caroline | ulemj | shibali

HONEYCOMB

a lightweight p2p communication platform for disaster situations

TABLE OF CONTENTS

01

Motivation

Disasters

Pictured: Nepal earthquake, 2015

- Urgent communication needs
- Low-bandwidth, unreliable and/or congested networks

Credit: Hilmi Hacaloğlu

Our System

System

- Short-form text messages
- p2p Chord-based network

Goals

- Available
  - Fault-tolerant, especially to churn
- Lightweight
- Scalable

ASSUMPTIONS

No bad actors

Everyone using Honeycomb as intended

Known IP

Users have at least one node's IP addr to join network with

Sufficient connectivity

Enough bandwidth for our lightweight system to run

Python + File

Users have python and the file downloaded prior to disaster

02

System Architecture

Node Structure

Each node is functionally homogeneous peer in the disaster network
Runs a **multi-threaded XML-RPC server**

Each Node Has:

- Unique Node ID (SHA-1 hash)
- Successor List & Predecessor
- Message list & message set
- Stabilization thread that runs every 1 second
- Finger table

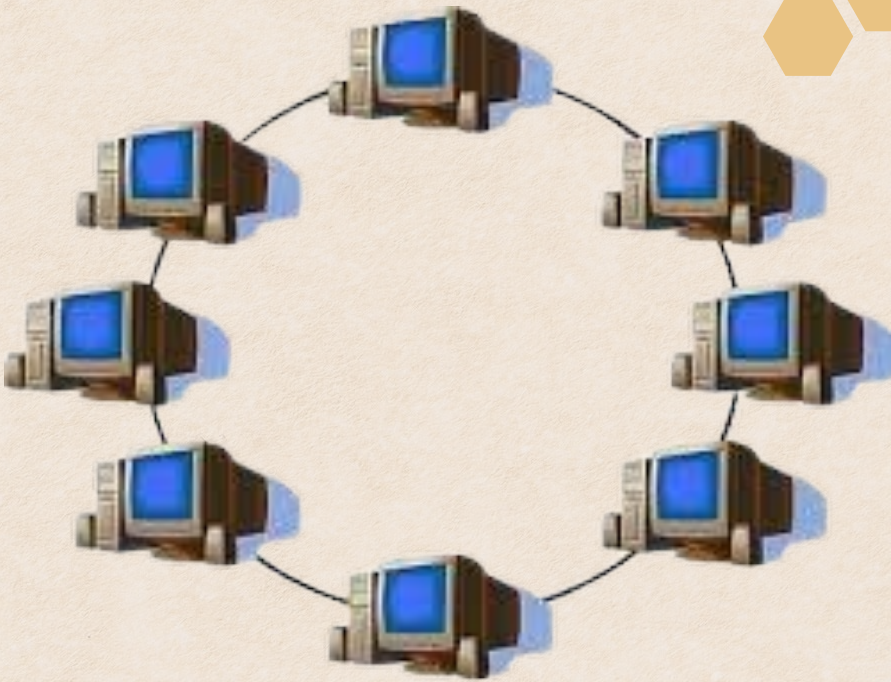
Ring Topology

Nodes form a **logical ring** Each node knows only its:

- **Successors (list of next 4 nodes in ring)**
- **Predecessor**
- **Finger Table**

