CSCI 565

Improving Gossip Protocol Performance in Unreliable Networks

Zinedine Boumghar, Shay Smith, Zakaria Khitirishvili

01

Background

What is Gossip Protocol and how is it used?

Introduction to Basic Gossip Algorithm

What is Gossip?

- A decentralized method for spreading information across nodes in a network.
- Inspired by how rumors spread in social networks.

Main idea:

Each node in a network communicates information (a "rumor") to a subset of its neighbors. Over multiple rounds, the information propagates through the network until all nodes receive it.

Assumptions:

- **Local Information Only:** Each node knows only its neighbors and cannot see the entire network.
- **Random Communication:** Nodes randomly choose which neighbors to communicate with.
- **Reliability:** Messages may be lost during transmission, but the protocol remains resilient.
- **Connected Network:** There is a path between any two nodes in the network.

Steps to Basic Gossip Algorithm

Initialization:

- A single node (or a small group) starts as "infected" with the information.
- All other nodes are "uninfected.“

Round Execution:

- Each infected node selects a subset of its neighbors randomly, determined by the fanout parameter. The infected node sends the rumor to the selected neighbors.

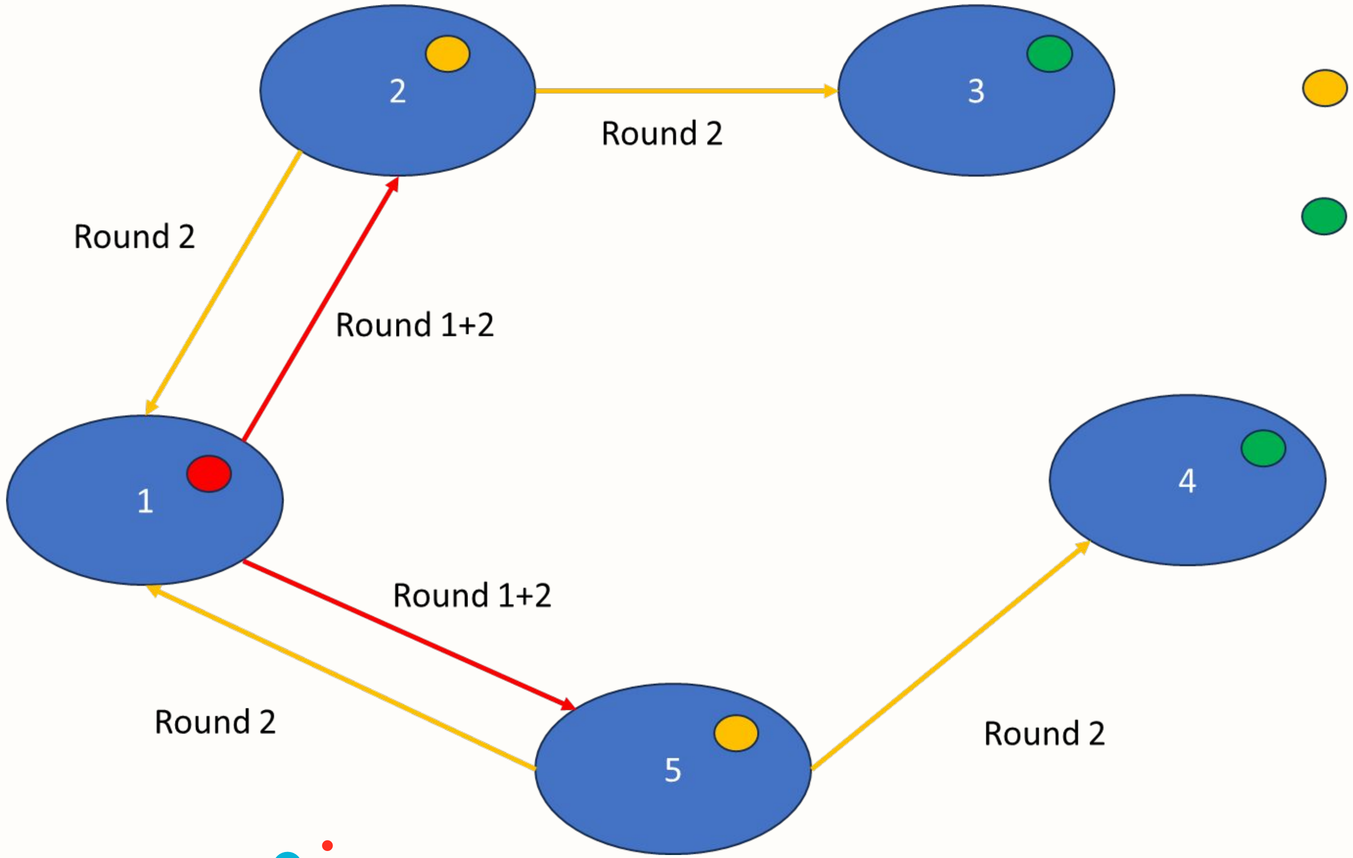
Infection Spread:

- If an uninfected node receives the rumor, it becomes infected.
- Newly infected nodes participate in subsequent rounds by spreading the rumor to their neighbors.

Termination:

- The process continues until all nodes are infected (i.e., have received the rumor), or a predetermined number of rounds is completed.

Example Run for 5 nodes



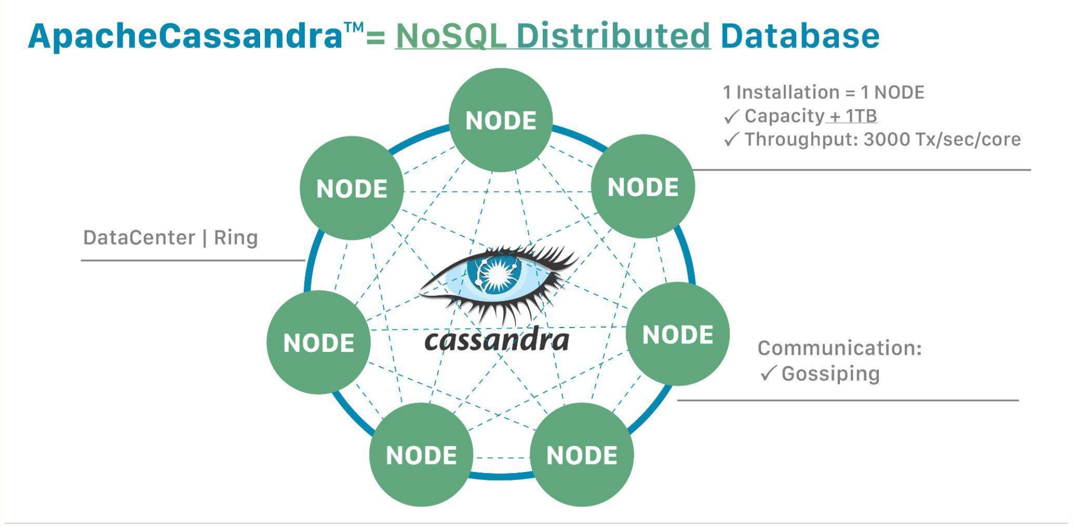
🔴 Has info Initially

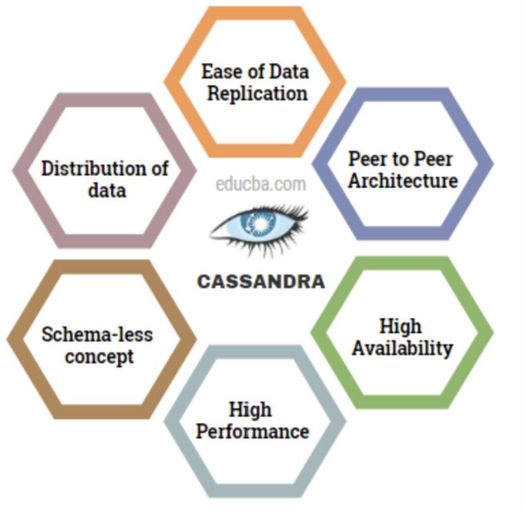
🟡 Has info after round 1

🟢 Has info after round 2

Fanout = 2

Application 1: Apache Cassandra

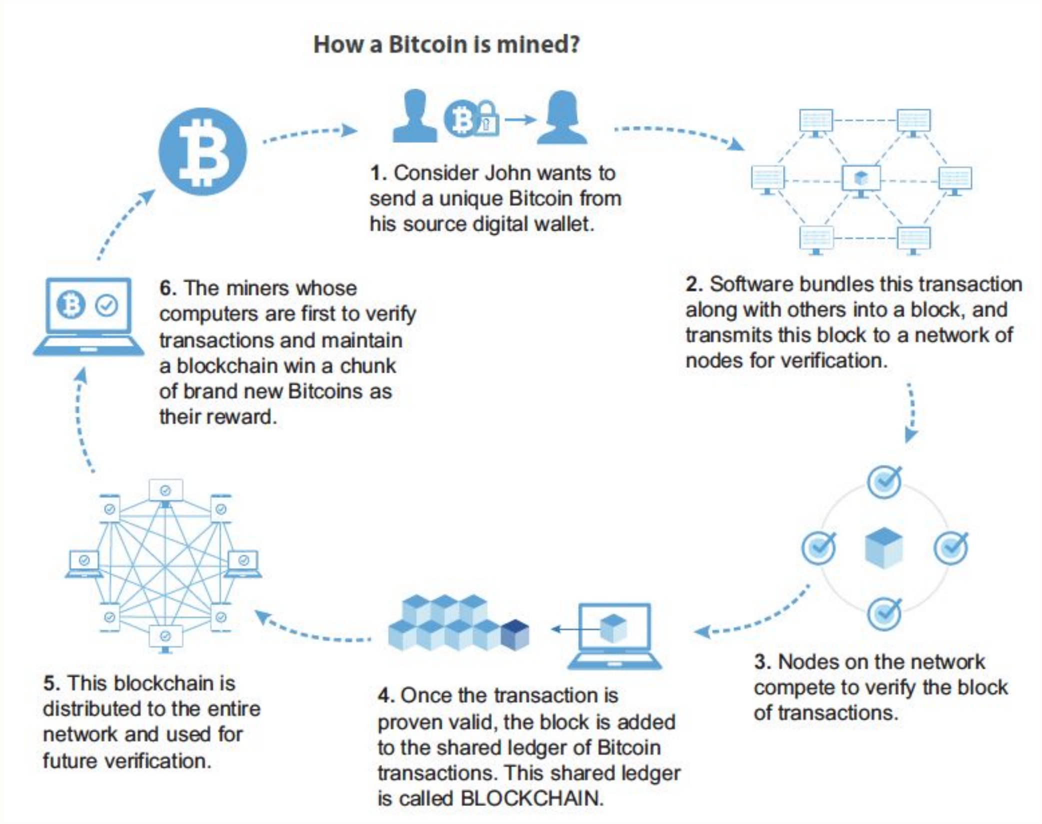




- Only state changes are broadcasted, and a maximum of three nodes are broadcasted to at a time (fanout = 3).
- "Heartbeats" are periodically sent between nodes to determine whether a node is UP/DOWN.
- Dynamic snitch feature: latencies of peers are collected and ranked, allowing a node to identify and avoid slower nodes.

Application 2: Blockchain Networks

- Due to blockchain networks being decentralized and peer-to-peer, gossip protocols naturally support them.
- Bitcoin and Ethereum use gossip protocols for propagating transactions and blocks.


How a Bitcoin is mined?
1. Consider John wants to send a unique Bitcoin from his source digital wallet.
2. Software bundles this transaction along with others into a block, and transmits this block to a network of nodes for verification.
3. Nodes on the network compete to verify the block of transactions.
4. Once the transaction is proven valid, the block is added to the shared ledger of Bitcoin transactions. This shared ledger is called BLOCKCHAIN.
5. This blockchain is distributed to the entire network and used for future verification.
6. The miners whose computers are first to verify transactions and maintain a blockchain win a chunk of brand new Bitcoins as their reward.

Problem Statement

Key Challenge:

- The traditional Gossip Protocol does not consider network reliability, leading to inefficiencies in unreliable environments.
- Assumes connected networks but fails to optimize for unreliable links.

