Distributed Systems
07. Group Communication

Paul Krzyzanowski
Rutgers University
Fall 2016

Modes of communication

• One-to-One
  – unicast
  • 1→1
  • Point-to-point
  – Anycast
    • 1→nearest 1 of several identical nodes
    • Introduced with IPv6; used with BGP

• One-to-many
  – multicast
  • 1→many
  • group communication
  – broadcast
    • 1→all

Groups

Groups allow us to deal with a collection of processes as one abstraction

Send message to one entity
  – Deliver to entire group

Groups are dynamic
  – Created and destroyed
  – Processes can join or leave
    • May belong to 0 or more groups

Primitives
  join_group, leave_group, send_to_group, query_membership

Design Issues

• Closed vs. Open
  – Closed: only group members can send messages

• Peer vs. Hierarchical
  – Peer: each member communicates with group
  – Hierarchical: go through dedicated coordinator(s)
  – Diffusion: send to other servers & clients

• Managing membership & group creation/deletion
  – Distributed vs. centralized

• Leaving & joining must be synchronous

• Fault tolerance
  – Reliable message delivery? What about missing members?

Failure considerations

• Crash failure
  – Process stops communicating

• Omission failure (typically due to network)
  – Send omission: A process fails to send messages
  – Receive omission: A process fails to receive messages

• Byzantine failure
  – A message is faulty

• Partition failure
  – The network may get segmented, dividing the group into two or more unreachable sub-groups

Implementing Group Communication Mechanisms
Hardware multicast

If we have hardware support for multicast
- Group members listen on network address

Hardware broadcast

If we only have hardware support for broadcast
- Software filters incoming multicast address
  - May use auxiliary address (not in the network address header)

Software: multiple unics

Sender knows group members

Software: hierarchical

Multiple unics via group coordinator
- Coordinator knows group members

Reliability of multics

Atomic multics

Atomicity
Message sent to a group arrives at all group members
- If it fails to arrive at any member, no member will process it

Problems
- Unreliable network
  - Each message should be acknowledged
  - Acknowledgements can be lost
- Message sender might die
Achieving atomicity

- General idea
  - Ensure that every recipient acknowledges receipt of the message
  - Only then allow the application to process the message

- Easier said than done!
  - What if a recipient dies after acknowledging the message?
  - Is it obligated to restart?
  - If it restarts, will it know to process the message?
  - What if the sender (or coordinator) dies partway through the protocol?

- Retry through network failures & system downtime
- Sender & receivers maintain a persistent log
- Each message has a unique ID so we can discard duplicates
- Sender
  - Send message to all group members
  - Write message to log
  - Wait for acknowledgement from each group member
  - Write acknowledgement to log
  - If timeout on waiting for an acknowledgement, retransmit to group member
- Receiver
  - Log received non-duplicate message to persistent log
  - Send acknowledgement
- NEVER GIVE UP!
  - Assume that dead senders or receivers will be rebooted and will restart where they left off

Reliable multicast

- All non-faulty group members will receive the message
  - Assume sender & recipients will remain alive
  - Network may have glitches
  - Retransmit undelivered messages

- Acknowledgements
  - Send message to each group member
  - Wait for acknowledgement from each group member
  - Retransmit to non-responding members
  - Subject to feedback implosion

- Negative acknowledgements
  - Use a sequence # on each message
  - Receiver requests retransmission of a missed message
  - More efficient but requires sender to buffer messages indefinitely

Unreliable multicast (best effort)

- Basic multicast
- Hope it gets there

Message ordering

Good Ordering

- Assume that dead senders or receivers will be rebooted and will restart where they left off
Bad Ordering

message a
order received
message b
a, b

Good Ordering

message a
order received
message b

Bad Ordering

message a
order received
message b
b, a

Sending versus Delivering

• Multicast receiver algorithm decides when to deliver a message to the process.

• A received message may be:
  - Delivered immediately (put on a delivery queue that the process reads)
  - Placed on a hold-back queue (because we need to wait for an earlier message)
  - Rejected/discarded (duplicate or earlier message that we no longer want)

Sending, delivering, holding back

Global time ordering

• All messages arrive in exact order sent

• Assumes two events never happen at the exact same time!

