CS 425 / ECE 428
Distributed Systems
Fall 2025

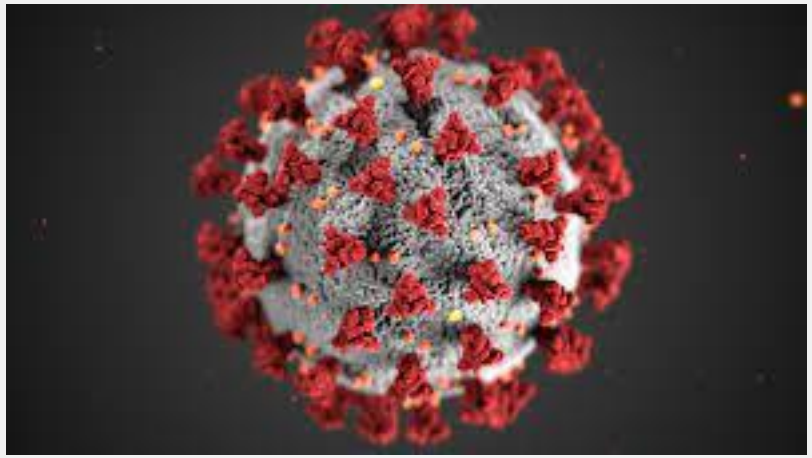
Indranil Gupta (Indy)

w/ Aishwarya Ganesan, Ram Kesavan

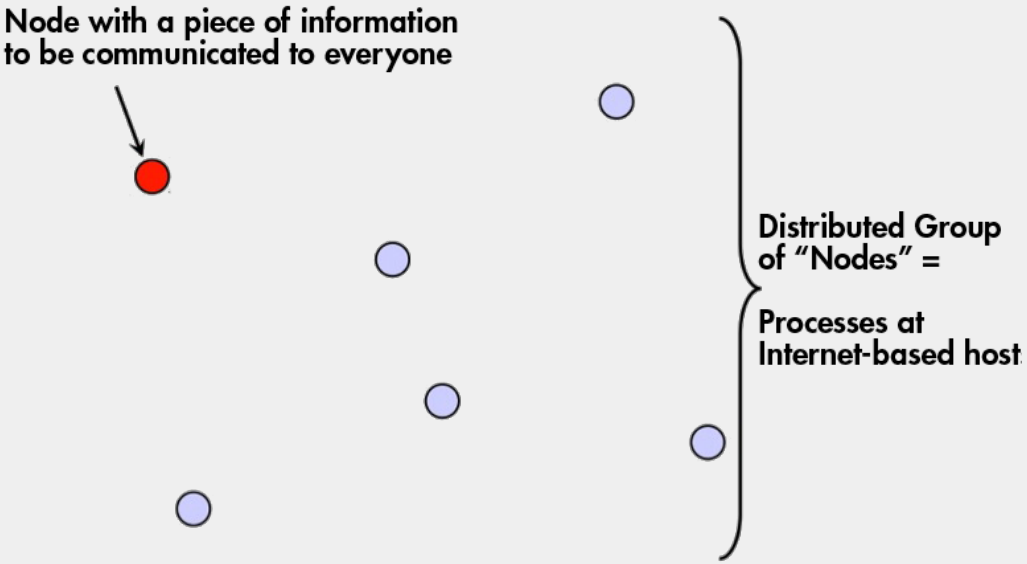
Lecture 5: Gossiping

Today's Agenda

- Epidemics, or how to use them to your advantage (to do good things)