Objective:

- Reduce message overhead: Minimize undelivered messages by prioritizing reliable communication links.
- Maintain efficiency: Avoid significant increases in the number of dissemination cycles.

Related Work in Improving the Protocol

- One paper proposes a protocol for real-time N-to-N multicast communication, which provides significantly less traffic load with higher performance improvements in lower latency networks. This is accomplished through redundancy suppression by use of a peer list that contains information about where a given peer received its information from, so messages are broadcasted back to already infected peers less.
  - https://jisajournal.springeropen.com/articles/10.1186/1869-0238-4-14

- One paper aimed to ensure all nodes in a network receive a desired message within a bounded latency at a very high probability. They argue that the most important parameter that controls the latency of message delivery is the fanout used during gossiping. The main contribution of this paper is that they introduced the idea of using variable fanouts in different rounds.
  - https://www.comp.nus.edu.sg/~ooiwt/papers/fanout-icdcs05-final.pdf

- One paper aims to improve the scalability of traditional heartbeating protocols for failure detection. It proposes a randomized probing protocol instead to monitor node aliveness.
  - https://www.cs.cornell.edu/projects/Quicksilver/public_pdfs/SWIM.pdf

02

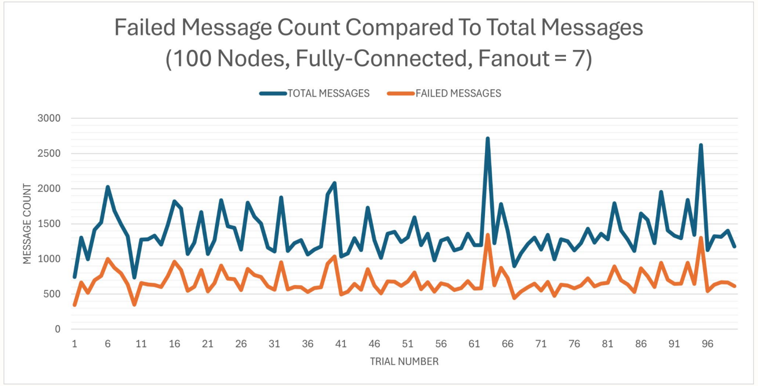
Research Outline

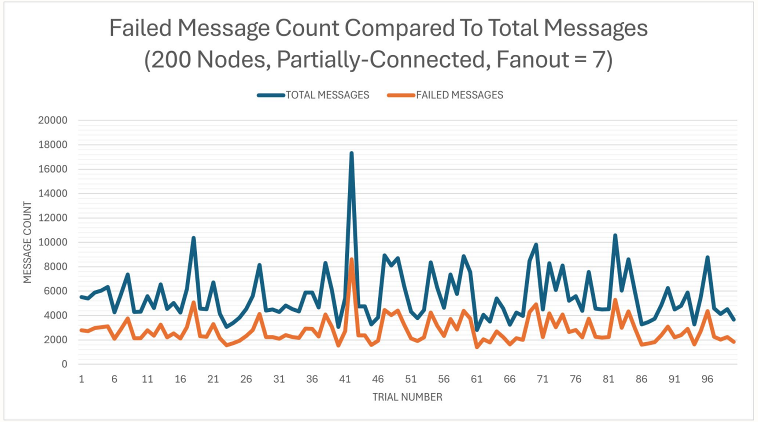
What specifically will we be doing?

Failed Messages

- **Defined as messages that couldn't be delivered due to an unreliable edge in the network**
- Less reliable edges = more failures
- Can we avoid using less reliable edges between nodes?
  - This is a probability problem

Intuition: Make unreliable paths the last resort option for infecting.





Proposal to Fix Failures

Initial Idea Considered

- **Make thresholds for minimum reliability needed to send a message**
  - If edge probability is too low, don't send ever
  - If edge probability is high, always send

- How do we determine these thresholds case-by-case?
- What if one node is "hard to reach"?

