Multicast protocols
(protocolos de radiado)

Ernesto Jiménez
Prometeo
Multicast Protocol one layer architecture

Application

Multicast protocol

Reliable channels
Multicast Protocol two layer architecture

Application

Multicast protocol (order guarantee)

Multicast protocol (delivery guarantee)

Reliable channels

X-multicast(m) → X-deliver(m)

Y-multicast(m) → Y-deliver(m)

send(m) → receive(m)
Reliable Channel properties

• **Validity:** If a correct process send(m), this message m must be eventually receive(m) in all correct processes.

• **Integrity:** If a process receive(m), then this message m has been previously send(m), and a process receive(m) at most once.
Reliable Multicast protocol architecture

Application

R-multicast(m)  ➔  R-deliver(m)

Reliable Multicast protocol

send(m)  ➔  receive(m)

Reliable channels
Reliable Multicast protocol properties

- **Validity**: if a correct process R-multicast a message $m$, $m$ must eventually R-deliver it in all correct processes.

- **Integrity**: if a process R-deliver($m$), then this message $m$ has been R-multicast by some process, and each process must R-deliver $m$ at most once.

- **Agreement**: if a correct process R-deliver a message $m$, it must eventually R-deliver $m$ in all correct processes.
Reliable multicast protocol

Process $p_i$ pseudocode

```plaintext
init:
  received$_i$ $\leftarrow$ $\emptyset$; //set of messages received by process $p_i$

when $R$-multicast $(m)$ is executed by $p_i$’s application:
  send $(m)$ to all processes.

when receive $(m)$ is executed by $p_i$’s system:
  if $(m \notin received$_i)$ then
    received$_i$ $\leftarrow$ received$_i$ $\cup$ $(m)$;
    $R$-deliver$(m)$;
    send $(m)$ to all processes
  end if.
```
Uniform Multicast Protocol architecture

Application

U-multicast(m)  U-deliver(m)

Uniform Multicast protocol

send(m)  receive(m)

Reliable channels
Uniform Multicast protocol properties

• **Validity**: if a correct process U-multicast a message m, m must eventually U-deliver it in all correct processes.

• **Integrity**: if a process U-deliver(m), then this message m has been U-multicast by some process, and a process must U-deliver m at most once.

• **Uniform Agreement**: if a process (correct or not) U-deliver a message m, it must eventually U-deliver m in all correct processes.
Uniform multicast protocol

init:
next_seqnum_i ← 1; received_i ← \emptyset; set of messages received by process p_i

when U-multicast (m) is executed by p_i’s application:
m.sender=i; m.seqnum ← next_seqnum_i;
send (m) to all processes; next_seqnum_i ← next_seqnum_i + 1;

when receive (m) is executed by p_i’s system:
if (m ∉ received_i ) then
    received_i ← received_i U {m}; pending_i ← pending_i U {m};
send(ACK, i) to m.sender;
send (m) to all processes
end if.

when receive (ACK, id_j) is executed by p_i’s system:
// set of ACK senders
let ACK-IDs(m) = \{id(p) : (ACK,id_j) has been received and
    (ACK,id_j).sender = p and m.sender= id_j\};
while (∃ m ∈ (pending_i) : m.sender = id_j) and
    (|ACK-IDs(m) U m.sender U \{i\} | > \|\Pi/2\) do // A majority of processes received m
    U-deliver(m);
    pending_i ← pending_i \ {m}
end while.

Process p_i pseudocode
FIFO Reliable Multicast Protocol architecture

Application

FIFO-multicast(m)

FIFO-Multicast protocol

R-multicast(m)

Reliable Multicast protocol

send(m)

Reliable channels

R-deliver(m)

receive(m)

FIFO-deliver(m)
FiFO-Reliable Multicast protocol properties

• **Validity:** if a correct process FiFO-multicast a message m, m must be FiFO-deliver in all correct processes.

• **Integrity:** if a process FiFO-deliver(m), then m has been FiFO-multicast by some process, and a process must FiFO-deliver m at most once.

• **Agreement:** if a correct process FiFO-deliver a message m, it must be FiFO-deliver in all correct processes.

