Lecture 14: Multicast Wrapup

14.0 Main Points

<previous lecture notes: routing and MBone>
Scoped multicast
Reliable multicast
Layered multicast (congestion control)
Ordered multicast

14.1 Scoped Multicast

How do we keep from sending multicast messages everywhere?

Recall that multicast routing works on "broadcast and prune", so that even announcements with purely local interest (e.g., pizza at 5 in Sieg 422), get broadcast everywhere, before being pruned back.

One approach taken in netnews, sender gives scope for news message (e.g., CS, UW, pacific northwest, usa, world).

Mbone does the same thing -- sender uses hop count (TTL) field in IP packet to limit how far a multicast is sent.

Further, instead of each hop counting 1, some hops count more -- e.g., exiting CS counts 8, exiting UW counts 16, ...

Sender targeting UW then puts TTL of 15; won't be sent beyond campus.

Alternative approach: assign multicast addresses in blocks, each block having specific scope. If want to send locally, use an address in a local block. If want to send everywhere in world, use an address in a global block. Routers must keep track of blocks of addresses, and forward only global ones.

14.2 Reliable Multicast

Motivation: multicast often used for audio/video teleconferencing. Real time constraints mean usually don't have time to retransmit dropped packets.

What if want reliable multicast transmission? For example, distributed whiteboard -- want everyone to see the same contents of the board.

Try #1: have every receiver ack every packet it receives, as in unicast ARQ

ack implosion : sender will be overwhelmed!

Try #2: hop-by-hop ARQ -- routers keep track of unacknowledged data, and resend if not ack.

Router complexity: requires routers to maintain state, timers, etc.

Try #3: negative acknowledgments (NAK) -- receiver sends NAK if it *doesn't* receive the packet

How?

If it sees a gap in the sequence #'s it receives (as in fast retransmit)

If it doesn't get a message for a while (could have missed last message in transmission). Requires sender to keep sending periodically, to avoid every receiver poking to see if there's been a later message. Seems inefficient, but less inefficient than ACK implosion!

What if drop packet near sender? *NAK implosion*

What if drop packet at last hop before many receivers? *NAK implosion*

Try #4: scalable reliable multicast (McCanne -- how the whiteboard app works on the MBone)

Challenge: can we make reliable multicast scalable?
Key idea: multicast NAK's and retransmissions, with increasing scope.

If receiver misses a packet, starting with local scope (e.g., 2 hops):

Start: pick a random timeout

after timeout, multicast NAK in local scope

if got packet and hear NAK, multicast retransmit within local scope
cancel timeout if hear retransmission (we're done!)

if hear NAK and no retransmit, then either second retransmit got dropped, or (more likely) no one in local scopegot packet
go back to start, using progressively bigger scope and an exponentially bigger average timeout

On average, one receiver in each scope times out, sends NAK. Everyone missing the packet either hears reply, or we start over, using a bigger scope and a bigger timeout (so in bigger scope, only one receiver times out).

14.3 Layered Multicast (congestion control)

When sending data to a number of receivers, likely that some receivers will have more bandwidth than others (e.g., desktop workstation vs. telephone modem).

What should sender do?

Send at maximum rate across all receivers? Packets to low bandwidth receivers will get dropped (possibly most packets to them will be dropped!)

Send at minimum rate across all receivers? Degrades performance to high bandwidth receivers (e.g., video goes from clear at 10Mb/s to highly fuzzy, when low bandwidth receiver connects).

Solution (at least for real time audio/video)? Layered multicast. Send each packet at multiple resolutions, one for each bandwidth.

High bandwidth receivers get entire packet; low bandwidth receiver gets only slice of packet targeted to it.

For example, if receivers connecting at 56Kbps, 1Mb/s, 100Mb/s: encode audio on 56Kbps channel (modem users get audio and no video), 1Mb/s receivers get audio and coarse video, 100 Mb/s receivers get full MPEG video.

14.4 Ordered Multicast

Can we deliver multicast packets in order from multiple senders to multiple receivers?

Easy for unicast -- just put a sequence # in each packet.

Same for multicast from a single sender -- packets from same sender can be easily delivered in order.

What about packets from multiple senders?

Packets may arrive at different receivers in different orders

For example: packet #1 from A may arrive at B before packet #1 from C, but packet from C may arrive at D before the one from A.

Does this matter? Ever get email response before you get the original message? Same thing can happen in netnews, chat rooms, etc.

Another example: distributed whiteboard. What if erase sent from one sender, while draw sent by another? Different users could see different content!

Potential cause of bugs -- and nearly impossible to track down, since non-deterministic (e.g., only if packet A got dropped, or queued behind a big transfer). If can't reproduce problem, can't fix it!

Almost every shared memory multiprocessor today uses message passing; similar problems occur if one processor sees a different memory state than another processor! Required for performance, but causes no end of bugs.

One approach: total order (sometimes called sequentially consistent). Everyone sees packets in same order.

Can be implemented with central sequence #'s. Give A's packet #1, C's packet #2; easy to deliver in order, but not very scalable (and what if A crashes before sending its packet??? No one else can send!)

Another approach: causal ordering. Packets can be delivered in different order to different machines, but packets "caused" by receipt of message aren't delivered before message causing them.

For example, in email, response message contains pointer to message being responded to; don't deliver response until original message arrives. Q: what if it never arrives? Unrelated messages can be interleaved in any fashion.

Yet another approach: weak ordering. Use some application-specific information to order messages.

Two examples: distributed whiteboard. Every message has timestamp; packets drawn as they arrive. If another packet arrives later with an earlier timestamp (e.g., because of a retransmission), then redraw screen with each packet applied in timestamp order.

Another example: in shared memory multiprocessors, use total order between synchronization points (e.g., mutual exclusive lock acquire) -- assume that if program doesn't use synchronization to enforce an order of operations, any order of operations is ok.