CS3205 A4 REPORT

Implementing and comparing Go-Back-N and Selective-Repeat protocol

Rudra Desai (CS18B012)

08-05-2021
3rd Year, BTech CSE

INDEX

AIM

The objective of this project is to implement the Selective-Repeat and Go-Back-N reliable transmission protocol and measure the round trip delays as well as the retransmission ratios.

INTRODUCTION

Go-Back-*N* ARQ is a specific instance of the **automatic repeat request (ARQ)** protocol, in which the sending process continues to send a number of frames specified by a *window size* even without receiving an acknowledgement (ACK) packet from the receiver. It uses frame-pipelining to send multiple frames without waiting for ACKs. It achieves better link-utilization, throughput so as to use bandwidth effectively. It is mainly used in low congested links, where the chance of getting timeouts is less.

Go-Back-N suffers from a major problem of resending the entire window when an error occurs, because the receiver can only accept frames in-order. So, we use **Selective-Repeat ARQ** to overcome this problem. It combines the advantages of both Stop-Wait and GBN protocols. Selective Repeat attempts to retransmit only those packets that are actually lost. Independent ACKs along with Negative acknowledgements are used in Selective-Repeat ARQ.

When Buffer Space is of more concern than bandwidth, Go-Back-N protocol is a good choice, while, if error rate is high, selective repeat is better in terms of number of retransmits .

EXPERIMENTAL SETUP

- We need to run both sender.py and the receiver.py on separate hosts or in separate sessions.
- Messages are exchanged between the routers using the TCP protocol.
- The only reason for changing from UDP to TCP was that the last assignment contained UDP programming, and so to get a feel for TCP programming too, I implemented TCP sockets here.
- Each router acts as a server as well as a client while sending and receiving.

ASSUMPTIONS

- All the packets get transmitted to the receiver and only get dropped randomly based on the probability supplied using the command-line arguments at the receiver's side.
- The equation used for timeout is : timeout = min(100 * rtt_avg, 0.3)
- There are no theoretical reasons for the above assumptions. This was decided based on the several experiments. Only the above assumption was found working for all the cases.

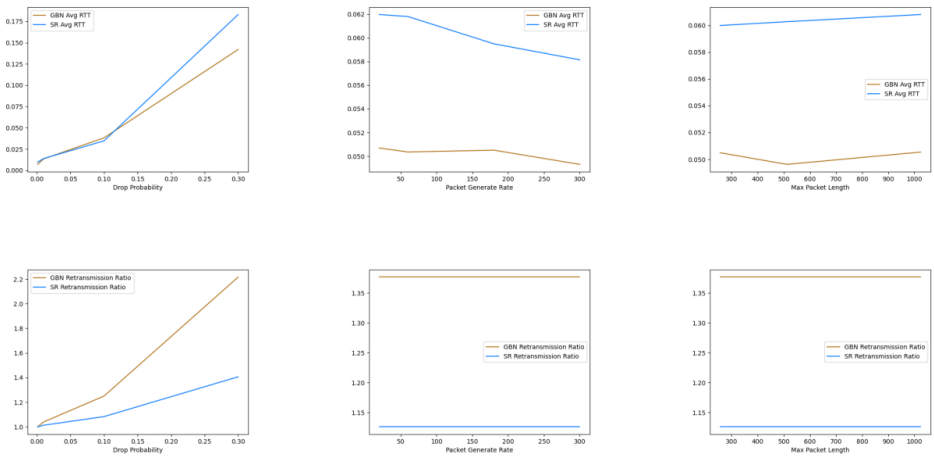
IMPLEMENTATIONAL DETAILS

1. **Go-Back-N**
    - *Sender*
        - This class uses 3 separate threads during its execution.
        - First thread generates a packet every second at a given rate and stores it in the buffer.
        - Second thread instantiates a TCP socket and continuously listens for messages. If ACK is received, it calls the associated functions.
        - Third thread sends packets to the receiver using another TCP socket and checks for timeout.
        - It maintains separate timers for each packet in transit and incase of a timeout, it clears all the timers and resends the entire window again.
        - I used locks here to ensure that receiver thread and timeout thread doesn't mix up.
    - *Receiver*
        - This class instantiates a TCP socket and continuously listens for incoming packets.
        - Based on a random toss, it decides whether to drop the packet or receive it.
        - Incase of a successful receival, it sends an ACK back to the sender, only if the received packet is in-order with the packet it expects.
        - It ignores any packet received out-of-order.

2. **Selective-Repeat**
    - *Sender*
        - Same as the Go-Back-N's sender, with the following changes.
        - Receivers can also send 'NAK' (Not Acknowledged) along with ACKs to specify if a packet prior to the current one is dropped.
        - To handle NAK, the sender instantly sends a packet again without changing the local variables like timers to not disturb the RTT.
    - *Receiver*
        - The receiver's window size is the same as the sender's window size.
        - Similar to the Go-Back-N's receiver, in case of in-order packets, it sends an ACK to the sender.
        - But, in case of out-of-order packets, it accepts it and sends an NAK with the next expected packet. So, to selectively repeat only the lost packet.