• **FiFO message delivery.** If a process p_i FiFO-multicast(m) and then FiFO-multicast(m’), then no process FiFO-deliver(m’) unless it has previously FiFO-deliver(m).
init:
- seqnumᵢ ← 1; //sequence number of process pᵢ
- ∀k, next_seqnumᵢ [k] ← 1; //next message to delivered by process pᵢ
- msg_setᵢ ← ∅; //set of messages waiting to be delivered by process pᵢ

when FIFO-multicast (m) is executed by pᵢ’s application:
- m.sender ← i;
- m.seqnum ← seqnumᵢ;
- R-multicast (m);
- seqnumᵢ ← seqnumᵢ + 1.

when R-deliver (m) is executed by pᵢ’s system:
- let j = m.sender;
- msg_setᵢ ← msg_setᵢ ∪ {m};
- while (∃m’ ∈ msg_setᵢ : (m’.sender = j) and (next_seqnumᵢ [j] = m’.seqnum)) do
  FIFO-deliver(m’);
  msg_setᵢ ← msg_setᵢ \ {m'};
  next_seqnumᵢ[j] ← next_seqnumᵢ[j] + 1
end while.

Process pᵢ pseudocode
Causal Order Reliable Protocol architecture

Application

Causal Order Multicast protocol

Reliable Multicast protocol

Reliable channels

CO-multicast(m) → CO-deliver(m)

R-multicast(m) → R-deliver(m)

send(m) → receive(m)
Causal Reliable Multicast protocol properties

- **Validity**: if a correct process CO-multicast a message $m$, $m$ must be delivered in all correct processes.

- **Integrity**: if a process CO-deliver($m$), then this message $m$ has been CO-multicast by some process, and a process must CO-deliver $m$ at most once.

- **Agreement**: If a correct process CO-deliver a message $m$, this message $m$ must be CO-deliver in all correct processes.

- **Causal message delivery** ($<_{cau}$): if CO-multicast($m$) causally precedes CO-multicast($m'$), then no process CO-deliver($m'$) unless it has previously CO-deliver($m$).
Causal Reliable Multicast protocol properties (cont.)

• **Causal Order** \((\text{CO-multicast}(m) <_{\text{cau}} \text{CO-multicast}(m'))\): if some of the following two cases happens:
  1. A process \(p_k\) invokes \(\text{CO-multicast}(m)\) before this same process \(p_k\) invokes \(\text{CO-multicast}(m')\).
  2. A process \(p_k\) invokes \(\text{CO-multicast}(m)\), and a process \(p_s\) invokes \(\text{CO-deliver}(m)\) before this same process \(p_s\) invokes \(\text{CO-multicast}(m')\).

• **Causal message delivery** \((<_{\text{cau}})\): every process must preserve causal order \(\text{CO-multicast}(m) <_{\text{cau}} \text{CO-multicast}(m')\), for all messages \(m\) and \(m'\), when it executes \(\text{CO-deliver}\).
Causal Reliable Multicast protocol properties (cont.)

\[ \text{CO-multicast}(m) \prec \text{cau} \text{CO-multicast}(m') \]

Causal message delivery
Causal Order reliable multicast protocol

**init:**
- \(\forall k, \text{next_seqnum}_i[k] \leftarrow 1; \) // seq. number of next messages to be applied by \(p_i\)
- \(\text{msg\_set}_i \leftarrow \emptyset;\) // set of messages waiting to be delivered

**when CO-multicast (m) is executed by \(p_i\)‘s application:**
- \(m.\text{sender} \leftarrow i; \quad m.\text{seqnum} \leftarrow \text{next_seqnum}_i;\)
- R-multicast (m);
- **CO-deliver (m);** // immediate delivery of \(p_i\)‘s message
- \(\text{next_seqnum}_i[i] \leftarrow \text{next_seqnum}_i[i] + 1.\)

**when R-deliver (m) is executed by \(p_i\)‘s system:**
- let \(j = m.\text{sender};\)
- if \((i \neq j)\) then
  - \(\text{msg\_set}_i \leftarrow \text{msg\_set}_i \cup \{m\};\)
  - while \((\exists m' \in \text{msg\_set}_i : \text{CausalCondition}(m'))\) do
    - let \(j = m'.\text{sender};\)
    - **CO-deliver(m’);**
    - \(\text{next_seqnum}_i[j] \leftarrow m'.\text{seqnum}[j] + 1;\)
    - \(\text{msg\_set}_i \leftarrow \text{msg\_set}_i \setminus \{m'\}\)
  - end while
- end if.

