Basic vs. Reliable Multicast

Basic multicast does not consider process crashes.

Reliable multicast does.

So far, we considered the basic versions of ordered multicasts. What about the reliable versions?

Reliable causal order multicast is feasible, but reliable total order multicast is not feasible in asynchronous systems, since it can be used to solve the consensus problem (how?), and we already know that asynchronous consensus cannot be solved in presence of crash failures.
Scalable Reliable Multicast

Too many ACKs may be a bottleneck particularly when the failure rates are low.

Floyd et al suggested that we use NACKs for non-delivery that will trigger selective retransmission from those who received it. Message deleted from the local buffer after a certain period.

NACKs can be combined. One NACK sent back to the sender would trigger a retransmission to all.
View Synchronous Communication

Processes may join or leave a group. A list of the current members is called a view.

Life would have been simple if all processes had the same view at all times, but that is not easy.

**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:

\[
\{0,1,2,3\}, \{0,1,3\} \{0,3,4\}, \quad \text{OR} \\
\{0,1,2,3\}, \{0,2,3\}, \{0,3\}, \{0,3,4\}, \quad \text{OR} \\
\{0,1,2,3\}, \{0,3\}, \{0,3,4\}
\]

Views should propagate in the same order to all.
Inconsistent views

\[ V(g) = \{0,1,2,3\} \text{ for } 0 \]
\[ V(g) = \{0,1,2,3\} \text{ for } 1 \]
\[ V(g) = \{0,1,2\} \text{ for } 2 \text{ (3 left the group) } \]

How will they divide a task of sending 144 emails?
**View Propagation**

While sending messages, send the view too.

{Process 0}: $v_0(g);$ send $m_1, \ldots ;$

$v_1(g);$ send $m_2, \text{send } m_3;$

$v_2(g) ;$

{Process 1}: $v_0(g);$ send $m_4, \text{send } m_5;$

$v_1(g);$ send $m_6;$

$v_2(g) \ldots ;$
Two obvious requirements for view delivery

If a process $j$ joins and thereafter continues its membership in a group $g$ that already contains a process $i$, then eventually $j$ appears in all views delivered by process $i$.

If a process $j$ permanently leaves a group $g$ that contains a process $i$, then eventually $j$ is excluded from all views delivered by process $i$. 
**View Synchronous Communication**

**Main idea.** With respect to each message, all correct processes have the same view

**Agreement.** If a correct process $i$ delivers a message $m$ in $v_i(g)$ before delivering the next view $v_{i+1}(g)$, then every correct process $j \in v_i(g) \subseteq v_{i+1}(g)$ must deliver $m$ before delivering $v_{i+1}(g)$.

**Integrity.** If a process $j$ delivers a view $v_i(g)$, then $v_i(g)$ must include $j$.

**Validity.** If a process $k$ delivers a message $m$ in view $v_i(g)$ and another process $j \in v_i(g)$ fails to deliver that message $m$, then the next view $v_{i+1}(g)$ delivered by $k$ must exclude $j$. 
Delivery of messages in a group of changing size

An example. Process 1 sends out a message $m$ to the group, and then crashes.

\[
\begin{array}{c}
\text{process 0} \\
\text{process 1} \\
\text{process 2} \\
\text{process 3}
\end{array}
\]

$v(g) = \{0,1,2,3\}$  \hspace{1cm}  $v(g) = \{0,2,3\}$

Possibility 1. No one delivers $m$, but each delivers the new view $\{0,2,3\}$.

Possibility 2. Processes 0, 2, 3 deliver $m$ and then deliver the new view $\{0,2,3\}$

Possibility 3. Processes 2, 3 deliver $m$ and then deliver the new view $\{0,2,3\}$ but process 0 first delivers the view $\{0,2,3\}$ and then delivers $m$, as shown (Violates the agreement criterion)
Network partitions

Disconnection due to router failure may partition a group. There are different policies or guarantees.

At most one partition will survive -- others will quit.

Two or more subgroups continue to operate (consider a video-conference). Consistency is restored when they regroup.
Replication

Replication improves *reliability* and *availability*.

Replicated web servers
RAID
Disconnected modes in mobile systems
Cache memory

Replication must be transparent. No one should find out whether (s)he is communicating with the master copy or a replica.
Architecture of Replicated data management

1. Passive Replication

Client communicates with the master. When the master crashes, a new master is elected. Ideally the switchover should be atomic.

*How many backups are required to survive $m$ failures?*
The Primary-backup protocol

The Requirements

1. At most one master at any time.
2. Each server maintains the identity of the server that it communicates with. Requests are queued by the master, and processed in order.

The protocol

1. Receive request from client
2. Broadcast it to all backups (totally ordered)
3. Send response to the client

If the primary server undergoes a fail-stop failure, then the backups detect it, elect a new master, and notify the clients.
**Replica coordination**

The agreement part boils down to consensus. The implementation of order boils down to stability detection, where a request or update can be delivered to a replica only if no request (from a correct replica) with a lower timestamp can be subsequently delivered to that replica.

**Use of null requests**
**Active Replication**

Each client transparently communicates with a group of servers with no master-slave relationship among them. These servers communicate with one another to maintain consistency.

How do we guarantee that all replicas are in the same state?

*Total order broadcast* is the answer. All updates must be sent to all the replicas in the same order.
Fault-tolerant state machines (Schneider)

Captures the formal theory of active replication.

The two requirements are:

- **Agreement.** Every correct replica receives all the requests.
- **Order.** Every correct replica receives the requests in the same order.

Reduces to the consensus problem, since all machines agree to the choice of the next request to update its own state.

*No solution exists for asynchronous systems.*
Dealing with crashed Servers

Only correct servers update their states properly. Crashed ones can be voted out.

![Diagram showing state machine, replicas, voting units, and actuators]

What if the voting unit is faulty?