TCP Implementation over UDP

Louay Hesham Saber 3303
Mahinour El Sheikh 3334

Introduction

In this project, we implement reliable TCP data transfer algorithms over UDP protocol using Python. The server runs in it own process, each client request opens a thread in the same process to handle the data transfer to that client. Each client instance will run in its own process. When the client is done receiving the file, it will write it to the hard disk so it can be viewed. We implemented the algorithms in a way that allows sending any file type.

Pseudo Code

Server

Main

```
● Open a new socket on port 30000 to receive new requests on
● Repeat forever
    ○ Wait for connection to be established from the client side
    ○ Open new socket for new request, port number increments for each request
    ○ Send new port number to client
    ○ Wait for connection on new socket to be established
    ○ Open new thread to handle data transfer
        ■ Receive requested file name from client
        ■ Determine which algorithm to use according to the configs
        ■ Start sending the requested file using the chosen algorithm
```

Selective repeat

```
● While EOF has not been reached yet
    ○ If packet_no to be sent is within the window
        ■ Read a chunk of the file and create the packet
        ■ Send the packet with corruption and loss probabilities in mind
        ■ Set ack[n] = false and start its timer
    ○ Check if there's an ack to be received
    ○ If ack received
        ■ Check sequence number and set ack[seq_no] = true
        ■ Stop timer[seq_no]
        ■ If seq_no = window base
            ● Shift window till the base has not been acked yet
    ○ Check on all timers, if a timer timeouts
        ■ Resend the packet with corruption and loss probabilities in mind
        ■ Reset its timer
● Close connection
```

Stop and Wait

Same as Selective Repeat but with window size = 1

Go Back N

- While EOF has not been reached yet
    - If packet_no to be sent is within the window
        - Read a chunk of the file and create the packet
        - Send the packet with corruption and loss probabilities in mind
        - Set ack[n] = false and start its timer
    - Check if there’s an ack to be received
    - If ack received
        - While window base <= seq_no of ack
            - Set ack[window_base] = true
            - Stop timer of packet at window base
            - Increment window base by 1
    - Loop on all timers
        - If timer[n] times out
            - Resend all the packets from n till the end of the window
- Close connection

Client

- Open new socket and connect it to port 30000 of the server (this socket is used only once)
- Receive new port number from server
- Open new socket and connect it to the received port number (this socket is used till the end)
- Randomly select a file name and send it to the server
- If algorithm = selective repeat: get window size from config
- Else: window size = 1 (Stop and wait & Go-Back-N)
- Repeat till EOF is received
    - Receive a packet and parse the checksum, seq_no and data
    - If seq_no is within the window
        - Compute checksum of packet data
        - If received checksum = computed checksum
            - Set prev_seq_no = seq_no
            - Send ack with packet loss probability in mind
            - If seq_no = window base
                - While window base packet has been received
                    - Append window base packet data into file data
                    - Increment window base by 1
        - Else (packet is corrupted)
            - If algorithm selected is Go-Back-N
                - Send ack[prev_seq_no]
            - Else do nothing
    - Else if seq_no < window base (before the window)
        - If algorithm selected is Go-Back-N: current_seq_no = prev_seq_no
        - Else: current_seq_num = seq_no
        - Send ack[current_seq_no]
- Make folder Client_instances/<timestamp>
- Output file data into file
- Close socket

Instructions to open client and server instances

You can change the TCP_PORT, the algorithm, window size, corruption probability and other parameters in config.py file.
The list of files is available at the config.py file, the client chooses a file from them randomly to request. When the file is received it will write it to the disk so it can be verified.

Method 1 (On Ubuntu):

1. Open terminal window and "cd" to project folder
2. Enter the command "python3"
3. "from main import *"
4. "start_server()", this command will start a new window with the server
5. "start_client()", this command will start a new window with a client instance
6. To open multiple clients, just run "start_client()" as many times as wanted

Method 2:

1. Open terminal window and "cd" to project folder
2. Enter the command "python3"
3. "import config, server"
4. "server.server(config)"
5. Open a new terminal window and "cd" to project folder
6. Enter the command "python3"
7. "import config, client"
8. "client.client(config)"
9. Repeat steps 5 through 8 for each client instance

Instructions to change configurations

Open "config.py" via any text editor to see available configurations and their property names
If using method 1 to start client and server instances, you can:

- From the same python3 terminal
    a. Enter "config.<property>=<value>"
    b. Restart server and client instances
- By editing config.py
    a. Edit config.py by any text editor
    b. From python3 terminal, enter "import importlib"
    c. "importlib.reload(config)"
    d. Restart server and client instances

If using method 2 to start client and server instances, you can only use the latter method to change the configurations. Or you can completely restart the python3 terminal after saving your changes in config.py

Stats for different PLP probabilities and window sizes

The following tables shows the average throughput (in bytes/sec) for all three TCP algorithms for different packet loss and corruption probabilities and window sizes.

