CS 455 PA2

Collaborators: Aaron Ang (GBN), Kian Boon Tan (SR)

Part I: SR with cumulative ACK

Overall Design

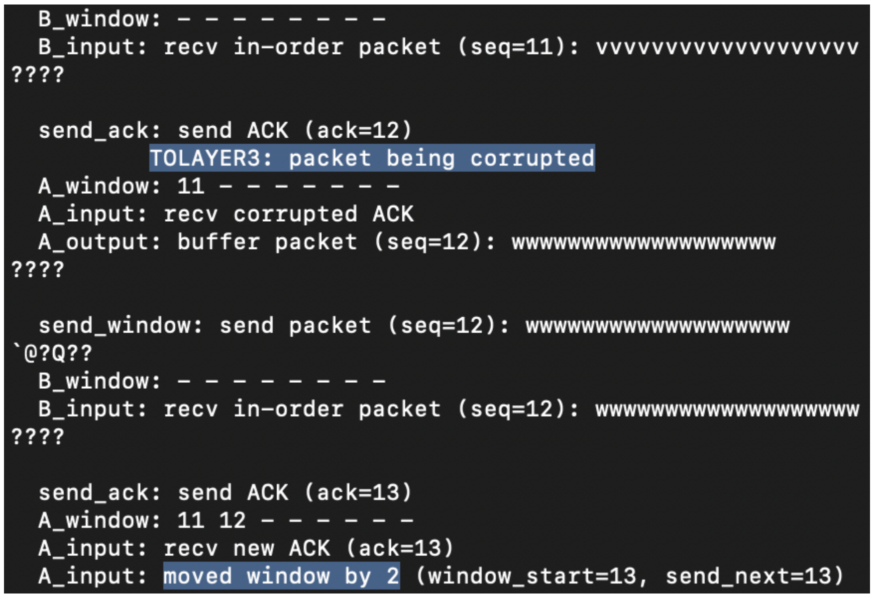
The window is implemented alongside a buffer array of fixed size. The buffer array contains pointers (initially NULL) to the pkt struct. There are three monotonically increasing indexes managing the buffer array in A: **window_start**, **send_next**, and **buffer_next**. Each time A_output is called, memory for a pkt is allocated and a pointer is initialized at the buffer_next index. Then, the **send_window()** routine will be called to send the packet objects to B's layer 3 if the current window allows. If a packet is sent, the send_next index will be incremented. When a packet is ACKed, memory will be freed and its pointer will be set to NULL. This way, advancing the window is simply a matter of incrementing window_start until a non-NULL pointer is found. Wraparound of the buffer is made possible using the modulo of the indexes.

Similarly, B contains a buffer array to handle out-of-order packets. Only one index: **window_start** is required to handle B's window. Other than packet corruption, The **B_input()** routine checks for duplicate and out-of-range packets and drops them if found. If a new packet is received, memory is allocated and a pointer is initialized at the specific index. Packets are only delivered to layer 5 when an in-order packet is received and the window slides.

To measure RTT and communication time, two separate arrays having the same length as the buffer are created in A. The first **packet_timer** array holds pointers to the **timespec** struct representing the packet transmission start time. Whenever a packet is sent, sys/time.h's **clock_gettime()** coroutine is called to place a value in the timespec object. The second **retransmitted** array holds boolean values indicating whether the packet was retransmitted. Whenever a packet is ACKed, the current clock time is taken and the value in the packet's timespec object is subtracted from it.

