Mohamed el Hadiyen   
Mustafa Yücesan

Solutions Unreliable network

Computer Networks

For some of the solutions we use custom headers which we include in the message. We have the following headers:

* A: Acknowledgement.
* N: Name. Was supposed to be used for error correction, which we decided not to do after all.
* M: Message. Lets the socket know to process incoming characters as a part of message,
* I: Initialize. Used to let the receiver know a window buffer must be made for the sender.
* P: Just a new packet from an existing sender, with the sequence number and window number.

To ensure that the header symbols the sender sends himself do not get confused with headers, we add the escape character /.

# Solution 1: Message Drop

To hide the message drops from the user, for each message we send, we do the following:

The sender socket sends the message. It then includes the message in the **retransmission\_list**. This list is being iterated in a thread to retransmit messages which have **not** been acknowledged yet within a fixed time slot, which is 0.05 seconds.

The receiver receives the message, checks whether it is a message packet or acknowledgement. If it is an acknowledgement, it stops retransmitting that specific message, which gets identified by **window number** and **sequence number**. If it is a message packet, it sends back an acknowledgement and processes the packet. If the packet is a duplicate, the receiver discards it, but keeps sending an acknowledgement back since it can drop at the sender too. The sender can then stop sending that specific message to the receiver.

# Solution 2: Message Delay

Delay causes the messages to be out of order. To fix this, we have added two dictionaries:

* Msg\_received\_list: keeps track of current window. Also keeps track of the messages sent by sender. It holds 2 window buffers in which messages can be stored.
* Msg\_send\_list: Keeps track of window and sequence number

What basically happens is as follows: lets say the sender’s messages get mixed up. If the incoming sequence number is **not** the expected sequence number, it just gets added into the right buffer, depending on window category. If the incoming sequence number is the expected sequence number, then it takes out every message that it has saved in that buffer, orders it based on sequence number and then checks whether there is a difference between two consecutive indexes that is higher than 1, which is checked with the difference between the sequence numbers. The application will only take the part of the buffer that does not skip sequence numbers inbetween. These messages will be sent in order after.

The expected sequence number then gets incremented to the highest sequence number possible ( highest sequence number in the part of the buffer we took + 1). If the new expected sequence number happens to be higher than 7, we know that this buffer has finished so we clear the window and switch to the other buffer. The inactivity of the lists are being checked every 10 minutes and get removed if they are idle.

We use two window buffers because it may happen that a message from another set of order, gets sent to the receiver with sequence number 0. To not mix this up with the previous set of order, as it may still be expecting sequence number 6 or 7, and thus get confused in order if it includes the new set of order, we separate two windows and keep switching between them.