Selective Repeat and Go-Back-N protocols

Report for Assignment 4 as part of the *CS3205 : Introduction to Computer Networks* course

Shreesha G Bhat (CS18B103)
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

I. Aim

The aim of the assignment is to implement the Go-Back-N and Selective Repeat Protocols and observe the effect of the various parameters involved.

II. Introduction

Go-Back-N and Selective Repeat are sliding window protocols used for reliable data transfer over a link. Both the protocols are sliding window protocols which make use of a pipelined methodology where multiple packets can be sent one after the other without waiting for acknowledgements. While Go-Back-N resends the entire window of sent but unacknowledged packets in case of a packet loss, Selective Repeat selectively re-transmits only those packets which were delivered erroneously or dropped by the network. In this assignment, we implement both Go-Back-N and Selective Repeat protocols. The sender and receiver are processes in the same machine which communicate via UDP sockets.

III. Implementation Details

A. Go-Back-N

The sender and receiver are processes communicating over a UDP socket. The sender and receiver parameters are taken via the command line arguments as described in the problem statement [2].

The sender sets up the required state variables and data structures to run the protocol such as the timers, various parameters such as window size, buffer size etc. The main thread also sets up the UDP socket connection to the receiver and then the process branches into 3 threads

1) `packet_generator` *thread:* The packet generator thread periodically generates packets based on the command line parameter for packet generation rate and stores them into the sender buffer, if the buffer has enough space. This is an abstraction of the upper layer in the protocol stack which would generate data packets at periodic intervals and deliver them to the layer which implements the Go-Back-N protocol. It also maintains a record of when the packet was generated

2) `sender` *thread:* The sender thread performs two actions. Periodically check whether the earliest unacknowledged packet has timed out (timeout value as provided in the problem description [2]) and if so, resends all those packets in the window which are unacknowledged but already sent and restarts their timers. The second action is to send packets based on the availability in the buffer. It checks if there is room to send packets in the current window, if so, checks the buffer for packets available and sends them to the receiver process.

3) `main` *thread:* The main thread continuously performs receive actions, whenever an acknowledgement is received, the main thread removes that packet from the data structure which contains the unacknowledged but sent packets and performs all necessary RTT computations based on the timer values.

As the receiver for Go-Back-N has exactly 1 window, it is relatively simple to implement. Initially, a UDP server is setup. Then, it maintains a random distribution generator which is used to decide if every arriving packet is dropped or accepted (based on a probability value provided in the problem statement [2]). In case the packet is dropped, no action is performed. If the packet is not dropped, it checks if the sequence number is the same as the expected sequence number in which case the acknowledgement is sent back (note we assume acks are not dropped) and sequence number is incremented.

B. Selective Repeat

Similar to Go-Back-N, sender and receiver are processes communicating over a UDP socket. The sender and receiver parameters are taken via the command line arguments as described in the problem statement [2].

The sender for Selective Repeat is not very different from Go-Back-N. The sender sets up the required state variables and data structures to run the protocol such as the timers, various parameters such as window size, buffer size etc. The main thread also sets up the UDP socket connection to the receiver and then the process branches into 3 threads

1) `packet_generator` *thread:* The packet generator thread periodically generates packets based on the command line parameter for packet generation rate and stores them into the sender buffer, if the buffer has enough space. In this case, the packet length can vary as given in the problem description [2] based on the command line parameter for maximum packet length. It also maintains a record of when the packet was generated for timer related outputs

2) `sender` *thread:* The sender thread performs two actions. Periodically checks whether the timer for any of the unacknowledged but sent packets has timed out (timeout value as provided in the problem description [2]) and if so, selectively resends only those packets in the window which are unacknowledged but already sent and restarts their timers. The second action is to send packets based on the availability in the buffer, this is the same as earlier. It checks if there is room to send packets in the current window, if so, checks the buffer if a packet is available and sends it to the receiver process.

3) `main` *thread:* The main thread continuously performs receive actions. Whenever an acknowledgement is received, the main thread removes that packet from the data structure which contains the unacknowledged but sent packets and performs all necessary RTT computations based on the timer values.

