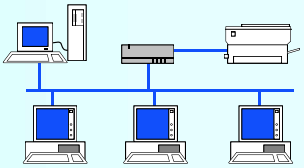


Distributed Systems
601.417
Multicast & Group Communication Services

Department of Computer Science
The Johns Hopkins University

Multicast & Group Communication Services

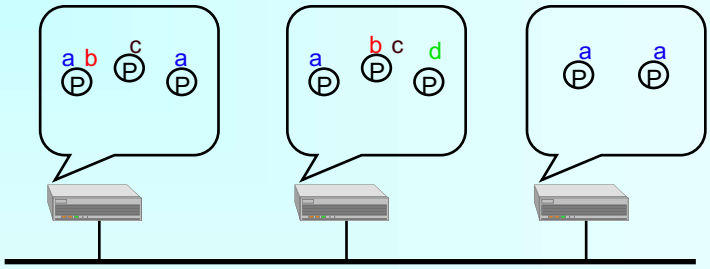
Lecture 3

Course recommended books and a survey paper at
http://www.dsn.jhu.edu/courses/cs417/ref.html

Accelerated Ring paper at
http://dsn.jhu.edu/papers/icdcs2016_AcceleratedRing.pdf

IP Multicast is documented in IETF RFC's and Internet-Drafts
which can be found at: http://www.ietf.org/

The Multicast Paradigm



- Ordering (Unordered, FIFO, Causal, Agreed)
- Delivery guarantees (Unreliable, Reliable, Safe/Stable)
- Open groups versus close groups
- Failure model (Omission, Fail-stop, Crash & Recovery, Network Partitions)
- Multiple groups

Using Traditional Transport Protocols for Multicast

Point to point (TCP/IP)

- Automatic flow control ✓
- Reliable delivery ✓
- Connection service ✓
- Complexity (n^2)
- Linear (?) degradation in performance

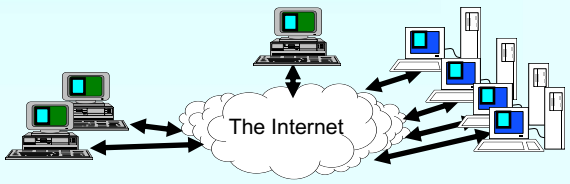
Using Traditional Transport Protocols for Multicast (cont.)

Unreliable broadcast/multicast
(UDP, IP-Multicast)

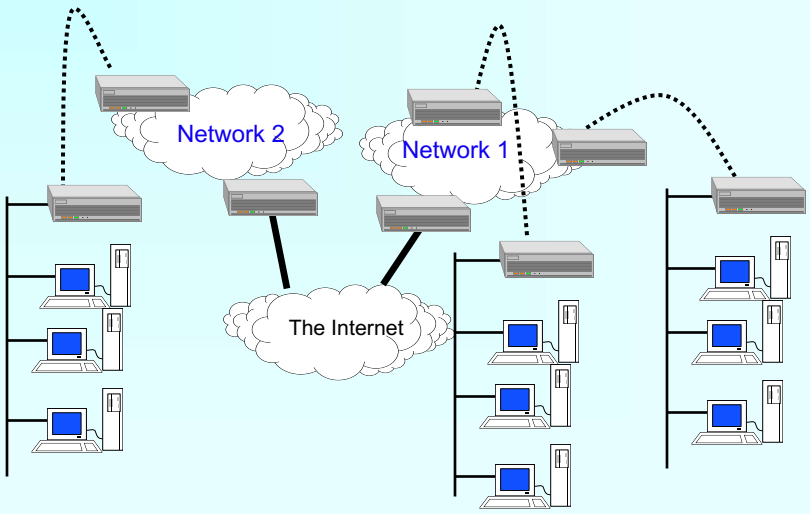
- Employs hardware support for broadcast and multicast ✓
- Message losses : 0.01% at normal load, 10%, 20%, 30% or more at high load
  - Buffers overflow (in the network and in the OS)
  - Interrupt misses
- Not a connection-oriented service

IP Multicast

- Multicast extension to IP
- Best effort multicast service
- No accurate membership
- Class D addresses are reserved for multicast: 224.0.0.0 to 239.255.255.255 and are used as group addresses
- The standard defines how hardware Ethernet multicast addresses can be used if these are possible



IP-Multicast Logical Design



IP Multicast (cont.)

Extensions to IP inside a host:

- A host may send IP multicast by using a multicast address as the destination address
- A host manages a table of groups and local application processes that belong to this group
- When a multicast message arrives at the host, it delivers copies of it to all of the local processes that belong to that group
- A host acts as a member of a group only if it has at least one active process that joined that group

IP Multicast Group Management

Extensions to IP within one local area network

The Internet Group Management Protocol (IGMP)

- A host that joins a group transmits a report message to IP multicast address 224.0.0.1 (all hosts group)
- A multicast router sends periodic general query messages to discover IP multicast groups with local hosts to 224.0.0.1
- A host replies after setting a random timer for each group it is a member of
  - The host sends a report message for that group only if no other host replied by the random timer expiration

IP Multicast Group Management

Extensions to IP within one local area network

The Internet Group Management Protocol (IGMP)

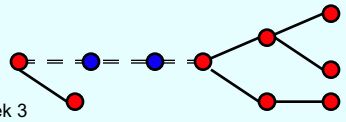
- When the host that replied last leaves the group, it sends a Leave Group message on IP multicast address 224.0.0.2 (all routers group)
- The multicast router then sends a group specific query to check whether there are additional members in the group
- After a timeout with no positive host responses for a certain group, the IP Multicast router stops participating in that group (beyond the local area network)

IP-Multicast Routing

Extensions to IP between routers in one network

Protocol Independent Multicast (PIM-SM, PIM-DM, ...)

