CS514: Intermediate Course in Computer Systems

Lecture 18: October 27, 2003

Reliable Multicast (part 2)

Quick recap of last lecture

CS514

- Reliable multicast is hard
  - Even more so if IP multicast is used
- Two main problems:
  - Implosion of feedback
  - Simultaneity of receivers ("weakest link" phenomenon)
- We looked at SRM and PGM
- Today we'll look at bimodal multicast, digital fountains, and overlay multicast

Bimodal multicast basic idea

CS514

- Also called pbcast, for probabilistic broadcast
- SRM used localized multicasts from receivers to request *and* send retransmissions
- BGM aggregates retransmission requests uptree
  - Source or routers send retransmissions along reverse path of retransmission requests

Bimodal multicast basic idea

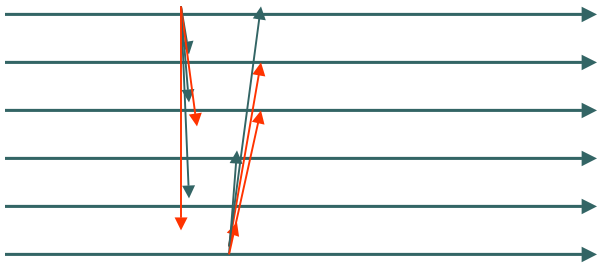
CS514

- Pbcast similar to SRM in that receivers request retransmissions from each other
- But differs from SRM in that pbcast uses *gossip*, not multicast
- Indeed, pbcast may be seen as essentially a gossip protocol, but “turbocharged” with multicast
  - Rather than multicast made reliable with gossip!

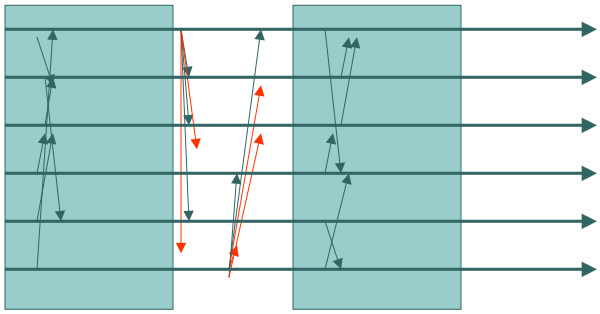
Bimodal multicast basic idea

CS514

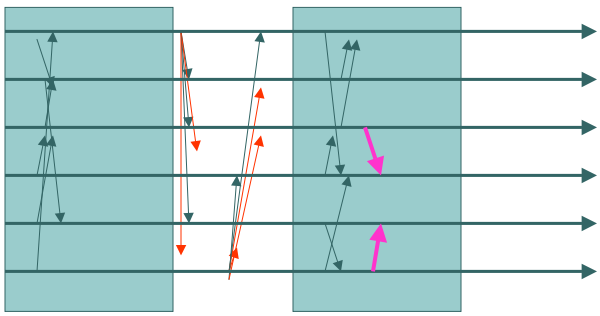
- Pbcast has two “phases”
  - Though both work continuously, “independently” of the other
- One phase is “multicast”
  - Could be IP multicast or overlay multicast
  - Gets a message to most processes
- Other phase is “gossip”
  - Repairs messages missed by the multicast



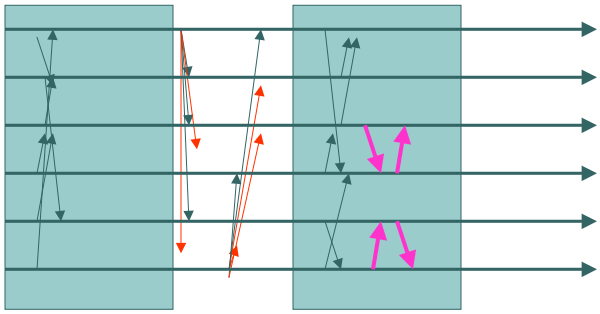
Start by using *unreliable* multicast to rapidly distribute the message. But some messages may not get through, and some processes may be faulty. So initial state involves partial distribution of multicast(s)



Periodically (e.g. every 100ms) each process sends a *digest* describing its state to some randomly selected group member. The digest identifies messages. It doesn't include them.