Multicast



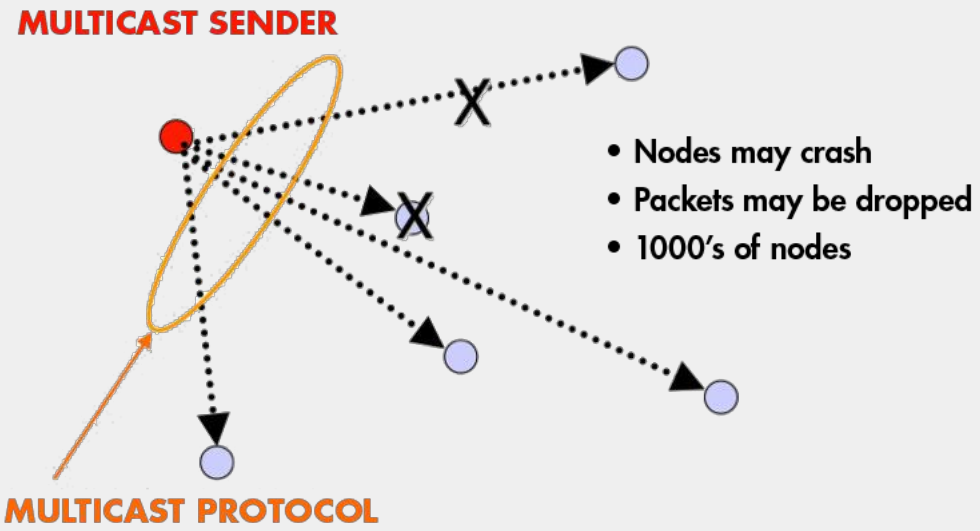
multicast vs broadcast

All nodes in a network

a certain set/group of nodes

multicast in this class

Fault-tolerance and Scalability



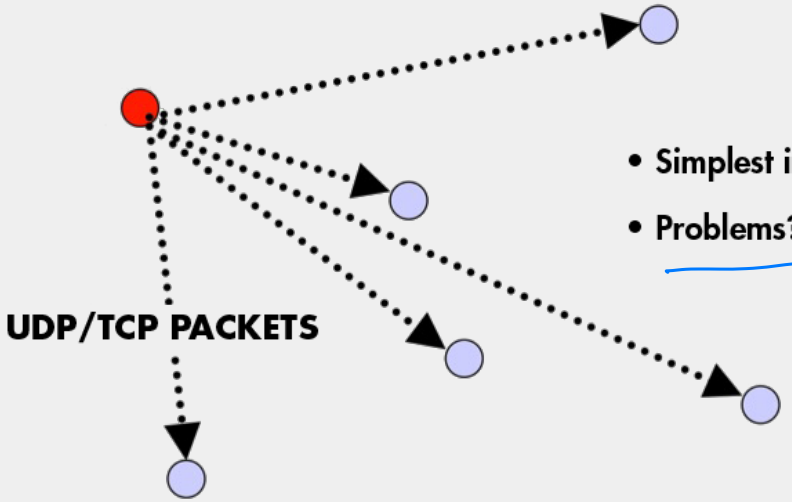
- Nodes may crash
- Packets may be dropped
- 1000's of nodes

Needs:

1. Reliability (Atomicity)
   - 100% receipt
2. Speed

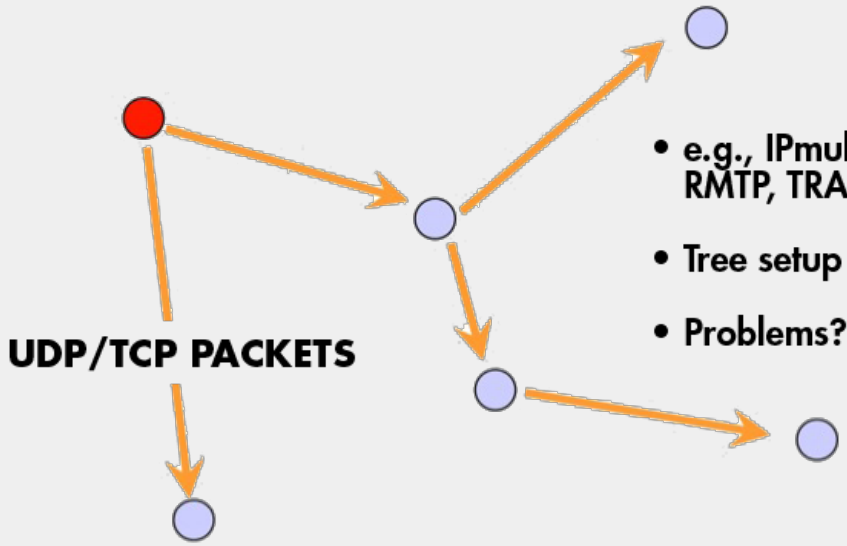
for all non-faulty nodes received fairly quickly

Centralized



- Simplest implementation
- Problems? → Single point of failure
  high latency
  sequential sends to all nodes

Tree-Based



- e.g., IPmulticast, SRM RMTP, TRAM, TMTP
- Tree setup and maintenance
- Problems?