Stop and Wait algorithm:

PLP	**0**	**0.01**	**0.05**	**0.1**	**0.3**
Throughput (bytes/sec)	352247	109668	29834	14820	3707

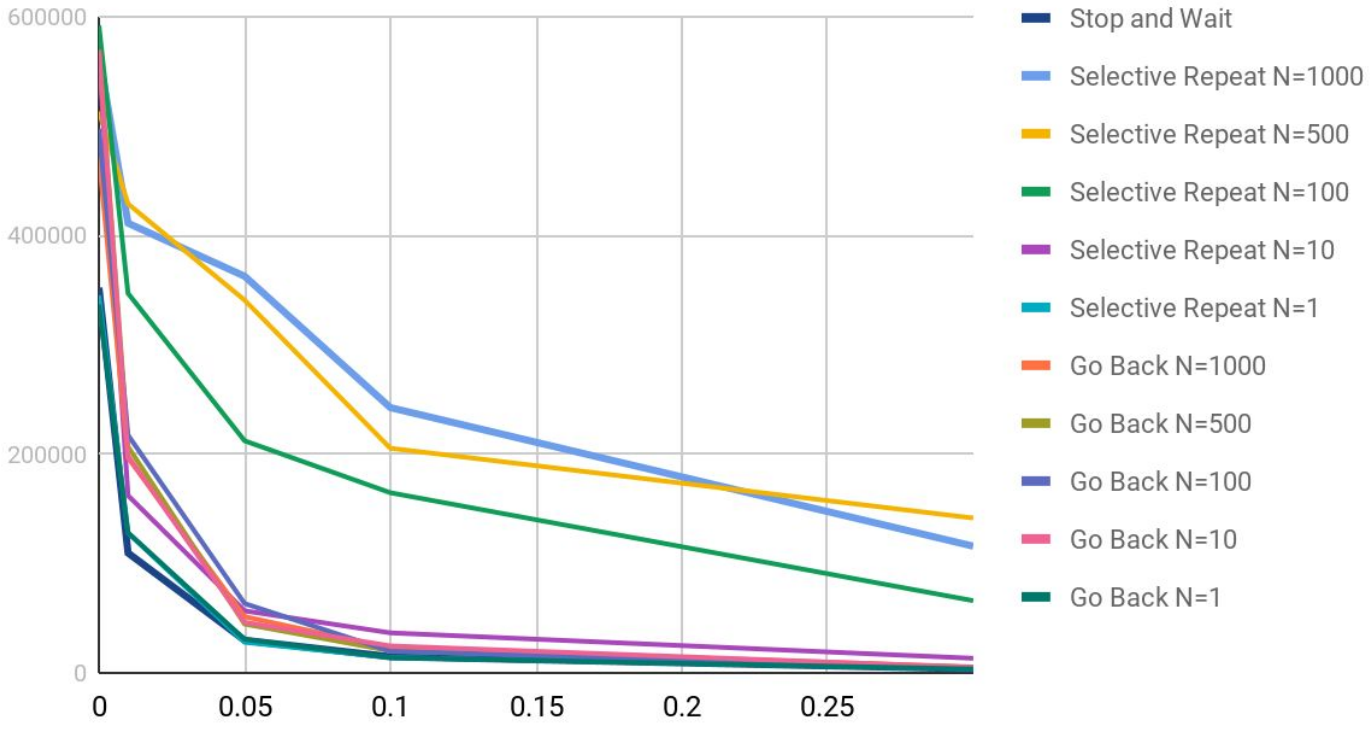
Selective Repeat algorithm:

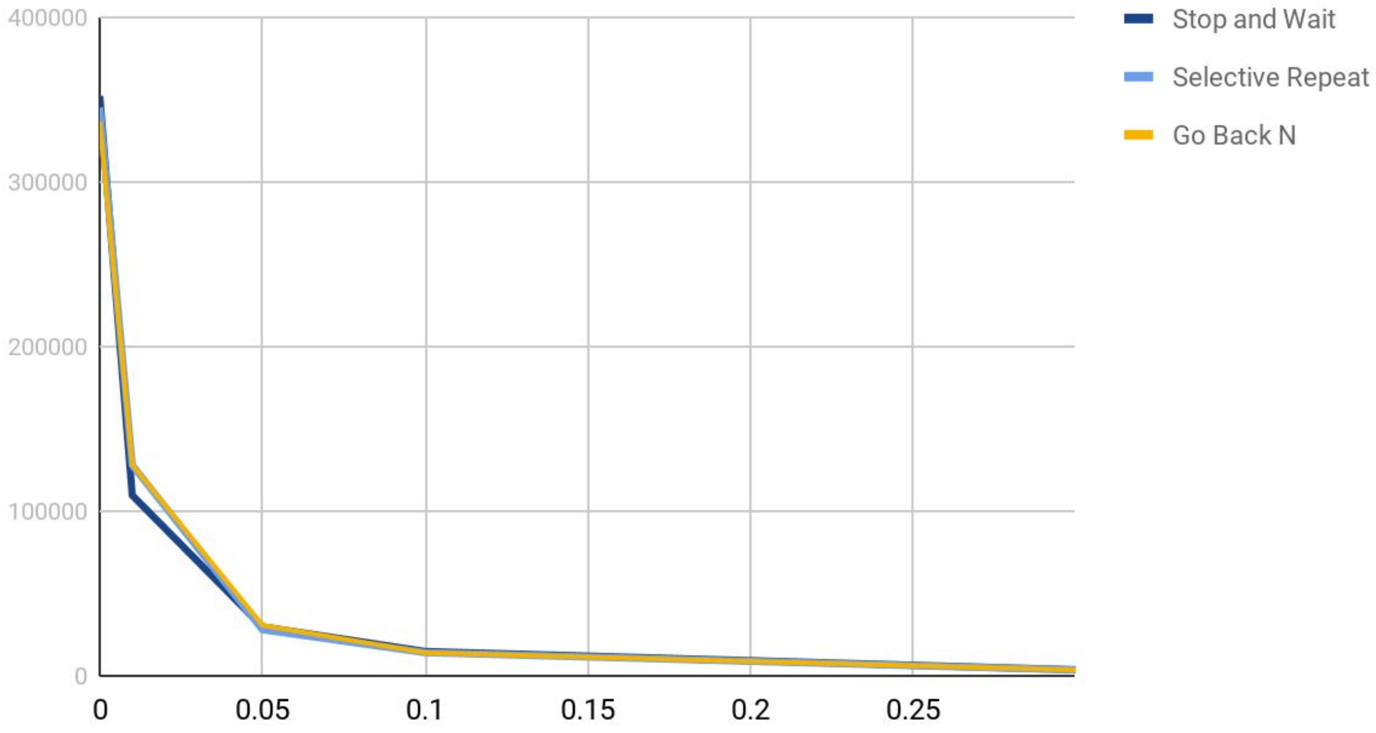
Window Size	**0**	**0.01**	**0.05**	**0.1**	**0.3**
1000	564791	411184	362654	242559	115868
500	513296	428850	340787	205401	141588
100	592026	347041	212211	164736	66027
10	564546	162122	56747	36627	13365
1	345240	127981	28025	14086	3562

Go-Back-N algorithm:

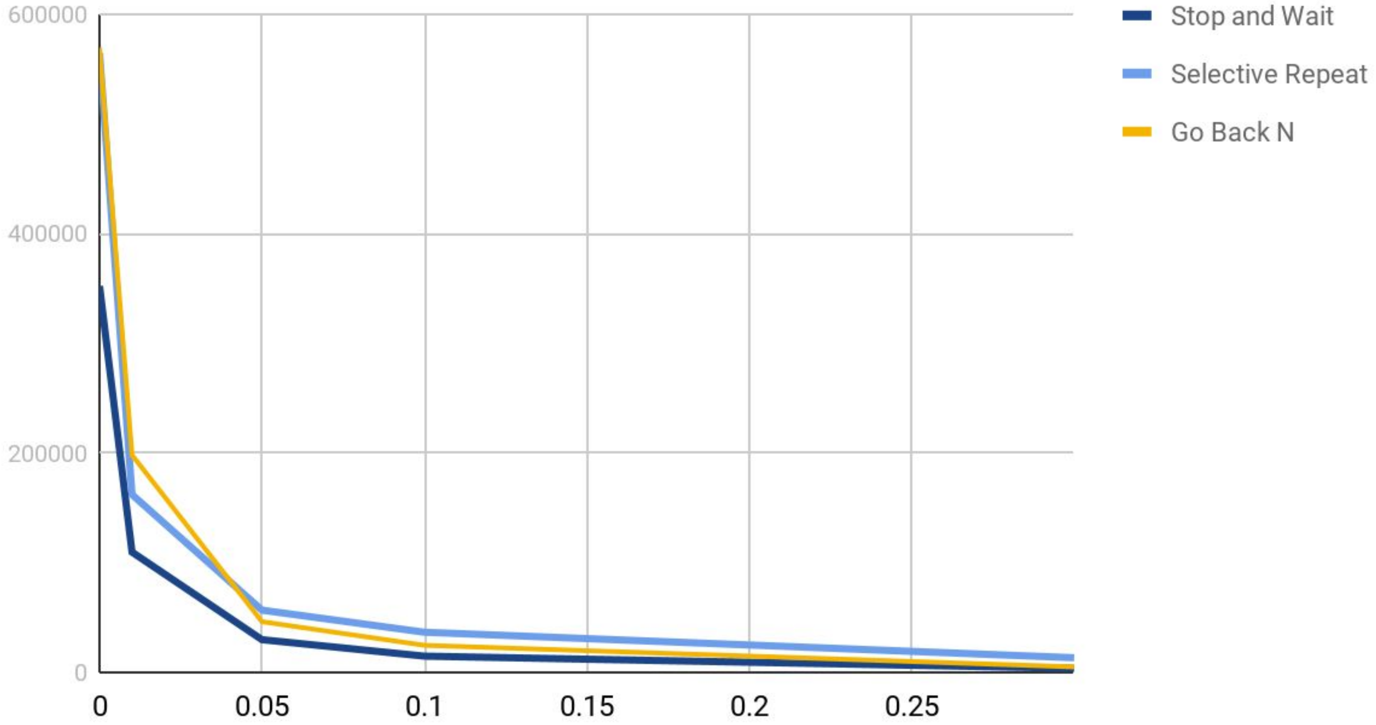
Window Size	**0**	**0.01**	**0.05**	**0.1**	**0.3**
1000	472508	196694	51461	22668	5471
500	569681	206785	44656	20138	5131
100	497644	217393	63438	19782	4744
10	569457	198170	46244	24709	5027
1	336747	128197	30718	14043	3602

