

Resilient Peer-to-Peer Messaging and Location Sharing Under Churn and Network Partitions

CSC 466: Overlay and Peer-to-Peer Networking, Project Proposal

Author: Sahil Sanghvi (V01039545) **Instructor:** Dr. Jianping Pan

Institution: Department of Computer Science, University of Victoria

1. Problem Statement and Motivation

Most messaging and location sharing services rely on central servers to relay messages, store them for offline users, and collect location data. This makes them simple to build, but a server outage or network failure stops the service, and the operator can see who talks to whom and where users are. Peer-to-peer designs avoid this dependence, but they face their own problems: peers come and go, links are unreliable, and groups of peers can be cut off from each other for a while.

The research question is: how well does a peer-to-peer messaging and location sharing system deliver messages and fresh location data under churn, packet loss, and network partitions, and what does it cost compared with a client-server design?

2. Related Literature and Research Context

Apple’s Find My network lets nearby devices relay encrypted reports about lost devices back to their owners. Heinrich et al. (2021) reverse engineered the protocol and analyzed its security and privacy. The system is crowd-sourced, but it still depends on Apple’s servers to store and serve the reports. The libp2p project provides peer discovery, NAT traversal, and encrypted transport, and its GossipSub protocol supports topic-based publish/subscribe messaging between peers (Vyzovitis et al., 2020).

Delay-tolerant networking addresses intermittent connectivity with in-network storage and store-and-forward delivery (Fall, 2003), and epidemic routing shows how messages can spread through partially connected mobile networks (Vahdat & Becker, 2000). Each of these covers one piece of the problem. This project combines them in a single application and measures the trade-offs between delivery, location freshness, and cost. It builds on an existing peer-to-peer stack rather than designing a new overlay, and focuses on application-level behavior under mobility and partitions.

3. Proposed System Architecture

The system is a web application built on js-libp2p (falling back to go-libp2p if browser support becomes a problem). It has three layers:

1. **Messaging:** Peers discover each other, exchange end-to-end encrypted messages, and use store-and-forward so messages for offline peers are held by other peers and delivered when the recipient returns.
2. **Location sharing:** Peers publish encrypted location updates to group topics using GossipSub. Identifiers rotate over time so relaying peers cannot easily link updates to a person. Sharing is opt-in and limited to chosen contacts.
3. **Offline finding (stretch goal):** Peers relay encrypted location reports for other peers, and the owner retrieves them later, in the style of Find My but without a central server.

A simple client-server version of the same application serves as a baseline for comparison.

4. Evaluation Methodology and Metrics

Many peers run in Docker containers, with Linux traffic control (tc/netem) injecting latency, jitter, and packet loss. Scripted peers join, leave, and move between network segments to emulate churn, mobility, and partitions. The testbed logs results to CSV files and plots them with Pandas and Matplotlib.

Message delivery: The delivery ratio is the fraction of sent messages that reach their recipient,

$$R = \frac{N_{\text{delivered}}}{N_{\text{sent}}}, \quad (1)$$

measured together with end-to-end latency as the number of peers, churn rate, and packet loss increase.

Location freshness: For each location update received at time t_r and sampled at time t_s , the staleness is

$$A = t_r - t_s, \quad (2)$$

which is compared against bandwidth cost as the update interval changes.

Partitions and baseline: The time to deliver queued messages after a partition heals is recorded, along with server load and failure behavior in the client-server baseline versus the peer-to-peer version.

5. Timeline

- **Mid Oct:** Review related work, set up the libp2p peer network, get basic encrypted chat working, and add logging.
- **Late Oct / Early Nov:** Add store-and-forward and location sharing, build the emulation setup, and prepare the midterm presentation.
- **Mid Nov:** Build the client-server baseline and run experiments across churn, loss, and partition scenarios.
- **Late Nov / Early Dec:** Analyze results, add the offline finding feature if time allows, prepare the demonstration, present, and submit the report and code.

6. Required Tools and References

Tools: js-libp2p (or go-libp2p), Docker, Linux Traffic Control (tc/netem), Leaflet for the map view, Matplotlib and Pandas for data plotting.

References:

- Fall, K. (2003). A Delay-Tolerant Network Architecture for Challenged Internets. *ACM SIGCOMM*, 27–34.
- Heinrich, A., Stute, M., Kornhuber, T., & Hollick, M. (2021). Who Can Find My Devices? Security and Privacy of Apple’s Crowd-Sourced Bluetooth Location Tracking System. *Proceedings on Privacy Enhancing Technologies*, 2021(3), 227–245.
- Vahdat, A., & Becker, D. (2000). Epidemic Routing for Partially Connected Ad Hoc Networks. *Technical Report CS-2000-06, Duke University*.
- Vyzovitis, D., Napora, Y., McCormick, D., Dias, D., & Psaras, Y. (2020). Gossip-Sub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networks. *arXiv:2007.02754*.