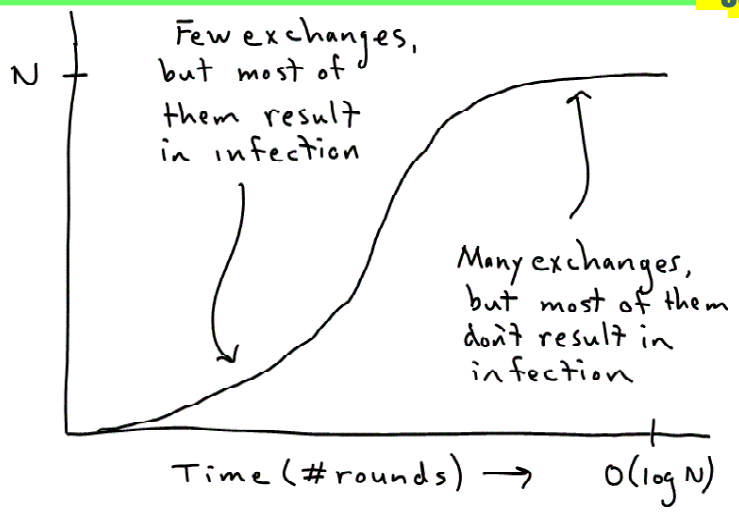
Recipient checks the gossip digest against its own history and *solicits* a copy of any missing message from the process that sent the gossip



Processes respond to solicitations received during a round of gossip by retransmitting the requested message. The round lasts much longer than a typical RPC time.

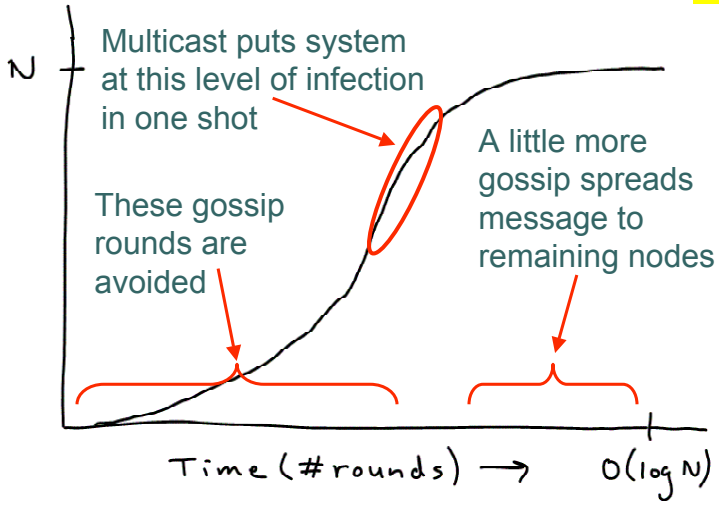
Recall gossip infection rate

CS514



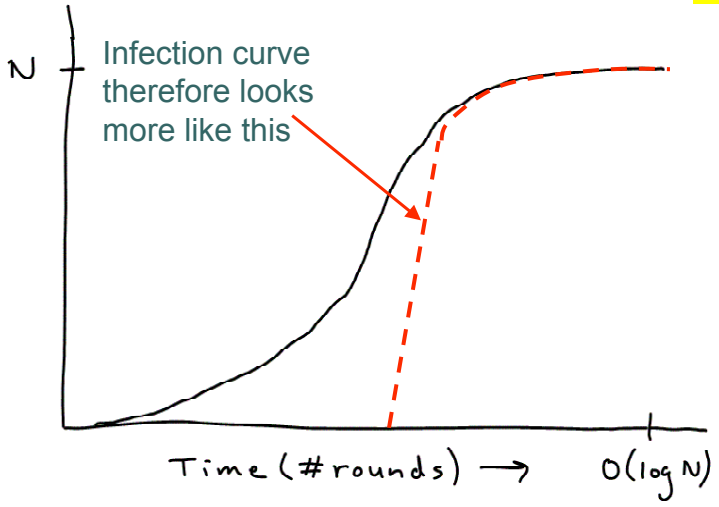
Initial multicast "bypasses" the early rounds

