# GRS CS 655

# Graduate Intro to Computer Networks

# PA2

Name: Haoxuan Sun

BUID: U58198360

Data: 11/03/2022

**How to compile**: ***cd*** into code folder and then compile all files using: ***javac *.java***

Then run the code with ***java Project***

Code and traceoutput file is submitted through gsubmit.

# Part 1

**Design:**

In part 1, Selective Repeat with Cumulative ACKs protocol is implemented.

A static retransmission timer value is used when implementing the protocol.

In sender end: An array ***senderWindow[]*** contains packets is used to simulate the sender window and a LinkedList ***senderBuffer*** is used to simulate the sender's buffer. Another array ***ifAck[]*** is used to record whether a packet has been ACKed. The initial value is -1; when the packet is sent, set value to 0; when receive an ACK for a packet, set value to 1. A variable ***tempSeqNum*** is used to track the next packet's sequence number which will send out.

When A receives data from upper layer, it news a packet with sequence ***tempSeqNum***. First send it to the sender's buffer. Move it to sender window if there are empty spaces in window. A sends packets in sender window and set its corresponding ***ifAck[]*** to 0.

In receiver end: An array ***receiverWindow[]*** contains packets is used to simulate the receiver window. A variable records the latest received sequence number, and a packet variable records the latest received packet.

If B receives corrupted packet (compare ***checksum()*** of the packet with the check sum number in its value), ignore the packet and send ACK of the latest received packet. If the packet is not corrupted, then check its sequence number:

If it is the expected packet: send the data to upper layer and send its ACK to A.

If it is not the expected packet but within the window size: buffer it and send ACK of the latest correct order packet to A.

At the meantime, check buffer to find whether the next expected packet is stored. If yes, send cumulative ACK to A.

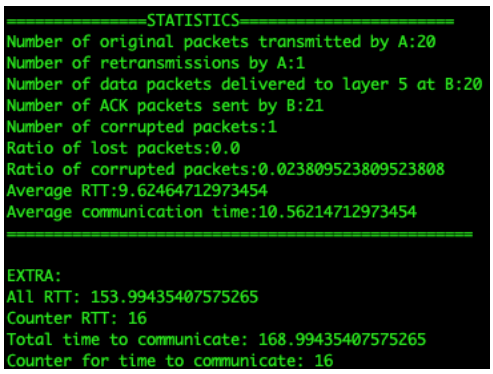
A will receive an ACK from B whenever B receives a packet. Check whether it is the first packet in sender window. If yes, slides sender window; if not, resend the B's expected packet.

At any time, if A finds out that a packet is time out without receiving its later message, resend the packet to B

**Testing:**

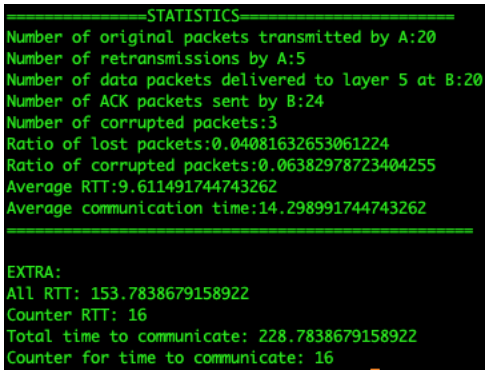
| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 1 | 20 | 0.0 | 0.0 | 1000 | 8 | 15.0 | 2 | 421 |

Result:



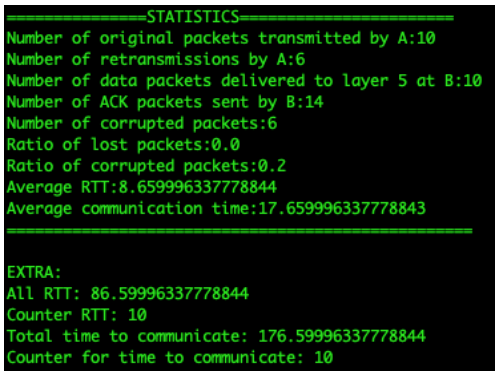

| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 2 | 20 | 0.1 | 0.0 | 1000 | 8 | 15.0 | 2 | 422 |

Result:



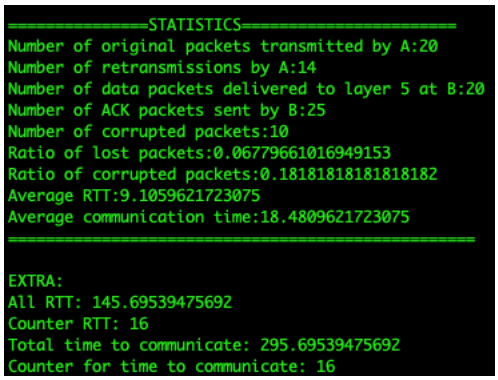
| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 3 | 20 | 0.0 | 0.1 | 1000 | 8 | 15.0 | 0 | 111 |

Result:




| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 4 | 20 | 0.1 | 0.1 | 1000 | 8 | 15.0 | 2 | 111 |

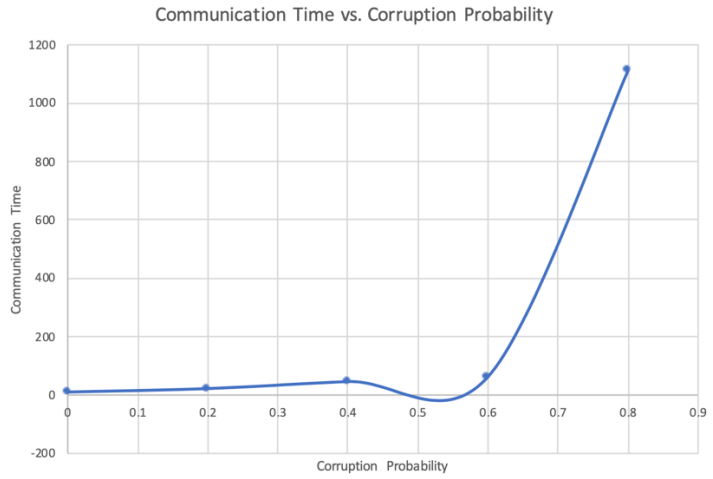
Result:



Average communication time/SSD/Confidence Intervals when setting corruption probability to 0/0.2/0.4/0.6/0.8:

| Corruption | T1 | T2 | T3 | T4 | T5 | AVG | SSD | CONFIDENCE 90% |
|---|---|---|---|---|---|---|---|---|
| 0 | 9.656 | 9.786 | 10.001 | 10.989 | 11.149 | 10.3162 | 0.700453924 | 0.51525454 |
| 0.2 | 20.855 | 22.866 | 22.677 | 22.019 | 22.677 | 22.2188 | 0.827440753 | 0.608666167 |
| 0.4 | 51.896 | 56.256 | 36.188 | 34.223 | 52.578 | 46.2282 | 10.22168201 | 7.519078536 |
| 0.6 | 73.782 | 53.715 | 73.276 | 58.278 | 53.561 | 62.5224 | 10.22631924 | 7.522489687 |
| 0.8 | 1687.281 | 977.055 | 818.026 | 272.623 | 1799.167 | 1110.8304 | 634.8867003 | 467.023231 |

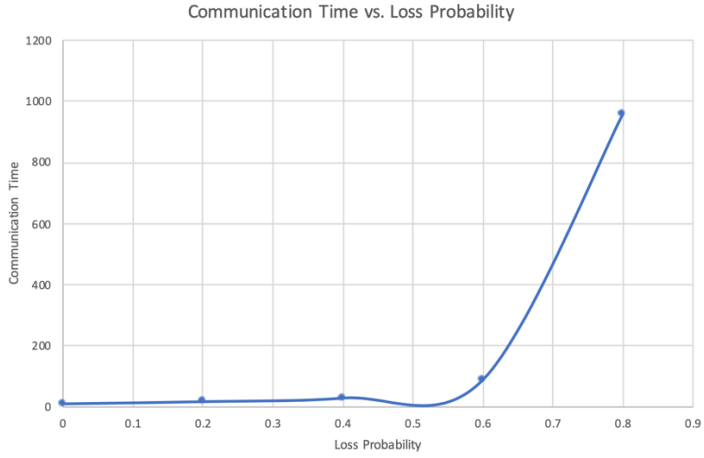
Line Graph of Comm Time vs. Corruption probability:




Average communication time/SSD/Confidence Intervals when setting lost probability to 0/0.2/0.4/0.6/0.8:

| Loss | T1 | T2 | T3 | T4 | T5 | AVG | SSD | CONFIDENCE 90% |
|---|---|---|---|---|---|---|---|---|
| 0 | 9.656 | 9.786 | 10.001 | 10.989 | 11.149 | 10.3162 | 0.700453924 | 0.51525454 |
| 0.2 | 19.561 | 19.138 | 15.322 | 18.523 | 15.328 | 17.5744 | 2.086317641 | 1.534697144 |
| 0.4 | 33.14 | 28.828 | 31.012 | 27.732 | 26.055 | 29.3534 | 2.77820622 | 2.043651008 |
| 0.6 | 103.852 | 77.828 | 91.899 | 81.102 | 91.899 | 89.316 | 10.29663797 | 7.574216205 |
| 0.8 | 622.429 | 506.551 | 2336.906 | 951.556 | 377.079 | 958.9042 | 799.2656783 | 587.9405559 |

