Network Protocol Specification

We have implemented **rdt3.0 Go-Back-N**(which is a **pipelined** protocol), with a slight modification. This protocol can handle corrupt packets as well as lost packets and reliably transfer data over a channel.

The sender side uses two separate threads to handle data packets and acknowledgements separately. Hence, while sending the data packets, the sender does not have to wait for the ACKs to be received, as it is running on an independent thread. Also, semaphores are used so that the threads don't try to update the sender window simultaneously.

The thread handling the data packets, maintains a **window of in-flight packets.** This window can have up to N consecutive unacked packets.

This protocol uses **Cumulative ACKs** and a timer is maintained for the oldest unacked packet.

As soon as an ACK is received for one of the packets in the sliding window, the base of the window is set as the next packet, and the window is hence shifted.

If the packet received is not the same as the expected packet, then the receiver sends a reACK for the previous packet, so that the sender is notified, and the sender retransmits the correct packet. The same is implemented on the sender side for the ACKs. This notifies the receiver side to resend ACKs.

Timeout

The timer starts as soon as the packet is sent. This timer is for the oldest unacked packet. If there is a timeout, all packets with a sequence number higher than that of the oldest unacked packet, are retransmitted.

The value of the Timeout changes **dynamically** to accommodate cases where the delay, i.e, the Round Trip Time(RTT) is greater than the value of the Timeout. If the Timeout value is set to be less than the RTT, then packets can never be delivered without the Timeout being exhausted. So, they will keep getting retransmitted. Hence, to avoid this scenario, the Timeout value is updated dynamically by setting it equal to the average of all the previous RTTs.

Connection Timeout

Our application is designed such that the receiver keeps listening till the sender begins to transmit. But if the sender/receiver goes down in the middle of the transmission, then the receiver/sender checks if there is a Connection Timeout.

In case the sender/receiver side stops responding, there should be a limit to the time the receiver/sender waits for a packet or an ACK. If there is **no response** from the other end until a specified time, the connection should be closed, notifying the sender/receiver that the connection has timed out.

Error Detection

The protocol implemented here uses a **CheckSum** value, to see if there is an error in the packet or the ACK being received. Only if the checksum of the received segment is equal to the checksum field value (expected checksum), the ACK or the next packet is sent. Otherwise, the packet is resent, or the acknowledgement is reACKed.

Closing Connection:

The number of packets to be expected by the receiver is sent in the first packet.

1. **Receiver Side**:

If a packet with sequence number M is received, it means the last packet is received. It marks Success to be true and acknowledges it.

If SYN packet is received, timer is turned on, sends SYN ACK and waits for FIN till timeout.

2. **Sender Side:**

If the ACK=M+1 is received, it means all packets from 0 to M reached the receiver. So mark the transfer status success to TRUE and send SYN. Start the timer. If status says transfer success is TRUE then it terminates till then it waits for SYN ACK.

If ACK=M+2 is received, it means ACK for SYN is also received. It simply terminates.

FIN = packet with seq no. M+2

SYN = packet with seq no. M+1

SYN-ACK = Packet with Ack value = M+2

M = no. Of data packets of the file

Here SYN, SYN-ACK, FIN are used for convenience.

Submitted by:

Sumanasa Somu (2017A7PS0114H)

Akshat Jain (2017A7PS0205H)

Prathma Chowksey (2017A7PS0059H)

JSNS Rahul (2017A7PS0262H)

Simran Malik (2017A7PS1631H)