Isis Total Order Algorithm

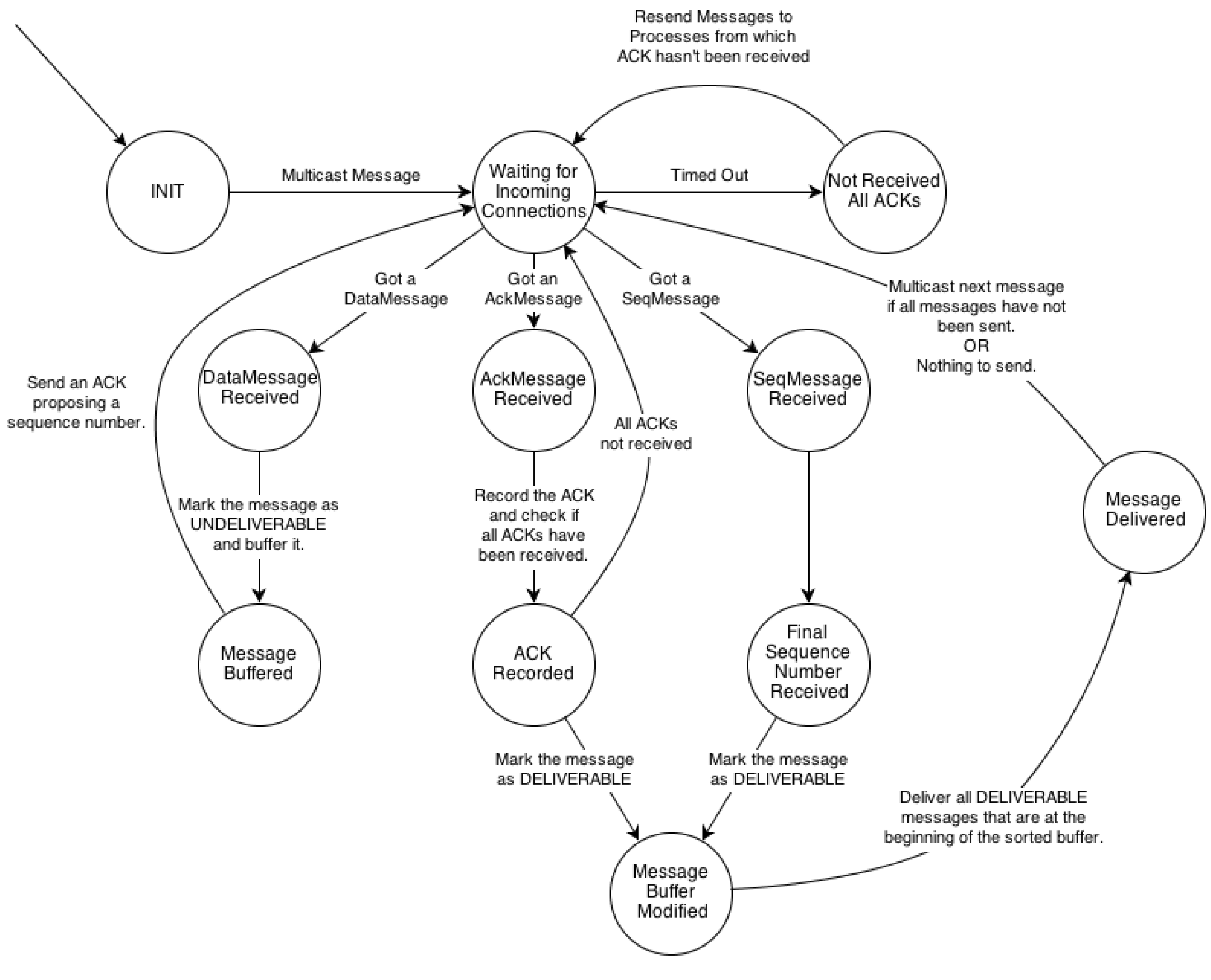
Kushal Lakhotia

March 3, 2013

Contents

1 State Diagram

The state diagram of a process in this algorithm is given below. It explains the states through which the process goes through as it deals with three different kinds of messages. Please note that there is no final state since the implementation loops in never-ending loop always listening for connections.



Resend Messages to Processes from which ACK hasn't been received
INIT
Multicast Message
Waiting for Incoming Connections
Timed Out
Not Received All ACKs
Got a DataMessage
Got an AckMessage
Got a SeqMessage
Multicast next message if all messages have not been sent. OR Nothing to send.
Send an ACK proposing a sequence number.
DataMessage Received
AckMessage Received
SeqMessage Received
All ACKs not received
Message Delivered
Mark the message as UNDELIVERABLE and buffer it.
Record the ACK and check if all ACKs have been received.
Message Buffered
ACK Recorded
Final Sequence Number Received
Mark the message as DELIVERABLE
Mark the message as DELIVERABLE
Deliver all DELIVERABLE messages that are at the beginning of the sorted buffer.
Message Buffer Modified

2 System Architecture and Design Decisions

The system has been implemented in C++ by the use of one class and a main function. The class `Isis` implements the actions which a process takes according to the algorithm. `Isis` has one public function `start()` that the `main()` method calls on the object for starting the system.

2.1 Data Structures

The main data structures that have been used to implement the algorithm are given below.

- **Message Buffer:** The message buffer has been implemented using a map with {*sequence number, process id*} as the key and {*message id, message marker*} as the value. This map always remains sorted according to *sequence number* and ties are broken using the *process id*.
- **Acknowledgment History:** The acknowledgments are stored using a map with *process id* as the key and *proposed sequence number* as the value. Since, a process collects ACKs only for one message at a time, we don't need to store an ACK for a message with its message id.
- **Message History:** All messages that been received are stored in a set. A message is represented by {*process id, message id*}. This ensures that a duplicate message can be detected.

2.2 Message Handling

Data messages and acknowledgments have been checked for duplicity. Any duplicate data message or acknowledgement will be discarded.

Messages have not been verified for correctness since malicious behavior from security point of view is not expected for this project. Thus, the implementation does not deal with the situation when a field of any message is mangled and assumes that every field is correct.

2.3 Implementation Overview

A process multicasts a message, records the time and then waits for incoming messages. As and when messages arrive, appropriate message handlers are called, depending on their type, which take necessary actions inline with the state diagram in section 1. The `revfrom()` call is non-blocking and, after every such call and message handling of data received (if any), the difference from the last recorded time is calculated. If this difference exceeds the acknowledgment timeout then the message is retransmitted to all processes from which an ACK has not been received and the time is recorded. This goes on forever till the process is killed.

A process delivers a message when either a final sequence number is generated by a process for a message it had multicast or, when process receives a final sequence number from another process.