Chart comparisons

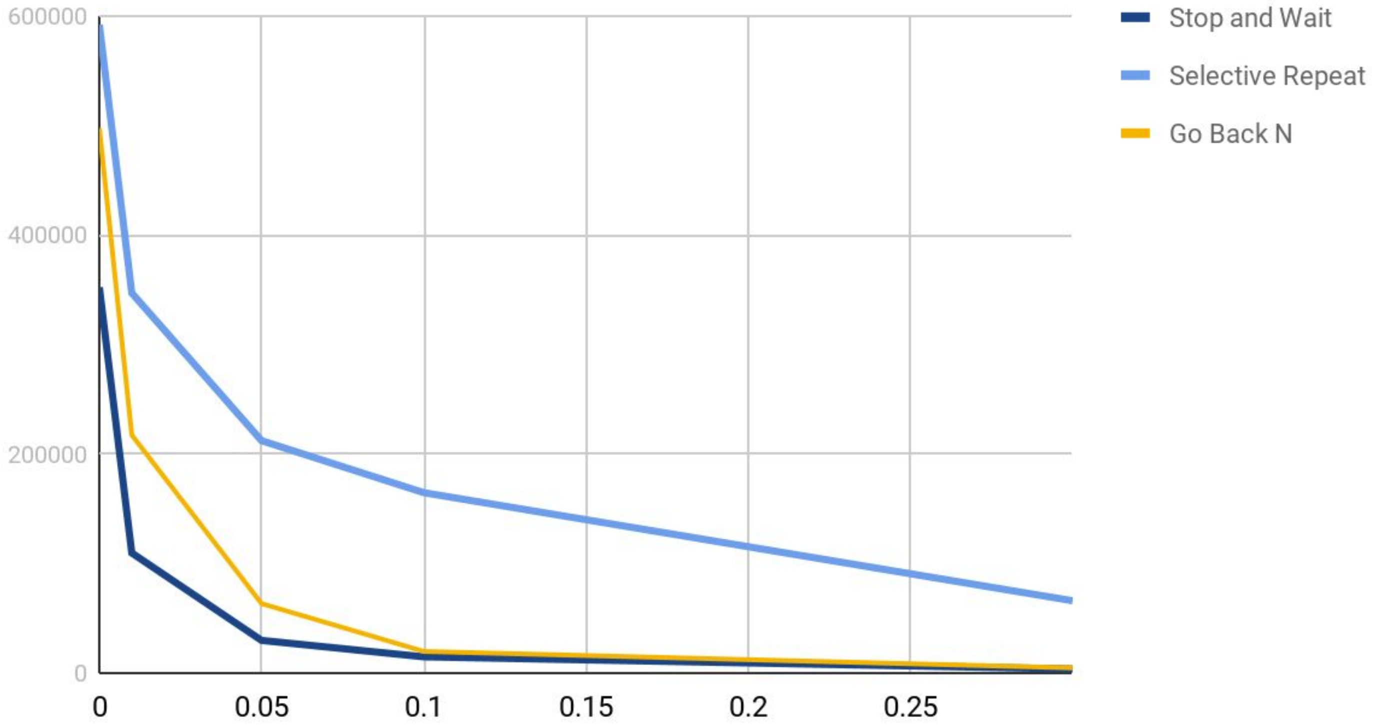

Average Throughput summary


Window Size = 1

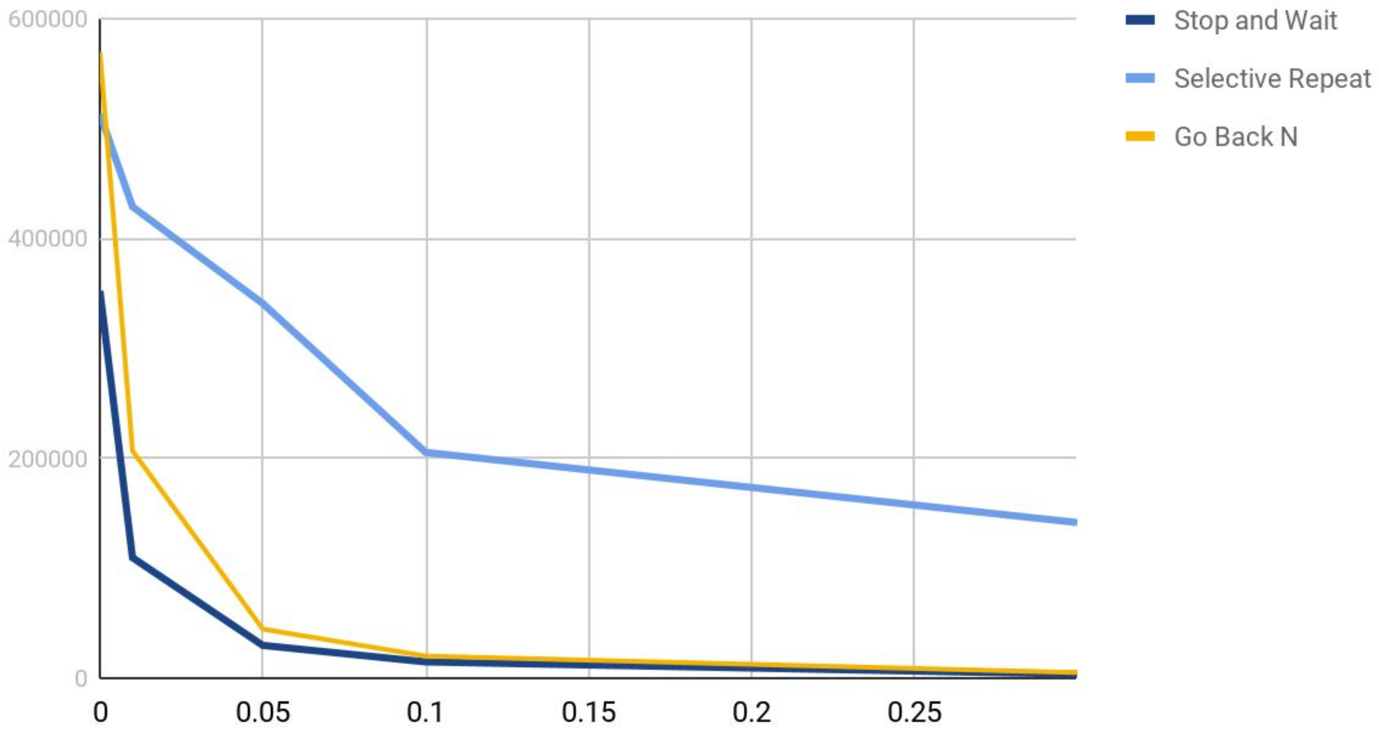