Line Graph of Comm Time vs. Loss probability:



From the above two tables and figures, we can guess that in both 2 situations: the bigger the loss/corruption probability, the more communication time is needed. From the limited collected data, we could guess the function grows exponentially. And the average commination time needed in both situations are almost the same.

# Part 2

**Design:**

In part 2, Go-Back-N protocol is implemented. Similar with the design idea of part 1.

The difference are:

In the receiver end, if the received packet is not the required one and hasn't been received before, send an ACK of the latest received packet.

In the sender end, when receives an ACK of an already sent packet, then resend all packets after it no matter whether have been sent before.

**Testing:**

| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 1 | 20 | 0.0 | 0.0 | 1000 | 8 | 15.0 | 2 | 421 |

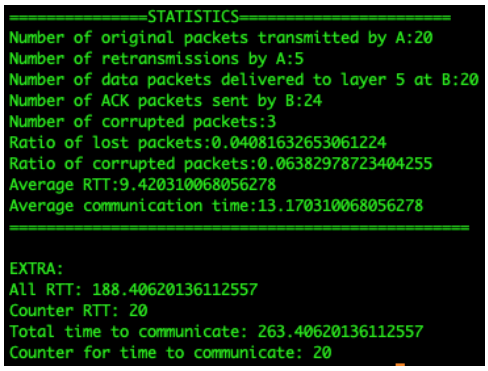
Result:

```
==============STATISTICS=======================
Number of original packets transmitted by A:20
Number of retransmissions by A:1
Number of data packets delivered to layer 5 at B:20
Number of ACK packets sent by B:21
Number of corrupted packets:1
Ratio of lost packets:0.0
Ratio of corrupted packets:0.023809523809523808
Average RTT:9.508853419599314
Average communication time:10.258853419599314
================================================

EXTRA:
All RTT: 190.17706839198627
Counter RTT: 20
Total time to communicate: 205.17706839198627
Counter for time to communicate: 20
```

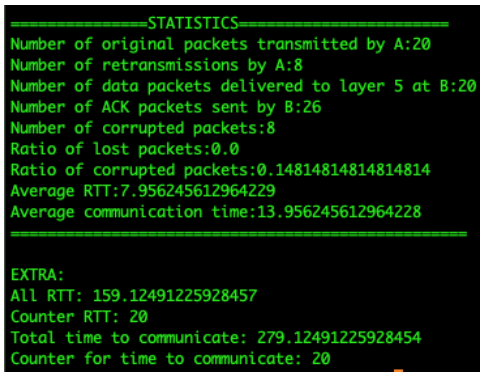
| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 2 | 20 | 0.1 | 0.0 | 1000 | 8 | 15.0 | 2 | 422 |

Result:




| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 3 | 20 | 0.0 | 0.1 | 1000 | 8 | 15.0 | 2 | 111 |

Result:




| CASE | # msg | Loss p | Corr p | Avg T | Win Size | Retra T | Trace Level | Seed |
|---|---|---|---|---|---|---|---|---|
| 4 | 20 | 0.1 | 0.1 | 1000 | 8 | 15.0 | 2 | 123 |

Result:

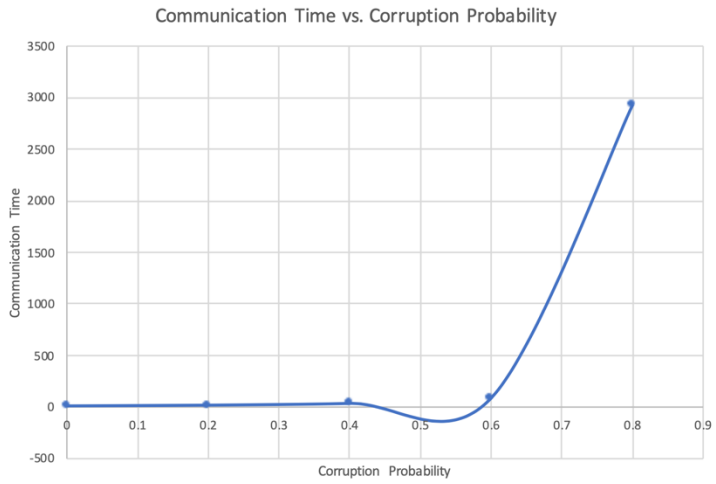
```
==========STATISTICS==========
Number of original packets transmitted by A:20
Number of retransmissions by A:16
Number of data packets delivered to layer 5 at B:20
Number of ACK packets sent by B:27
Number of corrupted packets:9
Ratio of lost packets:0.1111111111111111
Ratio of corrupted packets:0.16071428571428573
Average RTT:8.393340861232591
Average communication time:20.39334086123259
==============================================

EXTRA:
All RTT: 167.8668172246518
Counter RTT: 20
Total time to communicate: 407.8668172246518
Counter for time to communicate: 20
```