A spanning tree

For balanced tree

O(log N) latency

because O(log N) depth

failure handling

need to maintain tree structure

children need to find new parents

Tree-based Multicast Protocols

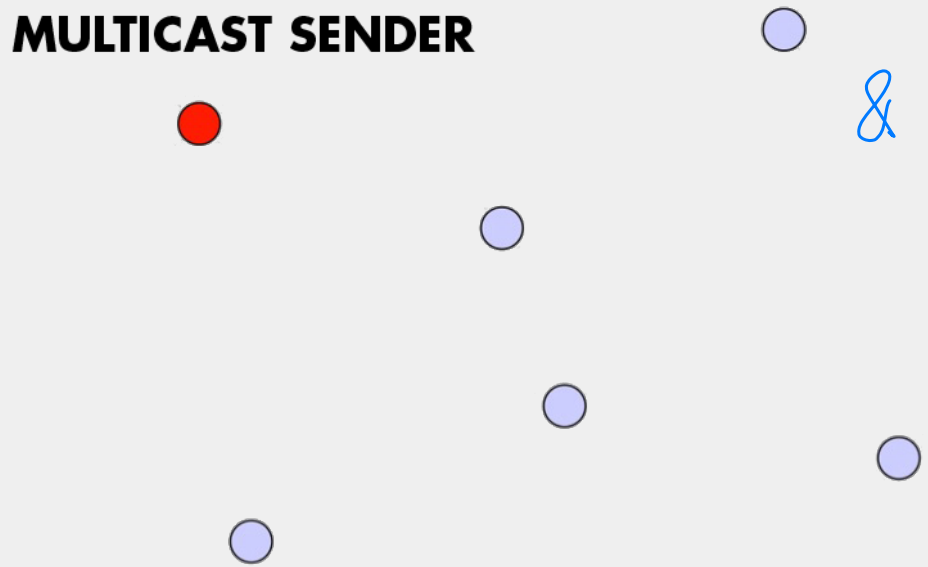
- Build a spanning tree among the processes of the multicast group
- Use spanning tree to disseminate multicasts
- Use either acknowledgments (ACKs) or negative acknowledgements (NAKs) to repair multicasts not received
- SRM (Scalable Reliable Multicast)
  - Uses NAKs
  - But adds random delays, and uses exponential backoff to avoid NAK storms
  - (Do you know why SRM is called a "talented" protocol?)
- RMTP (Reliable Multicast Transport Protocol)
  - Uses ACKs
  - But ACKs only sent to designated receivers, which then re-transmit missing multicasts
- These protocols still cause an O(N) ACK/NAK overhead [Birman99]

Notify peers for not receiving a multicast (e.g. by counting holes in the message seq #

