‘Better RDT over UDP’ than TCP

Sri Hari Malla - CS19BTECH11039

G Venkata Surya Sai - CS19BTECH11014

Veeramalli Sathwik - CS19BTECH11022

Vajire Pramod Kishanrao - CS18BTECH11046

K Sri Teja - CS19BTECH11002

February 20, 2022

Contents

1 Task - 1

1.1 Base:

Two linux vms are connected directly through a linux bridge as shown in the reference modules. Now that we have installed a direct connection between the two, we ran ping command between the two vms and it's output suggested that the connection is well established.

1.2 FTP server:

We know that FTP is a way to upload, download and transfer one file from one place to another in a network. By default ftp clients are installed in the linux vms but one needs to install FTP server in order to manage transfer of files. File first gets transferred to server and then to the machine targeted. So FTP server is mandatory for file transfer. We've used vsFTPd server in the server to manage the file transfer.

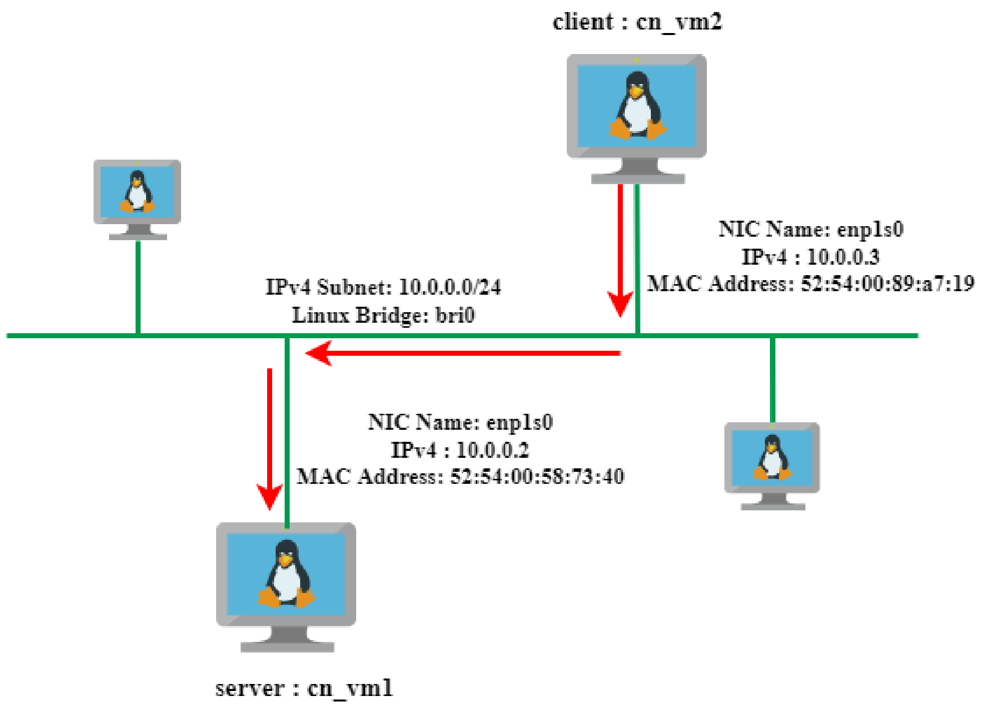


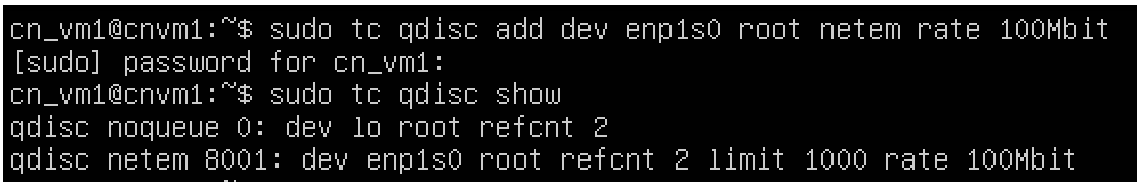
Figure 1: Network Diagram

1.3 Establishing link with speed 100Mb without delay and packet loss:

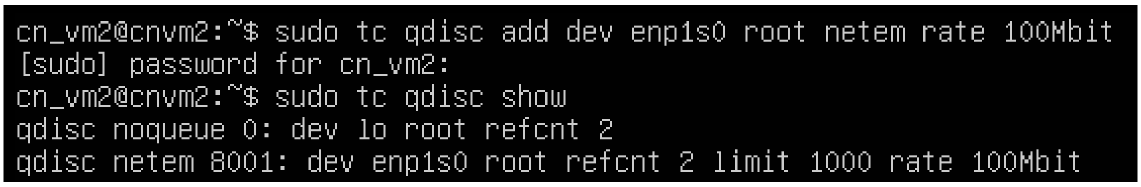
With the reference to the `tc`(traffic control) manual given, we've established a link speed of 100Mb in both the servers. Let's go through the one command which establishes the speed.
command : `$ sudo tc qdisc add dev enp1s0 root netem rate 100Mbit`

- sudo : root previliges
- tc : traffic control
- qdisc : modifying the scheduler(queuing discipline)
- dev enp1s0 : applied on device `enp1s0`
- root : to modify the out traffic scheduler
- netem : using the network emulator
- rate 100Mbit : network property that is modified and value to be applied.

While applying any command with sudo, we need to be extra careful, so we did a small research on what each of the thing does in the command. Below is the picture depicting the same being done on both the vms.



```
cn_vm1@cnvm1:~$ sudo tc qdisc add dev enp1s0 root netem rate 100Mbit
[sudo] password for cn_vm1:
cn_vm1@cnvm1:~$ sudo tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8001: dev enp1s0 root refcnt 2 limit 1000 rate 100Mbit
```

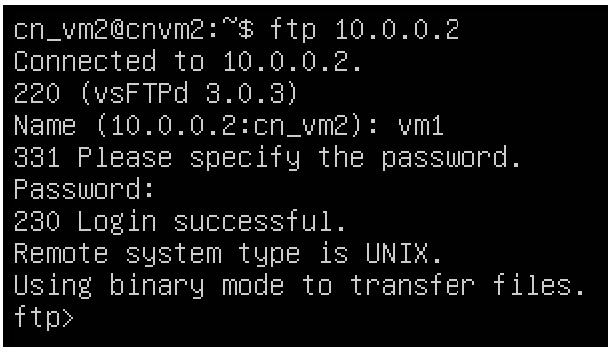


```
cn_vm2@cnvm2:~$ sudo tc qdisc add dev enp1s0 root netem rate 100Mbit
[sudo] password for cn_vm2:
cn_vm2@cnvm2:~$ sudo tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8001: dev enp1s0 root refcnt 2 limit 1000 rate 100Mbit
```

Using command : `$ sudo tc qdisc show` one can see the values applied.

1.4 FTP login:

As server is installed and set up, we need to login to the server to get the services running. There are three modes of login in to FTP server, but we use local user mode to connect and set things up.