- Messages *ABOUT* groups are sent on the special *all hosts* group 224.0.0.1
- Time to live: limits the distance messages travel
- Sparse Mode: A unidirectional shared tree toward a rendezvous point (RP) router. Source-based trees optimization for high rate flows is possible
- Dense Mode: Flood & Prune. All routers get packets initially, then prune out parts of the network that do not have group member hosts
- Tunneling: encapsulates multicast packets in regular packets in order to pass through routers that do not support IP Multicast

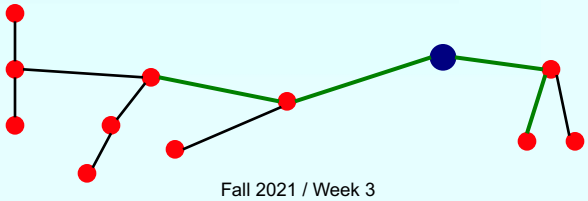


IP-Multicast Routing (cont.)

IP Multicast between routers in one network

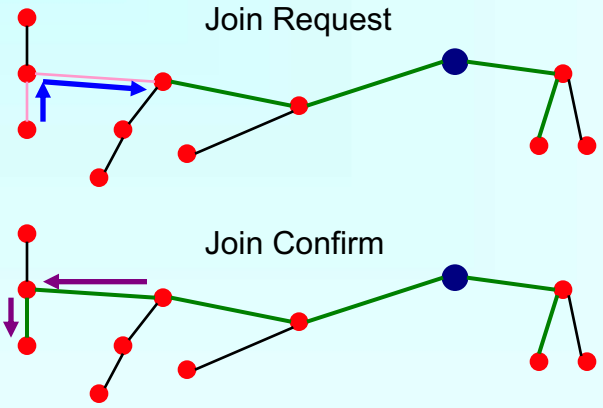
PIM-SM

- Sparse Mode for better scalability
  - only routers that participate, or are on the way to routers that participate, get IP multicast messages
  - In contrast to Dense Method that employs Flood and Prune
- Utilizes rendezvous points for each group
  - Rendezvous point router is determined via hashing the group address into a list of possible RP routers in the network (maintained by a bootstrap router)



IP-Multicast Routing(cont)

PIM-SM (Sparse Mode) Join Operation



IP Multicast Challenges

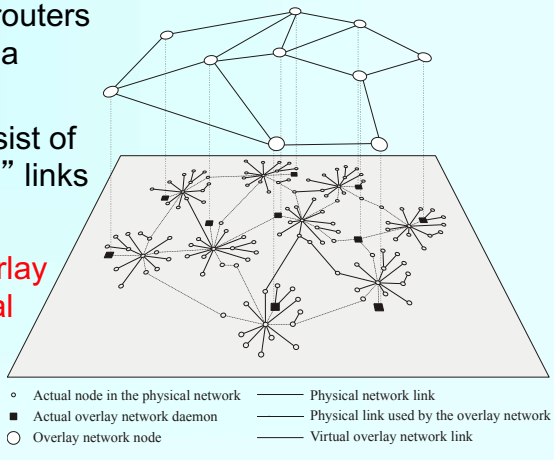
- Scalability with the number of applications / groups
  - How many groups are needed on a world-wide basis?
  - What happens to the core routers with many global groups?
- Turned off by ISPs
  - Can you think why?
- What can be done about that?

IP Multicast Challenges

- Scalability with the number of applications / groups
  - How many groups are needed on a world-wide basis?
  - What happens to the core routers with many global groups?
- Turned off by ISPs
  - Can you think why?
- What can be done about that?
  - Private networks – using IP multicast – e.g. for IPTV
  - Overlay networks – using unicast

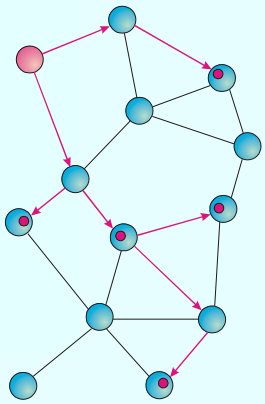
The Overlay Networks Approach

- Application-level routers working on top of a physical network
- Overlay links consist of multiple "physical" links
- Incurs overhead
- Placement of overlay routers not optimal
- **Flexible use of peer-protocols**
- **Provides added value**



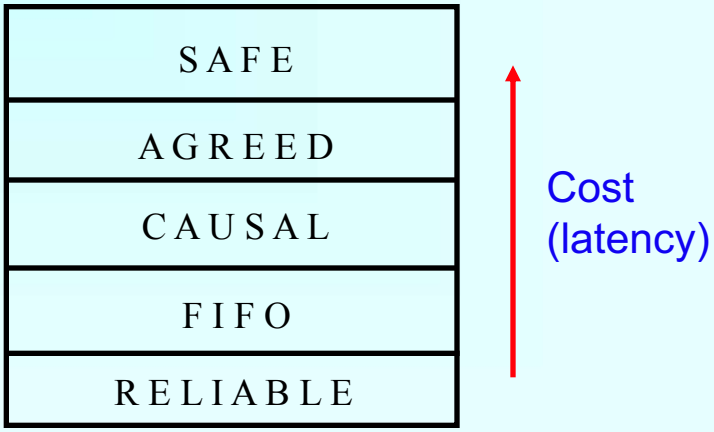
Multicast Using Overlay Networks

- Routing is not optimal. But functional and does not require state at intermediate routers – just at overlay routers.
- Multiple overlay networks can coexist in the Internet without overhead to Internet routers
- All the multicast traffic is seen as unicast packets at the network level. No need for hardware support
- Group names space extends only to the scope of the application (no longer global)



Reliable Multicast Services

Service-Type



Reliable Multicast Services (cont.)

