Multicast Communication
(aka. group communication)

Brian Nielsen
bnielsen@cs.aau.dk
Communication modes in DS

- **Uni-cast**
  - Messages are sent from exactly one process to one process

- **Broad-cast**
  - Messages are sent from exactly one process to all processes on the network.

- **Multi-cast**
  - Messages are sent from exactly one process to several processes on the network (*named group*).

- **Any-cast**
  - Message is sent to one (e.g., “best” or “nearest”) of a set of possible receivers

- **Geo-cast:**
  - Message sent to geographically close neighbors
Example: video-conferencing

Multicast address group 224.2.0.1
Reliable Multicast

• **Bulk Data**
  – Corporate data, server cluster (eg. replication), software distribution
  – Files, large memory segments
  – Static
  – Full reliability, no real-time, one sender

• **Streaming Data**
  – Stock quotes, news, video, audio
  – Messages, a/v formats
  – Dynamic
  – Full-to-none reliability reqs, varying real-time reqs, one/few sender(s)

• **Collaborative**
  – Whiteboard interaction, multimedia conference, gaming
  – Short messages, a/v formats
  – Dynamic and/or static
  – Full-to-moderate reliability reqs, moderate real-time reqs, many senders
Middleware Systems

- **JavaGroups**: Reliable, ordered group communication for Java.
- The jGCS library provides a **generic interface for Group Communication**.
- PGM (for MSMQ), Pragmatic General Multicast. [RFC 3208](https://tools.ietf.org/html/rfc3208)
- **GROF#**: Group Oriented Framework for C#.
- The Group Communication Toolkit ([GCT](https://www.globus.org/gct/)) is a .NET version of JavaGroups

**Enterprise “Middleware”**
- Tibco:
  - Rendezvous “reliable broadcast” or multicast
  - 60-second limit, probably Nack mechanism
  - Routing daemons: subnet and wide-area
- CorbaEvent services (?)
- DCS
LAN IP Multicast

• Class D IP address
• Hardware support = 1 message is sent
Unicast to multiple receivers

Sender

128.146.199.0/24

128.146.116.0/24

Receiver

128.146.199.0/24

128.146.116.0/24

Receivers

128.146.222.0/24

128.146.226.0/24
Unicast

• With 4 receivers, sender must replicate the stream 4 times.
• Consider good quality audio/video streams are about 1.5Mb/s (a T1)
• Each additional receiver requires another 1.5Mb/s of capacity on the sender network
• Multiple duplicate streams over expensive WAN links
IP-Multicast Efficiency

- IP-multicast more **Efficient** than n sends!
  - Source transmits one stream of data for n receivers
  - Replication happens inside routers and switches
  - WAN links only need one copy of the data, not n copies.

- IP datagram multicast:
  - Hosts join/leave on a class D address
  - IGMP constructs and maintains multicast tree
IP-Multicast Failures

• HW- and IP-multicast Failure model ~ UDP
  – Omission failures
    • Delivery to none
    • Delivery to some
  – No ordering guarantees
    • Consequentive multicasts may be received in different order
      – At same receiving node
      – At different nodes

• However, ordering and reliability are required by many applications
• Reliable & Ordered multicast requires “fancy” algorithms
Replicated Bank Account

B1  100

B2  100

B3  100

Add(amount)
pct(interest)
Replicated Bank Account

B1
200
Add(100)

B2
200
Add(100)

B3
200
Add(100)
Replicated Bank Account

UNRELIABLE Multicast \(\Rightarrow\) INCONSISTENCY
Replicated Bank Account

UNORDERED Multicast $\Rightarrow$ INCONSISTENCY
Replicated Bank Account

FIFO-ORDERING

B1

Add(100)
pct(10)

B2

Add(100)
pct(10)

B3

Add(100)
pct(10)

FIFO Multicast ⇒ CONSISTENCY??
Replicated Bank Account

FIFO-ORDERING

B1
220
Add(100)
pct(10)

B2
220
Add(100)
pct(10)

B3
210
pct(10)
Add(100)

FIFO Multicast ⇒ INCONSISTENCY
Replicated Bank Account

TOTAL ORDERING

B1
Add(100) pct(10)
220

B2
Add(100) pct(10)
220

B3
Add(100) pct(10)
220

TOTAL Multicast ⇒ CONSISTENCY??
Multicast-API

- **X-multicast(g,m)**
- **X-deliver(m)**
- **X** is one of
  - B: Basic,
  - R: Reliable
  - FO: FIFO,
  - CO: Causal,
  - TO: Total
  - ...

