CONTENTS:

Task 1:

```
C:\Users\hrao1\Desktop\Ple>python client.py 152.46.18.206 7735 Sample.pdf 1 500
```

The VCL IP address image:

```
hbalaji@bn18-206:~/Desktop/Project$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:10:01:fe
          inet addr:10.25.5.255  Bcast:10.25.15.255  Mask:255.255.240.0
          inet6 addr: fe80::250:56ff:fe10:1fe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14068 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1424 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1462340 (1.4 MB)  TX bytes:512008 (512.0 KB)

eth1      Link encap:Ethernet  HWaddr 00:50:56:10:01:ff
          inet addr:152.46.18.206  Bcast:152.46.23.255  Mask:255.255.248.0
          inet6 addr: fe80::250:56ff:fe10:1ff/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62676 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14271 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25375721 (25.3 MB)  TX bytes:13757352 (13.7 MB)
```

The picture above shows the client connection (localhost) to the server (VCL machine) whose IP is 152.46.18.206

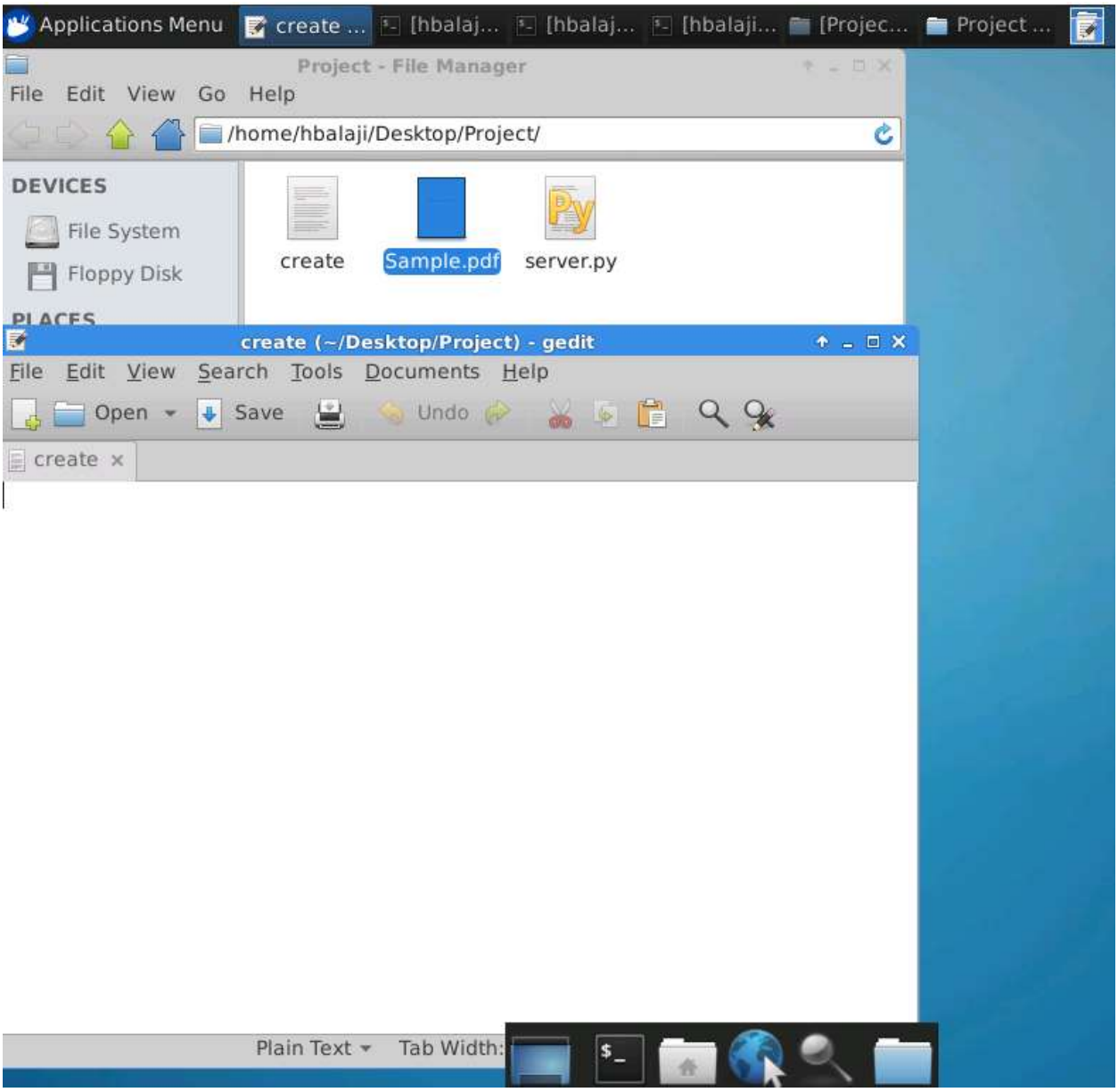
The picture below shows the traceroute from the localhost to the server

```
Tracing route to bn18-206.dcs.mcnc.org [152.46.18.206]
over a maximum of 30 hops:

  1     *       23 ms     4 ms  192.168.1.1
  2    18 ms    17 ms    19 ms  142.254.207.229
  3   220 ms    98 ms   109 ms  cpe-174-111-117-197.triad.res.rr.com [174.111.117.197]
  4    59 ms    10 ms    13 ms  cpe-024-025-063-144.ec.res.rr.com [24.25.63.144]
  5    15 ms    13 ms    42 ms  be10.drhmncev02r.southeast.rr.com [24.93.64.82]
  6    80 ms    14 ms    81 ms  24.93.67.200
  7    20 ms    20 ms   103 ms  gig10-0-0.chrlncsa-rtr1.carolina.rr.com [24.93.64.27]
  8   138 ms    23 ms    28 ms  cpe-024-074-247-065.carolina.res.rr.com [24.74.247.65]
  9    28 ms   104 ms    99 ms  rrcs-24-172-68-245.midsouth.biz.rr.com [24.172.68.245]
 10    22 ms    21 ms    21 ms  rrcs-98-101-20-135.midsouth.biz.rr.com [98.101.20.135]
 11    41 ms    27 ms    50 ms  rrcs-24-172-64-46.midsouth.biz.rr.com [24.172.64.46]
 12   197 ms   106 ms    96 ms  rtp-ip-asr-gw-to-hntvl-ip-asr-gw.ncren.net [128.109.9.217]
 13    49 ms    99 ms    99 ms  mcnc-dcs-to-rtp-ip-asr-gw.ncren.net [128.109.191.98]
 14    67 ms   103 ms    36 ms  152.46.46.18
 15    42 ms   104 ms    96 ms  152.46.55.4
 16   145 ms    99 ms    99 ms  bn18-206.dcs.mcnc.org [152.46.18.206]

Trace complete.
```