Fifo Order

$m \overset{cause}{\dashrightarrow} m'$ if $send_q(m) \dashrightarrow send_q(m')$

Causal Order

$m \overset{cause}{\dashrightarrow} m'$ if $deliver_q(m) \dashrightarrow send_q(m')$

Agreed Order

- Total order
- Consistent with Causal order and overlapping groups

Safe Delivery

* Not ordering

- Consistent with Agreed order
- Message is delivered after received by all processors

Reliable Multicast Protocols

- Free-access protocols
  - Vector Timestamps
  - Direct Acyclic Graph
  - Lamport Timestamps
- Token-based protocols
  - Single Ring Protocol
  - Accelerated Ring Protocol

Vector Timestamps Protocol: Reliability and Causal Ordering

- Each process maintains a logical time vector of size n; initially VT[i] = 0
- When p sends a new message m: VT[p]++
- Each message is stamped with VTm which is the current VT of the sender
- When p delivers a message, p updates its vector: for k in 1..n:

$$VT[k] = \max\{ VT[k], VTm[k] \}$$

Vector Timestamps protocol Causal Order (Cont)

Comparing messages:

$$VT1 < VT2 \text{ iff } \text{for } k = 1..n\ VT1[k] \le VT2[k] \text{ and } \exists k\ VT1[k] < VT2[k]$$

Determining causality:

$$m1 \rightarrow m2 \text{ iff } VT1 < VT2$$

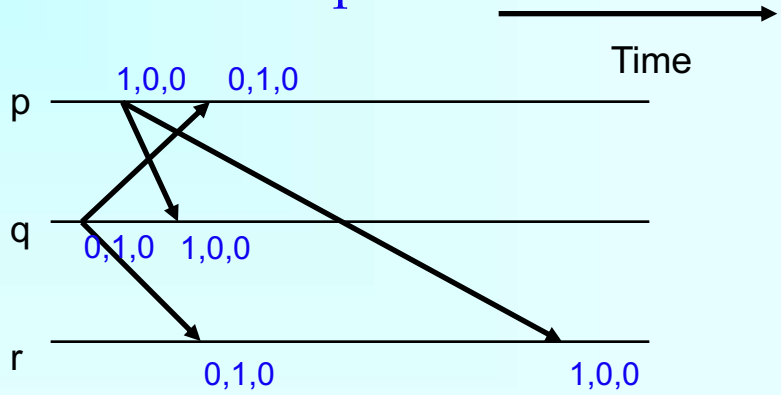
Determining whether a message sent by q can be delivered:

for any k in 1..n:

$VTm[k] = VT[k]+1$ if $k=q$.

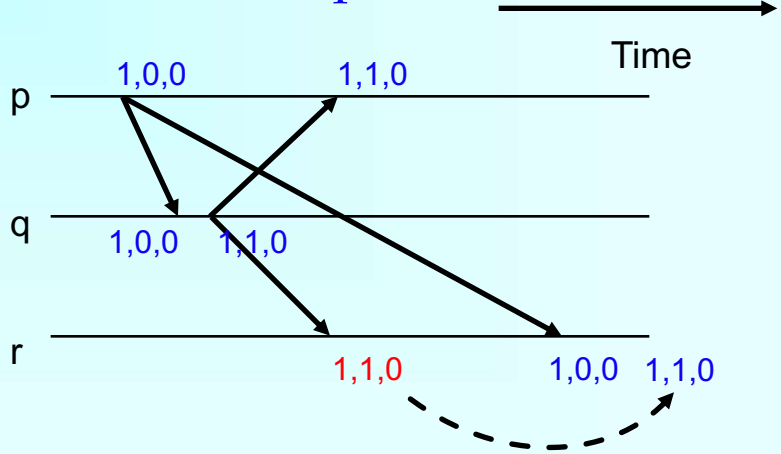
$VTm[k] \le VT[k]$ otherwise.

Example 1



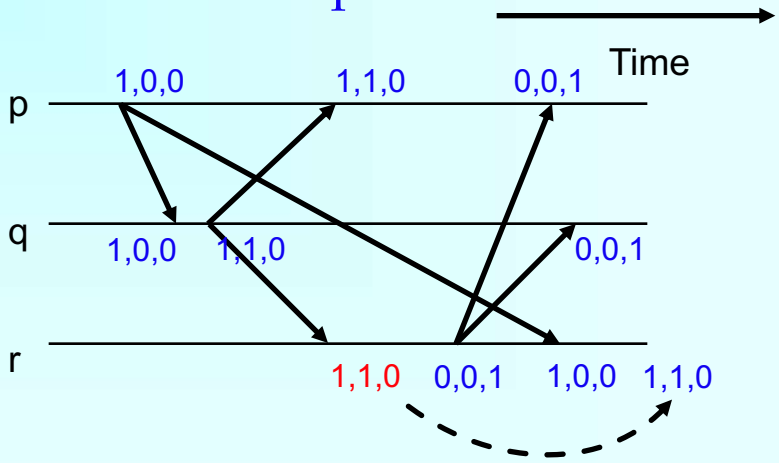
Messages {1,0,0} and {0,1,0} are not causally related, so they can be delivered in any order

Example 2



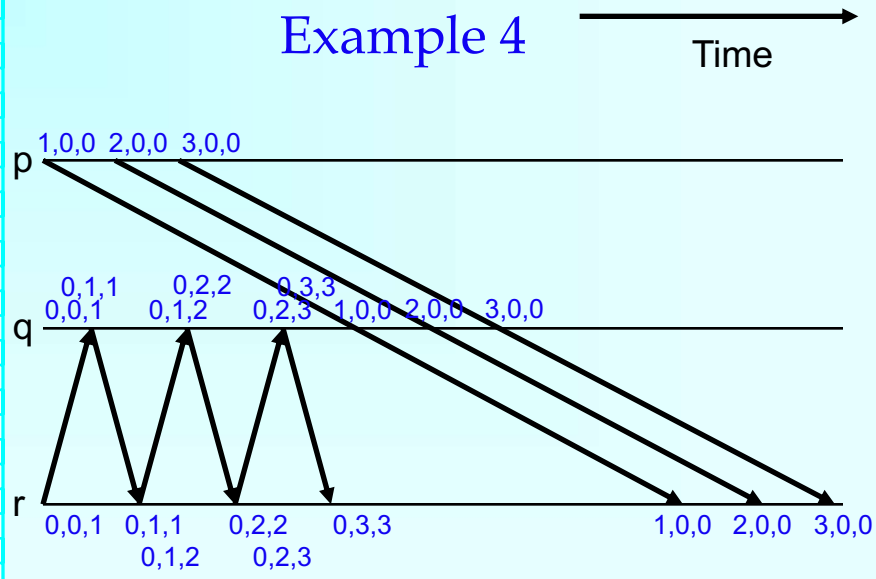
Message {1,0,0} causally precedes {1,1,0}, so {1,0,0} must be delivered before {1,1,0}