**MULTICAST PROTOCOL**

Incoming messages (Receive)

Application (process p)

Host OS/ Protocol Stack
The Hold-back queue

“stable” messages

Message processing

Hold-back queue

Delivery queue

When delivery guarantees are met

Incoming messages
Basic Multicast

• A **basic multicast** primitive guarantees
  – *All correct process eventually delivers the message, as long as the sender (multicasting process) does not crash*
  – A “**correct**” process = a process that exhibits no failures at any execution point under consideration
  – NB: *NOT* satisfied by HW (IP) multicast

• A *straightforward way to implement B-multicast is to 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).*
B-Multicast

- If $P_n$ crashes, message not delivered in $p_4$ and $p_5$
- Hence, Unreliable
Reliable Uni-cast

- **Integrity**: A correct process $p$ delivers a message $m$ at most once. Furthermore, $m$ is unmodified and was destined for $p$.

- **Validity**: If $m$ was sent and the receiver is correct, it eventually delivers $m$. 
Reliable multicast

• **Integrity**: A correct process $p$ delivers a message $m$ at most once. Furthermore, $p \in group(m)$ and $m$ was supplied to a multicast operation by $sender(m)$.

• **Validity**: If a correct process multicasts message $m$, then it will eventually deliver $m$.

• **Agreement**: If a correct process delivers $m$, then all other correct processes in $group(m)$ will eventually deliver $m$.

• **Liveness**=$Validity+agreement
Reliable multicast

Algorithm 1 with B-multicast

On initialization

\[ \text{Received} := \{ \} ; \]

For process \( p \) to R-multicast message \( m \) to group \( g \)

\[ \text{B-multicast}(g, m) ; \quad \text{// } p \in g \text{ is included as a destination} \]

On B-deliver(\( m \)) at process \( q \) with \( g = \text{group}(m) \)

\[ \text{if } ( m \not\in \text{Received} ) \]
\[ \text{then} \]
\[ \quad \text{Received} := \text{Received} \cup \{ m \} ; \]
\[ \quad \text{if } ( q \neq p ) \text{ then B-multicast}(g, m) ; \text{ end if} \]
\[ \quad \text{R-deliver } m ; \]
\[ \text{end if} \]

Each R-multicast message is sent \(|g|\) times, ie \( O(N^2) \).
Reliable multicast

- Correct?
  - Integrity
  - Validity
  - Agreement

- Efficient?
  - NO: each message transmitted $|g|$ times
R-multicast using IP multicast

- Each process maintains sequence numbers
  - $S^p_g$ next message to be sent
  - $R^q_g$ (for all $q \in g$) latest message delivered from $q$
- On **R-multicast** of $m$ to group $g$, attach $S^p_g$ and all pairs $<q, R^q_g>$
- **R-deliver** in process $q$ happens iff $S_m = R^p_g + 1$
  - if $S_m < R^p_g + 1$, process $q$ has seen the message before,
  - if $S_m > R^p_g + 1$ or if $R_m > R^p_g$ for some pair $<q, R_m>$ in message a message has been lost
R-multicast using IP multicast

Data structures at process p:

- \( S^p_g \) : sending sequence number
- \( R^q_g \) : sequence number of the latest msg p delivered from q (for each q)

On initialization:

\[ S^p_g = 0, \quad R^q_g = -1, \text{ for all } q \in g \]

For process p to R-multicast message m to group g:

- IP-multicast \((g, \langle m, S^p_g, \langle R_g \rangle \rangle)\)
- \( S^p_g ++ \)

On IP-deliver \((<m, S, \langle R \rangle>)\) at q from p

(continued)
R-multicast using IP multicast

On IP-deliver (<m, S, <<R>>>) at q from p

save m

if S = R_g^p + 1
then
   R-deliver (m)
   R_g^p ++
   check hold-back queue

else
   if S > R_g^p + 1
   then store m in hold-back queue
       request missing messages
   endif
endif

if ∃p. r_g^p ∈ R and r_g^p > R_g^p
then
   request missing messages
endif
R-multicast using IP multicast

