Multicast classes

1. **unreliable multicast**: an attempt is made to transmit the message to all members without acknowledgement; at-most-once semantics with respect to available members; message ordering is not guaranteed.

2. **reliable multicast**: the system transmits the messages according to “best-effort”, i.e. the “at-least-once” semantics is applied.
   - B-multicast primitive: guarantees that a correct process will eventually deliver the message as long as the multicaster does not crash.
   - B-deliver primitive: corresponding primitive when a message is received.

3. **serialized multicast**: consistent sequence for message delivery; distinction between
   - totally ordered
   - causally ordered (i.e. virtually synchronous)

4. **atomic multicast**: a reliable multicast which guarantees that either all operational group members receive a message, or none of them do.

5. **atomic, serialized multicast**: atomic message delivery with consistent delivery sequence
**Multicast messages** for constructing distributed systems based on group communication; different multicast communication semantics

**Multicast classes**

**Relationship between multicast classes**
Multicasting can be realized by using IP multicast which is built on top of the Internet protocol IP.

Java API provides a datagram interface to IP multicast through the class `MulticastSocket`.

---

**Introduction**
Group communication facilities the interaction between groups of processes.

**Motivation**

**Important issues**

**Conventional approaches**

Groups of components
Management of groups
Message dissemination
Message delivery

**Taxonomy of multicast**
Group communication in IS-IS

**JGroups**
Atomic broadcast supports a total ordering for message delivery, i.e. all messages to the group G are delivered to all group members of G in the same sequence.

Atomic realizes a serialized multicast

Atomic is based on a 2-phase commit protocol; message serialization is supported by a distributed algorithm and logical timestamps.

Phase 1
Sender S sends the message N with logical timestamp T_N (N) to all group members of G (e.g., by multicast).

Each g ∈ G determines a new logical timestamp T_g (N) for the received message N and returns it to S.

Phase 2
S determines a new logical timestamp for N; it is derived from all proposed timestamps T_g (N) of the group members g.

T_{S,new} (N) = \max (T_g (N)) + |G|, with j being a unique identifier of sender S.

S sends a commit to all g ∈ G with T_{S,new} (N).

Each g ∈ G delivers the message according to the logical timestamp to its associated application process.

Let n be the number of group members of G. Each g ∈ G has a unique number of \{1, \ldots, n\} and a state vector z which stores information about the received group messages.

The state vector represents a vector clock.

Each message N of sender S has a unique number; message numbers are linearly ordered with increasing numbers.

Let j be a group member of the group G.

the state vector z_j = (z_{j1}, \ldots, z_{jn}) specifies the number of messages received in sequence from group member i.

Example: z_{ij} = k; k is the number of the last message sent by member i ∈ G and received in correct sequence by the group member j.

at group initialization all state vectors are reset (all components are 0).

Sending a message N, j ∈ G sends a message to all other group members.

z_j := z_j + 1; the current state vector is appended to N and sent to all group members.

Receiving a message N sent by member i ∈ G.

Message N contains state vector z_i. There are two conditions for delivery of N to the application process of j.

\text{(C 1): } z_j = z_i + 1.

\text{(C 2): } \forall k \neq i: z_k \leq z_i.