CS514



Results in a faster infection

CS514



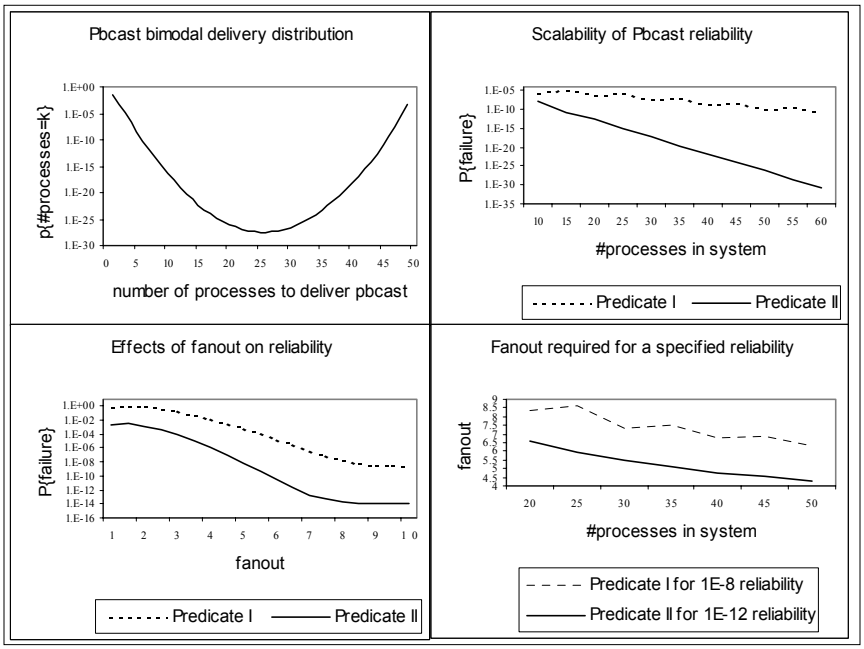


Figure 5: Graphs of analytical results

Pbcast versus SRM

CS514

- Pbcast has higher “good times” overhead
  - SRM has no overhead when there is no packet loss
- Pbcast has lower “bad times” overhead
  - SRM can send a lot of traffic during period of loss
  - This traffic can exacerbate problem

Pbcast versus SRM

CS514

- SRM reacts to loss by doing more work
  - Eventually melts down
- Pbcast reacts to loss by taking more time to deliver packets
  - Furthermore, from gossip messages, sender can see when receivers are getting behind, and can slow down
  - More graceful degradation

Digital Fountains

CS514

- Even though pbcast degrades more gracefully than SRM, in the end sender and receivers are still tightly coupled
  - Receivers receive at the same time (or shortly after) that sender sends
- Digital fountains break the simultaneity completely

Digital fountain basic idea

CS514

- Sender continuously sends content (say, a file)
  - Over a multicast “channel”
- Receivers join the multicast group when they want, receive until they have all content, and then quit
- Sender quits eventually
  - When there are no listeners for an extended period
  - Or when all receivers indicate they got it

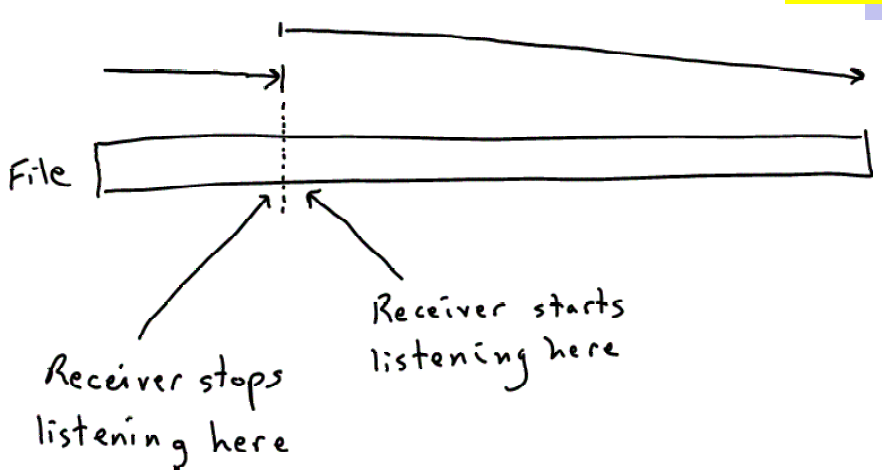
Naive digital fountain:

CS514

- Sender simply sends the file over and over
- Problem:
  - With even a little loss, receivers have to listen for a long time
  - For instance, 100 packet file, 1% loss
  - Most receivers will loose one packet, have to keep listening until that packet comes around again!

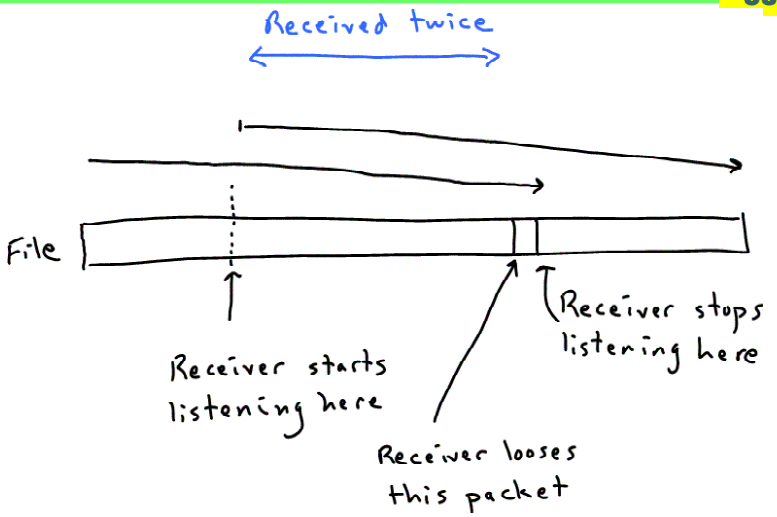
Naïve case with no loss

CS514



Naïve case with loss

CS514



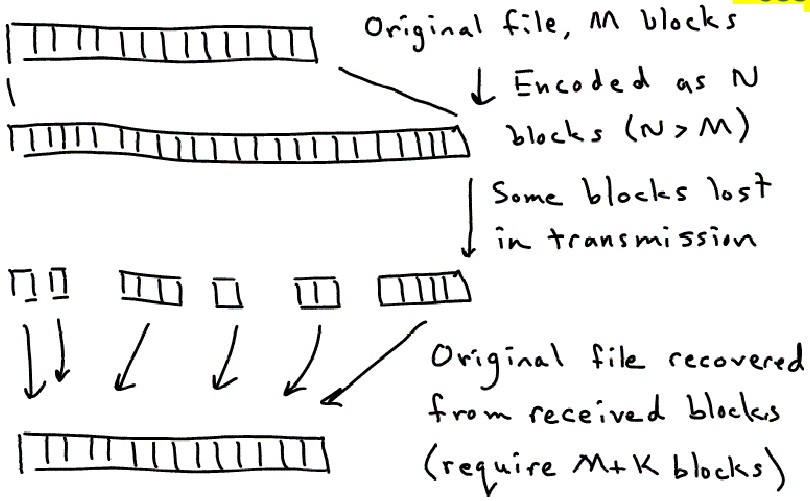
Digital fountain with erasure code

CS514

- Erasure code:
  - Original file has M packets (I.e. packets)
  - Encode as N packets (N > M)
    - N typically 2M or 3M
  - If any M+K packets received, original file can be reconstructed

Erasure code example

CS514



Erasure code performance measures

CS514

- Encoding/decoding cost
  - CPU/memory to encode and decode
- Reception efficiency
  - Ratio of number of packets needed to decode to packets in original file
  - (M+K) / M
  - Ideal is ratio of one (i.e. K = 0)

Erasure codes

CS514

- Reed-Solomon codes
  - Encoding and decoding scale quadratically with file size (I.e. as M^2)
- Tornado codes
  - Invented in late ’90s by Michael Luby
  - Encoding and decoding scales linearly with file size
  - Reception efficiency on the order of a few percent!