Idea in the Works

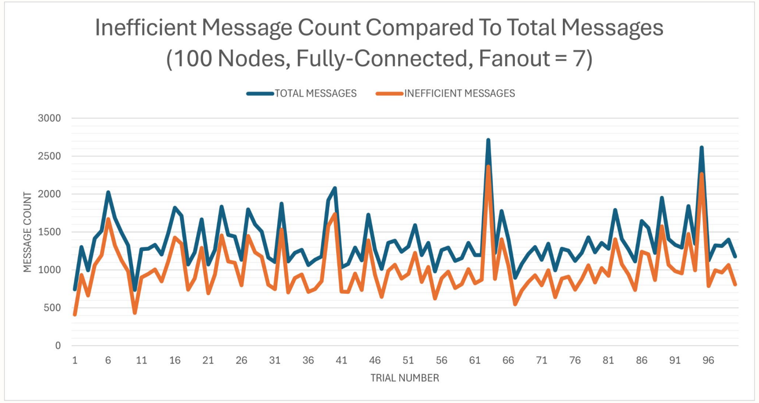
- **Make a "sliding window" which starts with high reliability links first**
- When node is first infected, it tries the best odds first
- As number of rounds increase, it tries worse odds
  - If already infected, then we don't need to take this chance

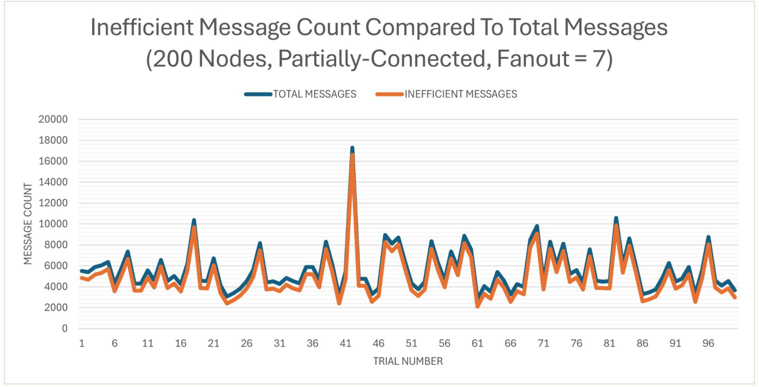
0.88	0.79	0.65	0.44	0.12	0.05
R1		*R2*		*R3*	

Inefficient Messages

- **Defined as messages sent to nodes infected in previous rounds**
- Efficiency gets worse as the number of rounds increases
- Can we prevent nodes from sending messages to infected nodes?
  - This is an information problem

Intuition: Have nodes update each other on their statuses via messages.





Proposal to Fix Inefficiency

Initial Idea Considered

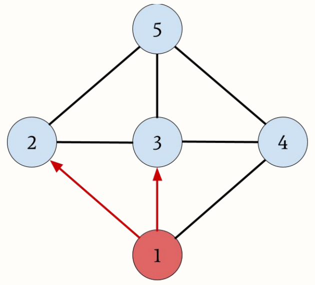
- **Have nodes send ACKs back to sender when infected**
    - Sender doesn't resend in later rounds

- This will double the total number of messages sent
- Doesn't account for an ACK not being delivered

Idea in the Works

- **Attach a list of known infections in each message**
- Each node that gets infected appends itself to the list
- Doesn't account for all situations, but still helps reduce trivial messages

What does node 2 know for sure about the network?



03

Conclusion

What other considerations are there in this research?

Limitations

Assumption of Known Probabilities:

- The protocol assumes that the reliability of communication links (probabilities) is known beforehand, which may not be realistic in all scenarios.

Overhead of Probability Estimation:

- Determining and updating link reliability metrics could introduce extra computational and communication overhead in dynamic environments.

Unexplored Network Scenarios:

- The current approach does not address networks with intermittent connectivity or scenarios where nodes may fail entirely.

Other Approaches Considered

- Algorithmically determining the optimal fanout value for a given network
  - Previous research has been done on this idea
  - Would be difficult to modify mid-execution on dynamic networks
- Improving infections via better routing in the network
  - It would be better to just store the entire network on each node
  - Doesn’t handle dynamic edges
- Restructure nodes or infection process to better handle unreliability
  - Run the risk of having inputs that can’t be restructured/optimized
  - Sacrifices the flexibility of Gossip Protocol

Thanks!

Do you have any questions?