• Difficult (impossible) to achieve
Total ordering

- Consistent ordering everywhere
- All messages arrive at all group members in the same order
  - They are sorted in the same order in the delivery queue

1. If a process sends \( m \) before \( m' \) then any other process that delivers \( m' \) will have delivered \( m \).
2. If a process delivers \( m' \) before \( m \) then every other process will have delivered \( m \) before \( m' \).

Implementation:
- Attach unique totally sequenced message ID
- Receiver delivers a message to the application only if it has received all messages with a smaller ID

Causal Ordering

Partial ordering
- Messages sequenced by Lamport or Vector timestamps

If multicast(\( G, m \)) \( \rightarrow \) multicast(\( G, m' \))
then every process that delivers \( m' \) will have delivered \( m \).

If message \( m' \) could be causally dependent on message \( m \), all processes must deliver \( m \) before \( m' \).

Algorithm
- Each entry = # of latest message from the corresponding group member that causally precedes the event

Causal Ordering: Example

- \( P_a \) receives message \( m_0 \) from \( P_b \) with \( V_a = (1,1,0) \)
- \( P_a \) compares current timestamp \( V_a \) with received \( V_b \) = \( (1,0,0) \)
  - Yes: \( V_a | V_b = (1,1,0) \) \( \rightarrow \) sequential order

- \( P_a \) receives message \( m_1 \) from \( P_b \) with \( V_a = (1,1,0) \)
- \( P_a \) compares current timestamp \( V_a \) with received \( V_b \) = \( (1,0,0) \)
  - No: \( V_a | V_b = (1,0,1) \) \( \rightarrow \) total ordering

Causal Ordering: Example

- \( P_a \) receives message \( m_0 \) from \( P_b \) with \( V_a = (1,1,0) \)
- \( P_a \) compares current timestamp \( V_a \) with received \( V_b \) = \( (1,0,0) \)
  - Yes: \( V_a | V_b = (1,1,0) \) \( \rightarrow \) sequential order

- \( P_a \) receives message \( m_1 \) from \( P_b \) with \( V_a = (1,1,0) \)
- \( P_a \) compares current timestamp \( V_a \) with received \( V_b \) = \( (1,0,0) \)
  - No: \( V_a | V_b = (1,0,1) \) \( \rightarrow \) total ordering
Causal Ordering: Example

(1) Is the held-back message \( m_1 \) in FIFO order from \( P_0 \)?

- Compare current \( V \) on \( P_2 \), \( V_2(1,0,0) \) with held-back \( V \) from \( P_0 \), \( V_1(1,1,0) \)
- Yes: \( V_2[1] = 0 \), received \( V_1[1] = 1 \) ⇒ sequential

(2) Is \( V_2[i] \leq V_1[i] \) for all other \( i \)?

- Now yes. Element 0: \( 1 \leq 1 \), element 2: \( 0 \leq 0 \).

Deliver \( m_1 \).

More efficient than total ordering:
- No need for a global sequencer.
- No need to send acknowledgements.

### Sync ordering

- Messages can arrive in any order
- Special message type
  - Synchronization primitive
  - Ensure all pending messages are delivered before any additional (post-sync) messages are accepted

### FIFO ordering

- Messages from the same source are delivered in the order they were sent.
- Message \( m \) must be delivered before message \( m' \) if \( m \) was sent before \( m' \) from the same host.

If a process issues a multicast of \( m \) followed by \( m' \), then every process that delivers \( m' \) will have already delivered \( m \).

### Unordered multicast

- Messages can be delivered in different order to different members.
- Order per-source does not matter.

### Multicasting considerations

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Atomic</th>
<th>Reliable</th>
<th>Unreliable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordering</td>
<td>unordered</td>
<td>FIFO</td>
<td>causal</td>
</tr>
</tbody>
</table>

### IP multicast routing

- Message Ordering
**IP multicast routing**

- Deliver messages to a subset of nodes
- How do we identify the recipients?
  - Enumerate them in the header?
  - What if we don’t know?
  - What if we have thousands of recipients?
- Use a *special address* to identify a group of receivers
  - A copy of the packet is delivered to all receivers associated with that group
  - Class D multicast IP address
    - 32-bit address that starts with 1110 (224.0.0.0 – 239.255.255.255)
  - Host group = set of machines listening to a particular multicast address