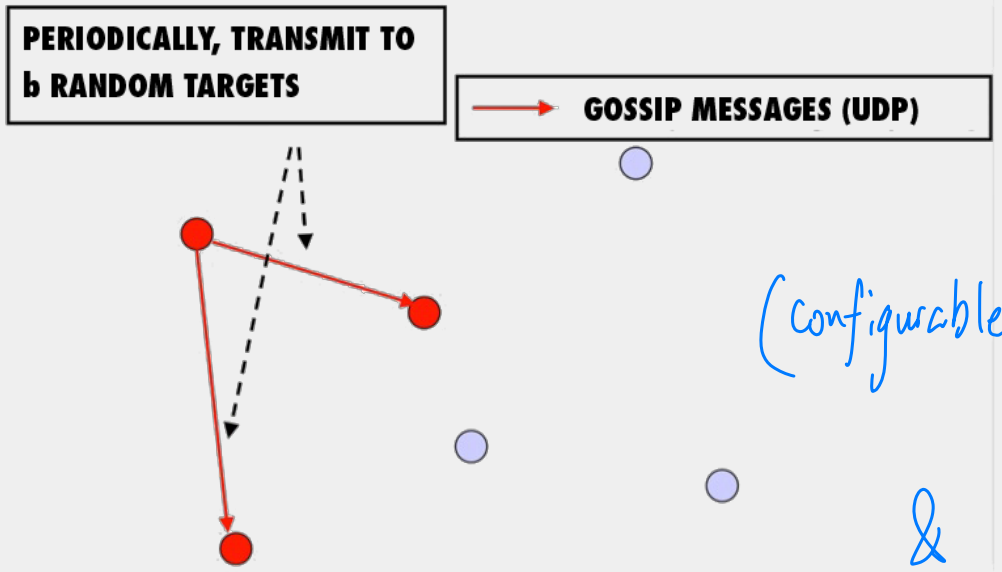
So not to flood the network

A Third Approach

So how to do multicast without feedbacks (ACK/NAK)? & also do almost 100% reachability