- 3 processes in group: P, Q, R
- State of process:
  - S: Next sequence number
  - R_q: Already delivered from Q
  - Set of Stored messages!
- Presentation:
  
  | P: 2 | Q: 3 | R: 5 |
  | << >|
R-multicast using IP multicast

• Initial state:

\[
\begin{align*}
\text{P: } & 0 \\
\text{Q: } & -1 \quad \text{R: } -1 \\
\end{align*}
\]

\[
\begin{align*}
\text{P: } & -1 \\
\text{Q: } & 0 \quad \text{R: } -1 \\
\end{align*}
\]

\[
\begin{align*}
\text{P: } & -1 \\
\text{Q: } & -1 \quad \text{R: } 0 \\
\end{align*}
\]
R-multicast using IP multicast

- First multicast by P:

  P: 1
  Q: -1  R: -1
  <mp0>

  P: m_p0, 0, <Q:-1, R:-1>

  Q: 0
  P: -1  R: -1
  <>

  R: 0
  P: -1  Q: -1
  <>
R-multicast using IP multicast

• Arrival multicast by P at Q:

\[ P: 1 \quad Q: -1 \quad R: -1 \quad < m_{p0} > \]

\[ P: m_{p0}, 0, <Q:-1, R:-1> \]

\[ Q: 0 \quad P: 0 \quad R: -1 \quad < m_{p0} > \]

\[ R: 0 \quad P: -1 \quad Q: -1 \quad <> \]
R-multicast using IP multicast

• New state:

\[ \begin{align*}
\text{P:} & \quad 1 \\
\text{Q:} & \quad -1 \quad \text{R:} \quad -1 \\
\langle m_{p0} \rangle &
\end{align*} \]

\[ \begin{align*}
\text{Q:} & \quad 0 \\
\text{P:} & \quad 0 \quad \text{R:} \quad -1 \\
\langle m_{p0} \rangle &
\end{align*} \]

\[ \begin{align*}
\text{R:} & \quad 0 \\
\text{P:} & \quad -1 \quad \text{Q:} \quad -1 \\
\langle \rangle &
\end{align*} \]
R-multicast using IP multicast

- Multicast by Q:

Q: \( m_{q0}, 0, <P: 0, R:-1> \)

P: 1
Q: -1  R: -1
< \( m_{p0} \) >

Q: 1
P: 0  R: -1
< \( m_{p0}, m_{q0} \) >

R: 0
P: -1  Q: -1
< >
R-multicast using IP multicast

• Arrival of multicast by Q:

- Q: \( m_{q_0}, 0, \langle P:0, R:-1 \rangle \)
- P: 1
  - Q: 0
  - R: -1
  - \( \langle m_{p_0}, m_{q_0} \rangle \)
- Q: 1
  - P: 0
  - R: -1
  - \( \langle m_{p_0}, m_{q_0} \rangle \)
- R: 0
  - P: -1
  - Q: 0
  - \( \langle m_{q_0} \rangle \)
R-multicast using IP multicast

• R detects missing message!
• When to delete stored messages?

\[ P: 1 \quad Q: 0 \quad R: -1 \]
\[ \langle m_{p0}, m_{q0} \rangle \]

\[ Q: 1 \quad P: 0 \quad R: -1 \]
\[ \langle m_{p0}, m_{q0} \rangle \]

\[ R: 0 \quad P: -1 \quad Q: 0 \]
\[ \langle m_{q0} \rangle \]
R-multicast using IP multicast

- **Correct?**
  - **Integrity:**
    - seq numbers (duplicate detection) + checksums in IP multicast
  - **Validity:**
    - Self delivery assumed for IP
  - **Agreement:**
    - if missing messages are detected
    - $\Rightarrow$ Correct processes *multicasts indefinitely*
    - if *copy of message* remains available

- IMPROVE IT!
Ordered multicast

- **FIFO ordering**
  - If a process multicasts message \( m \) and subsequently multicasts message \( m' \), every process will deliver \( m \) before \( m' \).
Ordered multicast

- **Total** ordering
  - If a process delivers message m before it delivers m’, then any other process will also deliver m before m’
Ordered multicast

- **Causal** ordering

If multicast( m ) “happens-before” multicast( m’ ), all processes will deliver m before m’

The **happened before** relation (→) causally relates two events.

- m1 → m2 Process P2 multicast m2 after it received message m1.
- m1 → m3 Process P0 multicast m3 after it multicast message m1.
- m2 ↳ m3 Process P0 multicast m3 **concurrently** with P2 multicasting m2.
FIFO multicast

• Analyse our algorithm for reliable multicast on top of IP-multicast.