File on the VCL server:



The entire experiment used the following parameters:

Probabilistic loss (as generated randomly from the server), p=0.05
Maximum segment size (MSS)=500

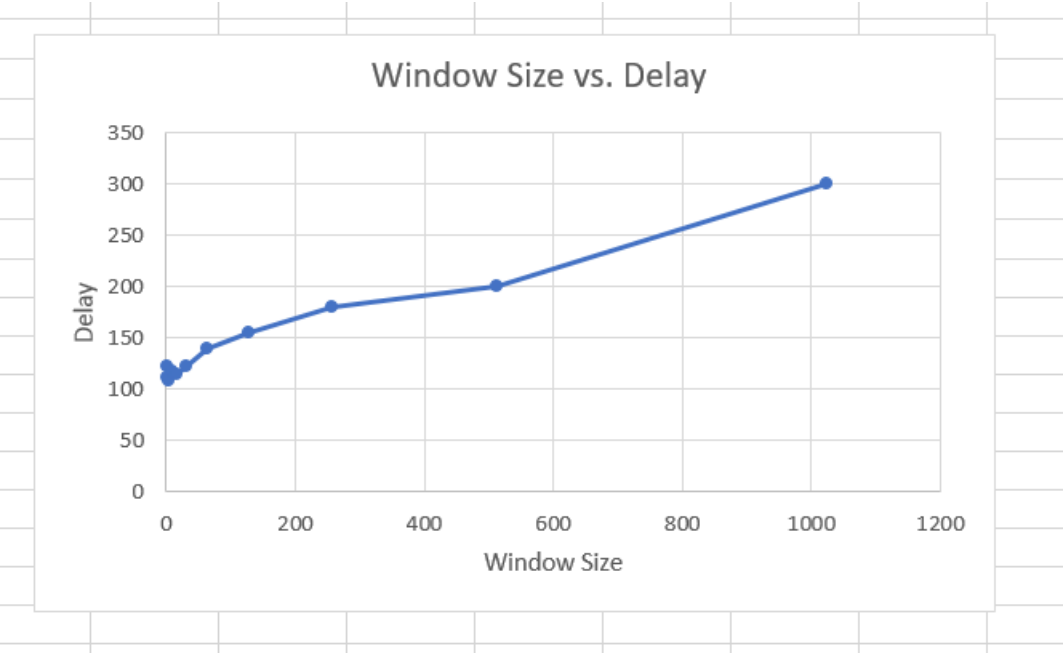
The window size was varied as shown in the table below. The table average RTT gives the average of five attempts of transfer of file from the client to the host.

