Testing Environment

I have tested the middleware programs on a lab machine to ensure compatibility with the university lab environment. All five middleware instances were able to run simultaneously without conflicts, and the network program successfully displayed communication between them.

Running Instructions

Follow these steps to run the program application:

1. Install the .NET Framework on your machine
2. Extract all files from the A1.7z archive, ensuring the directory structure is maintained
3. Open the “Developer Command Prompt for VS 2022” and navigate to the main assignment directory
4. Run the batch file by typing: “run.bat”

The batch file will:

- Compile all source files using the C# compiler (csc)
- Launch the network program (network.exe)
- Start all five middleware programs (middleware8082.exe through middleware8086.exe)

Each middleware will open in its own window with a GUI interface displaying three lists: Sent Messages, Received Messages, and Ready Messages. Initiate message sending by clicking the "Send" button on any middleware instance.

Message Identification

Each multicast message in the system is uniquely identified by combining several pieces of information:

- A sequential message number (Msg#X)
- The sender middleware identifier (from Middleware Y)
- A timestamp (at HH:mm:ss)

For example: "Msg#1 from Middleware 1 at 12:20:45"

This combination ensures that each message can be uniquely identified across the distributed system, even if multiple middleware instances send messages simultaneously. The message counter is maintained locally by each middleware instance and increments with each sent message.

Application Scenario for Total-Order Multicast

An example application scenario requiring total-order multicast is a distributed banking system with multiple branches processing transactions against the same accounts. Consider a banking application where:

- Multiple branches can process deposits and withdrawals for the same account
- Account balance must remain consistent across all branches
- Transactions must be processed in the same order at all branches to maintain consistency

For example, if a customer has $1000 in their account and simultaneously:

1. Makes a $800 withdrawal at Branch A
2. Makes a $500 withdrawal at Branch B

Without total ordering, Branch A might process the $800 withdrawal first and approve it, while Branch B also approves the $500 withdrawal, resulting in an overdraft. With total ordering, all branches would process these transactions in the same sequence (e.g., the $800 withdrawal first, then reject the $500 withdrawal), maintaining consistency across the distributed system.

The sequencer-based approach implemented in this middleware ensures that all branches see the same transaction sequence, preventing inconsistencies in account balances.

Total Order vs. Causal Order Multicast

Total order and causal order are different multicast ordering guarantees with distinct properties:

Total Order Multicast: Ensures that all messages are delivered in the same order to all recipients, regardless of when they were sent or received.

Causal Order Multicast: Ensures that messages are delivered in an order that respects their causal relationships (if event A could have caused event B, then A is delivered before B at all recipients).

Example:

Consider three middleware nodes (M1, M2, M3) and the following sequence of events:

1. M1 sends message A
2. M2 receives A and then sends message B in response
3. M3 sends message C concurrently (not in response to any message)

With **Causal Order** multicast:

- A must be delivered before B at all nodes (because B causally depends on A)
- C can be delivered in any order relative to A and B (as it's concurrent)
- Different nodes might see different orderings: [A,B,C] or [A,C,B] or [C,A,B]

With **Total Order** multicast (as implemented in our system):

- All nodes will see exactly the same sequence, e.g., [A,B,C]
- Even concurrent messages (like C) will be placed in a consistent order
- The sequencer (Middleware 1) determines this global ordering

The key difference is that causal order preserves only the "happened-before" relationships, while total order creates a global, consistent sequence for all messages, including concurrent ones. Our implementation uses a sequencer-based approach where Middleware 1 assigns sequence numbers to ensure total ordering across all middleware instances.