Reed-Solomon versus Tornado

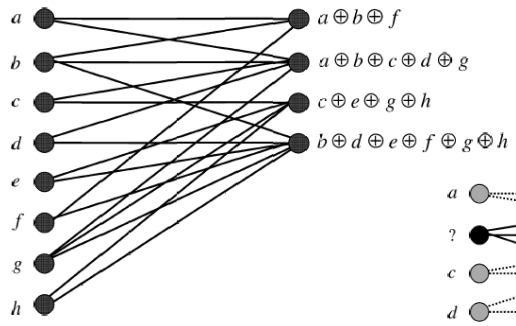
CS514

Encoding time (seconds), 1K packets		
Size	Reed-Solomon	Tornado
250 K	4.6	0.06
500 K	19	0.12
1 MB	93	0.26
2 MB	442	0.53
4 MB	1717	1.06
8 MB	6994	2.13
16 MB	30802	4.33

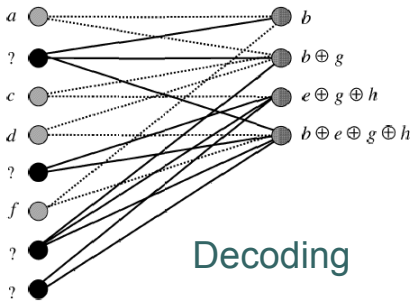
Decoding time (seconds), 1K packets		
Size	Reed-Solomon	Tornado
250 K	2.06	0.06
500 K	8.4	0.09
1 MB	40.5	0.14
2 MB	199	0.19
4 MB	800	0.40
8 MB	3166	0.87
16 MB	13829	1.75

Tornado Encoding and Decoding

CS514



Encoding



Decoding

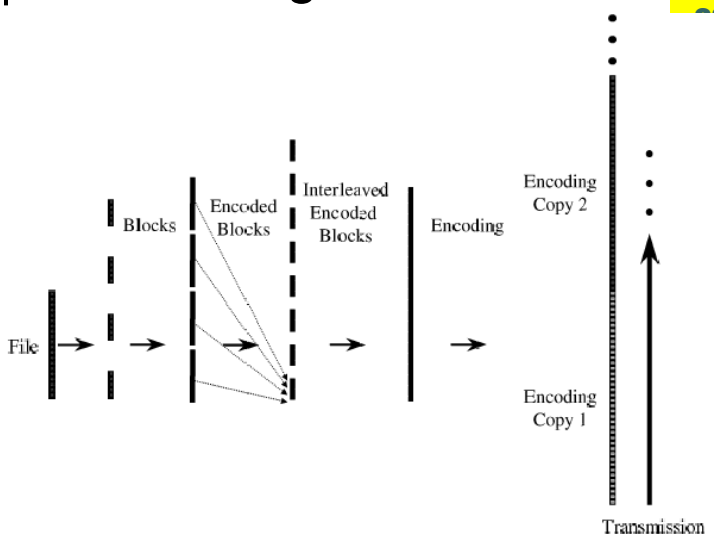
Digital fountain using Reed-Solomon codes

CS514

- Because of high encoding/decoding cost, file is broken into smaller blocks
  - Each block is encoded independently
- Now, receiver must get M+K packets for each block
  - M+K packets for block 1, M+K packets for block 2, etc.