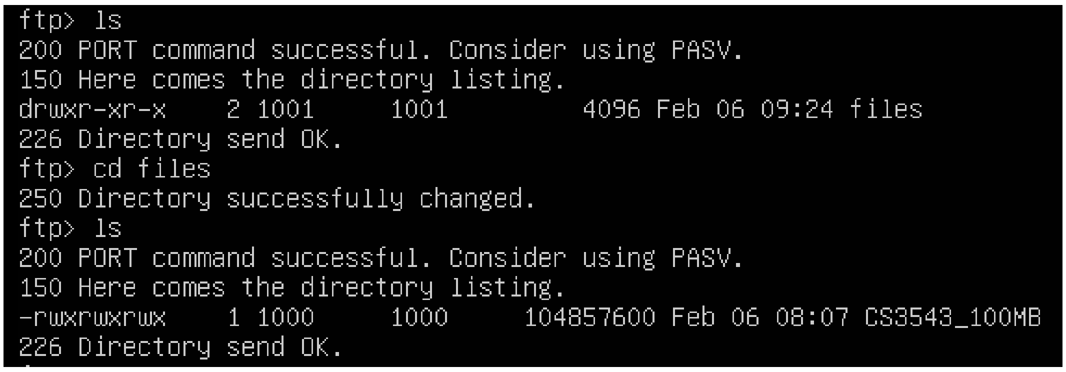


```
cn_vm2@cnvm2:~$ ftp 10.0.0.2
Connected to 10.0.0.2.
220 (vsFTPd 3.0.3)
Name (10.0.0.2:cn_vm2): vm1
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Figure 2: vm2 ftp login

1.5 FTP Navigation:

Navigate to the file's path and we are set to transfer files.
Now we can see the file `CS3543_100MB` which is actually 100Mb in size.



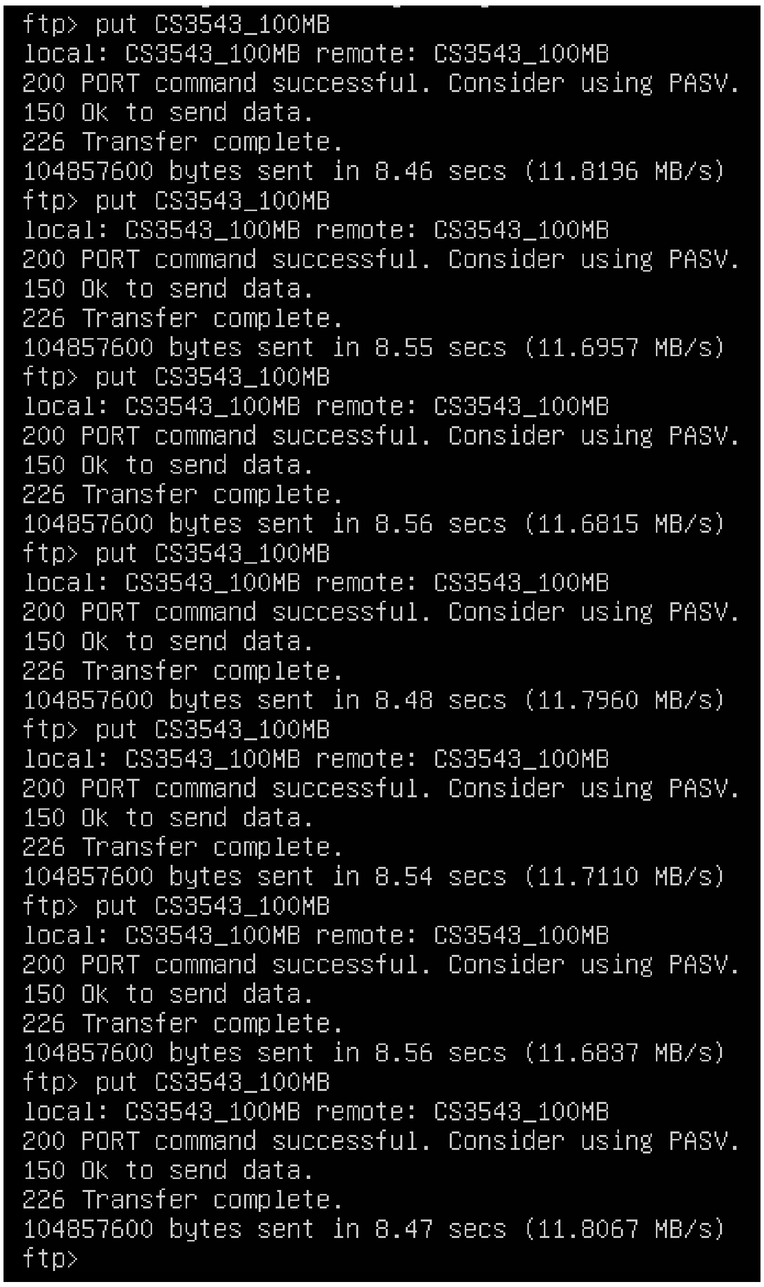
```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 1001     1001         4096 Feb 06 09:24 files
226 Directory send OK.
ftp> cd files
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rwxrwxrwx    1 1000     1000    104857600 Feb 06 08:07 CS3543_100MB
226 Directory send OK.
```

Figure 3: vm2 ftp navigation

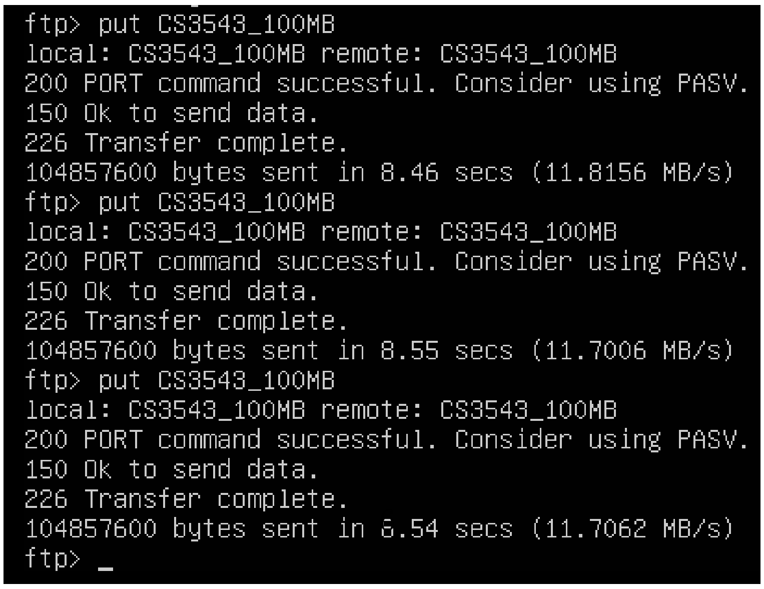
1.6 Measuring performance in vms:

As explicitly mentioned in the assignment statement, we are set to send file from client to server using FTP. Set the permissions to read and write in order to use both `get and put commands`. We've also played using get command. Some Observations:

- pasv - passive mode file transfer which works behind firewalls.
- 226 - reply code is sent by the server before closing the data connection after successfully processing the client command.

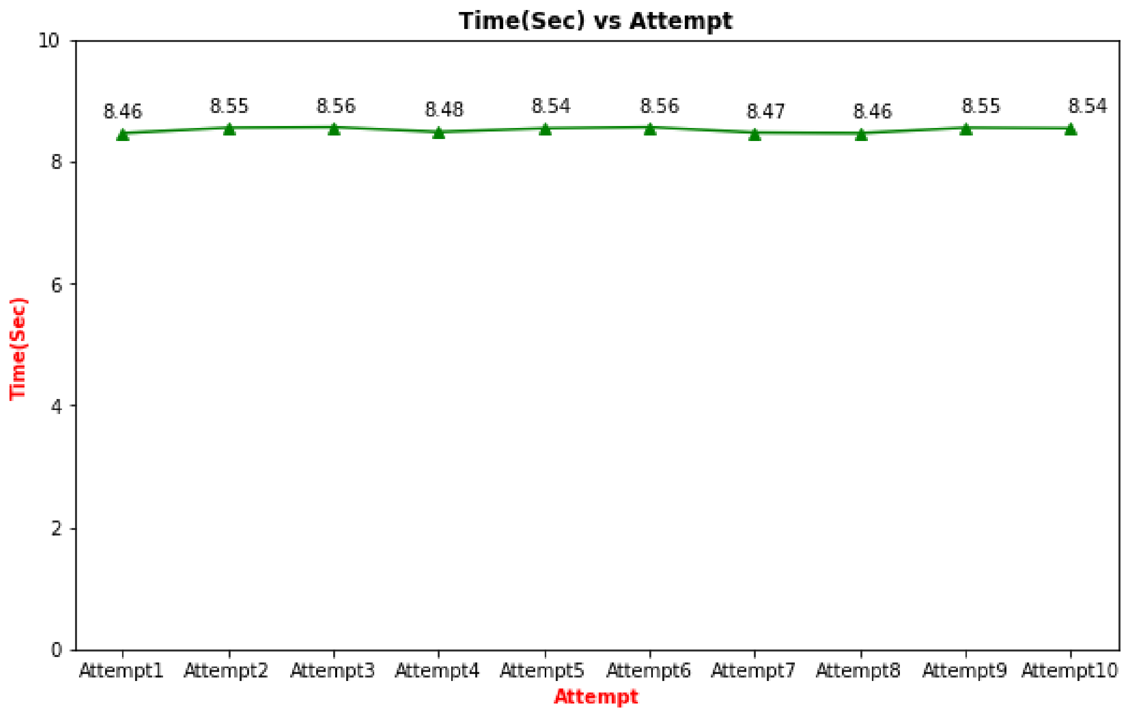


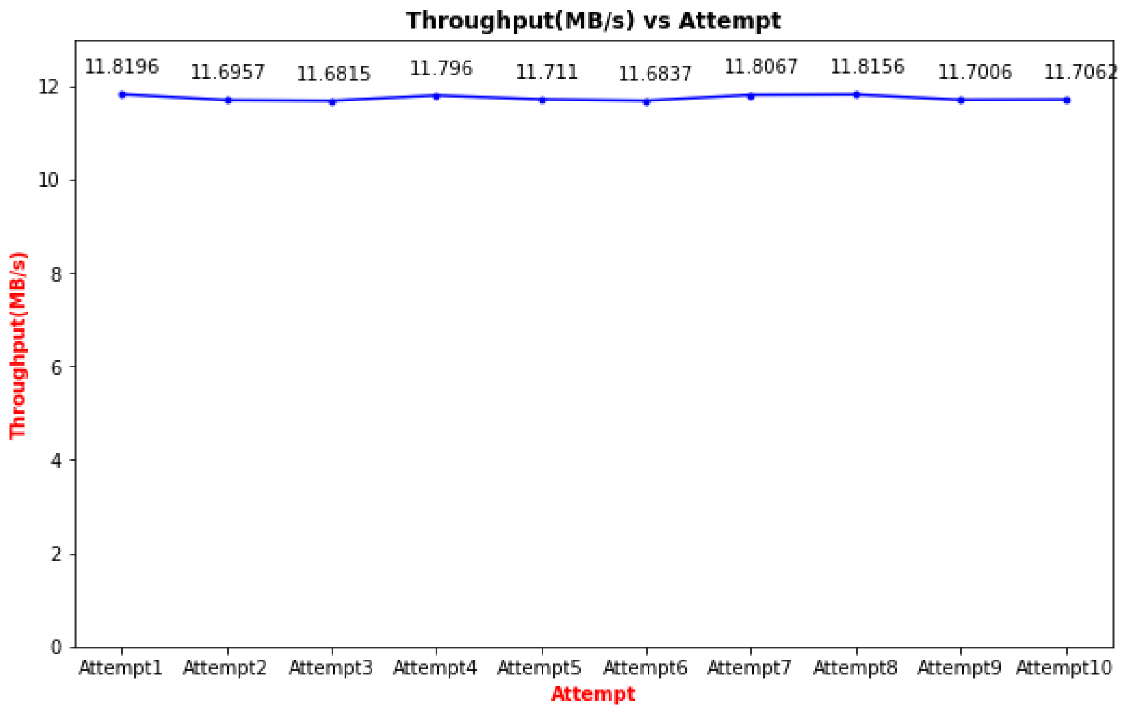
```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.46 secs (11.8196 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.55 secs (11.6957 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.56 secs (11.6815 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.48 secs (11.7960 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.54 secs (11.7110 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.56 secs (11.6837 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.47 secs (11.8067 MB/s)
ftp>
```