Example 3

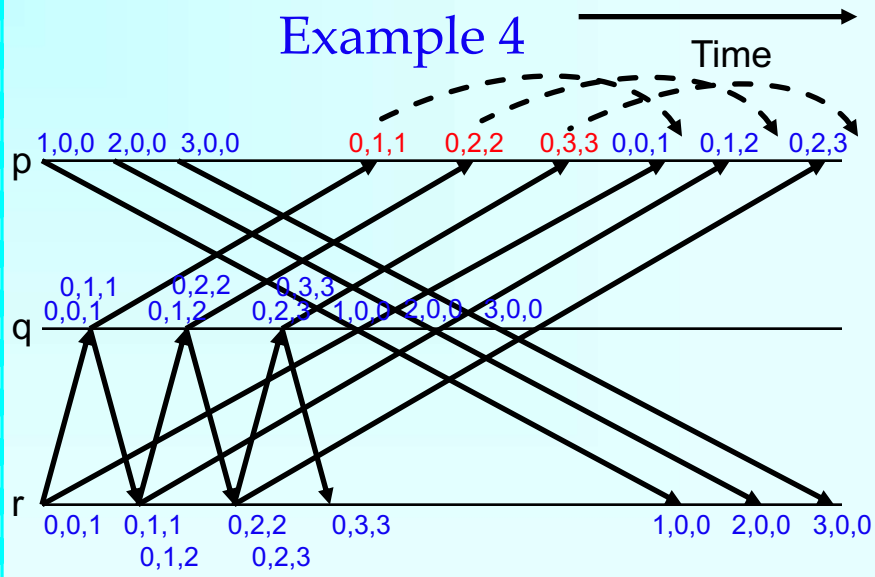


A process does not update its timestamp until it delivers a message, so r sends {0,0,1}, even though it received {1,1,0}

Example 4



Example 4



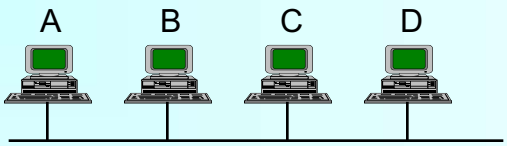
Vector Timestamps Protocol Agreed (Total) Order

- Preserves causality
- Option 1 (token-based method):
  - One process holds the token. From time to time, the token holder sends an "ordering" message for all the previous Agreed-order messages it knows that are not yet ordered
  - Non-token holders cannot deliver Causal messages that are causally after an Agreed message that is not yet ordered
  - A new token holder may be determined after a membership change
- Option 2 (all-message method):
  - A message can be agreed-ordered once there is a message (in FIFO order) from each process. At that point, causally parallel messages are ordered lexicographically

Reliable Multicast Protocols

- Free-access protocols
  - Vector Timestamps
  - Direct Acyclic Graph
  - Lamport Timestamps
- Token-based protocols
  - Single Ring Protocol
  - Accelerated Ring Protocol

Direct Acyclic Graph (DAG) Protocol



A Emits: A_1 A_2 A_3 ...

Scenario: A_1 a_1B_1 b_1B_2 b_2C_1 ...

Direct Ack: $a_1B_1 \rightarrow A_1$

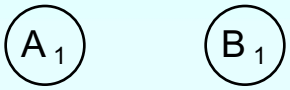
Indirect Ack: $b_2C_1 \rightarrow A_1, B_1$

Example

Scenario: A_1 B_1 $a_1 b_1 B_2$ $a_1 C_1$ $c_1 b_2 C_2 \ldots$

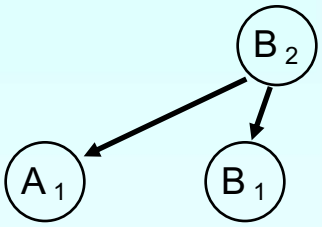
Example

Scenario: A_1 B_1 $a_1 b_1 B_2$ $a_1 C_1$ $c_1 b_2 C_2 \ldots$



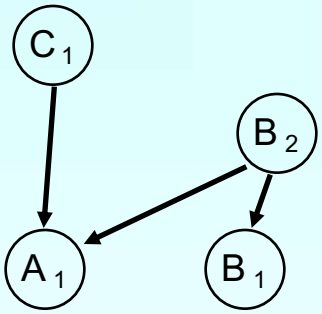
Example

Scenario: A_1 B_1 $a_1 b_1 B_2$ $a_1 C_1$ $c_1 b_2 C_2 \ldots$



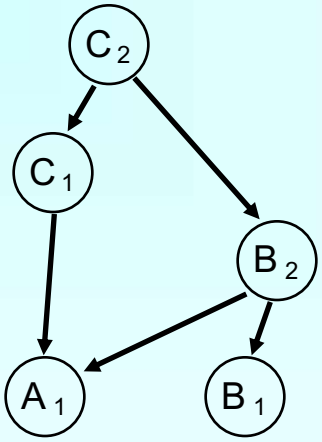
Example

Scenario: A_1 B_1 $a_1 b_1 B_2$ $a_1 C_1$ $c_1 b_2 C_2 \ldots$



Example

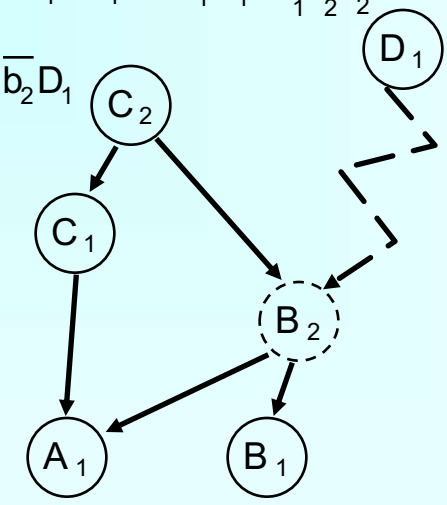
Scenario: A_1 B_1 $a_1 b_1 B_2$ $a_1 C_1$ $c_1 b_2 C_2 \ldots$



Example (Cont.)