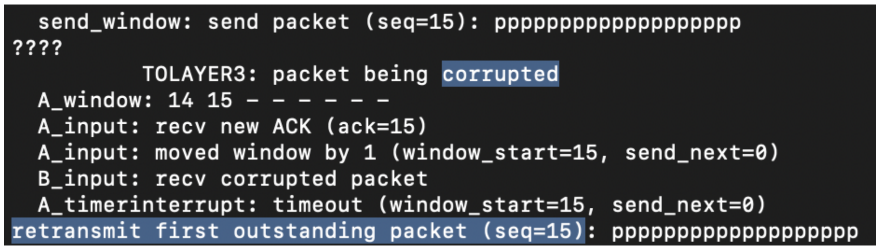
Sample Annotated Output

Case 2



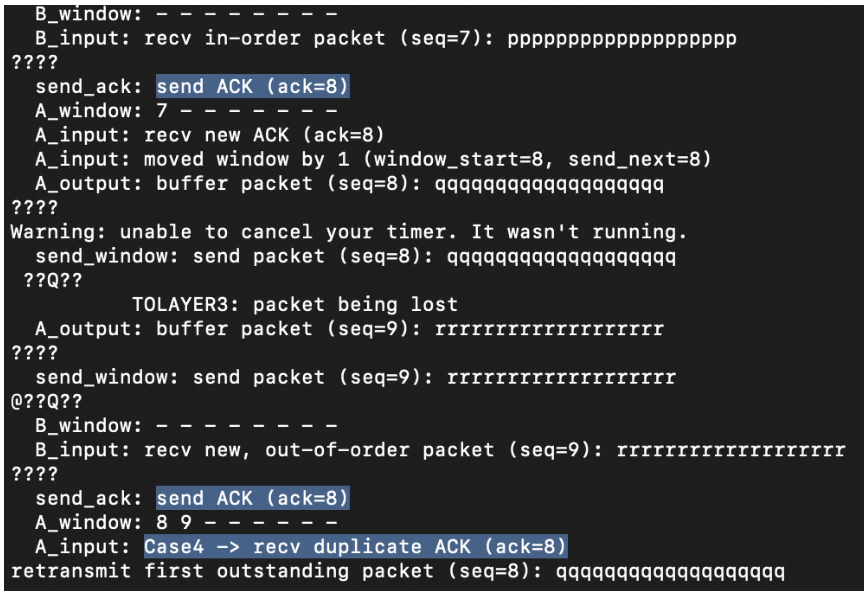
```
  B_window: - - - - - - - - -
  B_input: recv in-order packet (seq=11): vvvvvvvvvvvvvvvvvvv
????

  send_ack: send ACK (ack=12)
          TOLAYER3: packet being corrupted
  A_window: 11 - - - - - - - -
  A_input: recv corrupted ACK
  A_output: buffer packet (seq=12): wwwwwwwwwwwwwwwwwwww
????

  send_window: send packet (seq=12): wwwwwwwwwwwwwwwwwwww
`@?Q??
  B_window: - - - - - - - - -
  B_input: recv in-order packet (seq=12): wwwwwwwwwwwwwwwwwwww
????

  send_ack: send ACK (ack=13)
  A_window: 11 12 - - - - - - -
  A_input: recv new ACK (ack=13)
  A_input: moved window by 2 (window_start=13, send_next=13)
```

Case 3



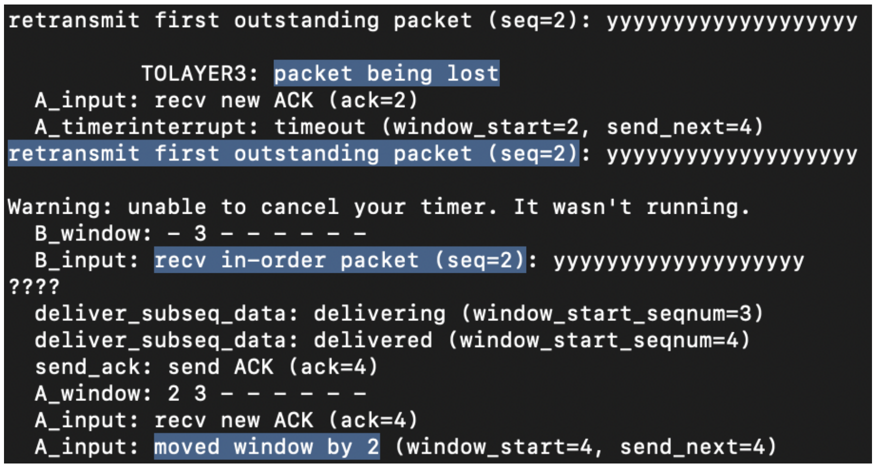
```
  send_window: send packet (seq=15): ppppppppppppppppppp
????
          TOLAYER3: packet being corrupted
  A_window: 14 15 - - - - - - -
  A_input: recv new ACK (ack=15)
  A_input: moved window by 1 (window_start=15, send_next=0)
  B_input: recv corrupted packet
  A_timerinterrupt: timeout (window_start=15, send_next=0)
retransmit first outstanding packet (seq=15): ppppppppppppppppppp
```

