M.Uday Theja
CS14B044

Lab6 - Go-Back-N (GBN)

README File:

GO-BACK-N (GBN) RELIABLE RETRANSMISSION PROTOCOL - UDP

Compilation : make

Run Server (Receiver) : java ReceiverGBN -p <PORT> -n <MAX_PACKETS> -e <RANDOM DROP PROB> -w <WINDOW SIZE> -d(optional - Debug Mode)

Run Client (Sender) : java SenderGBN -s <HOST> -p <PORT> -l <PACKET LENGTH> -r <PACKET GEN RATE> -n <MAX PACKETS> -w <WINDOW SIZE> -b <MAX BUFFER SIZE> -d(optional - Debug Mode)

Summary of Command Line Options: The command line options provided to the sender are listed below:

- -d – Turn ON Debug Mode (OFF if -d flag not present)
- -s string – Receiver Name or IP address.
- -p integer – Receiver's Port Number
- -l integer – PACKET LENGTH, in bytes
- -r integer – PACKET GEN RATE, in packets per second
- -n integer – MAX PACKETS
- -w integer – WINDOW SIZE
- -b integer – MAX BUFFER SIZE

Summary of Command Line Options: The command line options provided to the receiver are listed below:

- -d – Turn ON Debug Mode (OFF if -d flag not present)
- -p integer – Receiver's Port Number
- -n integer – MAX PACKETS
- -e float – Packet Error Rate (RANDOM DROP PROB)

The Sender terminates after:
• MAX PACKETS (a command-line parameter) have been successfully ACKNOWLEDGED
(OR)
• If the maximum retransmission attempts for any sequence number exceeds 5.

The Receiver terminates after:
• Acknowledging MAX PACKETS (a command-line parameter).

COMMENTS File:

COMMENTS:

- This is a Sliding Window Go-Back-N(GBN) reliable transmission protocol implementation.
- The Sender initially send a Window of Packets to the Receiver
- The Receiver send an ACK packet if the received packet is the expected packet
- For every ACK received, the Window slides by one unit, sending the next packet
- In case a packet is dropped, Timeout occurs and a new Window from last Unacknowledged packet is sent

ISSUES:

- Timeouts for few packets are less than 1 millisecond, Java socket timeout rounds it to 0. For such cases I am setting timeout to 1 millisecond.
- Had no prior experience with MultiThreading, wasted a lot of time for thread synchronisation.

SUGGESTIONS:

- If possible, give a small MultiThreading assignment as In Lab before giving this Assignment.

OBSERVATIONS:

Observations have been recorded with variables set to :

- MAX_PACKETS = 400
- WINDOW_SIZE=3

- MAX_BUFFER_SIZE=10
- PACKET_GEN_RATE=10

Packet Error Rate : 0.0000001	
Packet Length : 128 bytes	Retransmission Ratio : 1.022500 Average RTT : 0.344727 milliseconds
Packet Length : 1024 bytes	Retransmission Ratio : 1.025000 Average RTT : 0.382018 milliseconds

Packet Error Rate : 0.000100	
Packet Length : 128 bytes	Retransmission Ratio : 1.041000 Average RTT : 0.333363 milliseconds
Packet Length : 1024 bytes	Retransmission Ratio : 1.035000 Average RTT : 0.341096 milliseconds

INFERENCE & CONCLUSION:

- Average RTT value depends on Packet Length.
- As Packet Length increases, Average RTT must also increase.
- Retransmission Ratio depends on Packet Error Rate.
- As Packet Error Rate increases, more number of packets are dropped and so more retransmissions occur.

DESIGN :

Sender :

The Sender has three modules:

- Packet Generation & Filling Buffer
- Sending Data Packets
- Receiving Ack Packets

Packet Generation :

We use a Timer and TimerTask module of Java to schedule the event of periodically generating packets and pushing them to buffer according to Packet_Generation_Rate and Max_Buffer_Size. The TimerTask actually creates a new thread which sleeps according to Packet_Generation_Rate.

Sending Data Packets :

We initially send a Window of data packets (wait till buffer is not empty) to the receiver. Then listen for ACKs from the receiver. Whenever we receive an ACK, we remove the corresponding packet from the buffer and send the next packet (i.e the Window slides by one unit). In case a Timeout occurs, we resend a Window of packets from the first unacknowledged data packet. While sending packets, we store the time at which the packet was sent to implement timeout as well as calculating RTT value.

Receiving Ack Packets :

We create Listener Thread for listening or receiving ACK packets from the receiver.
Timeout is implemented as Socket Timeout. Socket Timeout is the remaining original timeout time after the duration of sending the packet and calling the receive function.
Whenever we receive an ACK, we calculate RTT value for the corresponding packet, clear that packet from the buffer and send the next packet available.

Case 1 : ACK packet arrives correctly before timeout
Action : Data packet cleared from buffer, next available packet sent i.e window slides by 1 unit

Case 2 : DATA packet is dropped or corrupted
Action : Timeout occurs, new window of packets from first unacknowledged is retransmitted.

Case 3 : ACK packet arrives late, after the timeout

Action : Before that ACK packet was received, a Timeout would have occurred and a window of packets would have been retransmitted. When this ACK is received, normal transmissions resume.

Receiver :

Receiver drops a packet if the packet is corrupted. It maintains a variable to know the Expected Packet arriving. For every packet arrived, it checks if the arrived packet has the same sequence number as the Expected Packet variable and a Corresponding Cumulative Ack is sent to the sender. ACK y means that all the packets till y are received and it is expecting packet y next.
If the received packet doesn't match with the expected packet, then the receiver ignores it and no ACK is sent to the sender.

Case 1 : DATA packet is corrupted
Action : Ignore, No ACK is sent

Case 2 : DATA packet is not corrupted and matches the Expected packet
Action : Corresponding ACK is sent and Expected packet variable is updated

Case 3 : DATA packet is not corrupted and does not match the Expected packet
Action : Ignore, No ACK is sent