**Exploring Broadcast Primitives: Atomic, Causal, and Ordered Multicast**

**Author: Iyiola Oluwaferanmi Peter**

**ID:UGR0202110333**

**. Introduction**

Broadcast primitives are fundamental communication operations in distributed systems where a single process sends a message to all other processes. These primitives ensure efficient and reliable message dissemination. This document explores three key types:

- Atomic Multicast

- Causal Multicast

- Ordered Multicast

## \*\*2. Atomic Multicast\*\*

### \*\*Definition\*\*

Ensures a message is delivered to \*\*all\*\* processes in a group \*\*or none at all\*\*, even if failures occur.

### \*\*Key Properties\*\*

- \*\*All-or-Nothing Delivery\*\*: Either all correct processes receive the message, or none do.

- \*\*Atomicity\*\*: The operation is indivisible (like a transaction).

### \*\*Types

1. \*\*Uniform Atomic Multicast\*\*: Guarantees atomicity even for failed processes.

2. \*\*Non-Uniform Atomic Multicast\*\*: Only guarantees atomicity for correct (non-failed) processes.

### \*\*Applications\*\*

- Distributed databases (e.g., replicating transactions).

- Financial systems (e.g., ensuring atomic money transfers).

---

## \*\*3. Causal Multicast\*\*

### \*\*Definition\*\*

Ensures messages are delivered in an order that respects \*\*causal relationships\*\* (e.g., if Message A happens before Message B, all processes receive A before B).

### \*\*Key Properties\*\*

- \*\*Causal Ordering\*\*: Preserves "happens-before" relationships.

- \*\*Partial Order\*\*: Only enforces ordering for dependent messages.

### \*\*Types\*\*

- \*\*Eager Causal Multicast\*\*: Immediate propagation of dependencies.

- \*\*Lazy Causal Multicast\*\*: Delays dependency tracking for efficiency.

### \*\*Applications\*\*

- Collaborative editing tools (e.g., Google Docs).

- Event-sourcing systems.

---

## \*\*4. Ordered Multicast\*\*

### \*\*Definition\*\*

Ensures messages are delivered in a \*\*specific order\*\* (e.g., FIFO, total order).

### \*\*Types\*\*

1. \*\*FIFO (First-In-First-Out) Multicast\*\*:

- Messages from the \*same sender\* are delivered in sending order.

- Example: Chat applications preserving message sequences.

2. \*\*Total Order Multicast\*\*:

- \*All\* messages delivered in the \*same global order\* across all processes.

- Example: Blockchain transactions.

3. \*\*Causal Order Multicast\*\*:

- Combines causal and total order guarantees.

### \*\*Applications\*\*

- Message queues (e.g., Apache Kafka).

- Replicated state machines.

---

## \*\*5. Comparison of Broadcast Primitives\*\*

| \*\*Primitive\*\* | \*\*Guarantee\*\* | \*\*Use Case\*\* |

|---------------|----------------------------------|----------------------------------|

| \*\*Atomic\*\* | All-or-nothing delivery | Database replication |

| \*\*Causal\*\* | Preserves causal relationships | Collaborative apps |

| \*\*FIFO\*\* | Per-sender order | Chat systems |

| \*\*Total Order\*\* | Global order for all messages | Consensus protocols (e.g., Paxos)|

---

## \*\*6. Challenges\*\*

- \*\*Latency\*\*: Atomic protocols (e.g., Paxos) introduce delays.

- \*\*Complexity\*\*: Causal tracking requires overhead.

- \*\*Network Partitions\*\*: May block delivery until resolved.

---

## \*\*7. Conclusion\*\*

Broadcast primitives are vital for consistency and reliability in distributed systems. Choosing the right primitive depends on the application’s needs:

- \*\*Atomic\*\* for fault-tolerant systems.

- \*\*Causal\*\* for dependency-aware apps.

- \*\*Ordered\*\* for sequential processing.

---

## \*\*8. References\*\*

1. Coulouris, G., et al. (2011). \*Distributed Systems: Concepts and Design\*.

2. Lamport, L. (1978). "Time, Clocks, and the Ordering of Events." \*CACM\*.

3. Apache Kafka Documentation. \*https://kafka.apache.org/\*

---

### \*\*Formatting Tips for Word\*\*

- Use \*\*headings\*\* (H1, H2) for sections.

- Add \*\*tables\*\* for comparisons (like in Section 5).

- Insert \*\*diagrams\*\* (e.g., message flow for multicast types).

Let me know if you'd like this converted into an actual Word file (I can provide a downloadable link or template)!