Case 4



```
  B_window: - - - - - - - -
  B_input: recv in-order packet (seq=7): ppppppppppppppppppp
????
  send_ack: send ACK (ack=8)
  A_window: 7 - - - - - - -
  A_input: recv new ACK (ack=8)
  A_input: moved window by 1 (window_start=8, send_next=8)
  A_output: buffer packet (seq=8): qqqqqqqqqqqqqqqqqqq
????
Warning: unable to cancel your timer. It wasn't running.
  send_window: send packet (seq=8): qqqqqqqqqqqqqqqqqqq
 ??Q??
          TOLAYER3: packet being lost
  A_output: buffer packet (seq=9): rrrrrrrrrrrrrrrrrrr
????
  send_window: send packet (seq=9): rrrrrrrrrrrrrrrrrrr
@??Q??
  B_window: - - - - - - - -
  B_input: recv new, out-of-order packet (seq=9): rrrrrrrrrrrrrrrrrrr
????
  send_ack: send ACK (ack=8)
  A_window: 8 9 - - - - - -
  A_input: Case4 -> recv duplicate ACK (ack=8)
retransmit first outstanding packet (seq=8): qqqqqqqqqqqqqqqqqqq
```

Case 5



```
retransmit first outstanding packet (seq=2): yyyyyyyyyyyyyyyyyyy

          TOLAYER3: packet being lost
  A_input: recv new ACK (ack=2)
  A_timerinterrupt: timeout (window_start=2, send_next=4)
retransmit first outstanding packet (seq=2): yyyyyyyyyyyyyyyyyyy

Warning: unable to cancel your timer. It wasn't running.
  B_window: - 3 - - - - - -
  B_input: recv in-order packet (seq=2): yyyyyyyyyyyyyyyyyyy
????
  deliver_subseq_data: delivering (window_start_seqnum=3)
  deliver_subseq_data: delivered (window_start_seqnum=4)
  send_ack: send ACK (ack=4)
  A_window: 2 3 - - - - - -
  A_input: recv new ACK (ack=4)
  A_input: moved window by 2 (window_start=4, send_next=4)
```

Statistics

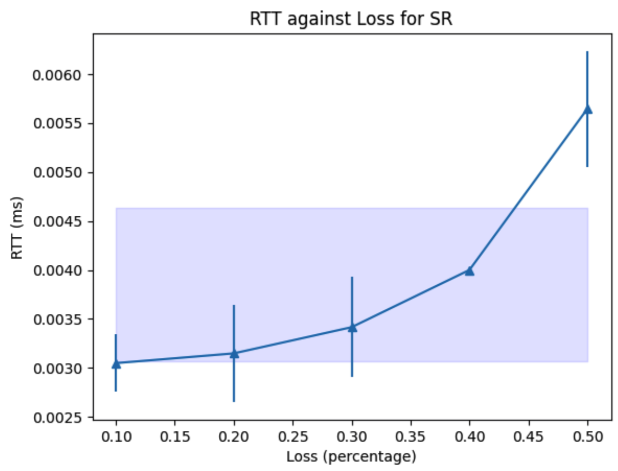
We implemented the **restart_rxmt_timer()** coroutine provided in the starter code. It simply calls the stoptimer() and starttimer() coroutines. We called restart_rxmt_timer() whenever new packets were sent or when the first outstanding packet was retransmitted. We experimented with calling the coroutine before or after the packets were sent with no observable difference in results. We expect differences in performance only if the retransmission timer value is very small. Otherwise, our current implementation should suffice.

