Distributed Systems

07. Group Communication

Paul Krzyzanowski
Rutgers University
Fall 2014
Modes of communication

• **unicast**
  – $1 \leftrightarrow 1$
  – Point-to-point

• **anycast**
  – $1 \rightarrow$ nearest 1 of several identical nodes
  – Introduced with IPv6; used with BGP

• **“netcast”**
  – $1 \rightarrow$ many but 1 at a time

• **multicast**
  – $1 \rightarrow$ many
  – group communication

• **broadcast**
  – $1 \rightarrow$ all
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 into two or more unreachable groups
Implementing Group Communication Mechanisms
Hardware multicast

Hardware support for multicast

– Group members listen on network address

\[ \text{send} \quad \text{addr} = m_1 \]

\[ \text{listen} \quad \text{addr} = m_1 \]

\[ \text{listen} \quad \text{addr} = m_1 \]

\[ \text{listen} \quad \text{addr} = m_1 \]

\[ \text{listen} \quad \text{addr} = m_1 \]
Hardware broadcast

Hardware support for broadcast
– Software filters multicast address
  • May be auxiliary address

broadcast(id=m)

- discard id=m
- accept id=m
- accept id=m
- discard id=m
- accept id=m
Multiple unicasts
  – Sender knows group members
Multiple unicasts via group coordinator
– coordinator knows group members

Diagram:
- Coordinator
- Listen local addr = $a_2$
- Listen local addr = $a_3$
- Listen local addr = $a_5$
- Send($a_2$)
- Send($a_3$)
- Send($a_5$)
- Send(c)
Reliability of multicasts
Atomic multicast

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 (2-phase commit variation)

Retry through network failures & system downtime
Sender and receivers maintain persistent log

1. Send message to all group members
   - Each receiver acknowledges message
   - Saves message and acknowledgement in log
   - Does not pass message to application

2. Sender waits for all acknowledgements
   - Retransmits message to non-responding members
     - Again and again… until responses from all are received
     - Now the sender knows all group members have received the message

3. Sender sends “deliver” message to all members
   - Each recipient delivers message to application
   - Sends reply to server
Achieving atomicity

- All members will eventually get the message
  - Phase 1:
    - Make sure that everyone gets the message
  - Phase 2:
    - Once everyone has confirmed receipt, let the application see it

- If a machine dies, it checks its persistent log to find its state in the protocol and recover the message

- Lots of other protocols
- All of them consume resources and time
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

order received

message $a$

message $b$

$a, b$

$a, b$

$a, b$
Bad Ordering

message $a$

message $b$

order received

$a, b$

$b, a$
Good Ordering

message $a$

order received

$a, b$

message $b$

$a, b$
Bad Ordering

message $a$

message $b$

order received $a, b$

order received $b, a$
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/discard**
    (duplicate or earlier message that we no longer want)
Sending, delivering, holding back

sender

send

Multicast sending algorithm

message transmission

receiver

deliver

delivery queue

hold-back queue

discard

Multicast receiving algorithm

September 30, 2014

© 2014 Paul Krzyzanowski
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) → 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’.
$m_1$ is causally dependent on the receipt of $m_0$. Hence, $m_1$ must be delivered after $m_0$ has been delivered.

$m_0$ and $m_1$ have no causal relationship (they are concurrent). Any process can deliver them in any order.
Causal ordering

• Implementation: $P_a$ receives a message from $P_b$
  – Each process keeps a precedence vector (similar to vector timestamp)
  – Vector is updated on multicast send and receive events
    • Each entry = # of latest message from the corresponding group member that causally precedes the event
  – Algorithm
    • When $P_b$ sends a message, it increments its own entry and sends the vector
      
      $V_b[b] = V_b[b] + 1$
      
      Send $V_b$ with the message
    
    • When $P_a$ receives a message from $P_b$
      – Check that the message arrived in sequential order from $P_b$
        
        $V_b[b] == V_a[b] + 1$ ?
      – Check that the message does not causally depend on something $P_a$ has not seen
        
        $\forall i, i \neq b: V_b[i] \leq V_a[i]$ ?
      – If both conditions are satisfied, $P_a$ will deliver the message
      – Otherwise, hold it until the conditions are satisfied
P₂ receives message \( m_1 \) from P₁ with \( V=(1,1,0) \). At this time, P₂ has \( V=(0,0,0) \)

1. Is this in sequential order from P₁? (i.e., is it the next sequential message from P₁?)
   - Yes: \( V₂[1] = 0 \), received \( V₁[1] = 1 \) ⇒ it’s sequential

2. Is \( V₁[i] \leq V₂[i] \) for all other \( i \)? (i.e., did we miss a message that P₁ saw?)
   - No. \( V₁[0] > V₂[0] \) (1 > 0)

**Test failed. So we hold back \( m₁ \) at P₂**
P₂ receives message m₀ from P₀ with V=(1,0,0)
At this time, P₂ has V=(0,0,0). Note: we did not update V from m₁ because we held m₁

(1) Is this in sequential order from P₀? (i.e., *is it the next sequential message from P₁?*)
Yes: V₂[0] = 0, received V₁[0] = 1 ⇒ sequential

(2) Is V₀[i] ≤ V₂[i] for all other i?

**Test passed. Deliver m₀.**
Now check hold-back queue. Can we deliver m₁?
Causal Ordering: Example

(1) Is held-back message $m_1$ in sequential order from $P_0$?
   Yes: $V_1[1] = 0$, received $V_1[1] = 1 \Rightarrow$ sequential

(2) Is $V_2[i] \leq V_1[i]$ for all other $i$?
   Now yes. Element 0: $(1 \leq 1)$, element 2: $(0 \leq 0)$.

**Test passed. 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 $m'$ iff $m$ 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

- Atomic
- Reliable
- Unreliable

Message Ordering:
- Unordered
- Unordered FIFO
- Sync
- Causal
- Total
- Global
IP multicast routing
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
      (224.0.0.0/4 = 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
  – 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
      224.0.23.173: Philips Health
      224.0.23.52:  Amex Market Data
      224.0.12.0-63: 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 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

Protocol Independent Multicast (PIM) provides Internet multicast routing. IGMP is used to control the membership in multicast groups.
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
Flooding: Dense Mode Multicast

- Advantage:
  - Simple
  - Good if the packet is desired in most locations

- Disadvantage:
  - wasteful on the network, wasteful extra state & packet duplication on routers
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
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
The End