A Third Approach



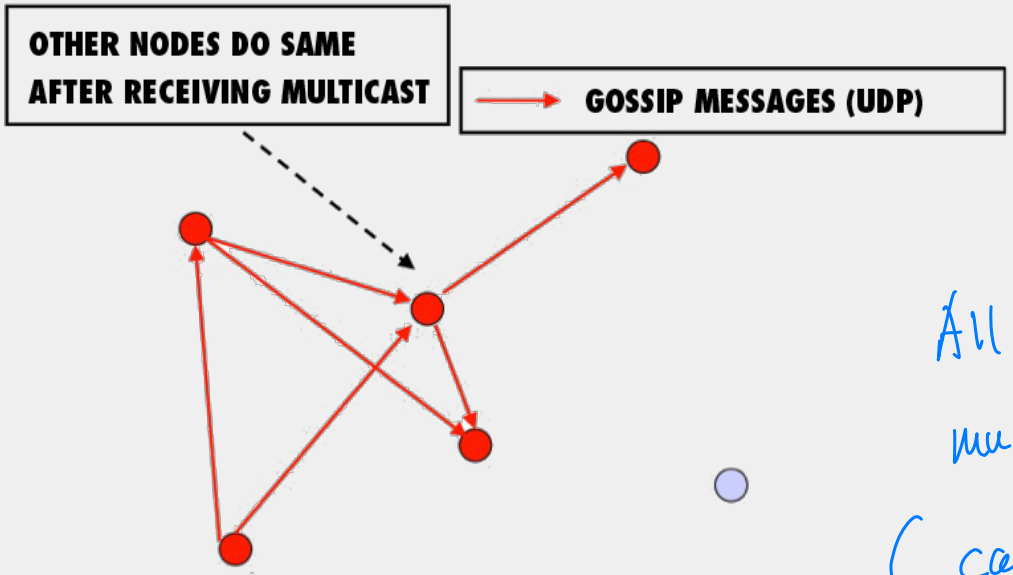
A node starts a multicast

It picks randomly pick

(configurable) b targets uniformly random

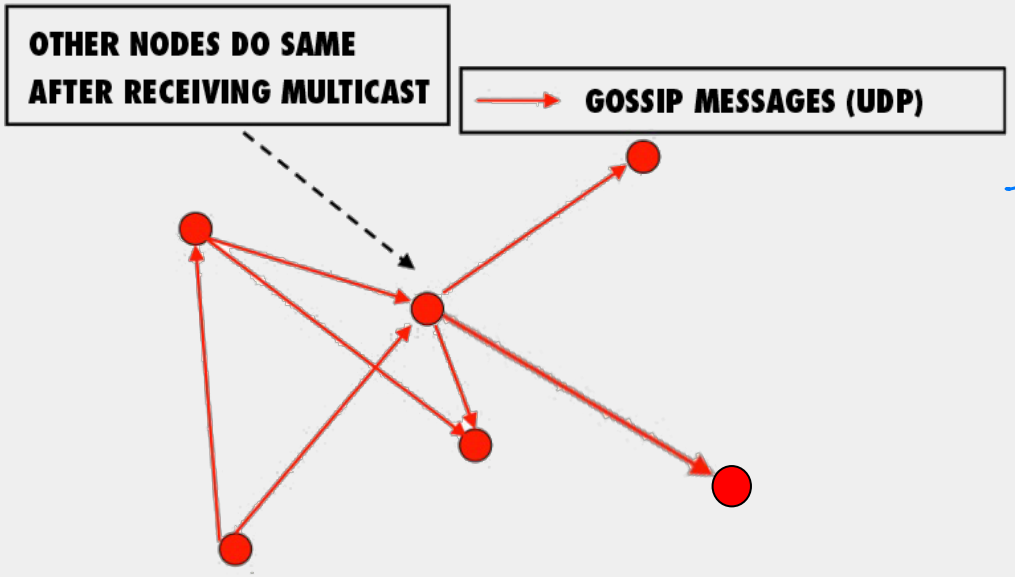
& send the multicast

A Third Approach



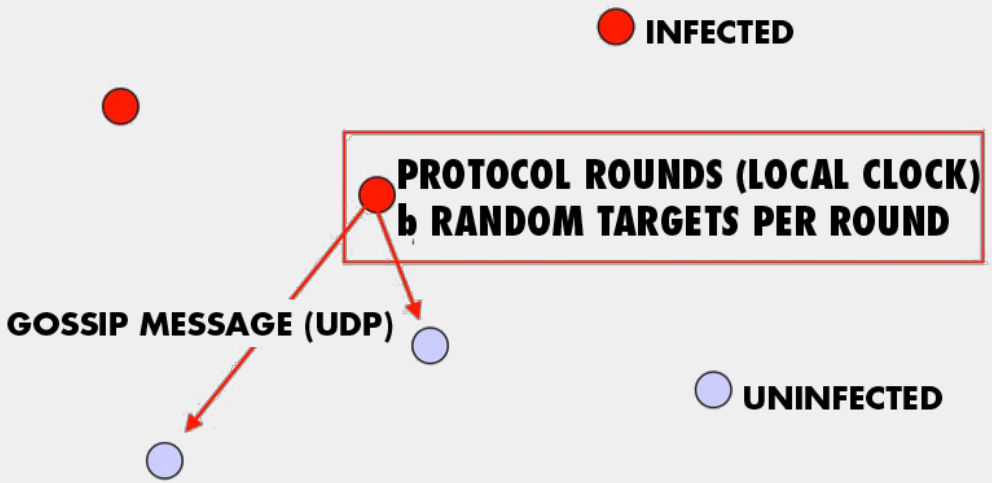
Periodically.
All nodes received the multicast do the same
(cast to b targets)
no need to pick new nodes

A Third Approach



Eventually, the message spreads to entire group

“Epidemic” Multicast (or “Gossip”)



This is similar to spread of viruses or rumors hence the name

Two type of nodes Infected & uninfected

Push vs. Pull

- So that was "Push" gossip
  - Once you have a multicast message, you start gossiping about it
  - Multiple messages? Gossip a random subset of them, or recently-received ones, or higher priority ones
- There's also "Pull" gossip
  - Periodically poll a few randomly selected processes for new multicast messages that you haven't received
  - Get those messages
- Hybrid variant: Push-Pull
  - As the name suggests

Properties

Claim that the simple Push protocol

- Is lightweight in large groups
- Spreads a multicast quickly
- Is highly fault-tolerant

You may recieve & send multiple copy of the same message but this is low

Also, the comm. can be UDP, package loss is fine

Even with node failures & packet drops

It can still spread

Analysis

From old mathematical branch of *Epidemiology* [Bailey 75]

- Population of $(n+1)$ individuals mixing homogeneously
- Contact rate between any individual pair is β
- At any time, each individual is either uninfected (numbering x) or infected (numbering y)
- Then, $x_0 = n, y_0 = 1$
  and at all times $x + y = n + 1$
- Infected–uninfected contact turns latter infected, and it stays infected

Any pair a,b has same probability of contact to any other pair

Eventually $x_t \to 0$, $y_t \to n+1$

Analysis (contd.)

- Continuous time process
- Then

$$\frac{dx}{dt} = -\beta xy \quad \text{(why?)}$$

with solution:

$$x = \frac{n(n+1)}{n + e^{\beta(n+1)t}}, y = \frac{(n+1)}{1 + ne^{-\beta(n+1)t}}$$

(can you derive it?)