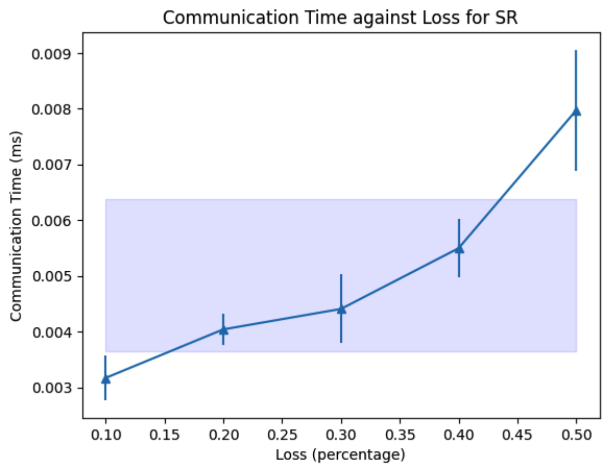
Average Performance

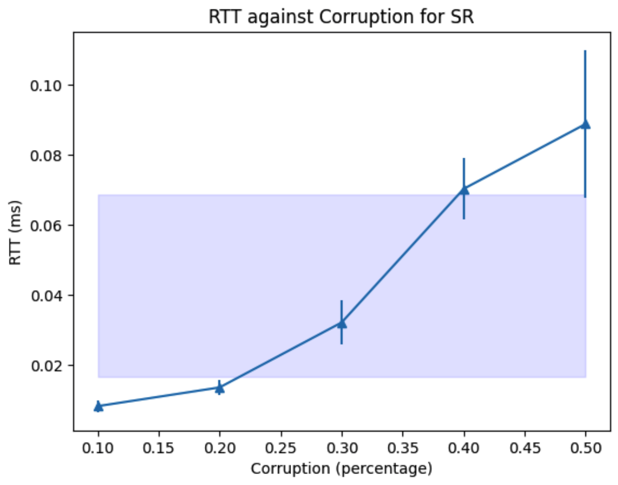
All tests were run on the csa1.bu.edu machine. The data points represent the mean of each set of results, while the line represents the standard deviation. The shaded rectangle represents the confidence interval of 90% for the data.

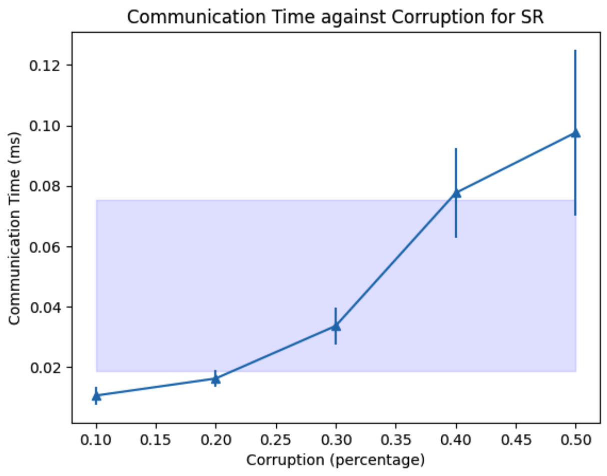
Other variables kept constant:

- Messages to simulate: 1000
- Average time between messages: 200
- Window size: 8
- Retransmission timeout: 30
- Trace level: 1
- Seeds: [1, 105, 205, …, 10001] for each result









Part II: GBN with SACK option

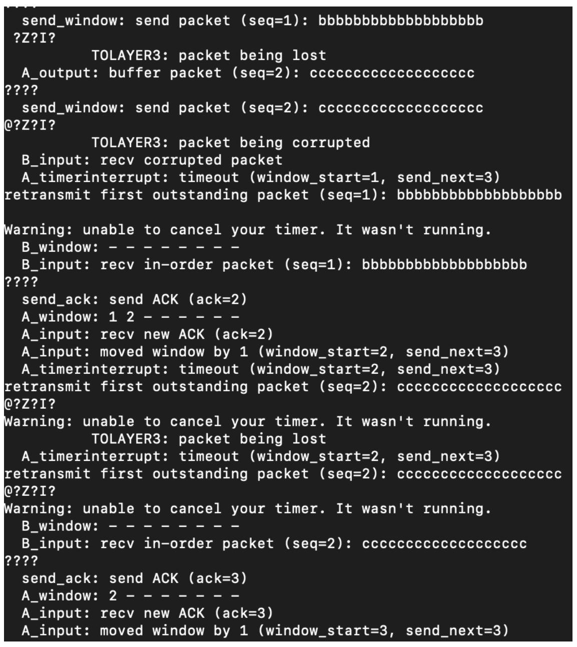
Overall Design

Most of the functionality is reused from our SR implementation. For the receiver side (B), only one index: window_start is used since it has a window size of 1. A_input checks if the ACK equals window_start's sequence number. If so, it will process SACKs. Otherwise, it will move the window forward. In B_input, the implementation is more strict. It will attempt to process out-of-order ACKs but will drop those that are repeated (SACK is already present) or out-of-range (seq no. does not exist within the SACK option). Compared to SR, this implementation will immediately deliver both in-order and valid out-of-order packets to layer 5, instead of delivering the packets when the window slides. Thus, we expect better performance out of this GBN implementation.