The file size=**1711KB** which is slightly over 1 MB. For convenience, the delay values were rounded from decimal to the nearest whole number

Round trip time (RTT) between server and client: **~127ms**

Window Size	Average delay(s) (sum/5)
1	**122**(121.5+122.3+123.2+119.2+120.1)
2	**111(**110.3+112.5+111.1+110.1+113.2)
4	**109**(108.5+110.2+108.8+110.3+111.2)
8	**118**(118.2+117.3+119.4+118.9+117.8)
16	**115**(114.8+115.2+115.8+113.9+114.9)
32	**123**(122.9+123.5+123.0+123.5+122.6)
64	**139**(138.2+139.2+139.1+137.1+139.5)
128	**155**(154.6+155.1+153.8+154.4+155.1)
256	**180**(179.6+181.2+182.1+180.1+181.6)
512	**201**(200.8+202.4+201.8+200.9+201.6)
1024	**300**(298.3+299.1+302.8+301.8+398.8)

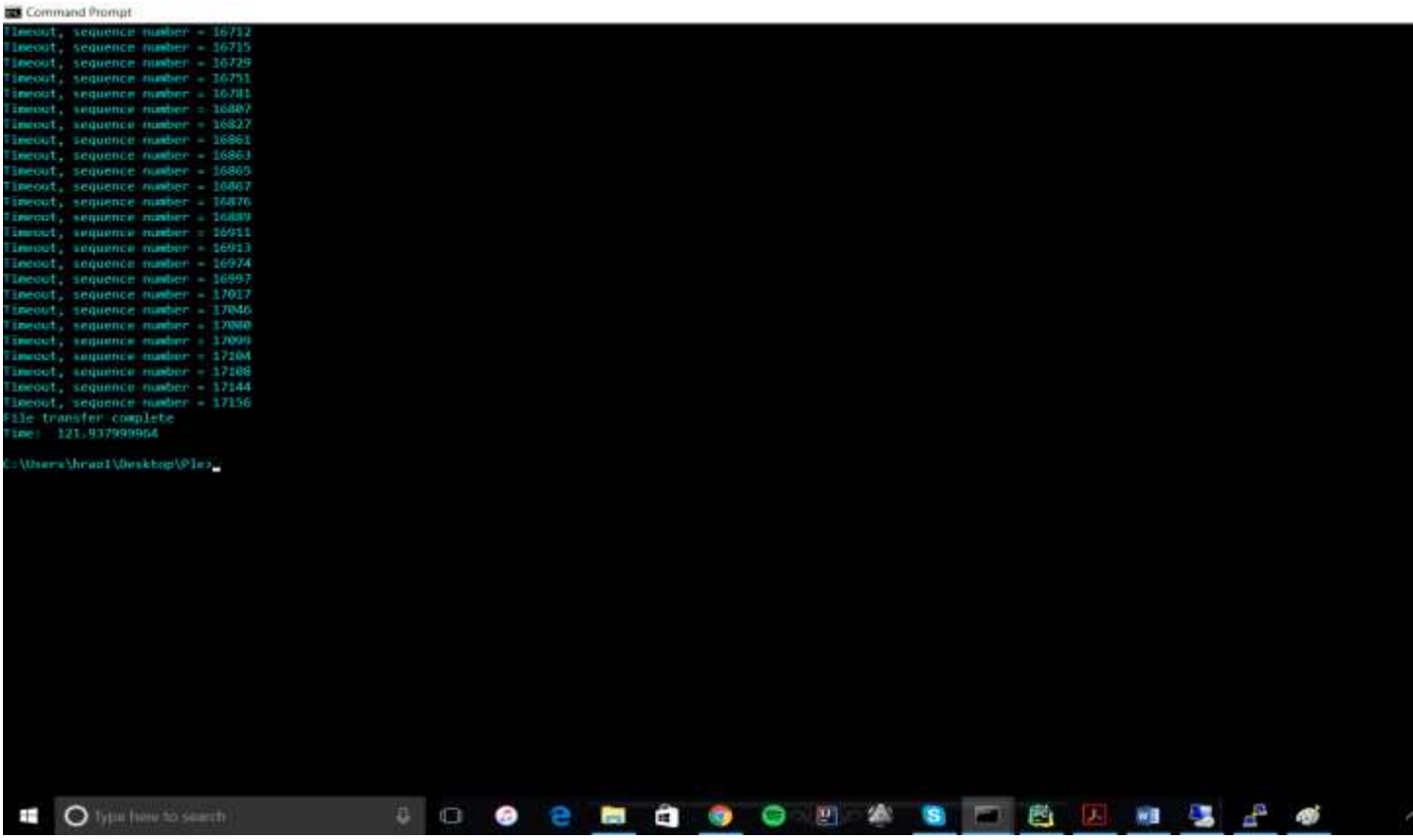
The plot for N vs. Delay is a linear graph with slight anomalies where for example the delay dips a bit when the window size is about 32 and then gradually increases with the rise in window size.