We can represent this process with diff. equations

why − ? A: x is going down

Why ? β is contact rate, $x \cdot y$ is total # of pairs

Now with $x+y = n+1$

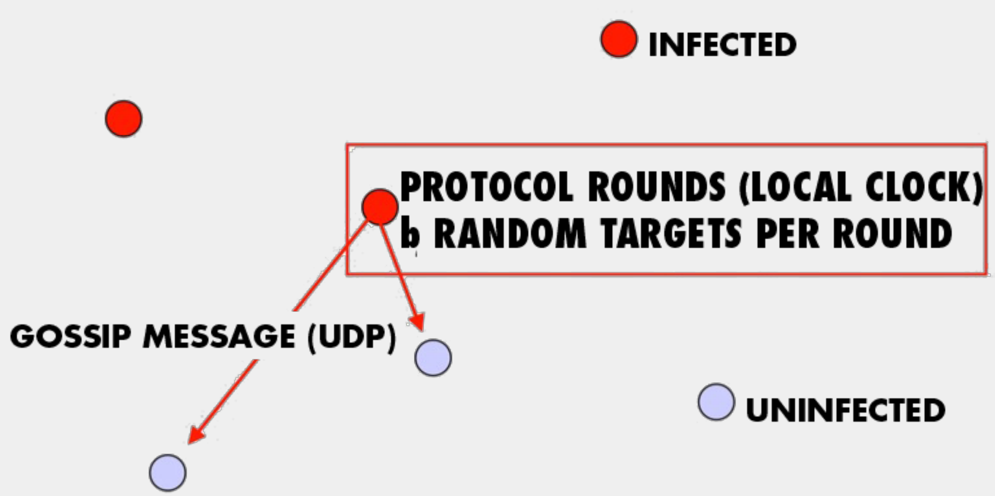
Solve ⟹

Sanity check: $t=0$, $e^0 = 1 \Rightarrow x_0 = n$, $y_0 = 1$

$t \to \infty$ $e^{+\infty} \to \infty \Rightarrow x_\infty \to 0$

$e^{-\infty} \to 0 \Rightarrow y_\infty \to n+1$

Epidemic Multicast



How to derive β?

Remem

each round each infected contact b nodes...

Epidemic Multicast Analysis

$$\beta = \frac{b}{n}$$ (why?)

Each infected pick b target uniformly random each round

Substituting, at time $t=c\log(n)$, the number of infected is

c constant

$$y \approx (n+1) - \frac{1}{n^{cb-2}} \rightarrow$$

if you let gossip run for c log n round, this is the # of infected node

↳ remaining uninfected node

(correct? can you derive it?)

By configuring c & b, can make close to 0. Also, note this scales with

Analysis (contd.)

O, meaning when n is large, the effect of gossip is even better

- Set c,b to be small numbers independent of n
- Within $clog(n)$ rounds, [**low latency**]

② theoretically, log n is not constant. But practically, it is usually very small

  - all but $\frac{1}{n^{cb-2}}$ number of nodes receive the multicast [**reliability**]

  - each node has transmitted no more than $cblog(n)$ gossip messages [**lightweight**]

① Why?
c log(n) round, b messages per round

Why is log(N) low?

- log(N) is not constant in theory
- But pragmatically, it is a very slowly growing number
- Base 2
  - log(1000) ~ 10
  - log(1M) ~ 20
  - log (1B) ~ 30
  - log(all IPv4 addresses) = 32
  - log(all IPv6 addresses) = 128

Fault-tolerance

- Packet loss
  - 50% packet loss: analyze with *b* replaced with *b/2*
  - To achieve same reliability as 0% packet loss, takes twice as many rounds
- Node failure
  - 50% of nodes fail: analyze with *n* replaced with *n/2* and *b* replaced with *b/2*
  - Same as above

Why? Note the term $\frac{1}{n^{ch-2}}$ $b \to b/2$ $f = c \log n \to 2c \log(n)$

Imagine virus, some contact does not infect, but does not prevent virus spreading

half node down, so comm. fails

Fault-tolerance

- With failures, is it possible that the epidemic might die out quickly?
- Possible, but improbable:
  - Once a few nodes are infected, with high probability, the epidemic will not die out
  - So the analysis we saw in the previous slides is actually behavior *with high probability*

  [Galey and Dani 98]
- Think: why do rumors spread so fast? why do infectious diseases cascade quickly into epidemics? why does a virus or worm spread rapidly?