Sample Annotated Output

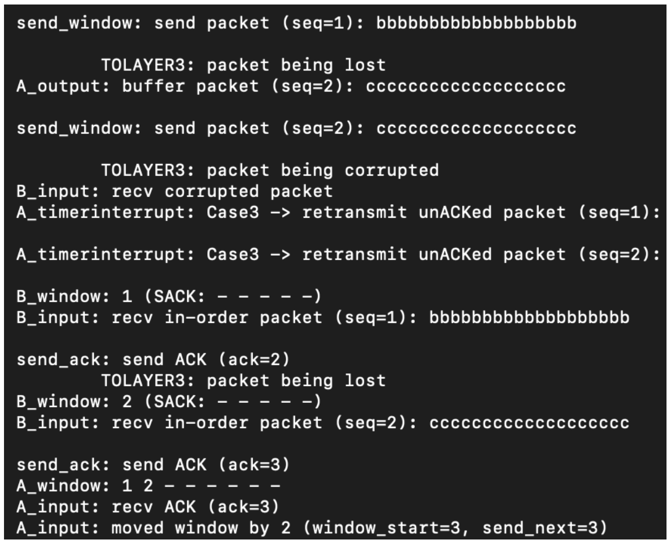
The following traces show differences between GBN+SACK and SR. They are run using identical input values given in the PA2 handout. During RTO, the GBN implementation will retransmit all unACKed packets, while the SR implementation will retransmit only the first outstanding packet.

SR



```
  send_window: send packet (seq=1): bbbbbbbbbbbbbbbbbbb
 ?Z?I?
          TOLAYER3: packet being lost
  A_output: buffer packet (seq=2): ccccccccccccccccccc
????
  send_window: send packet (seq=2): ccccccccccccccccccc
@?Z?I?
          TOLAYER3: packet being corrupted
  B_input: recv corrupted packet
  A_timerinterrupt: timeout (window_start=1, send_next=3)
retransmit first outstanding packet (seq=1): bbbbbbbbbbbbbbbbbbb

Warning: unable to cancel your timer. It wasn't running.
  B_window: - - - - - - - -
  B_input: recv in-order packet (seq=1): bbbbbbbbbbbbbbbbbbb
????
  send_ack: send ACK (ack=2)
  A_window: 1 2 - - - - - -
  A_input: recv new ACK (ack=2)
  A_input: moved window by 1 (window_start=2, send_next=3)
  A_timerinterrupt: timeout (window_start=2, send_next=3)
retransmit first outstanding packet (seq=2): ccccccccccccccccccc
@?Z?I?
Warning: unable to cancel your timer. It wasn't running.
          TOLAYER3: packet being lost
  A_timerinterrupt: timeout (window_start=2, send_next=3)
retransmit first outstanding packet (seq=2): ccccccccccccccccccc
@?Z?I?
Warning: unable to cancel your timer. It wasn't running.
  B_window: - - - - - - - -
  B_input: recv in-order packet (seq=2): ccccccccccccccccccc
????
  send_ack: send ACK (ack=3)
  A_window: 2 - - - - - - -
  A_input: recv new ACK (ack=3)
  A_input: moved window by 1 (window_start=3, send_next=3)
```

GBN



```
send_window: send packet (seq=1): bbbbbbbbbbbbbbbbbbb

        TOLAYER3: packet being lost
A_output: buffer packet (seq=2): ccccccccccccccccccc

send_window: send packet (seq=2): ccccccccccccccccccc

        TOLAYER3: packet being corrupted
B_input: recv corrupted packet
A_timerinterrupt: Case3 -> retransmit unACKed packet (seq=1):

A_timerinterrupt: Case3 -> retransmit unACKed packet (seq=2):

B_window: 1 (SACK: - - - - -)
B_input: recv in-order packet (seq=1): bbbbbbbbbbbbbbbbbbb

send_ack: send ACK (ack=2)
        TOLAYER3: packet being lost
B_window: 2 (SACK: - - - - -)
B_input: recv in-order packet (seq=2): ccccccccccccccccccc

send_ack: send ACK (ack=3)
A_window: 1 2 - - - - - -
A_input: recv ACK (ack=3)
A_input: moved window by 2 (window_start=3, send_next=3)
```