**IP multicasting**

- Can span multiple physical networks
- Dynamic membership
  - Machine can join or leave at any time
- No restriction on number of hosts in a group
- Machine does not need to be a member to send messages
- Efficient: Packets are replicated only when necessary

**IP multicast addresses**

- Addresses chosen arbitrarily for an application
- Well-known addresses assigned by IANA
  - Internet Assigned Numbers Authority
  - See [http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml](http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml)
  - Similar to ports – service-based allocation
    - For ports, we have:
      - FTP: port 21, SMTP: port 25, HTTP: port 80
    - For multicast, we have:
      - 224.0.0.1: all systems on this subnet
      - 224.0.0.2: all multicast routers on subnet
      - 224.0.0.3: Philips Health
      - 224.0.0.4: Amex Market Data
      - 224.0.0.5: Microsoft & MSNBC

**IGMP**

- Internet Group Management Protocol (IGMP)
  - Operates between a host and its attached router
  - Goal: *allow a router to determine to which of its networks to forward IP multicast traffic*
  - IP protocol (IP protocol number 2)
  - Three message types
    - Membership_query
      - Sent by a router to all hosts on an interface to determine the set of all multicast groups that have been joined by the hosts on that interface
    - Membership_report
      - Host response to a query or an initial join or a group
    - Leave_group
      - Host indicates that it is no longer interested
      - Optional: router infers this if the host does not respond to a query

**Multicast Forwarding**

IGMP allows a host to subscribe to receive a multicast stream

What about the source?
- There is no protocol for the source!
- It just sends to a class D address
- Routers have to do the work

**IGMP & Wide-Area Multicast Routing**

- No protocol!
**Multicast Forwarding**

- **IGMP:** Internet Group Management Protocol
  - Designed for routers to talk with hosts on directly connected networks

- **PIM:** Protocol Independent Multicast
  - Multicast Routing Protocol for delivering packets across routers
  - Topology discovery is handled by other protocols

**Flooding: Dense Mode Multicast**

- **Relay multicast packet to all connected routers**
  - Use a spanning tree and use reverse path forwarding (RPF) to avoid loops
  - Feedback & cut off if there are no interested receivers on a link
  - A router sends a prune message.
  - Periodically, routers send messages to refresh the prune state
  - Flooding is initiated by the sender’s router

- **Reverse path forwarding (RPF):** avoid routing loops
  - Packet is duplicated & forwarded ONLY IF it was received via the link that is the shortest path to the sender
  - Shortest path is found by checking the router’s forwarding table to the source address

**Sparse Mode Multicast**

- **Initiated by the routers at each receiver**
  - Each router needs to ask for a multicast feed with a PIM Join message
  - Initiated by a router at the destination that gets an IGMP join
  - Rendezvous Point: meeting place between receivers & source
  - Join messages propagate to a defined rendezvous point (RP)
  - Sender transmits only to the rendezvous point
  - RP announcement messages inform edge routes of rendezvous points
  - A Prune message stops a feed

- **Advantage**
  - Packets go only where needed
  - Creates extra state in routers only where needed

**IP Multicast in use**

- **Initially exciting:**
  - Internet radio, NASA shuttle missions, collaborative gaming

- **But:**
  - Few ISPs enabled it
  - For the user, required tapping into existing streams (not good for on-demand content)
  - Industry embraced unicast instead

**IP Multicast in use: IPTV**

- **IPTV has emerged as the biggest user of IP multicast**
  - Cable TV networks have migrated (or are migrating) to IP delivery

- **Cable TV systems: aggregate bandwidth ~ 4.5 Gbps**
  - Video streams: MPEG-2 or MPEG-4 (H.264)
  - MPEG-2 HD: ~30 Mbps ⇒ 150 channels ⇒ ~4.5 Gbps
  - MPEG-4 HD: ~6-9 Mbps; DVD quality: ~2 Mbps

- **Multicast**
  - Reduces the number of servers needed
  - Reduces the number of replicated network streams

---

*© 2014-2016 Paul Krzyzanowski*
IP Multicast in use: IPTV

- Multicast allows one stream of data to be sent to multiple subscribers using a single address
- IGMP from the client
  - Subscribe to a TV channel
  - Change channels
- Use unicast for video on demand