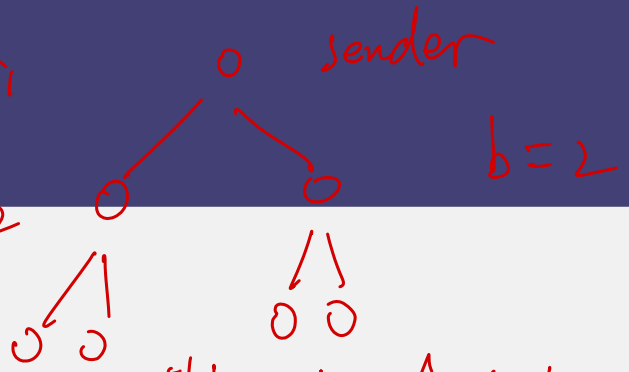
Possible : All infected nodes fails at very beginning. but once a few nodes infected & survives. hard to stop

Pull Gossip: Analysis

- In all forms of gossip, it takes O(log(N)) rounds before about N/2 processes get the gossip
  - Why? Because that's the fastest you can spread a message – a spanning tree with fanout (degree) of constant degree has O(log(N)) total nodes (height of tree)
- Thereafter, pull gossip is faster than push gossip
- After the ith round let p_i be the fraction of non-infected processes. Let each round have k pulls. Then

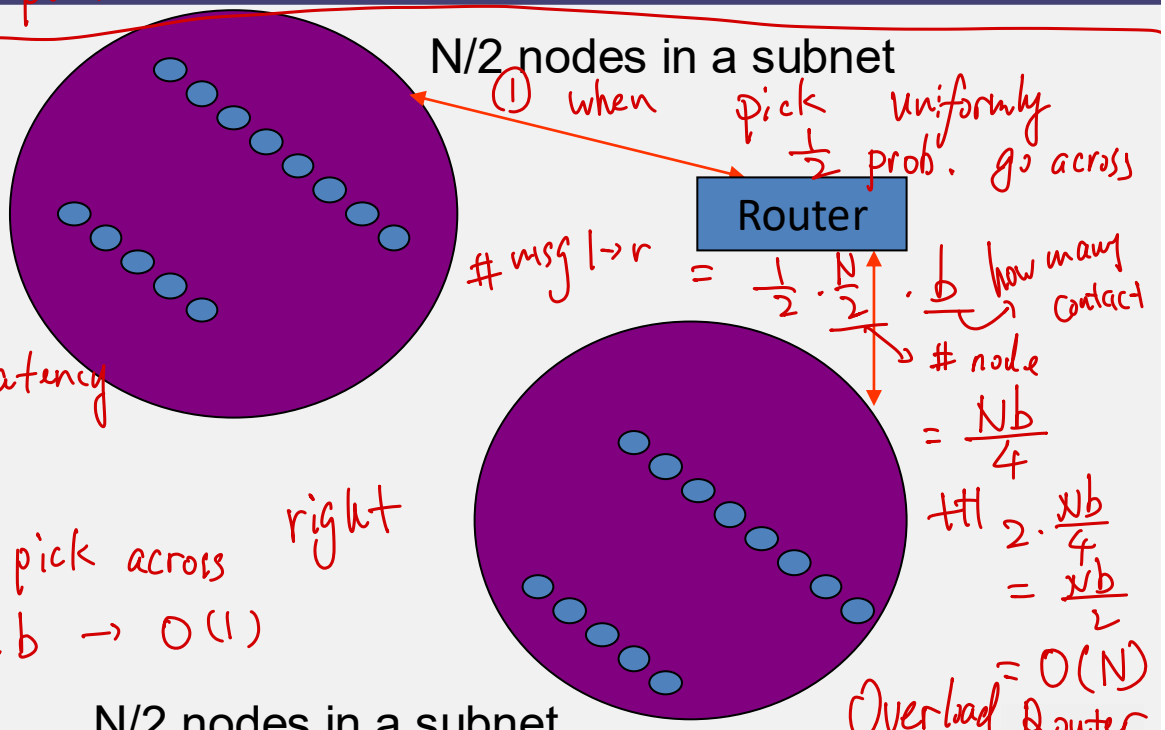
$$p_{i+1} = (p_i)^{k+1}$$

- This is super-exponential
- Second half of pull gossip finishes in time O(log(log(N))