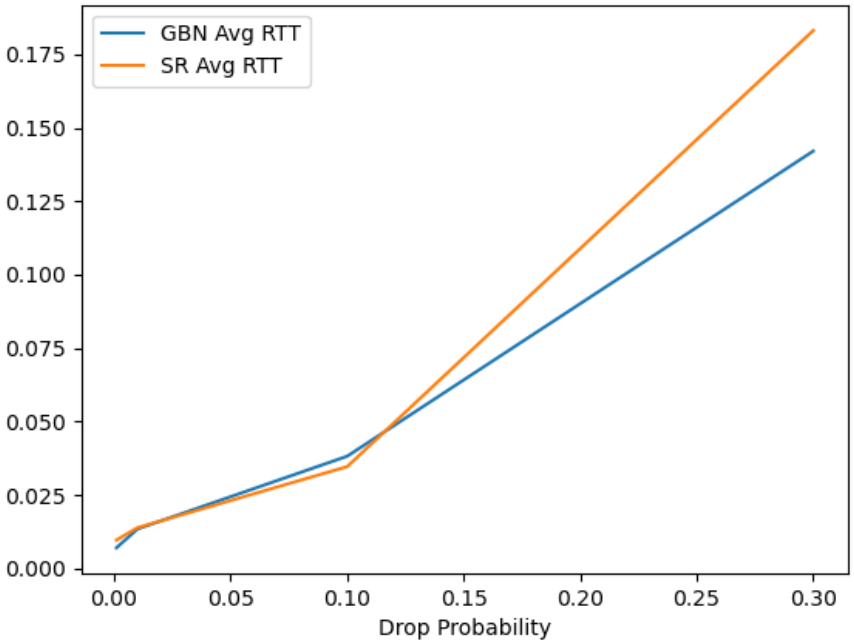
PLOTS

1. Collaged Plots

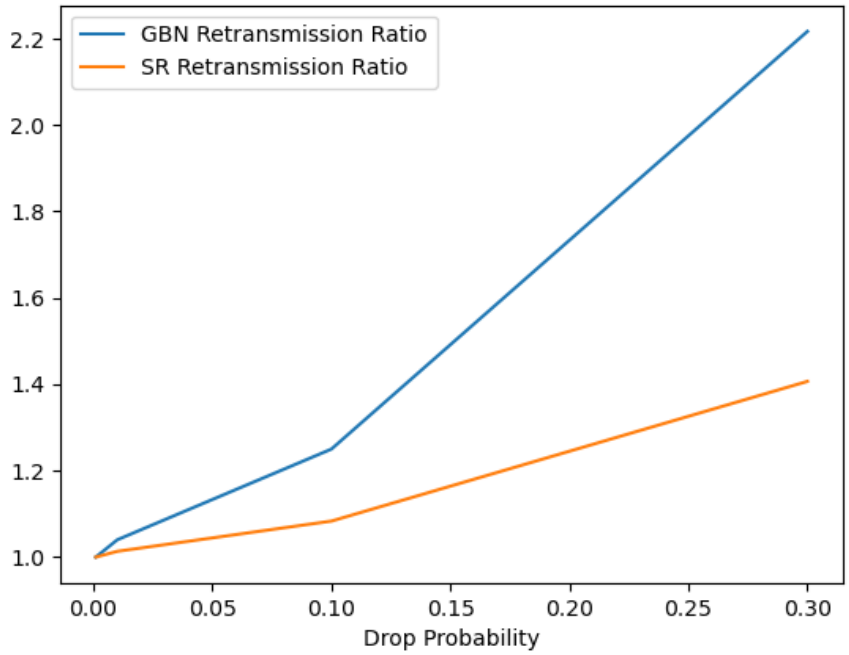


Note: You can view the zoomable plots here.

2. Drop Probab vs Avg RTT



3. Drop Probab vs Retransmission Ratio



OBSERVATIONS

- **Pack length and size doesn't change avg RTT as well as the Retransmission ratio** much as the TCP sockets used in this assignment are implemented very efficiently by python developers ;)
- Although, Drop probability has a great effect on avg RTT as expected. One can clearly see that **as drop probability increases, avg RTT of Selective-Repeat grows rapidly as compared to Grow-Back-N**.
- Also, Drop probability has a great effect on Retransmission Ratio. **As drop probability increases, Retransmission ratio for Go-Back-N grows exponentially,** while Retransmission Ratio for Selective-Repeat, increases in polynomial time.

RESULTS

1. The above plots display the major differences between Go-Back-N and Selective-Repeat protocols.
2. Go-Back-N protocol is efficient when error-rate is low, thereby reducing the average RTT.
3. But, incase of erroneous links, the retransmission ratio of Go-Back-N becomes a major concern.
4. Selective-Repeat protocol is mainly efficient when error-rate is high. By selectively asking for only the lost packets, it decreases the Retransmission ratio by a lot.
5. But, naturally, because of more timeouts, avg RTT increases.

CONCLUSION

As the plots display, each protocol is efficient for different situations. Selective-Repeat is more complex but works better in erroneous links, while Go-Back-N is simple and works very efficiently with less error-prone links. The difference between the above protocols is quite similar to TCP vs UDP, one ensures higher reliability while the other ensures faster delivery.