```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.46 secs (11.8156 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.55 secs (11.7006 MB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 8.54 secs (11.7062 MB/s)
ftp> _
```

Graphical Representation:





Tabular format of recorded observations:

Attempt Number	Time Taken(sec)	Throughput(MB/s)
1	8.46	11.8196
2	8.55	11.6957
3	8.56	11.6815
4	8.48	11.7960
5	8.54	11.7110
6	8.56	11.6837
7	8.47	11.8067
8	8.46	11.8156
9	8.55	11.7006
10	8.54	11.7062
Average	**8.517**	**11.74166**

1.7 Verification:

It is clear from the above average calculations that average time taken is 8.517 seconds and average throughput observed is 11.7416 MB/s.

The observed throughput is 11.7416 MB/s which in turn is almost equals to 94Mbps which is close to the speed we established using the link. The loss in speed must be due to synchronization of the packets sent.

```
we know that,
    avg_time_taken = file_size/avg_throughput
    => 100(MB)/11.7416(MB/s) = 8.51667 s
```

which is exactly what we got from the observed calculation.

1.8 Establishing 100Mbit speed link with 5% loss and 50ms delay :

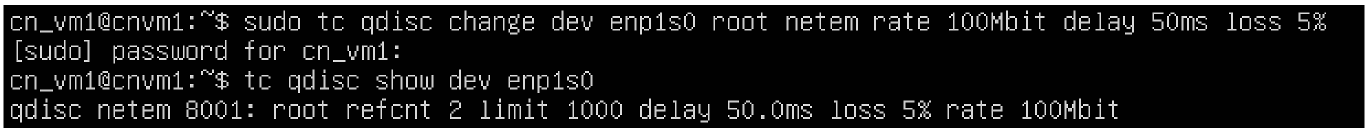
We've already seen how the link can be established with "tc" command between two vms. Now we need to add packet loss and delay.
command : `sudo tc qdisc change dev enp1s0 root netem rate 100Mbit delay 50ms loss 5%`

We've seen the command in detail in the above sections and let's see the new terms:

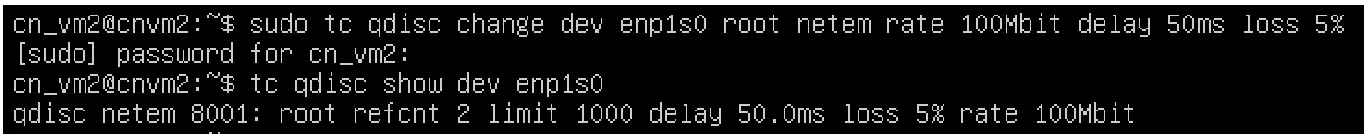
- change : change the existing configuration
- delay value : add delay of value(in ms)

- loss value : add packet loss of value(in %)

One can find the attached screenshots of the same being applied on both the vms. As explained above, use `show` command to view the actual link set in the vms.