The Receiver for Selective Repeat is significantly more complicated than Go-Back-N as it now has to buffer out-of-order packets and print the output to the screen in-order. So, essentially, some sorting has to be performed. The implementation is as follows. After setting up the UDP server and Random Distribution as before, for every received packet that is not dropped, if the sequence number is not the same as the expected sequence number, it checks if the sequence number is within

the receiver window and if there is enough buffer space, in which case it is buffered. If the sequence number is same as the expected sequence number, after printing the relevant information for the current packet, the relevant information for all packets which where delivered out-of-order are now printed in order (upto the next packet in the receiver window which has not yet arrived).

IV. Results

The protocol was run for the following set of changing parameter values

$$\texttt{packet\_gen\_rate(r)} = \{20, 300\}$$
$$\texttt{packet\_length (l)} = \{256, 1500\}$$
$$\texttt{packet\_drop\_probability (p)} = \{10^{-3}, 10^{-5}, 10^{-7}\}$$

Parameters other than those above were set as follows

TABLE I: Parameter Values for GBN

max_packets	200
max_buffer_size	60
window_size	10

TABLE II: Parameter Values for SR

sequence_number_bits	8
max_buffer_size_sender	100
max_buffer_size_receiver	100
window_size	4

The tables obtained are as follows

TABLE III: Go-Back-N

r	l	p	Re-transmission Ratio	Average RTT (μ secs)
20	256	1e-3	1.055	311.025
20	256	1e-5	1.04	210.615
20	256	1e-7	1.045	481.87
20	1500	1e-3	1.065	296.965
20	1500	1e-5	1.025	259.915
20	1500	1e-7	1.025	633.5
300	256	1e-3	1.085	661.6
300	256	1e-5	1.055	298.735
300	256	1e-7	1.045	352.205
300	1500	1e-3	1.045	384.98
300	1500	1e-5	1.04	545.3
300	1500	1e-7	1.01	219.45

TABLE IV: Selective Repeat

r	l	p	Re-transmission Ratio	Average RTT (μ secs)
20	256	1e-3	1.015	12451.4
20	256	1e-5	1	12085.8
20	256	1e-7	1.01	14329.2
20	1500	1e-3	1.025	32097.2
20	1500	1e-5	1	11146.3
20	1500	1e-7	1	17132.7
300	256	1e-3	1.04	64989.4
300	256	1e-5	1.02	39166.1
300	256	1e-7	1.02	39567.6
300	1500	1e-3	1.02	72698.8
300	1500	1e-5	1.02	77868.6
300	1500	1e-7	1.02	57894.5

V. Observations

A. Effect of the packet_gen_rate, packet_length, packet_drop_probability on the protocols

The following observations are made based on the tables III, IV.

- We can immediately notice the efficiency in-terms of Re-transmission ratio. The selective repeat protocol, which selectively resends only those packets which were dropped or erroneously received, has a lower re-transmission ratio in general as compared to the Go-Back-N protocol which resends the entire window in case of a packet drop.
- RTT seems to be higher for Selective Repeat. This is expected and can be attributed to the increased processing time at the receivers end. As well as to the increased complexity of the protocol as a whole. It could also be affected by the implementation of the protocol, an *event-based programming* approach might lead to lesser time spent in multi-threading, acquiring the various locks etc and lead to a lower RTT in general.
- We see that apart from a few anomalies, in general, the re-transmission ratio decreases as the probability of the packet-drop decreases as expected. However, with very low probabilities like 10^{-7} and 10^{-5}, even though it is expected to be 1, the RTT comes slightly higher at $\approx 1.01/1.02$. While the reason for this is not apparent, it could possibly be attributed to timer related errors. In this implementation, the timers are checked in a linear manner for timeouts periodically. Because of this, the occasional un-intended timeouts can lead to un-intended retransmissions and increase the re-transmission ratio. In an ideal implementation, a timeout signal would be more effective and efficient.
- The effect of increasing the length of the packets is not clear. But from the tables, a slight increase in RTT is visible (but not very distinctive).
- The effect of increasing packet generation rate however is more striking. A clear increase in RTT is visible in case of Selective Repeat but not so apparent in case of Go-Back-N.