Statistics

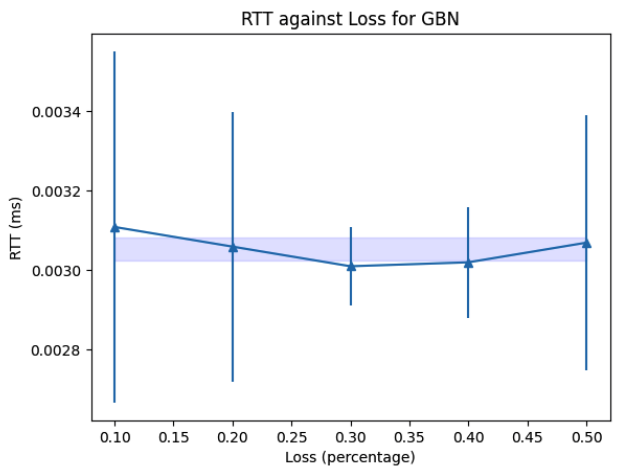
Compared to SR implementation, we only call restart_rxmt_timer() in GBN when sending new packets. We call starttimer() during retransmissions since retransmissions are only executed during RTO in our implementation and we can safely assume that no timers are running during RTO.

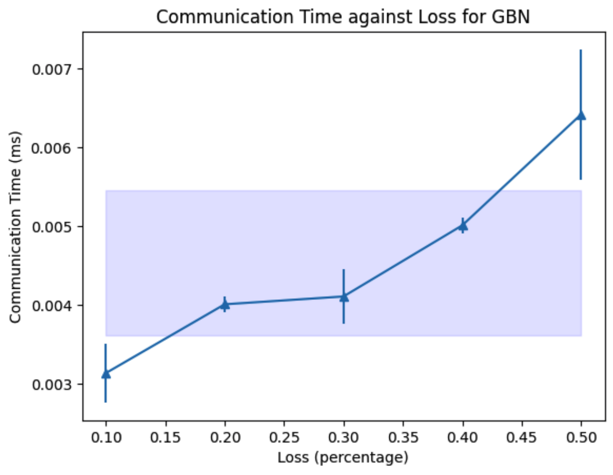
Average Performance

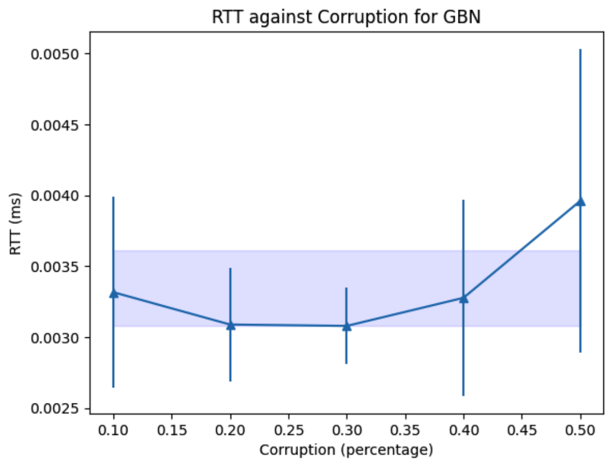
Again, the tests were run on the csa1.bu.edu machine. The data points represent the mean of each set of results, while the line represents the standard deviation. The shaded rectangle represents the confidence interval of 90% for the data.

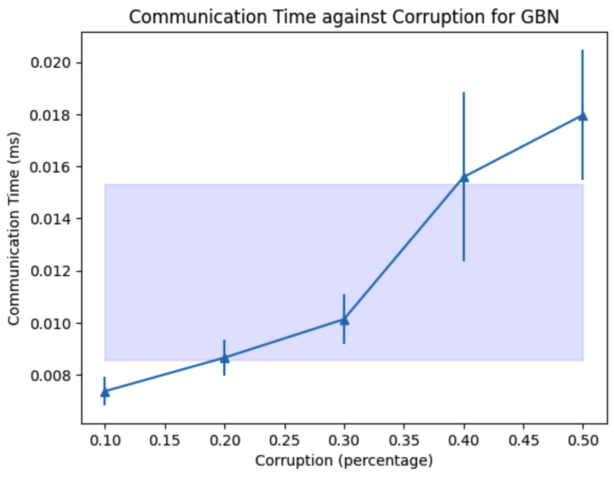
Other variables kept constant:

- Messages to simulate: 1000
- Average time between messages: 200
- Window size: 8
- Retransmission timeout: 30
- Trace level: 1
- Seeds: [1, 105, 205, …, 10001] for each result









Comparing SR with GBN

From the results, we can observe that on average, GBN results in lower RTT and Communication Time, showing that it results in better throughput and goodput than SR.