**CausalCondition(m’) \equiv ( \forall k : \text{next_seqnum}_i[k] \geq m'.\text{seqnum}[k] )**

Process \(p_i\) pseudocode
Total order reliable Protocol architecture

Application

Total Order Multicast protocol

Reliable Multicast protocol

Reliable channels

TO-multicast(m)  \rightarrow  R-multicast(m)  \rightarrow  send(m)  \rightarrow  Application

R-multicast(m)  \rightarrow  R-deliver(m)

TO-deliver(m)  \rightarrow  TO-multicast(m)

receive(m)  \rightarrow  R-deliver(m)
Total Order Reliable Multicast protocol properties

- **Validity**: if a correct process TO-multicast a message $m$, this message $m$ must be delivered in all correct processes.

- **Integrity**: if a process TO-deliver($m$), then $m$ has been TO-multicast by some process, and a process must TO-deliver $m$ at most once.

- **Agreement**: if a correct process TO-deliver a message $m$, this message $m$ must be TO-deliver in all correct processes.

- **Total order message deliver**: If a process TO-deliver($m$) and then TO-deliver($m'$), then no process TO-deliver($m'$) unless it has previously TO-deliver($m$).
Total Order reliable multicast

Process $p_i$ pseudocode

`init:`

next_seqnum$_i$ $\leftarrow$ 1;  //next message to delivered
msg_set$_i$ $\leftarrow$ $\emptyset$;  // set of messages waiting to be delivered

`when TO-multicast (m) is executed by $p_i$‘s appl.:`

send (m) to $p_s$;  // sequencer

`when R-deliver (m) is executed by $p_i$‘s system:`

msg_set$_i$ $\leftarrow$ msg_set$_i$ $\cup$ \{m\};
while ($\exists$ m’ $\in$ msg_set$_i$ : next_seqnum$_i$ = m’.seqnum) do

TO-deliver(m’);

msg_set$_i$ $\leftarrow$ msg_set$_i$ \{m’\};
next_seqnum$_i$ $\leftarrow$ next_seqnum$_i$ + 1;
end while.

Process sequencer ($p_s$) pseudocode

`init:`

seq_globs $\leftarrow$ 1;  // global sequence number

`when receive (m) is executed by $p_i$‘s system:`

m.seqnum $\leftarrow$ seq_globs;
R-multicast (m);
seq_globs $\leftarrow$ seq_globs + 1;

NOTE: if the process sequencer crashes, then alive processes must detect it reliably and elect a new sequencer.
FIFO+Total Order reliable multicast

**Process p_i pseudocode**

**init:**

```
next_seqnum_i ← 1;  // next message to delivered
msg_set_i ← Ø;  // set of messages waiting to be delivered
```

**when FIFO+TO-multicast (m) is executed by p_i’s appl.:**

```
m.sender ← i;
m.segnum ← next_seqnum_i;
send (m) to p_s;  // sequencer
next_seqnum_i ← next_seqnum_i + 1;
```

**when R-deliver (m) is executed by p_i’s system:**

```
msg_set_i ← msg_set_i U {m};
while (∃ m’ ∈ msg_set_i:
    next_seqnum_i = m’.seqnum) do
    FIFO+TO-deliver(m’);
    msg_set_i ← msg_set_i \ {m’};
    next_seqnum_i ← next_seqnum_i + 1;
end while.
```

**Process sequencer (p_s) pseudocode**

**init:**

```
seq_glob_s ← 1;  // global sequence number
∀j, next_FIFO_s [j] ← 1;
msg_set_s ← Ø;
```

**when receive (m) is executed by p_i’s system:**

```
msg_set_s ← msg_set_s U {m};
while ((∃ m ∈ msg_set_s) and (∃ k = m.sender) :
    next_FIFO_s[k] = m.seqnum) do
    m.seqnum ← seq_glob_s;
    R-multicast (m);
    seq_glob_s ← seq_glob_s + 1;
    msg_set_s ← msg_set_s \ {m};
    next_FIFO_s[k] ← next_FIFO_s[k] + 1;
end while.
```

**NOTE:** if the process sequencer crashes, then alive processes must detect it reliably and elect a new sequencer.