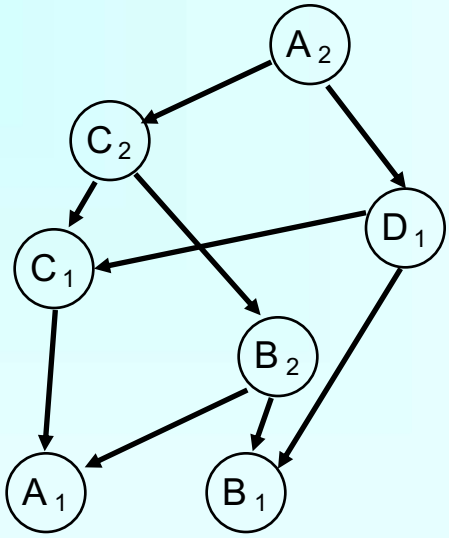
D received: A_1 B_1 $a_1 C_1$ $c_1 b_2 C_2 \ldots$

Nack:: $c_1 b_1 \overline{b_2} D_1$



The DAG

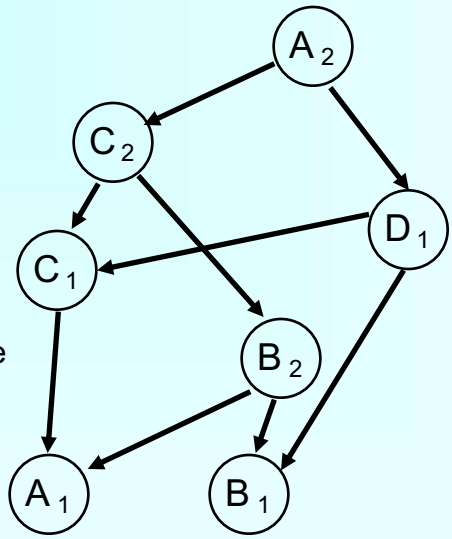
The DAG may be revealed in a different way, but its structure will be identical at all the processors!!



The DAG

The DAG may be revealed in a different way, but its structure will be identical at all the processors!!

Total order can be based on the structure of the graph, e.g., All-message method



Reliable Multicast Protocols

- Free-access protocols
  - Vector Timestamps
  - Direct Acyclic Graph
  - Lamport Timestamps
- Token-based protocols
  - Single Ring Protocol
  - Accelerated Ring Protocol

Lamport Timestamps Protocol

- A Lamport Time Stamp (LTS) contains two fields:
  - **Counter.**
  - **Process id.**
- When sending a message.
  - **Increment your counter.**
  - **Stamp your message.**
  - **Send your message.**
- When receiving a message
  - **Adopt the counter on the message if it is bigger than your local counter.**
- Unique for every message.

Lamport Timestamps Protocol

- A Lamport Time Stamp (LTS) contains two fields:
  - **Counter.**
  - **Process id.**
- When sending a message.
  - **Increment your counter.**
  - **Stamp your message.**
  - **Send your message.**
- When receiving a message
  - **Adopt the counter on the message if it is bigger than your local counter.**
- Unique for every message.
- It is useful to add an index next to the LTS, such that the index is incremented only when sending new messages.
  - The index helps track how many messages were sent by a process as well as how many were missed from that process.

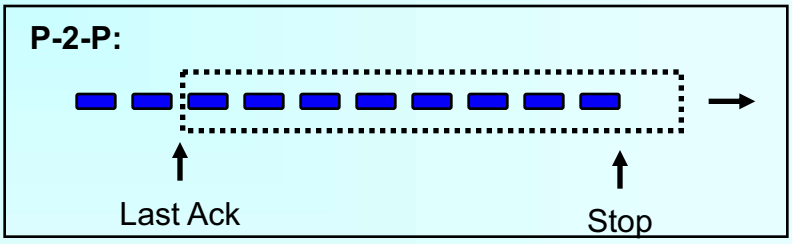
Lamport Timestamps Protocol

- A Lamport Time Stamp (LTS) contains two fields:
  - **Counter.**
  - **Process id.**
- When sending a message.
  - **Increment your counter.**
  - **Stamp your message.**
  - **Send your message.**
- When receiving a message
  - **Adopt the counter on the message if it is bigger than your local counter.**
- Unique for every message.
- It is useful to add an index next to the LTS, such that the index is incremented only when sending new messages.
  - The index helps track how many messages were sent by a process as well as how many were missed from that process.
- Agreed order of messages can be achieved using all-message method by comparing (counter, process id) of messages.
- FIFO and Causal order as a by-product.

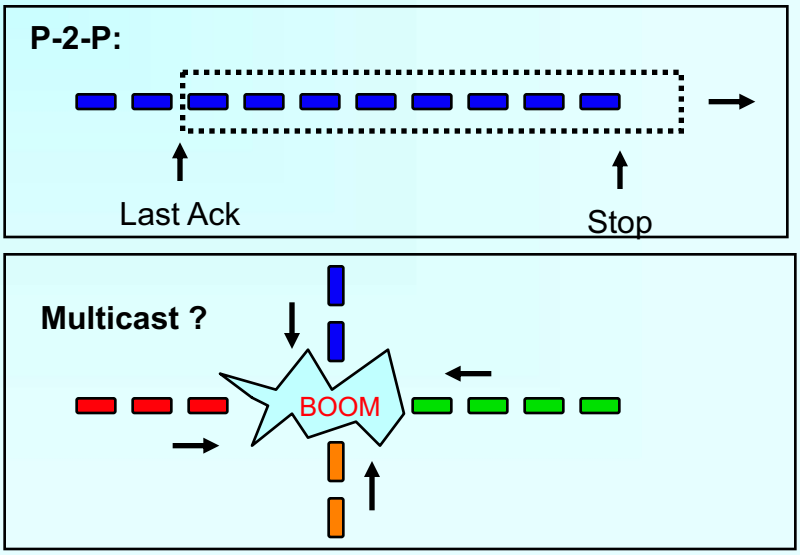
Vector Timestamps vs DAG vs Lamport Timestamps

- DAG representation is a compaction of a vector timestamps representation. Both method provides accurate causality information
- The DAG representation is more efficient network-wise compared with vector timestamps and therefore can scale better, but requires maintaining a more sophisticated data structure
- Lamport timestamps are even more compact than a DAG. The method is very simple to implement. It loses accurate causality information while still guaranteeing causality
- All protocols could implement a similar all-message method for Agreed Delivery (where a message can be agreed-ordered if there is a message (in FIFO order) from each participant