Reed-Solomon with cyclic interleaving

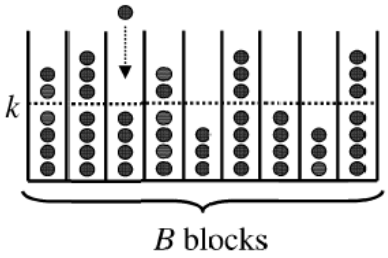
CS514



Problem with cyclic interleaving

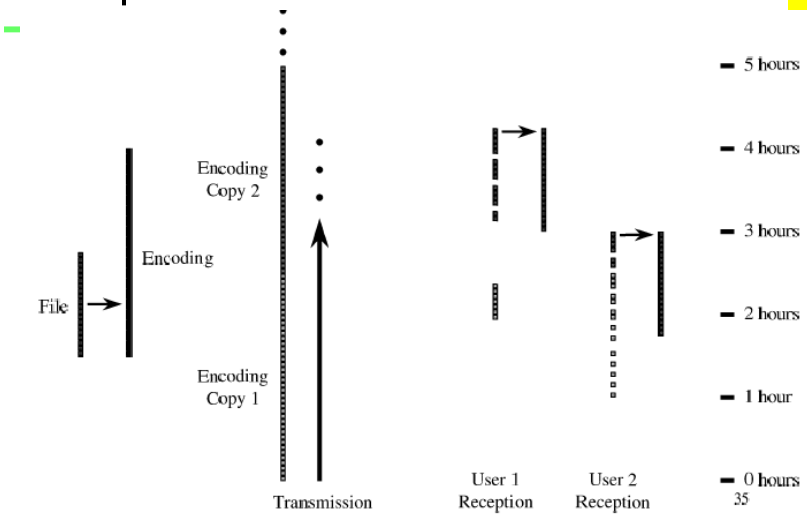
CS514

- Need a lot of blocks
- The “coupon collectors” problem
  - Receive a lot of packets waiting for the last blocks



Digital fountain with Tornado codes

CS514



Digital fountains haven't quite taken off either

CS514

- One problem is that Luby patented the technology
  - Apparently new, unpatented tornado codes are becoming available, so we'll see what happens
- Another is the lack of IP multicast
- Another is the relatively narrow application space (large files…though still important)
- Plus there are acceptable alternatives…
  - Like pub/sub?

Overlay multicast

CS514

- Actually, we've already seen overlay multicast systems
  - Pub/sub
- Overlay means: not using IP multicast
  - Also called "application level multicast"
- As we've discussed, advantage here is that you can decouple sender and receivers with buffering at forwarders

Overlay multicast and the promise of IP multicast

CS514

- Pub/sub systems are typically carefully engineered and static
  - Primary/backup “routes”
  - By the way, this is how you want to operate systems if at all possible
- The promise of IP multicast was that it would allow dynamic formation of multicast groups
- But IP multicast didn’t take off, and pub/sub systems don’t allow dynamic formation of groups

Overlay multicast: one definition

CS514

- An application-level multicast system that
  - Allows dynamic formation of groups, possibly among administratively separate hosts
  - Can (optionally) take advantage of buffering in forwarding nodes
- In other words, tries to get the best of both worlds…

Functions in overlay multicast

CS514

- *Discovery*: A joining node must find one or more nodes already in the group
- *Tree building*: The nodes must dynamically form a multicast tree among themselves, and repair it when nodes leave
  - Normally with small fanout---2, 3, 4…
- *Optimizing*: The nodes must form good trees
  - Neighbors are close, diameter not too big . . .
- *Forwarding*: The nodes must forward messages over the tree

Two early overlay multicast implementations

CS514

- Yoid (Paul Francis, NTT)
- Narada (Hui Zhang, CMU)
- Both allowed hosts to dynamically form single trees
- Both were conceived as IP multicast replacements
  - We didn't focus on buffering
- More similarities than differences