Window Size = 10



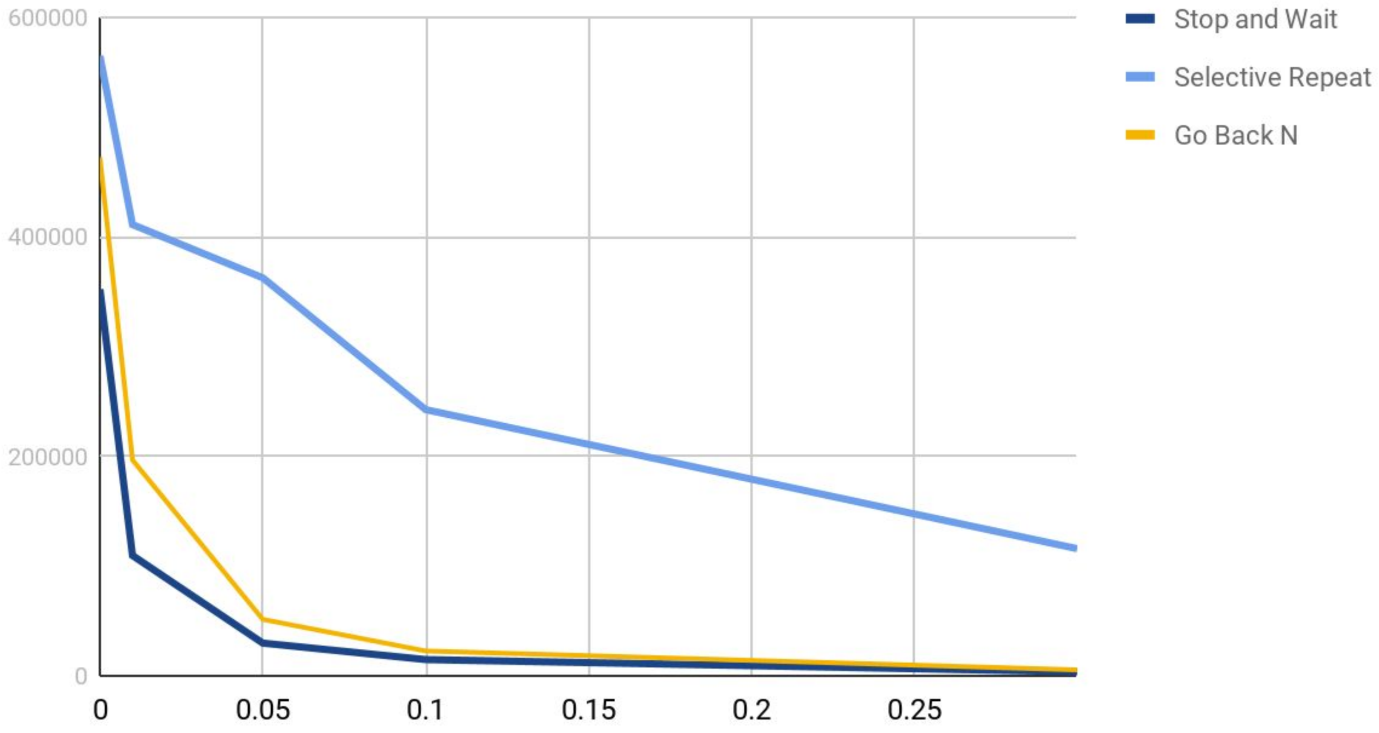
Window Size = 100



Window Size = 500



Window Size = 1000


Stop and Wait
Selective Repeat
Go Back N