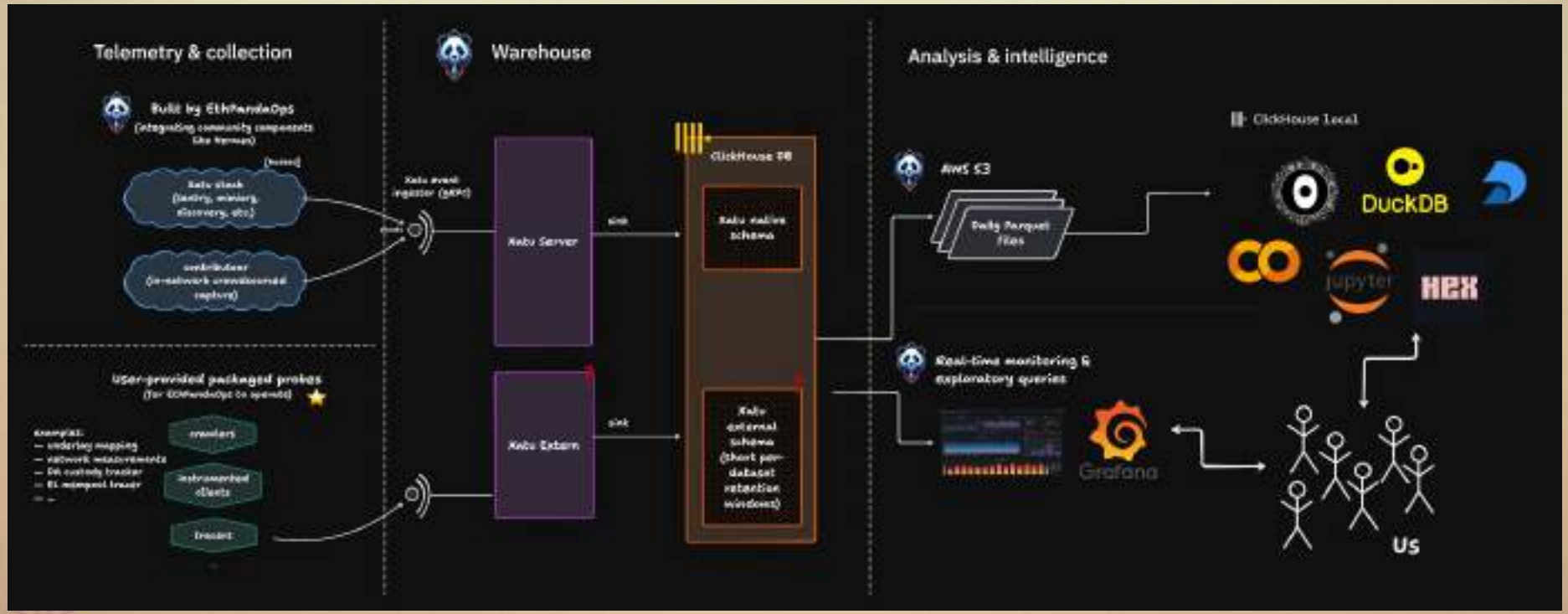
Capture

- Open up the firehose for raw gossipsub traces (sampling!).
- Adopt a libp2p tracing standard to capture connection, stream, discovery events (WIP).
- Build small focused probes that collect targeted measurements (chatting with EthPandaOps to arrange hosting.)

