Quick recap of last lecture

- 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

- 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

- 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

- 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

Few exchanges, but most of them result in infection.

Many exchanges, but most of them don't result in infection.

\[ \text{Time (rounds)} \rightarrow O(\log N) \]
Initial multicast “bypasses” the early rounds

Multicast puts system at this level of infection in one shot

These gossip rounds are avoided

A little more gossip spreads message to remaining nodes

Results in a faster infection

Infection curve therefore looks more like this
**Pbcast versus SRM**

- 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

- 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

- 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

- 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:

- 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 lose one packet, have to keep listening until that packet comes around again!
Naïve case with no loss

Naïve case with loss
Digital fountain with erasure code

- **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

Original file, $M$ blocks

$\downarrow$ Encoded as $N$ blocks ($N > M$)

$\downarrow$ Some blocks lost in transmission

Original file recovered from received blocks (require $M+K$ blocks)
Erasure code performance measures

- 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

- 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

<table>
<thead>
<tr>
<th>Encoding time (seconds), 1K packets</th>
<th>Decoding time (seconds), 1K packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Reed-Solomon</td>
</tr>
<tr>
<td>250 K</td>
<td>4.6</td>
</tr>
<tr>
<td>500 K</td>
<td>19</td>
</tr>
<tr>
<td>1 MB</td>
<td>93</td>
</tr>
<tr>
<td>2 MB</td>
<td>442</td>
</tr>
<tr>
<td>4 MB</td>
<td>1717</td>
</tr>
<tr>
<td>8 MB</td>
<td>6994</td>
</tr>
<tr>
<td>16 MB</td>
<td>30802</td>
</tr>
</tbody>
</table>

Tornado Encoding and Decoding

Encoding

Decoding
Digital fountain using Reed-Solomon codes

- 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
Problem with cyclic interleaving

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

Digital fountain with Tornado codes
Digital fountains haven’t quite taken off either

- 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

- 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

- 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

- 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

- **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

- **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

- 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
Tree building

- 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

First form an unstructured mesh

Then routing alg builds a tree
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.
Forwarding

- Overlay nodes can use UDP or TCP (or TCP friendly transport) between nodes.
- If TCP, then nodes buffer to deal with slowdowns 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.