```
cn_vm1@cnvm1:~$ sudo tc qdisc change dev enp1s0 root netem rate 100Mbit delay 50ms loss 5%
[sudo] password for cn_vm1:
cn_vm1@cnvm1:~$ tc qdisc show dev enp1s0
qdisc netem 8001: root refcnt 2 limit 1000 delay 50.0ms loss 5% rate 100Mbit
```

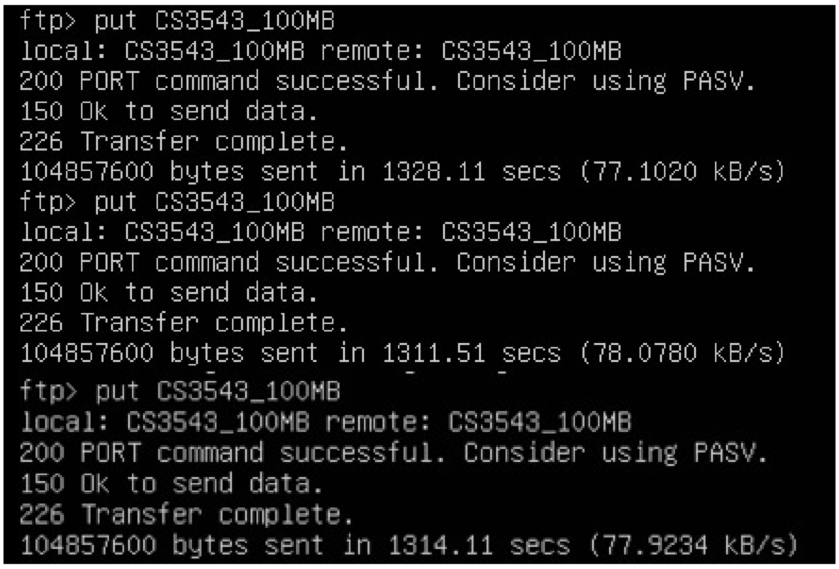


```
cn_vm2@cnvm2:~$ sudo tc qdisc change dev enp1s0 root netem rate 100Mbit delay 50ms loss 5%
[sudo] password for cn_vm2:
cn_vm2@cnvm2:~$ tc qdisc show dev enp1s0
qdisc netem 8001: root refcnt 2 limit 1000 delay 50.0ms loss 5% rate 100Mbit
```

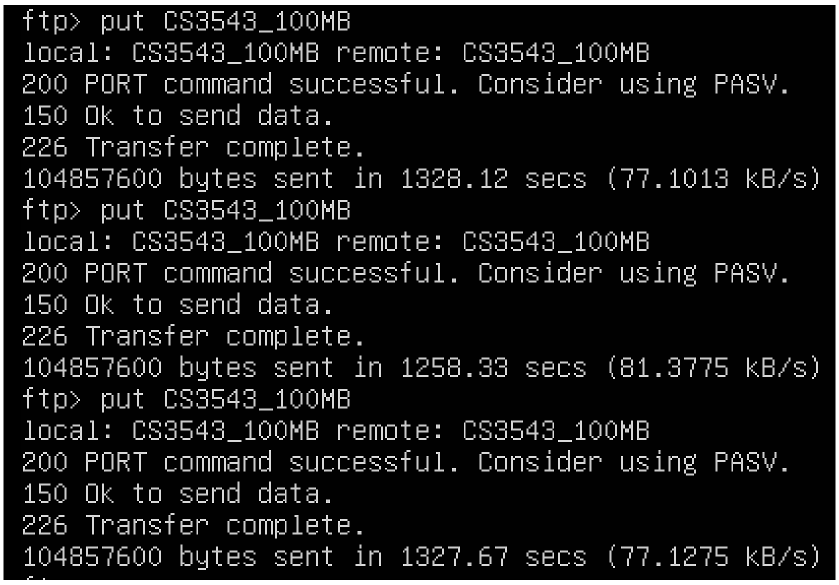
Followed the FTP login and FTP navigation same as explained above in 1.4 and 1.5

1.9 Measuring performance in vms:

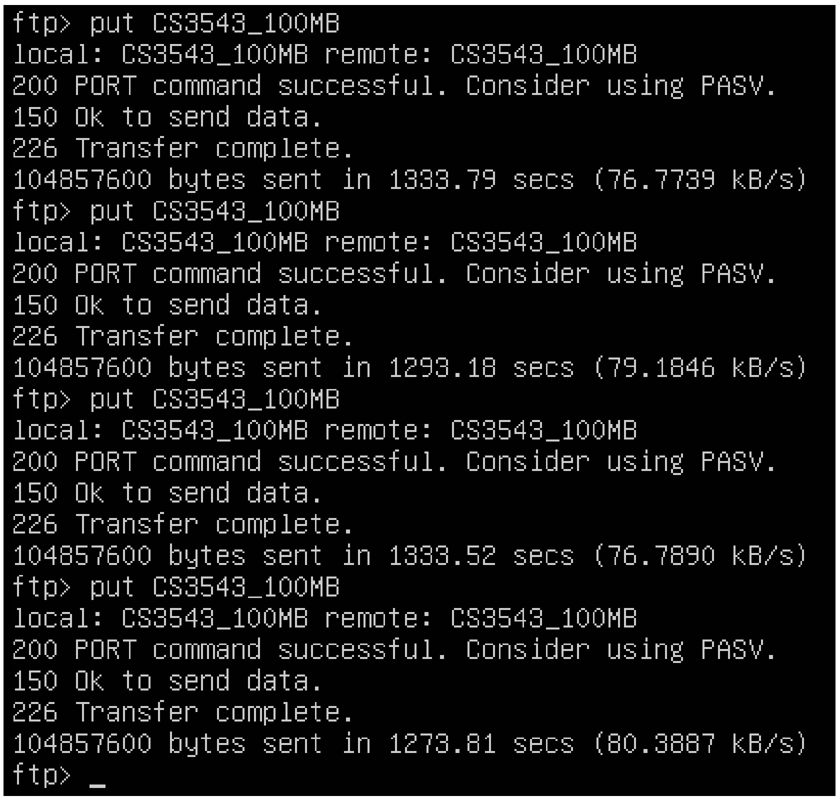
Now that we have established a link with desired packet loss and delay, and logged in to the server, now we are set to use the put command via ftp.