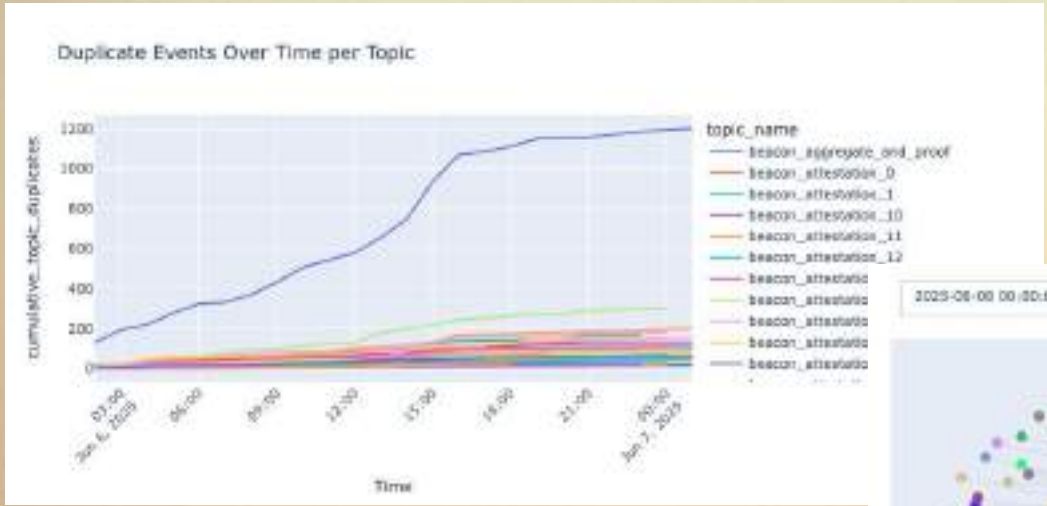
Collaborate

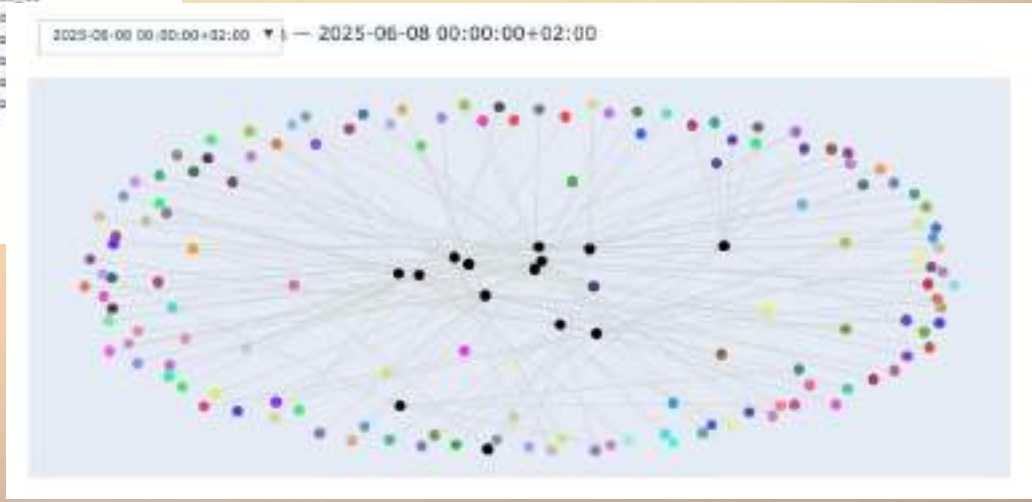
- Create and share Grafana charts based on Clickhouse data.
- For deeper analyses, consider collaborative notebook platforms with DuckDB / Clickhouse-local over Parquet files

Telemetry and Analysis



Gossipsub traces exploration





Shoutout to Wonbin! 🙌

Closing the loop

Ongoing experiment: use telemetry data to build statistical models of mainnet inter-peer latencies, and generate a mainnet-like Shadow configuration (mainnet-in-a-box) that refreshes daily.

v0 statistical model will be quite crude and only scoped to the underlay.

As we improve on gossip trace collection, we can build higher-fidelity models.

(Discussion)
Reliability, Security, Privacy, Validator Anonymity

Seeding the discussion

Haven't had a lot of time to focus on these aspects, but we should.

Some ingredients:

- Ephemeral identities (PeerID)
- Oblivious routing (think OHTTP)
- Mixnets / Onion routing => latency?
- SSLE
- IPv6 tricks
- Metadata leakage
- ZK proofs of some kind to deal with beacon node / validator duality without revealing location
- Self-healing, redundancy, chaos testing
- ...

(Discussion)
First-principles network stack design

First principles thinking network stack design

Some thoughts to seed discussion.

- QUIC:
  - The modern transport choice; it's pervasive and well supported.
  - Native multiplexer, no reason to have muxer modularity.
  - Cannot leverage QUIC specificities via libp2p abstraction (e.g. RTT, congestion control stats, reliable resets, etc.)
  - Unreliable datagrams are now a thing in QUIC.
- Other transports:
  - Do we need WebTransport, WebRTC, WSS in the L1 networking stack?
- Legacy stack: TCP + Yamux + Noise (future: Yamux => QMux?)
- Drop modularity where none is needed => vertical integration and mechanical sympathy.
- MTU optimization.
- Large fanouts are now possible (e.g. pre-handshake and store session resumption keys).
- EL/CL networking planes unification? What can this look like?

Takeaways and conclusions

Thank you!