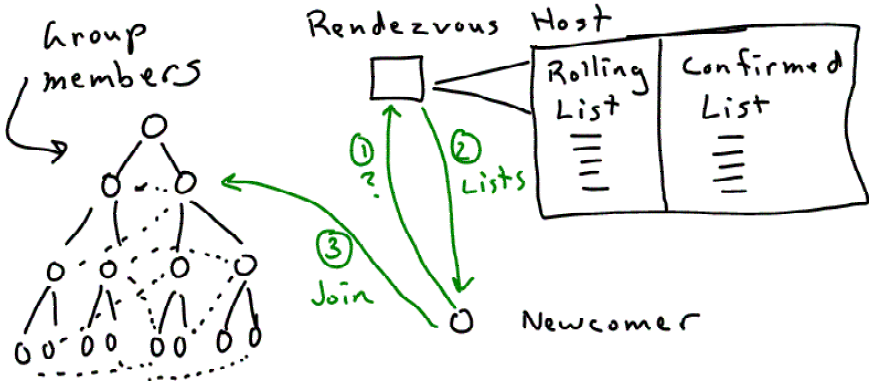
Discovery

CS514

- Typically requires some kind of “rendezvous” node
  - A node that is aware of at least some active group members
- Rendezvous node has a well-known name/address
  - So that joining nodes can contact it
- In Yoid, the rendezvous kept both a “rolling list” of recently joined nodes, plus a “confirmed list” of older nodes known to be still in the group (by pinging)
  - Newly joining nodes would try contacting rolling list nodes first, then confirmed list nodes

Discovery

CS514



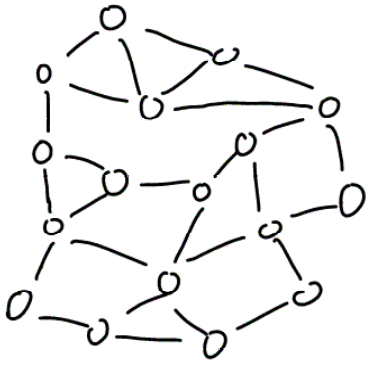
Tree building

CS514

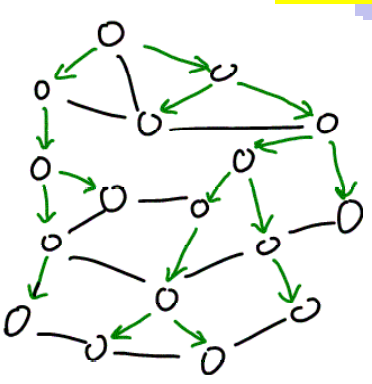
- In Narada, nodes gossip identity of all other nodes
  - Yoid didn't require full membership knowledge
- Each node selects 5 or so neighbors
  - Unstructured mesh network is formed
- Run a "routing algorithm" over this mesh to create a tree

Tree building

CS514



First form an unstructured mesh



Then routing alg builds a tree

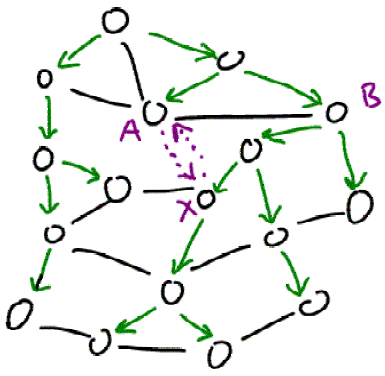
Optimizing

CS514

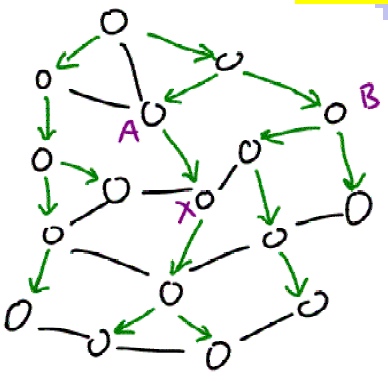
- Each node (say A) periodically pings a randomly selected node (say X) in the network
- If that node is closer than one of its existing neighbors (say B), then drop the neighbor B and add X as a new neighbor
  - Re-run the routing algorithm to optimize the tree

Optimizing

CS514



A pings X, finds that X is closer than B



A becomes a neighbor of X instead of B

Forwarding

CS514

- Overlay nodes can use UDP or TCP (or TCP friendly transport) between nodes
- If TCP, then nodes buffer to deal with slow-downs in TCP throughput
  - Nodes can also apply back-pressure up the tree, for instance if they buffer too much
- This buffering helps prevent congestion, and makes reliability much easier
  - Because each hop is reliable
  - But changes in topology, or buffer overflow can still cause losses