```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1328.11 secs (77.1020 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1311.51 secs (78.0780 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1314.11 secs (77.9234 kB/s)
```

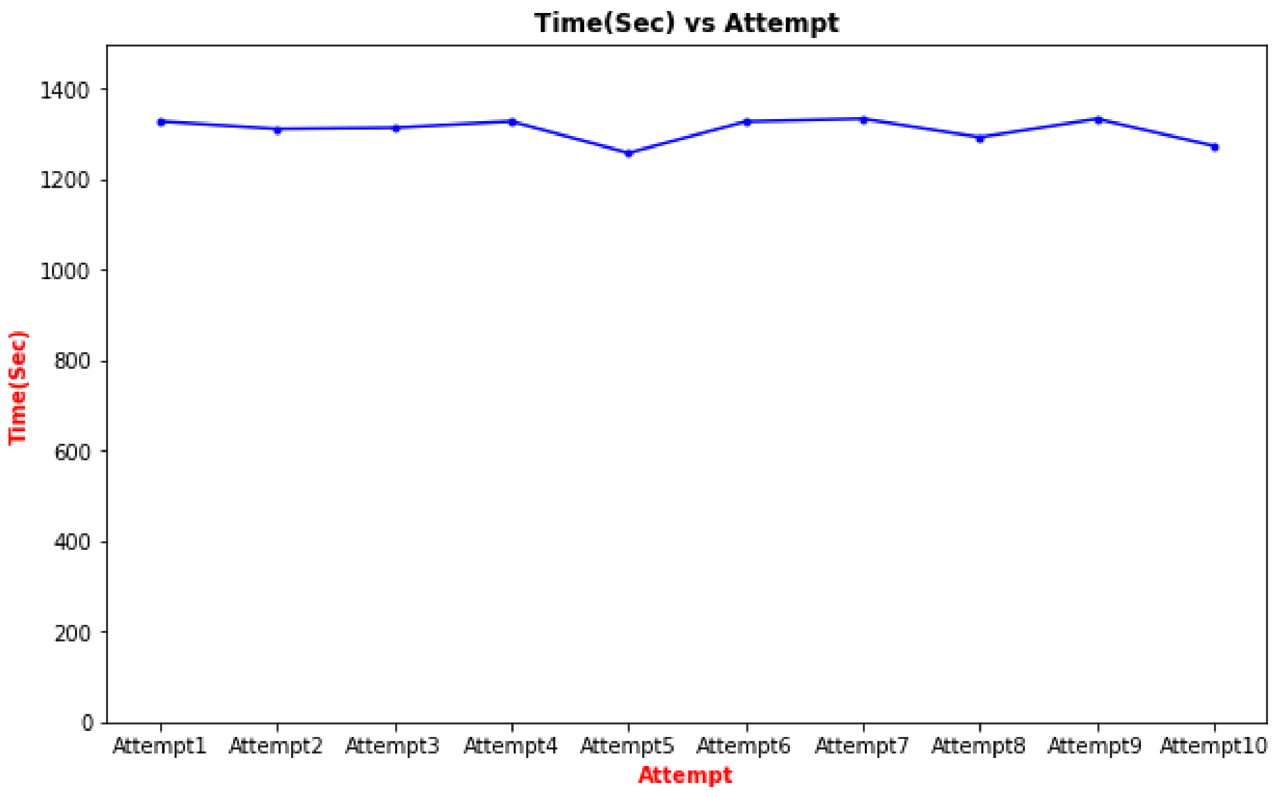


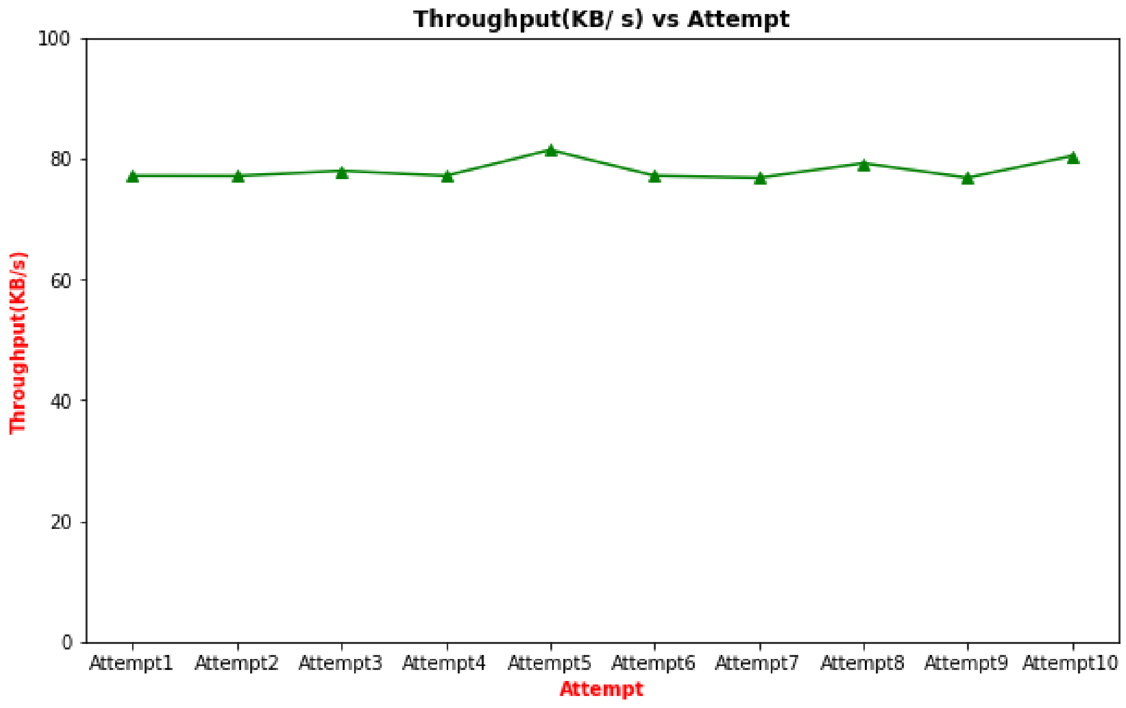
```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1328.12 secs (77.1013 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1258.33 secs (81.3775 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1327.67 secs (77.1275 kB/s)
```



```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1333.79 secs (76.7739 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1293.18 secs (79.1846 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1333.52 secs (76.7890 kB/s)
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 1273.81 secs (80.3887 kB/s)
ftp> _
```

Attempt Number	Time Taken(sec)	Throughput(kB/s)
1	1328.11	77.1020
2	1311.51	78.0780
3	1314.11	77.9234
4	1328.12	77.1013
5	1258.33	81.3775
6	1327.67	77.1275
7	1333.79	76.7739
8	1293.18	79.1846
9	1333.52	76.7890
10	1273.81	80.3887
average =	**1310.21**	**78.184**





1.10 Justification:

It is clear from the above average calculations that average time taken is 1310.21 seconds and average throughput observed is 78.184 kB/s.

The observed throughput is 78.184 kB/s as we added a packet loss and delay time. This affects the transmission and need to retransmit a packet if it was lost in order.
Observed throughput is 78.184 kB/s which is 0.07635 MB/s.

```
we know that,
    avg_time_taken = file_size/avg_throughput
    => 100(MB)/0.07635(MB/s) = 1309.7 s
```

which is exactly what we got from the observed calculation.

2 Task 2

2.1 Header:

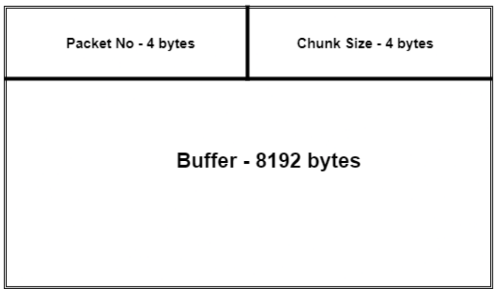
As packets need to be sent via sockets and data need to be sent we need to structure a packet which we can get maximum out of it. So we created a class named packet which stores the packet number, sizeof data being carried in buffer and buffer itself of fixed size.

Why packetno? chunksize? Is buffer alone not sufficient?
As we never know which packet is received at the server end we need to maintain the packet number in order to keep track of where in the file the packet needs to be written. Also we should be able to request the sender to send a specific chunk of data which should be easy if we maintain packet number.
Also the C++ has ostream support to use redirection operator to write data to files. The problem with that is the operator stops when it encounters the first null terminated character and binary file itself may contains many null terminated characters so the data will be lost. So we keep track of chunk size which makes it easier to write to the file specifying the size to write. And the presence of buffer is trivial.

The max buffer size is set to 8192. Theoretically the max size is set to 64kb and we are set to use the structure.

```
structure of packet:
class packet{
public:
    int packet_no;
    int chunk_size;
    char buffer[bufferSize];
};

