Group Communication

Group oriented activities are steadily increasing. There are many types of groups:

- Open and Closed groups
- Peer-to-peer and hierarchical groups

Important issues

- Atomic multicast
- Ordered multicast
- Dynamic groups

Sometimes, certain features available in the infrastructure of a distributed system simplify the implementation of multicast. Examples are (1) multicast on an ethernet LAN (2) IP multicast
**Atomic multicast**

A multicast by a group member is called *atomic*, when the message is delivered to **every correct** (i.e. functioning) member, or to **no member** at all.

<table>
<thead>
<tr>
<th>Sender’s program</th>
<th>Receiver’s program</th>
</tr>
</thead>
<tbody>
<tr>
<td>i:=0;</td>
<td>if m is new ∅</td>
</tr>
<tr>
<td><strong>do</strong> i ≠ n ∅</td>
<td>accept it;</td>
</tr>
<tr>
<td>send message to i;</td>
<td>multicast m;</td>
</tr>
<tr>
<td>i:= i+1</td>
<td>m is duplicate ∅ discard m</td>
</tr>
<tr>
<td><strong>od</strong></td>
<td><strong>fi</strong></td>
</tr>
</tbody>
</table>
Basic vs. Reliable Multicast

Basic multicast does not consider process crashes. Reliable multicast does.

Three criteria for basic multicast:

Liveness. Each process must receive every message
Integrity. No spurious message received
No duplicate. Accepts exactly one copy of a message
**IP multicast**

Practical form of multicast - utilizes the inherent multicasting capability of the underlying medium. Class D addresses assigned to multicast groups.

![Source tree](image)

(a) Source tree

![Shared tree](image)

(b) Shared tree

Distribution tree is dynamically updated as members join or leave a group.
Ordered multicasts

Total order
Causal order
Local order (single source FIFO)

Why are they important?

Total order multicast is useful in the consistent update of replicated servers

Causal order multicast is relevant in implementing bulletin boards

Local order multicast is useful in updating cache memories in multi-computers
Implementing ordered multicasts
(basic version)

Total Order Multicast using a sequencer

Every process forwards the data to the sequencer.

{The sequencer $S$}

```plaintext
define seq: integer (initially seq=0)
do  receive m Æ multicast (m, seq); seq := seq+1;
    deliver m
od
```

Every process accepts and delivers the messages in the increasing order of $seq$. 
Total order multicast without a sequencer

Uses the idea of two-phase commit.

Step 1. Sender i sends \((m, ts)\) to all

Step 2. Receiver j saves it in a holdback queue, and sends \((a, ts)\)

Step 3. Receive all acks, and pick the largest ts. Then send \((m, commit)\) to all.

Step 4. Receiver removes it from the holdback queue and delivers \(m\).
Implementing causal order broadcast

Use vector clocks. (Note the difference from the classical model)

![Diagram showing vector clocks and message delivery]

The recipient $i$ delivers a message from $j$ iff

1. $\text{VC}_j(j) = \text{LC}_j(i) + 1 \quad \{\text{LC is the local vector clock}\}$

2. $\square k: k \neq j :: \text{VC}_k(j) \leq \text{LC}_k(i)$

What is the rationale behind these rules?
Dealing with open groups

Processes may join or leave a group, but life will be simpler, if everyone has a consistent view of the current membership. A list of the current members is called a view.

Views should propagate in the same order to all.

Example.
Current view $v_0(g) = \{0, 1, 2, 3\}$.
Let 1, 2 leave and 4 join the group concurrently.
These view change can be serialized in many ways:

- $\{0,1,2,3\} \cup \{0,1,3\} \cup \{0,3,4\}$, OR
- $\{0,1,2,3\} \cup \{0,2,3\} \cup \{0,3\} \cup \{0,3,4\}$, OR
- $\{0,1,2,3\} \cup \{0,3\} \cup \{0,3,4\}$

Collected from local observations and send by a total order multicast.