Flow Control (Free Access)

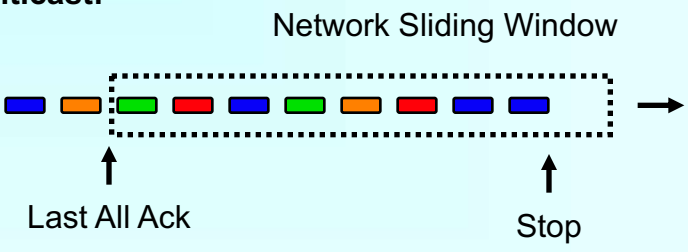


Flow Control (Free Access)



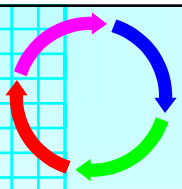
Flow Control (Free Access)

Multicast:



Reliable Multicast Protocols

- Free-access protocols
  - Vector Timestamps
  - Direct Acyclic Graph
  - Lamport Timestamps
- Token-based protocols
  - Single Ring Protocol
  - Accelerated Ring Protocol



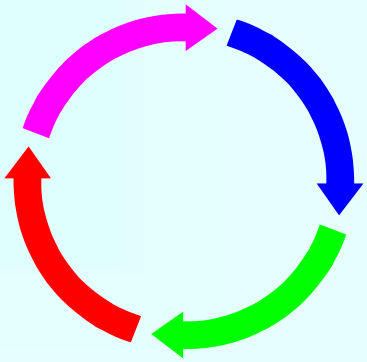
The Single Ring Protocol

- The communication is multicast (UDP/IP).
- Services: Agreed (which is also FIFO and Causal), Safe.
- supports message omissions, network partitions, crashes and recoveries.

The Single Ring Protocol (cont)

Token fields

- type - {regular, form}.
- seq - of last message.
- **aru** - replaces acks.
- rtr - retrans. requests
- fcc - flow control.



The Single Ring Protocol (cont)

How to update the token aru?

- If token.aru = token.seq and have all the messages then should raise aru together with the seq (when sending new messages).
- If the token.aru is higher than the highest in-order message (local aru), lower the token.aru to the local aru.
- If is the one that lowered the aru, and the token.aru is still the same, should set token.aru to its local aru.

The trick: Everyone has all the messages up to:

min(token.aru, previous token.aru)

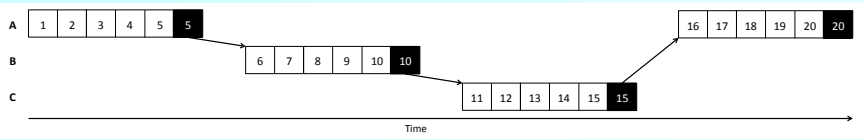
Reliable Multicast Protocols

- Free-access protocols
  - Vector Timestamps
  - Direct Acyclic Graph
  - Lamport Timestamps
- Token-based protocols
  - Single Ring Protocol
  - Accelerated Ring Protocol

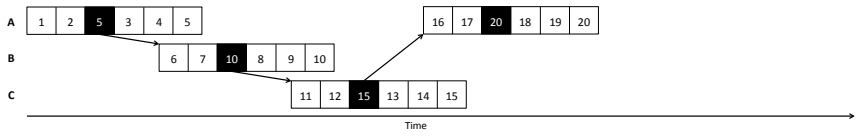
Accelerated Ring Protocol

- Original Ring Protocol
  - Token is passed around a ring of participants
  - A participant multicasts while it holds the token, then passes the token to the next participant
- Accelerated Ring Protocol
  - Participants pass the token while multicasting
  - Circulates the token faster, allowing more rounds of sending per second
  - Allows controlled parallelism, while maintaining semantics
  - Designed for modern data centers

Accelerated Ring Protocol



Original Protocol



Accelerated Protocol

Accelerated Ring Protocol

Updating token fields

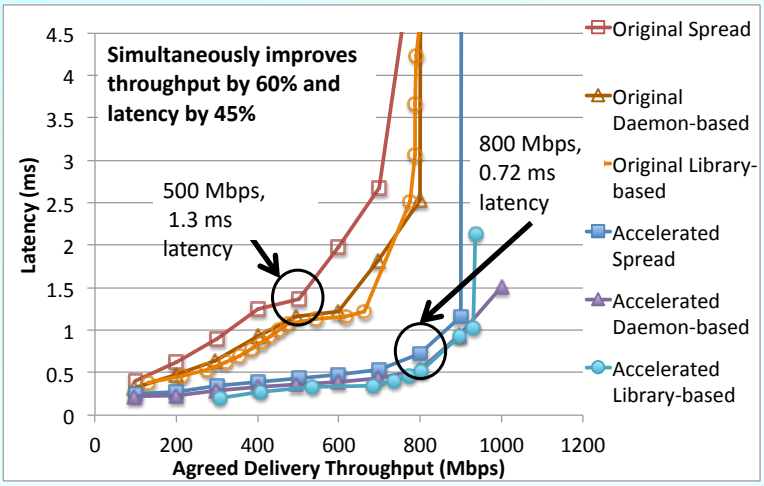
•seq

- Original: sequence number of last message sent
- Accelerated: last sequence number claimed (message will be sent by the time the next token is processed)

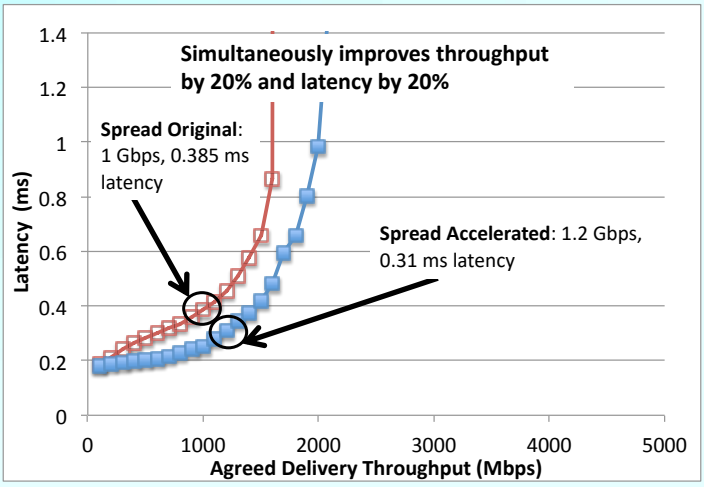
•rtr - how do you decide what to request?