• A process q delivers all messages from p in p sending order \((S^p_g)\) by comparing to local expected sequence number \(R^p_g\).
(Unreliable) TO-multicast

• Basic approach as FIFO:
  – Uses globally unique IDs instead of per process unique IDs (as FIFO)
  – Receiver: deliver as for FIFO ordering

• Alg. 1: use a (single) sequencer process
• Alg. 2: participants collectively agree on the assignment of sequence numbers
**TO-multicast: sequencer**

1. Algorithm for group member \( p \)

   **On initialization:** \( r_g := 0; \)

   **To TO-multicast message \( m \) to group \( g \)**

   \[ B\text{-multicast}(g \cup \{\text{sequencer}(g)\}, <m, i>); \]

   **On B deliver(\(<m, i>\)) with \( g = \text{group}(m) \)**

   Place \(<m, i>\) in hold-back queue;

   **On B-deliver(\(<\text{“order”}, i, S>\)) with \( g = \text{group}(m) \)**

   wait until \(<m, i>\) in hold-back queue and \( S = r_g + 1; \)

   **TO-deliver \( m; \) // (after deleting it from the hold-back queue)**

   \( r_g = S; \)

2. Algorithm for sequencer of \( g \)

   **On initialization:** \( s_g := 0; \)

   **On B-deliver(<m, i>) with g = group(m)**

   \[ B\text{-multicast}(g, <\text{“order”}, i, s_g>); \]

   \( s_g := s_g + 1; \)

---

\( r_g: \) seq nr of last delivered message

\( s_g: \) global unique seq nr

i: Unique message id
(Unreliable) TO-multicast: ISIS

- Approach:
  - Sender:
    - B-multicasts message
  - Receivers:
    - Propose sequence numbers to sender
  - Sender:
    - uses returned sequence numbers to generate agreed sequence number
The ISIS algorithm for total ordering

- **P₂**
  - 1 Message
  - 2 Proposed Seq
  - 3 Agreed Seq

- **P₃**
  - 1 Message
  - 2 Proposed Seq
  - 3 Agreed Seq

- **P₁**
  - 1 Message
  - 2 Proposed Seq

- **P₄**
  - 1 Message
  - 2 Proposed Seq

Diagram shows the communication between nodes P₂, P₃, P₁, and P₄.
The ISIS algorithm

• Process q maintains sequence numbers
  – $A^q_g$, the largest agreed seq nr q has observed for g
  – $P^q_g$, q’s own largest proposed sequence number q

• Process p performs $B$-multicast($<m,i>,g$), where $i$ as a unique identifier for message $m$.

• Each process q replies p with a proposed sequence number $P^q_g := \max(A^q_g, P^q_g) + 1$.

• Process p collects proposed sequence numbers and chooses the largest, let’s call it $a$. Then p performs $B$-multicast($<i,a>,g$).

• Each process q in g sets $A^q_g := \max(A^q_g, a)$ and attach sequence number $a$ to message $m$
TO-multicast: ISIS alg.

• Correct?
  – Processes will agree on sequence number for a message
  – Sequence numbers are monotonically increasing
  – No process can prematurely deliver a message

• Performance
  – 3 serial messages!
CO-multicast

• Each process \( p_i \) maintains vector clock
  
  - \( V_g^i[j] \) is the number of messages from each process \( P_j \) that happened-before next message to be multicast

• To **CO-multicast**\((m)\): \( P_i \) increments \( V_g^i[i] \) and \( B\)-multicasts\((g, < V_g^i,m>)\)

• \( P_i \) **CO-delivers**\((m)\) from \( P_j \) iff
  
  a) It has delivered any earlier message send by \( P_j \)
     \( V_g^j[j] = V_g^i[j] + 1 \), and
  
  b) It has delivered any message that \( P_j \) had delivered at the time it multicast the message:
     \( V_g^j[k] \leq V_g^i[k] + 1, k \neq j \)

**E.g.** message: \( V^2 = [3,6,2] \) Receiver \( V^3 = [2,5,2] \)

  I.e \( p_3 \) needs to deliver a message from \( p_1 \) first
Summary

• So you thought multi-cast was simple??!!

• Applications have different semantic ordering, reliability and cost requirements
  – Unreliable / reliable multicast
  – FiFo, Causal, Causal-Fifo, Total, …
  – FiFo+Total (Exercise)

• Many algorithms available with different cost / ordering tradeoff

• Did you see an algorithm for totally ordered reliable multicasting ????
END