int(packet no) -> 4 bytes
int(chunk size) -> 4 bytes
char(bufferSize) -> 8192 bytes
```



2.2 Why C++?

We've done the coding part in C++. Now that a question arises on why use C++ when there are many high languages which offers ease of use. The main reason we used C++ is because it is closer to the OS and hardware which results in faster execution of code to a considerable extent. As Unix sockets are implemented in C/C++, we are more than convinced to use the C++ language to code the programs which also offers high level language specifications which is not available in C. It also gives a clear impression of huge control over packets and the as we need to take more care of every line it is less error prone while programming.

2.3 Features:

- packet loss detection:
- Acks/nAcks to inform Missing packets
- Retransmission of missed packets
- flow control

2.4 Implementation:

2.4.1 Packet loss detection:

Ultimately while sending a large number of packets via udp socket, there will be a considerable packet loss and we need to tackle this. One straight forward way to deal with this is to acknowledge every packet and this takes way more time. We need to send acknowledgements or neg acknowledgements to sender such that it retransmits the missed packet based on the acknowledgements. One way to encounter an acknowledgement is to set time out at the sender end and there are several problems with that being set. If time is very less, then the packet might be in the middle of the transfer and again there will be a waste retransmission packet and that packet could also cause other packets to miss. To arrive at a timeout which is reasonable is a hectic task. So we send the acknowledgements and neg acknowledgements to signal the sender. We run a loop over number of packets and send the acks and nacks to sender and nacks tells the sender about the lost packets. Thus we are detecting the lost packets. Every packet has a packet number associated with it so as to communicate without any problem.

2.4.2 Acks/nAcks:

Why acks/nacks? As just said above we need to send acks/nacks to sender to notify about the missing packets. One needs to send direct ack or nack to the sender. We included both the acks and nacks to the same packet and sent it to the server. We paired the packet number and corresponding status of the packet whether it is received or not in one packet and sending it to the server. Thus both acks and nacks are being sent as a similar packet. One thread sends acks and nacks to the server. Thus making most of the multitasking.

2.4.3 Retransmission:

One the packets are sent, there is no guarantee that the packet is sent successfully. So we are sending acks/nacks from receiver to sender such that sender gets notified about missing packets. Once the nack is received for a certain packet, then it will be retransmitted. If the packet is sent successfully then the sender gets a corresponding ack and if the packet is not sent then it receives nack again and requests for retransmission until the packet gets reached.

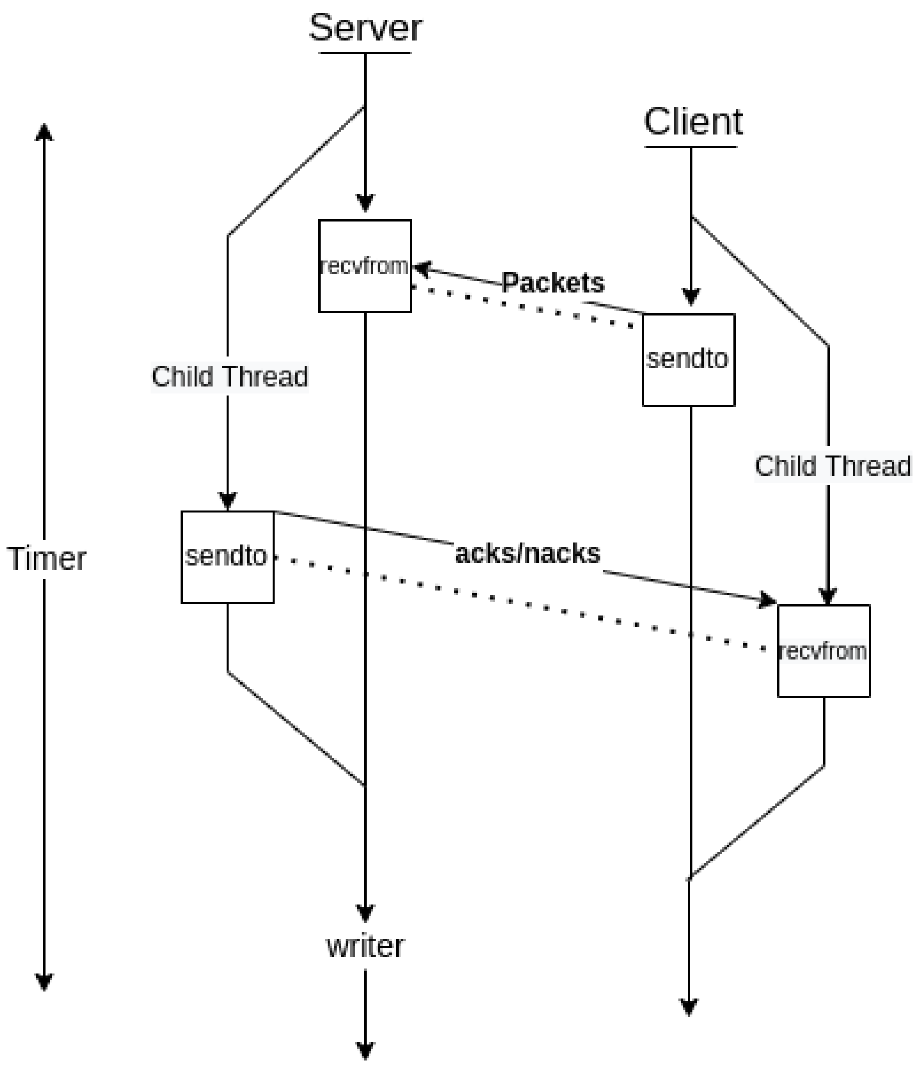
2.4.4 Flow Control:

With all being set correctly, one observes a high packet loss. We also need to deal with this issue. With Reliable data transfer being our top priority, we also should look at the packet loss to manage the loads on Vms. With that taking into consideration, we set some sleep while on sending to reduce the sending rate. Thus receiver receives at a slower rate than the usual speed reducing the nacks/acks to be sent reducing the retransmission packets thus reducing packet loss and mainly the load will be balanced on the vms or servers.