- Original: request any missing messages with sequence numbers less than seq
- Accelerated: request any missing messages with sequence numbers less than the value of seq on the token received in the previous round
  - seq may reflect messages that are still on their way or even not yet sent; you don't want to request them unless they are really lost

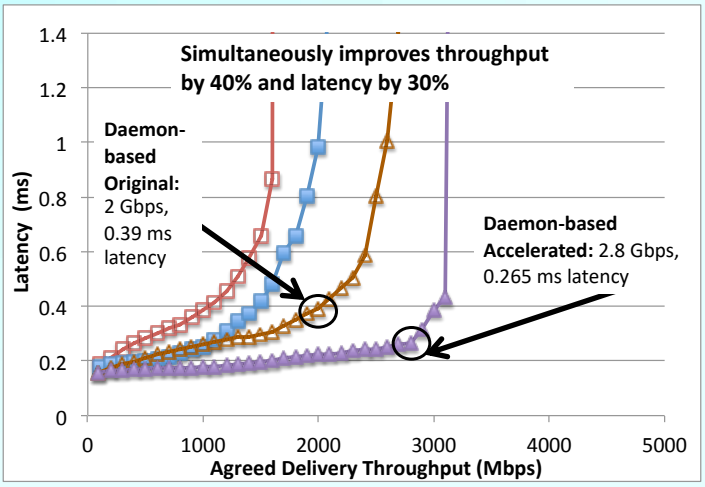
1G Network Results



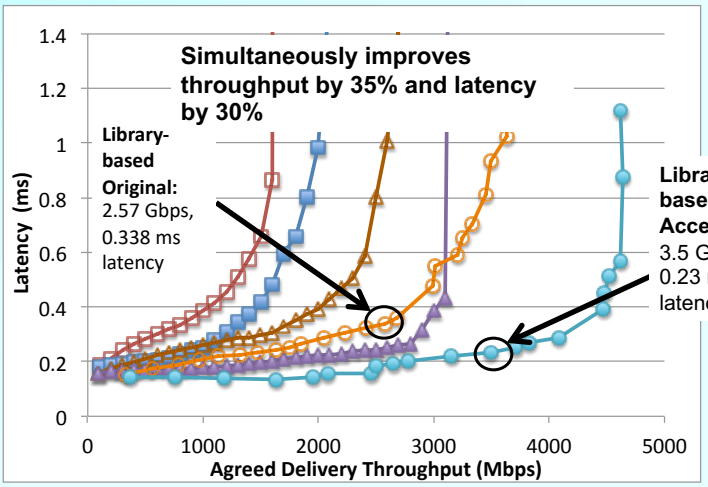
10G Network Results



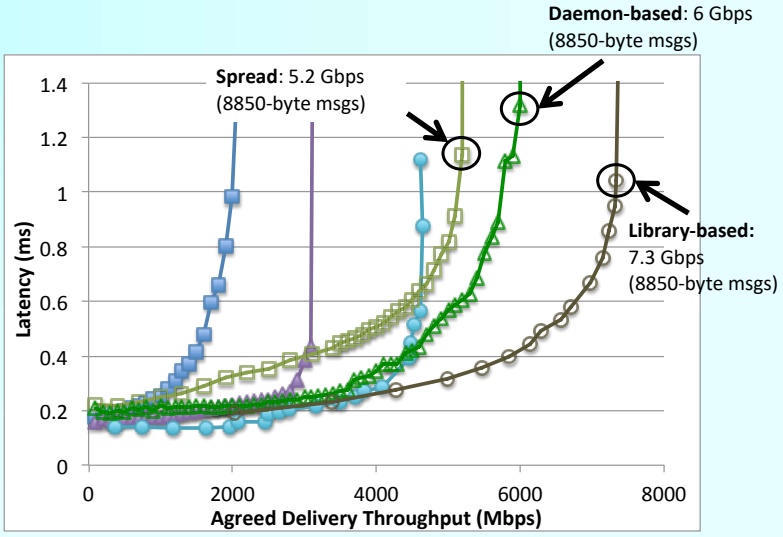
10G Network Results



10G Network Results



10G Network Results



Failure Models

Possible faults:

- Message omissions and delays
- Processor crashes and recoveries
- Network partitions and re-merges

Most of the time it is assumed that:

- Message corruption is detected
- There are no malicious faults

Free-Access Membership Protocol

- Utilizes broadcast or multicast
- Ordering and Reliability optimized by DAG
- Handles crashes and recoveries
- Handles network partitions and merges
- Terminates in a bounded time (to do that, it allows the extraction of live but "inactive" processors).
- Guarantees virtual synchrony (relationship between messages and membership events)

Free-Access Membership (cont.)

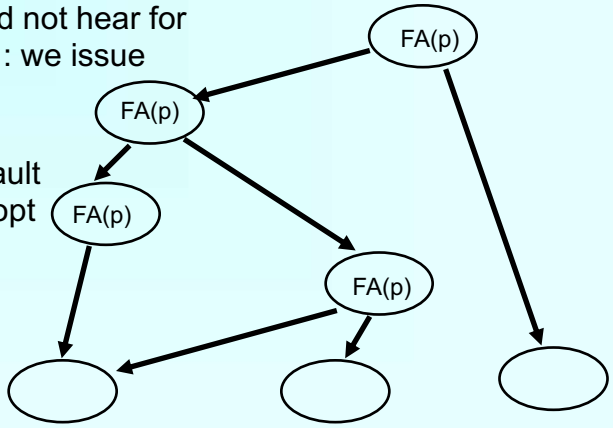
- Partitioning / crashes detection
  - Invoked by timeout
- Merging
  - Symmetric: no joining-side / accepting-side
  - Spontaneous: invoked after receiving Join messages or hearing "foreign" messages
- Faults may occur at any time (even while merging)