VI. Additional Experiments and Observations

A. Effect of drastically varying the packet-drop probability

For the following plots, we consider the effect of drastically varying the packet drop probabilities over a wide range

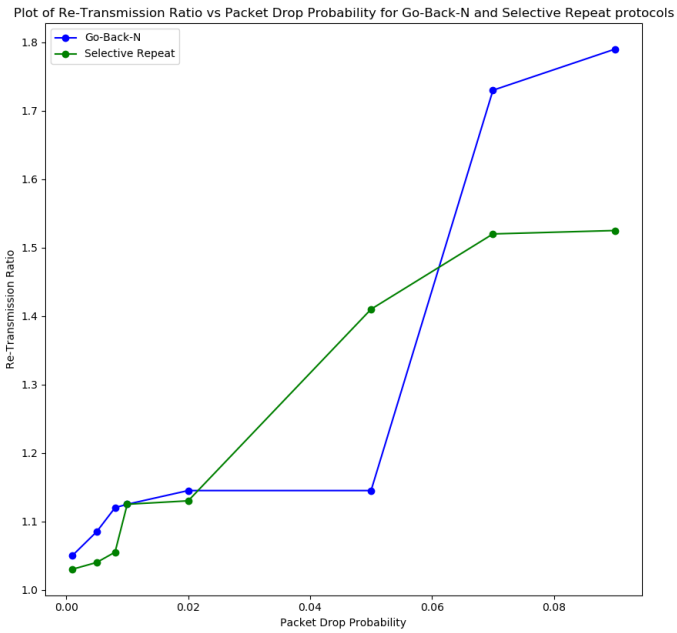

Fig. 1: Plot of Re-Transmission Ratio vs Packet Drop Probability for Go-Back-N and Selective Repeat

The drastic rise in retransmission ratio is visible as the network progressively rejects a higher and higher number of packets. After 0.1 probability, the program crashes due to possible implementation loop-holes or due to very high timeouts set on packets. On comparing Go-Back-N vs Selective Repeat, we see that although there is a rise for both protocols, Selective Repeat performs relatively better in networks which are very noisy with a high probability of packet drops.

The effect on RTT was also visible on conducting the experiment. RTT increases drastically as the network progressively gets noisier. This is due to the given rule of including the total time taken by a packet from when it was first sent to when it was finally received (after possibly many re-transmissions) as the RTT for the packet. As the network gets worser, the number of re-transmissions for a given sequence number increases and hence even the RTT.

B. Effect of Increasing Window Size in GBN

The following experiments were conducted in a relatively noisy network with 0.09 drop probability. The following results were obtained for Go-Back-N

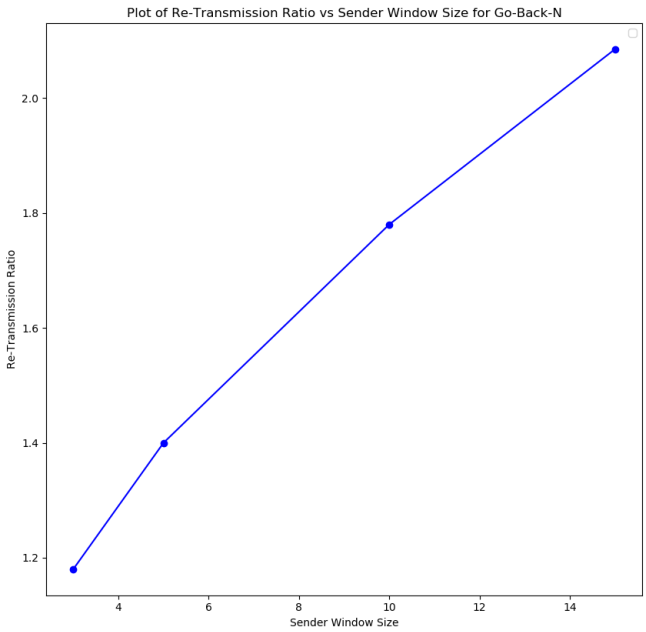

Fig. 2: Plot of Re-Transmission Ratio vs Window Size for Go-Back-N

We observe a steady increase in Re-Transmission Ratio. This is due to the fact that when a timeout occurs, the entire window (packets sent but unacknowledged) has to be resent, therefore, a larger window implies higher number of retransmissions for the same network conditions due a higher number of packets in the pipeline at any instant of time.

VII. Learning Outcomes & Conclusion

- In this assignment, we learnt about the Go-Back-N and Selective Repeat protocols for reliable data transfer.
- Coded up a simplified version of the both protocols.
- Observed the effect of the parameters involved through graph plotting and tabulation. Reasoning about the effects of the parameters was also done.

References

[1] Kurose, J. F. Ross, K. W. (2016), Computer Networking: A Top-Down Approach , Pearson , Boston, MA .
[2] A4 problem description PDF provided by TAs during class hours.