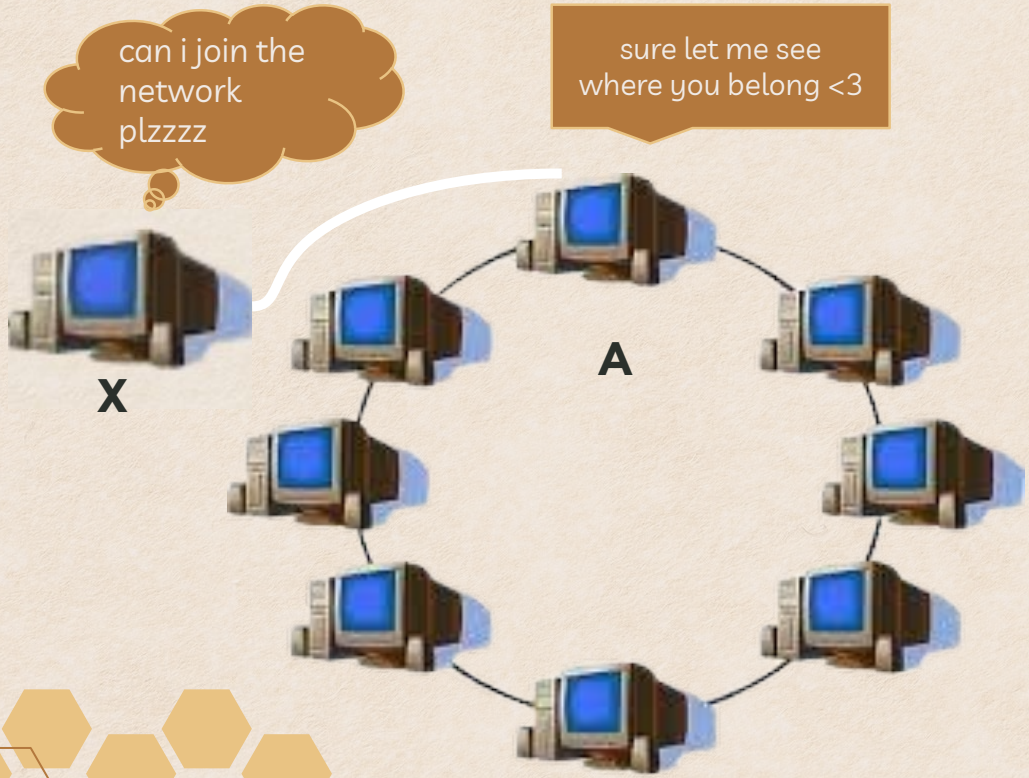
New nodes join by finding their correct successor
Stabilization runs every second
Notify updates predecessor links

Network self repairs on joins & failures



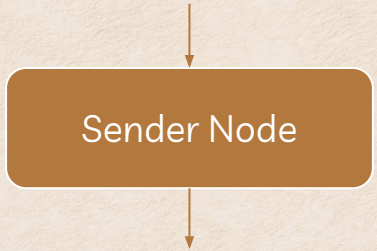
How do nodes join?

*Imagine the ring already has some nodes. A new node **X** wants to join via an existing node **A**.*



Sending Message

User types: “Bridge Collapsed”



Generate MessageID:
hash(message + UTC time + node ID)

Insert Locally

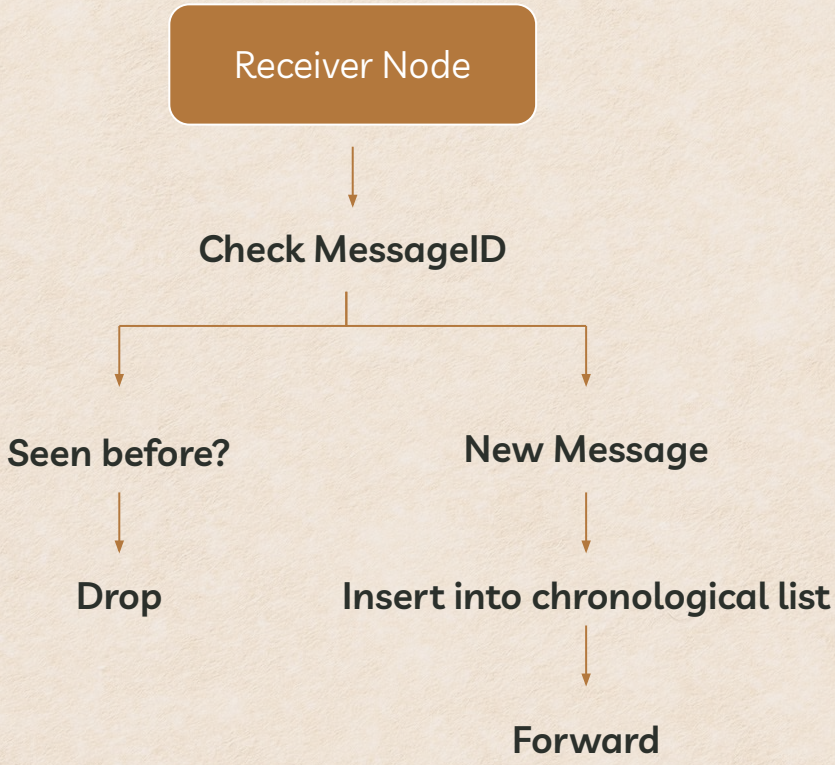
Forward via XML-RPC

Finger Table Node
(p = 0.5)

