CS3205 Computer Networks Assignment 4 Report

(Jan - May 2021 - Prof. Siva Ram Murthy)

Assignment 4: **Go Back N and Selective Repeat Protocol**

Name: **Rushabh Nirdosh Lalwani**

Roll No: **CS18B046**

Index

Aim

The objective of this assignment is to implement the Go Back N ARQ protocol and Selective Repeat reliable transmission protocol and measure the average round trip delays varying the packet size and packet generation rate.

Introduction

Go Back N ARQ

This is a data link layer protocol similar to the Stop & Wait protocol which uses the sliding window method. The packets in the sender window are sent continuously and use cumulative acknowledgments to slide the window.
The frames in the window are numbered sequentially called the sequence number of the packet. If n bits are allotted for the sequence number, the maximum window size for the sender can be 2^n - 1.

Selective Repeat Protocol

This protocol is modified Go Back N where the client-side also maintains the sliding window. It uses selective acknowledgment and the sender only retransmits the non-acknowledged packets from the window.
The frames in the window are numbered sequentially. To avoid inconsistency, the maximum window size can be 2^(n-1) at both sender and client-side.

I have implemented both the protocols and calculated the average RTT (round trip delay/time) required for the acknowledgment of the packets. The input details can be found in the README document. Below is the basic explanation of the implementation. For specific method details, check the code.

Implementation

UDP packets are used to set the communication and study both the protocol. The implementation of both protocols has a lot of similarities. I have created a packet class and buffer class common to both the protocols and they share most of the methods in it.

Packet and Buffer Structure

The two classes in *packet.h* are used as the underlying structure for buffer and individual packet in the sender code for both the protocols. They have internal locking and ensure no deadlock and runtime inconsistent memory issues.
The packet structure stores the sequence number, time of creation, the latest time it got sent from the server, the message stored in the packet, and total transmissions of the packet. The acknowledgment time is also stored and used in debugging, calculating RTT, and updating the timeout.
The buffer structure stores the dynamic array (vector) of packet structures. It is used to add new packets and delete the acknowledged packets (according to sliding window protocol). There is a mutex used to ensure memory inconsistency and avoid the index out-of-bound exception.

Sender

There are 3 threads used to concurrently perform the sender-side implementation.

1. Random packets are generated and added to the buffer if the buffer is not full frequently.
2. The receiver thread independently receives the acknowledgment packet from the client-side and it invokes the corresponding method to acknowledge the given packet according to the protocol.
3. The main thread keeps sending the unacknowledged packet from the window after shifting the window first in a loop.

For GBN, the timeout is set to 100msec while for SRQ, the timeout is set to 300msec.

Client

The client receives the packets from the server. It randomly disregards the packet with the given error rate without acknowledging it. In case, it doesn't disregard (mostly), it extracts out the sequence number.
For SRQ, it checks if this sequence number corresponds to one waiting in the window. It acknowledges if not done and sends the acknowledgment packet. The assignment instruction asks to assume that the acknowledgment packet gets successfully sent every time. But in my implementation, I am sending the ACK packet despite been acknowledged previously to avoid termination of the algorithm. If the first packet of the window gets acknowledged, I shift the window (as per the sliding window protocol).
For GBN, only the specific sequence number is expected and others are ignored. On being successfully received, the ACK packet is sent which is assumed to be a cumulative acknowledgment at the sender-side.

Observations

The implementation is run by varying the packet generation rate, the maximum size of the packet, and the error rate at the client side. I have ignored error rate = 10^-7 as that gives the same result as 10^-5 because no re-transmissions occur in both cases.

SRQ

Packet Gen Rate	Max packet size	Error rate	Avg RTT (msec)
20	256	0.01	4.05
		0.001	1.10
		0.00001	0.73

	1500	0.01	39.3
		0.001	7.57
		0.00001	0.96
300	256	0.01	42.7
		0.001	8.81
		0.00001	2.90
	1500	0.01	8.52
		0.001	3.80
		0.00001	5.77

GBN

Packet Gen Rate	Packet size	Error rate	Avg RTT
20	256	0.01	11.2
		0.001	7.5
		0.00001	4.3
	1500	0.01	3.43
		0.001	1.40
		0.00001	0.90
300	256	0.01	35.7
		0.001	13.5
		0.00001	2.60
	1500	0.01	11.91
		0.001	8.32
		0.00001	4.50

Conclusions

As the packet size is increased, the RTT also increases naturally. SRP requires lesser retransmission because only non-acknowledged packets are re-transmission, unlike GBN where the whole window is sent. The change in RTT is not much in both protocols. SRP has better RTT as the network traffic doesn't get bulkier and the trip occurs fast.
The effect of the error rate is significant. As we increase it, the re-transmissions increase. The experiment fails (re-transmission more than 10) if the error rate becomes in the magnitude of 0.1.

Summary

This experiment helped me understand both protocols better. I also experienced the impact of a lot of print statements on networking of UDP packets. I got to practice the use of mutexes and threading in CPP. Also, I understood and successfully debugged the segmentation faults I got because of index array out-of-bound access and string to const char* conversion.