Sample Screenshots:

P.S: We decided to attach just a few of the 50 screenshots for the various readings to give a gist of how the variations were observed with the change in window size

Window size 1 with completion time (below):



The captures below were taken for window **size 2** :

```
C:\Users\hrao1\Desktop\Ple>python client.py LAPTOP-00EHEO26 7735 fake.txt 2 500
Timeout, sequence number = 7
Timeout, sequence number = 138
Timeout, sequence number = 172
Timeout, sequence number = 211
Timeout, sequence number = 222
Timeout, sequence number = 267
Timeout, sequence number = 269
Timeout, sequence number = 306
Timeout, sequence number = 323
Timeout, sequence number = 348
Timeout, sequence number = 349
Timeout, sequence number = 384
Timeout, sequence number = 404
Timeout, sequence number = 424
Timeout, sequence number = 427
Timeout, sequence number = 443
Timeout, sequence number = 499
Timeout, sequence number = 500
Timeout, sequence number = 501
Timeout, sequence number = 502
Timeout, sequence number = 528
Timeout, sequence number = 544
Timeout, sequence number = 614
Timeout, sequence number = 643
Timeout, sequence number = 649
Timeout, sequence number = 17139
Timeout, sequence number = 17162
File transfer complete
Time:  110.056999922
```

For window **size 4** (below):

```
le>python client.py LAPTOP-00EHEO26 7735 fake.txt 4 500
Timeout, sequence number = 17160
File transfer complete
Time:  108.698999882

C:\Users\hrao1\Desktop\Ple>
```

Captures for window **size 8**:

```
File transfer complete
Time:  120.570000172

C:\Users\hrao1\Desktop\Ple>
```

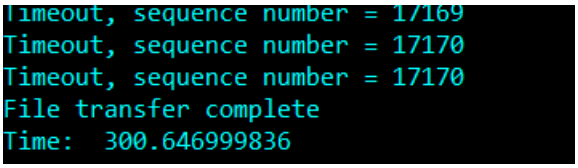
Window size **16**:

```
C:\Users\hrao1\Desktop\Ple>python client.py LAPTOP-00EHEO26 7735 fake.txt 16 500
Timeout, sequence number = 19
Timeout, sequence number = 26
Timeout, sequence number = 48
Timeout, sequence number = 65
Timeout, sequence number = 85
Timeout, sequence number = 141
Timeout, sequence number = 141
Timeout, sequence number = 154
```

```
Timeout, sequence number = 17148
Timeout, sequence number = 17166
File transfer complete
Time:  115.921999931
```