2.5 Program flow :

- Sets and inits sockets at both server and client.
- Loads the file details which needs to be sent
- Client sends the file size to server and then server calculates the number of packets to be received.
- The server will then be ready to recv packets from socket.
- The client sends the file packets one by one to server.
- The server receives the packets.
- A child thread of server sends the acks/nacks to inform the missing packets
- A child thread of client receives the acks/nacks and sends the missing packets.
- The server keeps requesting the client until all the packets are received.

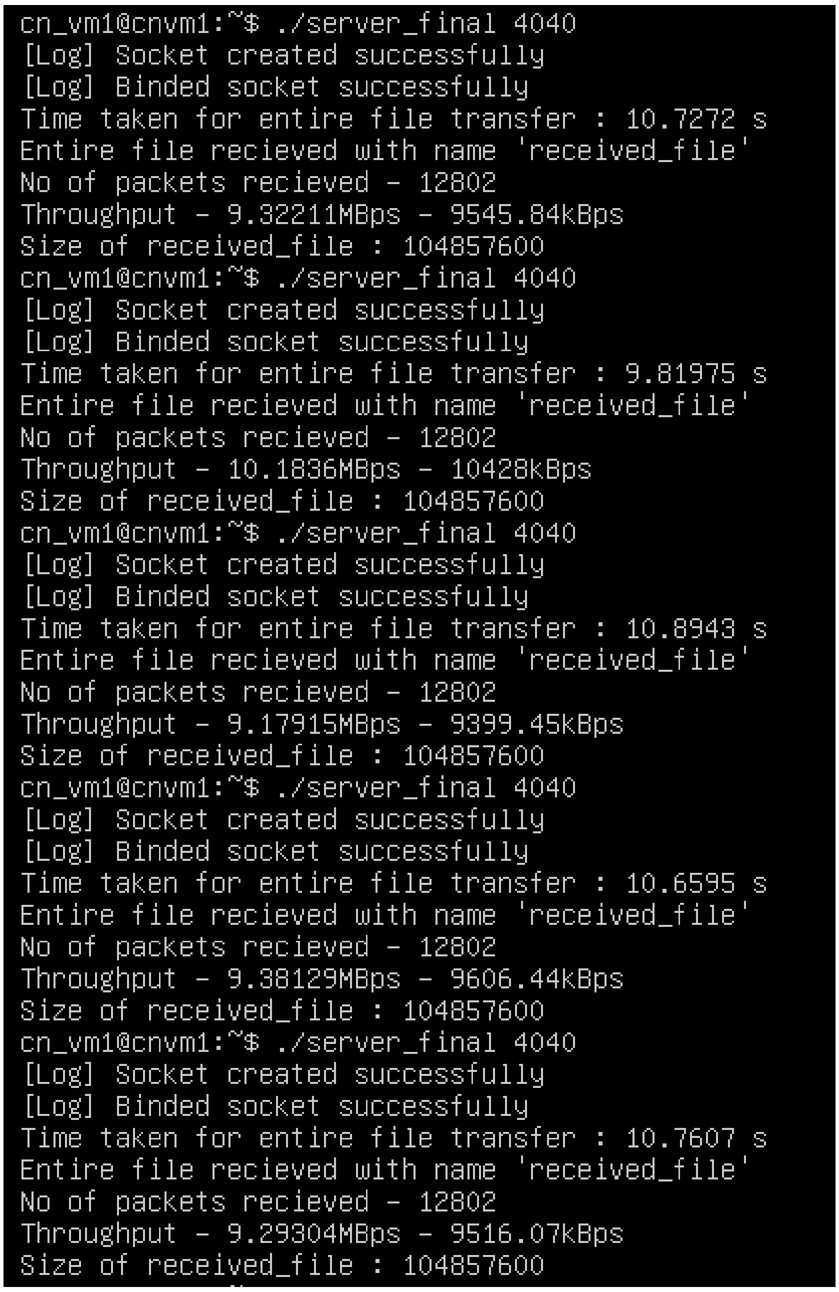
2.6 Pictorial Representation:



2.7 Measuring performance in Vms:

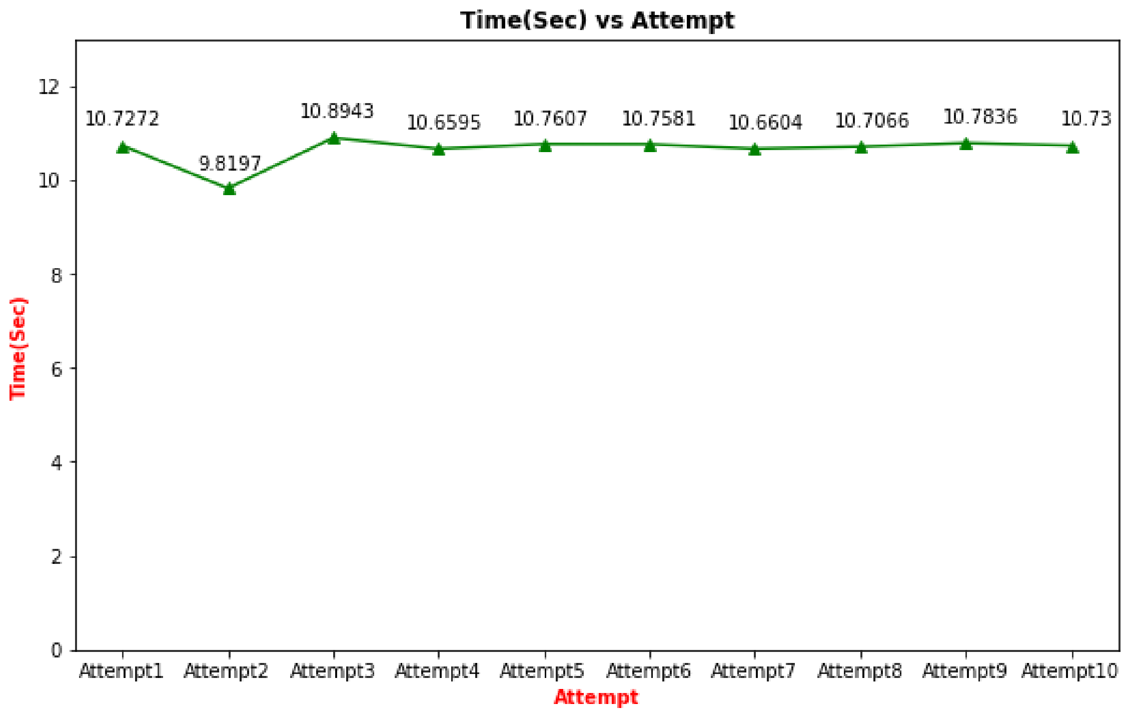
2.7.1 With 100Mbit speed link:

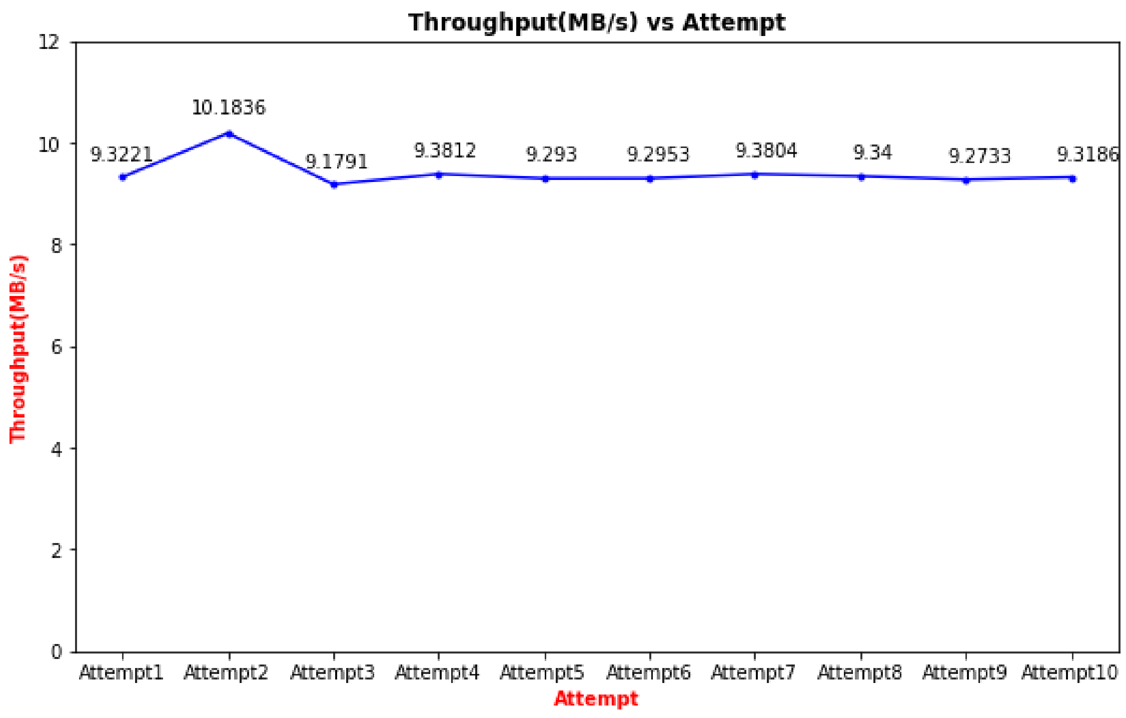
Find the attached screenshots with only 5 trails. As the report is getting longer, we decided not to attach every screenshot. One can find all the observations it in the tabular form.



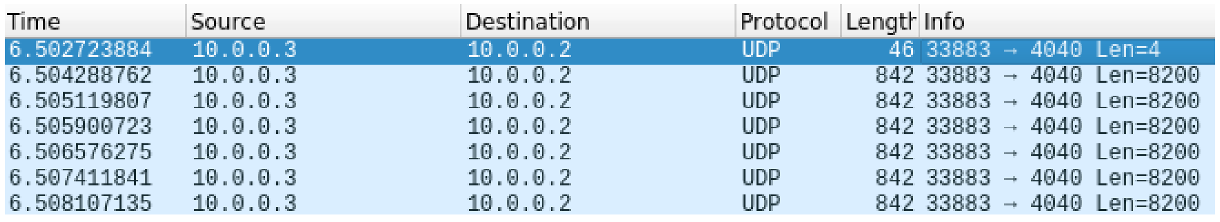
```
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 10.7272 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 9.32211MBps - 9545.84kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 9.81975 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 10.1836MBps - 10428kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 10.8943 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 9.17915MBps - 9399.45kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 10.6595 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 9.38129MBps - 9606.44kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 10.7607 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 9.29304MBps - 9516.07kBps
Size of received_file : 104857600
```

Attempt Number	Time Taken(sec)	Throughput(MB/s)
1	10.7272	9.3221
2	9.8197	10.1836
3	10.8943	9.1791
4	10.6595	9.3812
5	10.7607	9.2930
6	10.7581	9.2953
7	10.6604	9.3804
8	10.7066	9.3400
9	10.7836	9.2733
10	10.7308	9.3189
Average	**10.6500**	**9.3967**

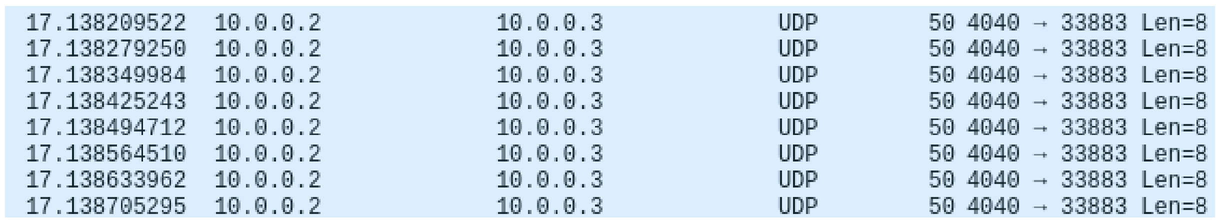




Wireshark screeshots:



Time	Source	Destination	Protocol	Length	Info
6.502723884	10.0.0.3	10.0.0.2	UDP	46	33883 → 4040 Len=4
6.504288762	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200
6.505119807	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200
6.505900723	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200
6.506576275	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200
6.507411841	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200
6.508107135	10.0.0.3	10.0.0.2	UDP	842	33883 → 4040 Len=8200



17.138209522	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138279250	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138349984	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138425243	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138494712	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138564510	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138633962	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8
17.138705295	10.0.0.2	10.0.0.3	UDP	50	4040 → 33883 Len=8

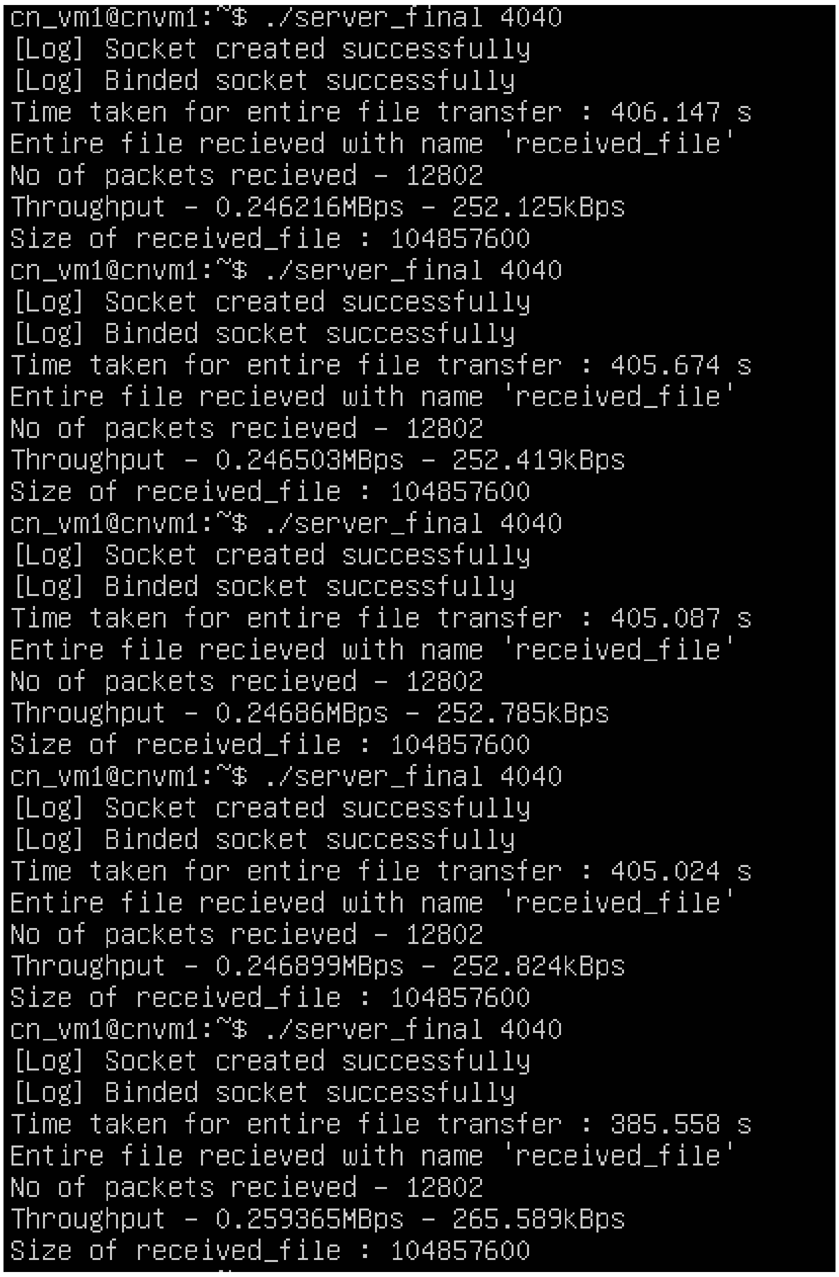
As one can see the first packet transferred as observed in the wireshark is at 6.5027 sec and the last acknowledgements were transferred at 17.1387.The observed diff is 10.636.

