Indirect communication

- Communication through an intermediary
  - No direct coupling between the sender and the receiver(s)
- Space uncoupling – no need to know identity of receiver(s) and vice versa
  - Participants can be replaced, updated, replicated, or migrated
- Time uncoupling – independent lifetimes
  - Requires persistence in the communication channel

Good for ...

- Scenarios where users connect and disconnect very often
  - Mobile environments, messaging services, forums
- Event dissemination where receivers may be unknown and change often
  - RSS, events feeds in financial services
- Scenarios with very large number of participants
  - Google Ads system, Spotify
- Commonly used in cases when change is anticipated
  - need to provide dependable services

... but there are also some disadvantages

- Performance overhead introduced by adding a level of indirection
  - Reliable message delivery, ordering \( \rightarrow \) effect on scalability
- More difficult to manage because lack of direct coupling
- Difficult to achieve end-to-end properties
  - Real time behavior
  - Security
Characteristics

- **Indirect communication**
  - Communication through an intermediary
  - No direct coupling between the sender and the receiver(s)
- **Group communication**
  - Messages sent to a group of processes and delivered to all members of the group

Groups (of processes)

- **One-to-many communication**
  - Provide reliability and ordering guarantees
- **Group management functionality**
  - Maintain membership
  - Detect failure of member(s)

Types of groups

**Closed or open**

A group is **closed** if only members of the group can multicast to it. A group is **open** if processes outside the group may send to it.

**Overlapping or non-overlapping**

In **overlapping** groups, processes may be members of multiple groups. In **non-overlapping** groups, processes may belong to at most one group.

Group membership management

- Interface for group membership changes
  - Create and destroy groups, add or remove members to a group
- Failure detection
  - Mark processes as suspected or unsuspected and remove those processes that have (suspected) failed
  - Notify members of membership changes
  - Processes that join or leave
  - Perform group address expansion

Receive versus Deliver

- **Receive:** message has arrived and will be processed
- **Deliver:** message is allowed to reach upper layer

Unreliable (basic) multicast (using reliable unicast)

- Send (unicast) to each other process in the group!
- What if sender fails halfway through?

Multicast
Basic Multicast

- Use a reliable one-to-one send operation
- B-multicast \((g, m)\):
  - for each process \(p \in g\), send \((p, m)\)
  - receive\((m)\) at \(p\):
    - \(B\)-deliver\((m)\) at \(p\)

Reliable multicast

- Integrity
  - Messages delivered at most once
- Validity
  - If a correct process multicasts message \(m\), it will eventually deliver \(m\)
- Agreement
  - If a correct process delivers \(m\), then all correct processes in the group will eventually deliver \(m\)

Reliable multicast algorithm

- Use basic multicast to send to all (including self)
- On initialization
  - Received := \(\emptyset\)
- For process \(p\) to B-multicast message \(m\) to group \(g\)
  - B-multicast \((g, m)\) if \(p \in g\) is included as a destination
- On B-deliver\((m)\) at process \(q\) with \(g\) = group\((m)\)
  - if \((m, m)\) \(\in\) Received then
    - Received := Received \(\cup\) \{m\};
  - if \((q \neq p)\) then B-multicast\((g, m)\);

Other approaches

- Scalable reliable multicasting (SRM) using feedback suppression, hierarchical feedback control
- IP multicast requires hardware support

View-synchronous group communication

- Group views to provide a type of synchrony
  - All messages must be delivered within a view
  - A view is a list of members of a group
  - Members decide which processes belong to the group
    - ... and so they decide which messages are accepted or rejected
Atomic multicast
- totally ordered reliable multicast

Message orderings
1. Unordered
2. FIFO
3. Total
4. Causal
5. Hybrid orderings such as Total-Causal & Total-FIFO

FIFO ordering
- Intuition
  Messages from a process should be delivered in the order in which they were sent
- Solution
  Sender numbers the messages, receivers hold back those that have been received out of order

FO-multicast
- Send $s(p_1, g)$ with message
- Increment $s(p_1, g)$ by 1

FO-deliver
- If $s = R(p_1, g) + 1$
- FO-deliver and set $R(p_1, g) = s$
- Place in hold-back queue until $s = R(p_1, g) + 1$
Total ordering

- Intuition
  Messages from all processes should get a (unique) group wide ordering number, so all processes can deliver messages in a single order!
  
  Mental pitfall: the order itself does not have to make any sense, as long as all processes abide by it!

Implementing total ordering

- Sequencer
  - Simple
  - Central server (= single point of failure)
  - ISIS-algorithm
    - Not as simple
    - Distributed
    - Study on your own!

Sequencer

- Sequencer is logically external to the group
- Messages are sent to all members, including sequencer
  - Initially, have no "ordering" number
- Sequencer maps message identifiers to ordering numbers
  - Multicasts mapping to group
  - Once a message has an ordering number, it can be delivered according to that number

Sequence – final notes

- Note, again, that the ordering is completely up to the sequencer
  - It could collect all messages for half an hour and then assign numbers according to how many a’s there are in the message
  - While annoying to use, this is still a total order, and all processes will have to follow it!
Causal ordering

- **Intuition**
  - Captures causal (cause and effect) relationships via happened-before ordering
  - Vector clocks ensure that replies are delivered after the message that they are replying to

Algorithm for group member $p_i$ ($i = 1, 2, ..., N$)

- **On initialization**
  - $V^i_j[0] = 0$ ($j = 1, 2, ..., N$)

- **To CO-multicast message $m$ to group $g$**
  - $V^i_j[0] = V^i_j[1] + 1$

- **B-multicast** ($v > V^i_j[0]$)

- **On B-deliver($V^i_j$, $m$) from $p_j$ with $g = \text{group}(m)$**
  - place $V^i_j$, $m$ in hold-back queue
  - CO-deliver $m$
  - after removing it from the hold-back queue

Hybrid orderings

- **Causal order is not unique**
  - Concurrent messages
  - ...neither is FIFO

- **FIFO only guarantees per process not inter-process**

- **Total order only guarantees a unique order**
  - Combine with others to get stronger delivery semantics!

Summary

- **Group communication**
  - One-to-many, indirect communication

- **Different types of groups**
  - Open, closed, overlapping, and non-overlapping

- **Reliability in group communication**
  - Integrity, validity, and agreement

- **Group membership management**
  - Changes, failure detection, notification of membership changes, group address expansion
Summary

- Multicast, reliable and unreliable
- Message ordering
  - The ordering in delivering messages is necessary in some cases
  - Ordering is expensive in terms of delivery latency and bandwidth consumption
  - FIFO – order messages from each sender
  - Causal – order messages across senders
  - Total – same message ordering on all recipients

Reading

Chapter 15 “Coordination and Agreement”, Distributed systems, 5th ed. By Coulouris, Dollimore, Kindberg and Blair

Next Lecture

Consensus