Topology-Aware Gossip

- Network topology is hierarchical
- Random gossip target selection => core routers face O(N) load (Why?)
- Fix: In subnet *i*, which contains n_i nodes, pick gossip target in your subnet with probability $(1-1/n_i)$
- Router load=O(1)
- Dissemination time=O(log(N))



Answer – Push Analysis (contd.)

⇒ $T_{left} \sim O(\log(\frac{N}{2}))$ & across $O(1)$
& $T_{right} \quad O(\log\frac{N}{2})$
⇒ $T(N) = O(\log\frac{N}{2}) + O(1) + O(\log\frac{N}{2})$
$= O(\log N)$

Using: $\beta = \frac{b}{n}$

Substituting, at time $t = c\log(n)$

$$y = \frac{n+1}{1+ne^{-\frac{b}{n}(n+1)c\log(n)}} \approx \frac{n+1}{1+\frac{1}{n^{cb-1}}}$$

$$\approx (n+1)(1-\frac{1}{n^{cb-1}})$$

$$\approx (n+1)-\frac{1}{n^{cb-2}}$$

SO,...

- Is this all theory and a bunch of equations?
- Or are there implementations yet?

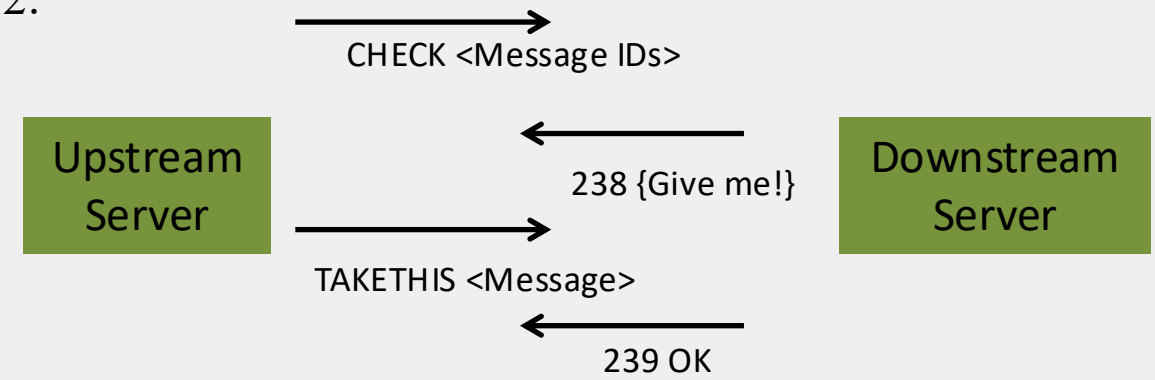
Some implementations

- Clearinghouse and Bayou projects: email and database transactions [PODC ‘87]
- refDBMS system [Usenix ‘94]
- Bimodal Multicast [ACM TOCS ‘99]
- Sensor networks [Li Li et al, Infocom ‘02, and PBBF, ICDCS ‘05]
- AWS EC2 and S3 Cloud (rumored). [‘00s]
- Cassandra key-value store (and others) use gossip for maintaining membership lists
- Usenet NNTP (Network News Transport Protocol) [‘79]

NNTP Inter-server Protocol

1. Each client uploads and downloads news posts from a news server

2.



Server retains news posts for a while,
transmits them lazily, deletes them after a while.

Summary

- Multicast is an important problem
- Tree-based multicast protocols
- When concerned about scale and fault-tolerance, gossip is an attractive solution
- Also known as epidemics
- Fast, reliable, fault-tolerant, scalable, topology-aware