Faults & Partitions

When Detecting a processor from which we did not hear for a certain timeout : we issue a fault message

When we get a fault message, we adopt it (and issue our copy)

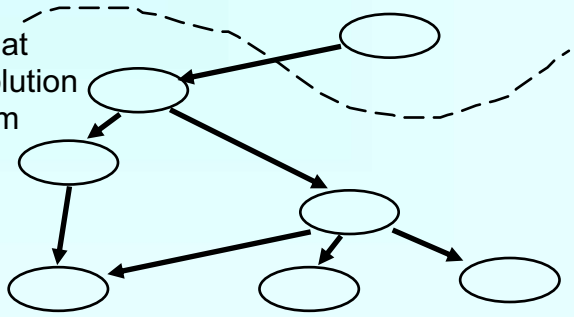
Problem: maybe p is only slow



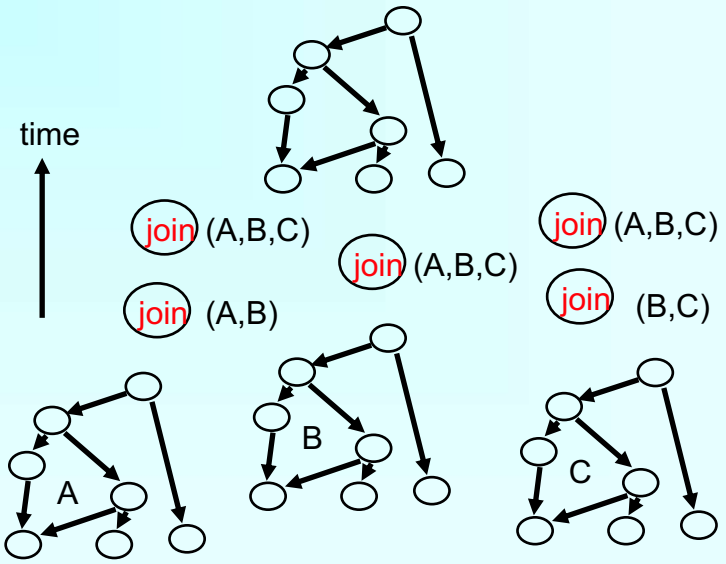
The Problem

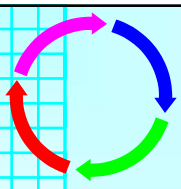
When a partition occurs, we cannot always completely determine who received which messages

It is proven that there is no solution to this problem (no common knowledge)



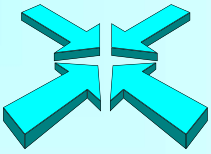
Merges





Single Ring Membership Protocol

- Membership has several stages:
  - Detect that old membership is lost
  - Gather together all alive members
  - Form a new ring and send old state
  - Transfer missing messages
  - Install new membership
- Supports message omissions, network partitions, crashes and recoveries



Membership

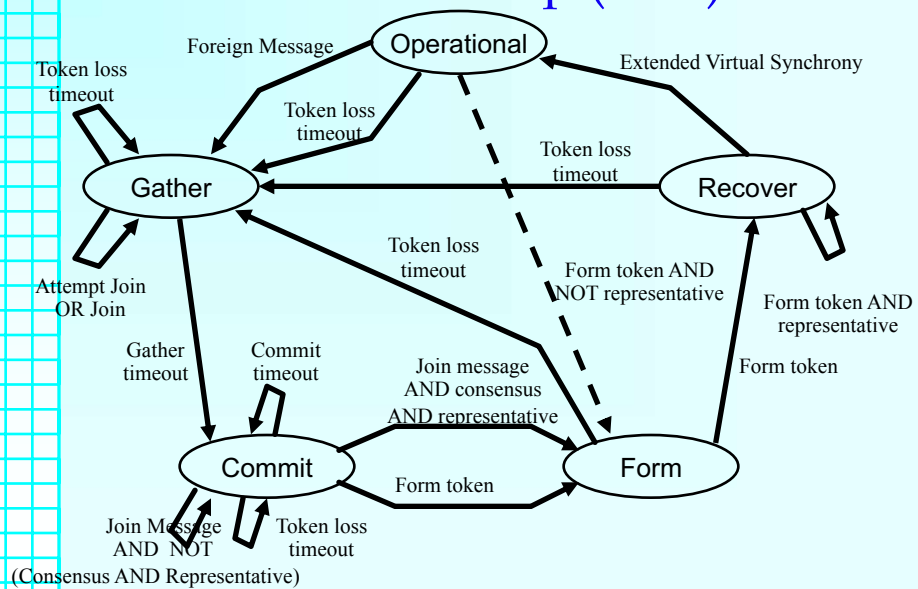
Events

- Foreign Message.
- Attempt join/ Join.
- Gather timeout.
- Commit timeout.
- Form token.
- Token loss timeout.

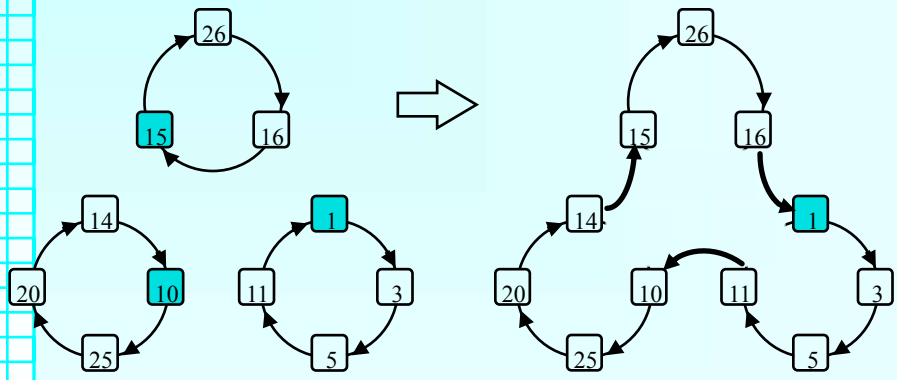
States

- Operational state.
- Gather state.
- Commit state.
- Form state.
- Recover state.

Membership (cont)



Membership (cont..)



Representatives are shown shaded