The above process was continued five times each for every window size in the table. Just for demonstration, we show for a window size of 1024.

```
:\Users\hrao1\Desktop\Ple>python client.py LAPTOP-00EHEO26 7735 fake.txt 1024 500
imeout, sequence number = 10
imeout, sequence number = 45
imeout, sequence number = 105
imeout, sequence number = 115
imeout, sequence number = 123
imeout, sequence number = 137
imeout, sequence number = 141
imeout, sequence number = 190
imeout, sequence number = 196
imeout, sequence number = 232
imeout, sequence number = 244
imeout, sequence number = 265
imeout, sequence number = 271
imeout, sequence number = 281
imeout, sequence number = 283
```



It is also to be noted that a file reaching the running VCL server took 46 minutes for a file size of 1.2MB and a window size of 32, 72 minutes (so on and so forth) for a window size of 1and we were not able to take readings for each repetitive value. We then decided to make sure that the average values are indeed computed and created another server on the localhost with a different port number and proceeded with our observations.

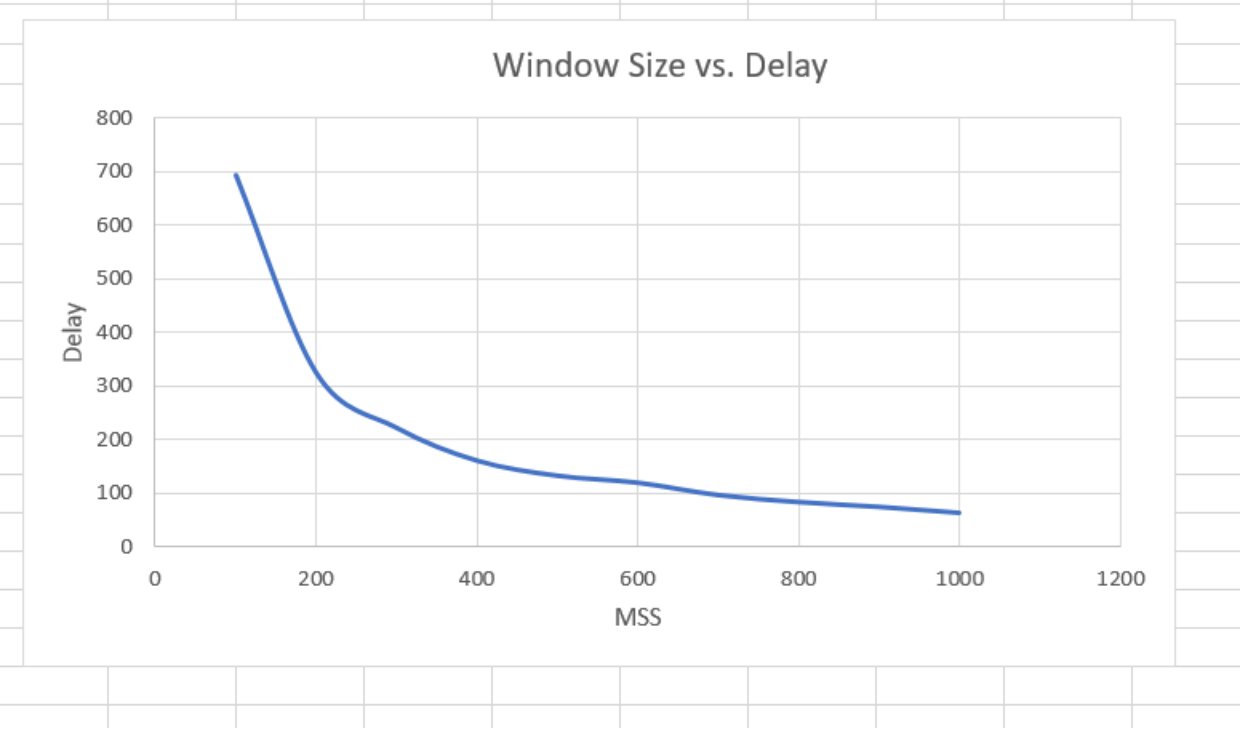
Task 2:

For this task, the constant factors are p=0.05, window size=64. We vary the MSS from 100 to 1000 and observe this effect on delay as plotted in the graph below. The readings are whole number average values as described in the first task.