Successor Nodes
(decreasing logarithmic p)

Successor	Probability of being forwarded the message
1st	100%
2nd	50%
3rd	25%
4th	12.5%

Receiving Message



Concurrency

Multi-threaded Server

- 3 background threads:
  1. Listening for XML RPC Calls
  2. UI Updates
  3. Periodic ring maintenance
- Listening thread creates new thread for each XML-RPC request

SimpleXMLRPCServer + ThreadingMixIn

- TCP
  - Tradeoff: additional overhead for greatly reduced complexity

Locks protect shared data:

Successor

Predecessor

Messages

Fault Tolerance

4 successors

Finger table

Periodic updates

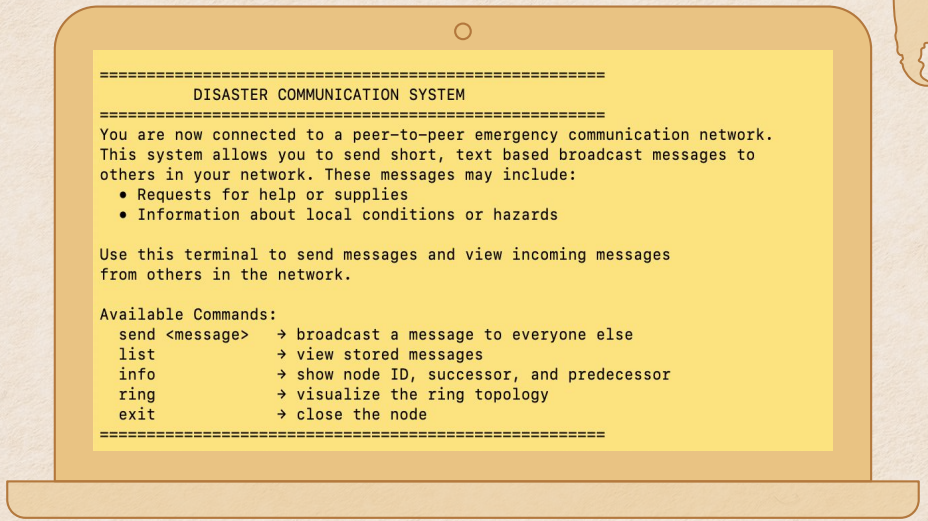
03

Current Progress + Demo

Current Progress

✔	Fully functioning version with concurrency but assumes no node failures.
✔	CLI as a user interface
✔	Scripts to run all 20 nodes
⏸	Introduce successor lists (we currently store only the immediate successor) and finger tables for improved fault tolerance.
⏸	Add failure detection via timeouts and implement disk-based flushing of old messages.
⏸	Add probabilistic message forwarding to reduce traffic
⏸	Conduct experiments

!! HONEYCOMB DEMO !!



```
======================================================
          DISASTER COMMUNICATION SYSTEM
======================================================
You are now connected to a peer-to-peer emergency communication network.
This system allows you to send short, text based broadcast messages to
others in your network. These messages may include:
  • Requests for help or supplies
  • Information about local conditions or hazards

Use this terminal to send messages and view incoming messages
from others in the network.

Available Commands:
  send <message>   → broadcast a message to everyone else
  list             → view stored messages
  info             → show node ID, successor, and predecessor
  ring             → visualize the ring topology
  exit             → close the node
======================================================
```

04

Experiments

01

Failure Recovery Time vs. Node Loss

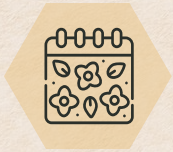
02

Message Delivery Success Under Failure and Load

03

Per-Node Load Scaling with Network Growth

Honeycomb is awesome because...



Available

System runs under node failure

Decentralized

No single point of failure

Lightweight

Minimal overhead

Questions?