Average communication time/SSD/Confidence Intervals when setting corruption probability to 0/0.2/0.4/0.6/0.8:

| Corruption | T1 | T2 | T3 | T4 | T5 | AVG | SSD | CONFIDENCE 90% |
|---|---|---|---|---|---|---|---|---|
| 0 | 11.935 | 11.261 | 11.213 | 11.569 | 11.621 | 11.5198 | 0.294175458 | 0.216395733 |
| 0.2 | 20.693 | 18.813 | 20.609 | 15.523 | 18.142 | 18.756 | 2.122387335 | 1.561230044 |
| 0.4 | 41.553 | 31.233 | 28.101 | 32.934 | 45.052 | 35.7746 | 7.194430714 | 5.292229741 |
| 0.6 | 112.348 | 60.318 | 77.192 | 88.613 | 91.752 | 86.0446 | 19.17854543 | 14.10775537 |
| 0.8 | 4654.681 | 1404.923 | 2918.826 | 3085.192 | 2569.718 | 2926.668 | 1167.726577 | 858.9807261 |

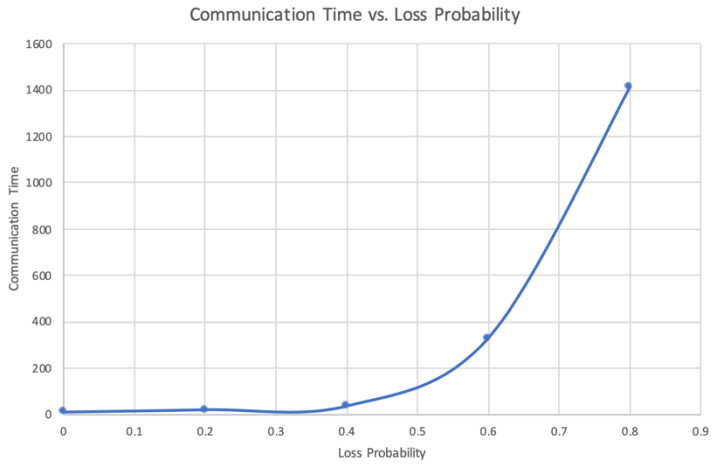
Line Graph of Comm Time vs. Corruption probability:




Average communication time/SSD/Confidence Intervals when setting lost probability to 0/0.2/0.4/0.6/0.8:

| Loss | T1 | T2 | T3 | T4 | T5 | AVG | SSD | CONFIDENCE 90% |
|---|---|---|---|---|---|---|---|---|
| 0 | 11.935 | 11.261 | 11.213 | 11.569 | 11.621 | 11.5198 | 0.294175458 | 0.216395733 |
| 0.2 | 18.839 | 20.396 | 25.922 | 25.929 | 17.629 | 21.743 | 3.942057141 | 2.899780799 |
| 0.4 | 42.181 | 37.834 | 31.012 | 37.781 | 39.796 | 37.7208 | 4.160056334 | 3.060141202 |
| 0.6 | 113.031 | 480.229 | 396.842 | 220.858 | 441.253 | 330.4426 | 156.8441735 | 115.3747159 |
| 0.8 | 1256.822 | 1988.685 | 1379.768 | 1228.732 | 1201.377 | 1411.0768 | 330.0112306 | 242.7565597 |

Line Graph of Comm Time vs. Loss probability:




From the above two tables and figures, we can guess that in both 2 situations: the bigger the loss/corruption probability, the more communication time is needed. From the limited collected data, we could guess the function grows exponentially. And the average commination time needed in both situations are almost the same.

Typically, GBN is not as efficient as SR, because sender has send all packets according to ACK it receives, although some of them could have arrived receiver end successfully. SR only resends the lost/corrupted packets.