The file size=**1711KB** which is slightly over 1 MB. For convenience, the delay values were rounded from decimal to the nearest whole number

MSS(bytes)	Average Delay(s)
100	**694**(692.8+694.1+694.3+692.1+694.3)
200	**325**(323.9+325.8+326.1+326.8+325.8)
300	**223**(223.8+222.5+223.2+223.1+222.9)
400	**162**(162.8+161.7+162.3+161.2+161.2)
500	**134**(133.8+134.2+134.1+133.6+134.5)
600	**121**(120.8+119.8+121.2+122.0+120.3)
700	**98**(97.3+99.1+98.20+98.6+98.3)
800	**85**(84.9+83.6+85.2+85.4+85.9)
900	**76**(75.8+76.1+76.3+75.6+76.3)
1000	**65**(63.8+64.9+65.6+66.1+65.8)

The plot obtained below is the desired one as the increase in maximum segment size increases the amount of data that can be transferred in each packet. This ensures that more data is transferred in a lesser amount of time. This in turn ensures the overall delay be reduced due to the very reason mentioned above.



Few screenshots for the experiment for various values of MSS

MSS=200

```
imeout, sequence number = 42896
imeout, sequence number = 42897
imeout, sequence number = 42909
imeout, sequence number = 42917
imeout, sequence number = 42926
ile transfer complete
ime:  325.717999935
```

MSS=400

```
Timeout, sequence number = 21318
Timeout, sequence number = 21333
Timeout, sequence number = 21360
Timeout, sequence number = 21394
Timeout, sequence number = 21447
Timeout, sequence number = 21454
File transfer complete
Time:  162.347000122
```

MSS=500

```
Timeout, sequence number = 17069
Timeout, sequence number = 17152
Timeout, sequence number = 17162
Timeout, sequence number = 17162
Timeout, sequence number = 17164
File transfer complete
Time:  134.838000059
```

MSS=1000

```
Timeout, sequence number = 8330
Timeout, sequence number = 8340
Timeout, sequence number = 8352
Timeout, sequence number = 8356
Timeout, sequence number = 8389
Timeout, sequence number = 8405
Timeout, sequence number = 8415
Timeout, sequence number = 8468
Timeout, sequence number = 8478
Timeout, sequence number = 8486
Timeout, sequence number = 8544
File transfer complete
Time:  64.7699999809
```

Task 3:

Constant factors:

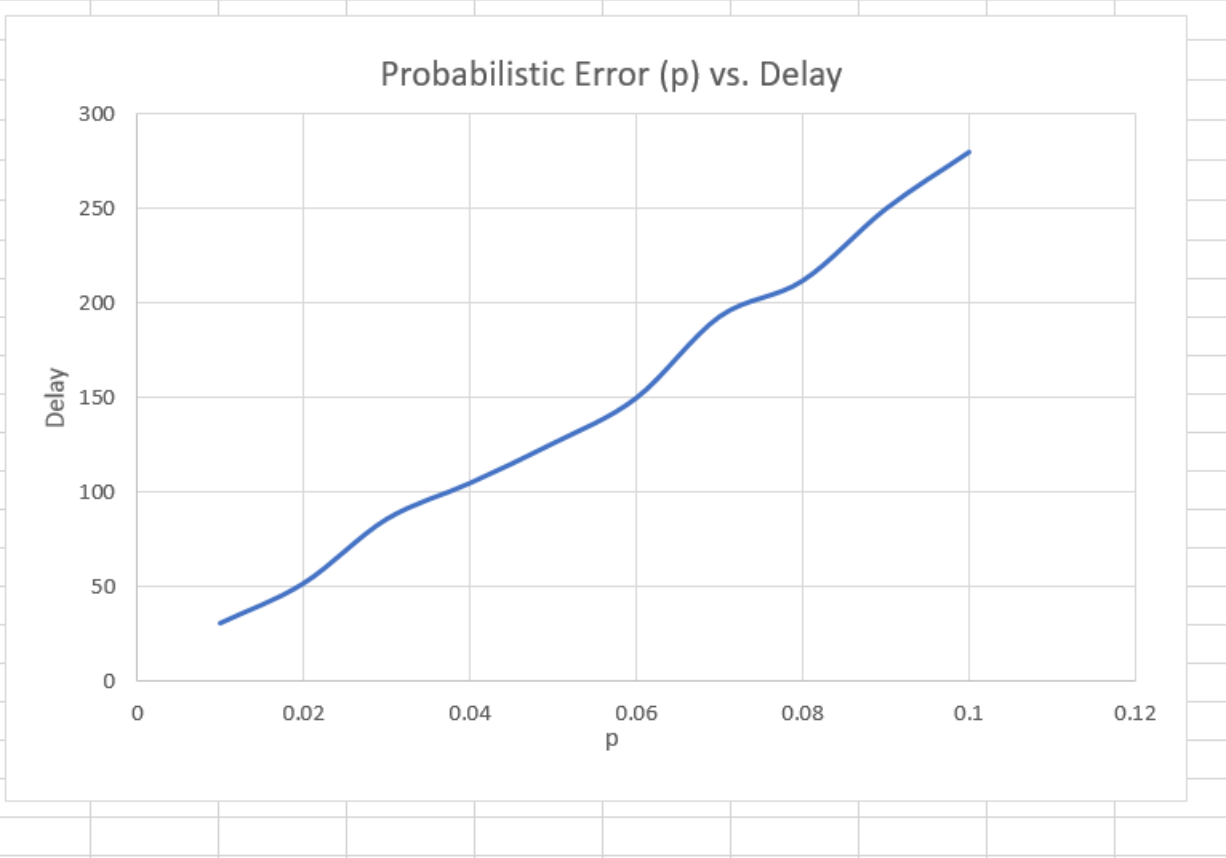
N=64
MSS=500

Table involves increasing the probabilistic error rate from 0.01 to 0.0 with an increment of 0.01 in each step.

The file size=**1711KB** which is slightly over 1 MB. For convenience, the delay values were rounded from decimal to the nearest whole number

Probabilistic Error(p)	Average Delay(s)
0.01	**31**(30.7+31.3+30.2+31.8+31.9)
0.02	**52**(51.8+54.3+48.2+51.1+52.0)
0.03	**86**(85.3+86.2+84.9+85.1+87.1)
0.04	**105**(105.2+106.1+104.3+106.2+105.1)
0.05	**126**(124.8+127.3+126.8+126.1+124.0)
0.06	**150**(149.8+150.2+149.6+150.3+148.4)
0.07	**193**(192.7+193.2+193.5+192.4+192.8)
0.08	**212**(210.8+211.9+213.1+212.8+212.9)
0.09	**250**(249.8+247.3+249.3+252.1+250.2)
0.1	**280**(282.1+281.6+280.5+279.3+280.3)

The plot below shows that the increase in the error rate, increases the delay. Yes, this is the expected out. When there is an error in any given packet, the sender is responsible for retransmitting the packet received with error. This cumulatively increases the delay of the transmission of the file from the client to the server.



There are a few screenshots to show the various probabilistic error rates we manually set and the various results:

(i) Manual setting of p:

```
C:\Users\hrao1\Desktop\Ple>python server.py 7735 create 0.09
Packet loss, sequence number = 7
Packet loss, sequence number = 15
Packet loss, sequence number = 20
Packet loss, sequence number = 20
Packet loss, sequence number = 62
Packet loss, sequence number = 78
Packet loss, sequence number = 81
Packet loss, sequence number = 91
```

(ii) P=0.04

```
Timeout, sequence number = 16812
Timeout, sequence number = 16840
Timeout, sequence number = 16845
Timeout, sequence number = 16880
Timeout, sequence number = 16910
Timeout, sequence number = 16927
Timeout, sequence number = 16930
Timeout, sequence number = 16968
Timeout, sequence number = 17032
Timeout, sequence number = 17096
Timeout, sequence number = 17130
Timeout, sequence number = 17152
File transfer complete
Time:  104.580999851
```

(iii) p=0.09

```
Timeout, sequence number = 16938
Timeout, sequence number = 16938
Timeout, sequence number = 16955
Timeout, sequence number = 16965
Timeout, sequence number = 16981
Timeout, sequence number = 17000
Timeout, sequence number = 17035
Timeout, sequence number = 17039
Timeout, sequence number = 17041
Timeout, sequence number = 17048
Timeout, sequence number = 17094
Timeout, sequence number = 17129
File transfer complete
Time:  249.693000078
```

(iv) p=0.1

```
Timeout, sequence number = 17099
Timeout, sequence number = 17099
Timeout, sequence number = 17101
Timeout, sequence number = 17103
Timeout, sequence number = 17128
Timeout, sequence number = 17128
Timeout, sequence number = 17128
Timeout, sequence number = 17138
Timeout, sequence number = 17140
Timeout, sequence number = 17145
Timeout, sequence number = 17150
Timeout, sequence number = 17156
Timeout, sequence number = 17156
Timeout, sequence number = 17166
File transfer complete
Time:  279.681000233
```

README

- The server file is server.py and the client file is client.py
- The file used for transfer is named Sample.pdf
- The file on the server to which it is written is named create.txt
- The checksum code is a standard code we obtained form: (http://stackoverflow.com/questions/1767910/checksum-udp-calculation-python)
- The other resources used were:
  https://github.com/codyharrington/python-udp-filetransfer
  https://github.com/confuzzed03/Python-UDP-File-Transfer
  https://github.com/blasrodri/File-Transfer-Python
  https://github.com/gibsjose/UDPFileTransfer
  https://docs.python.org/2/library/select.html
  http://andrewtrusty.com/2006/11/22/reliable-udp-file-transfer-2/

- Further, there were some very standard functions and data fields that were mentioned in the specification of the question like use of rdt_send() and data and acknowledgement fields. We directly used them as per the instructions
- We varied the timeout values to check various scenarios (on both server and client)
- Both the server and client codes are run according to the following use cases:

  Client:

  (i) Take in the arguments from the command line
  (ii) Checksum calculation as per the standard code
  (iii) To create a packet, generate various fields and then append them in a serial manner. This appending was done using the 'pickle' serializer in python
  (iv) Once the packet is prepared, the transmission of this packet is done using the rdt_send() function, which checks for various conditions as specified in the Go-Back-N protocol

  Server:

  (i) The server takes in the arguments as specified in the question
  (ii) The server is the listener whereas the client is the sender
  (iii) The server creates the file (where data is to be written/transferred) on the fly as specified in the input (as command line arguments)
  (iv) The checksum code is the same as used in the client end
  (v) The server is responsible for generating errors (packet losses) using probabilistic packet loss generator)

- **To run the server first:** python server.py server port number (7735) file to be created (create) probabilistic error/packet loss generator (p=between 0 and 1) Hence, command: **python server.py 7735 create.txt 0.05**

- **To run the client:** python client.py hostname(local/remote host) server port number (7735) file to be transmitted (Sample.pdf) window-size(N) MSS
  Hence, command: **python client.py localhost 7735 Sample.pdf 64 1000**
  P.S:
  <Localhost>: specifies hostname of device being run on
  we obtained it using socket.hostname()

Each program- the server and the client is divided into several use cases and the logic for each of the use case is commented in the code with a marking, **UC** and explained:

Client:

UC I: Read the arguments from the command line

UC II: Preparation of packet- append various fields (serialize) using Pickle module. The specifications of the various fields are given in the project description

UC III: call the function packet_generator() for preparation of packet with various fields
buffer the packet and send to socket for transfer
check if the correct acknowledgement is received

Server:

UC I: On receiving a packet, ensure the checksum for correctness of the packet using the checksum code

UC II: On receiving the packet, check if the packet is in sequence

UC III: writing data into a specified file