```
we know that,
    avg_throughput = file_size/avg_time_taken
    => 100(MB)/10.636(s) = 9.402 Mbps
```

Which is close to the observed average throughput.

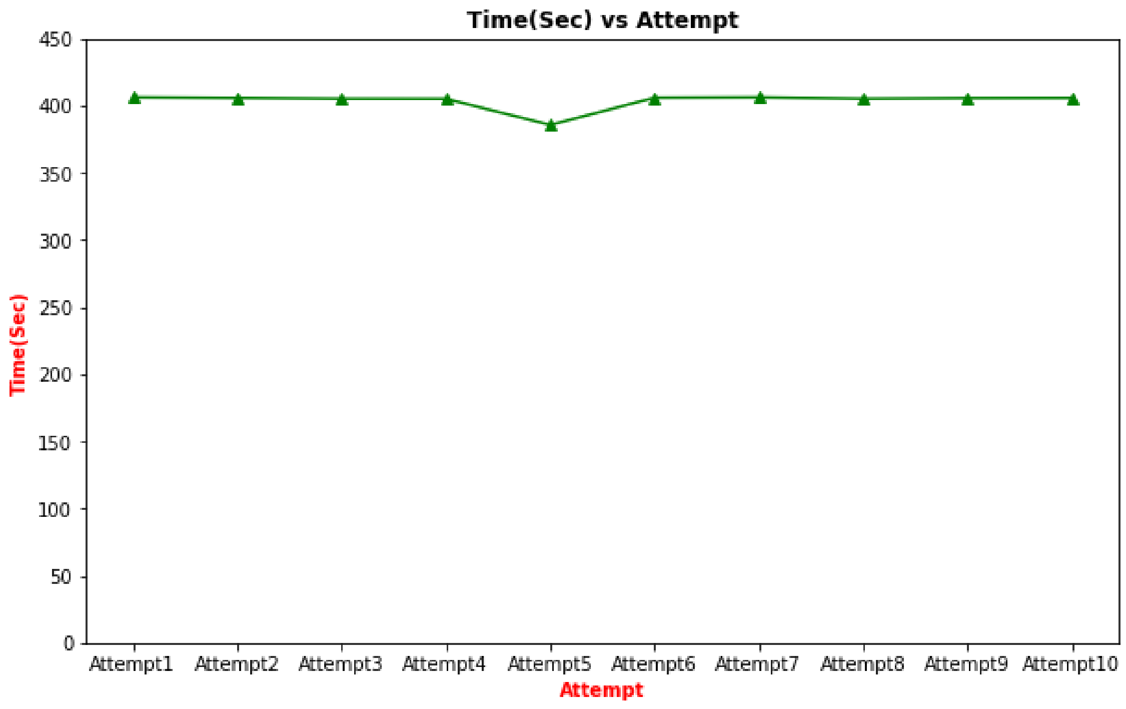
2.7.2 With delay 50ms loss 5%

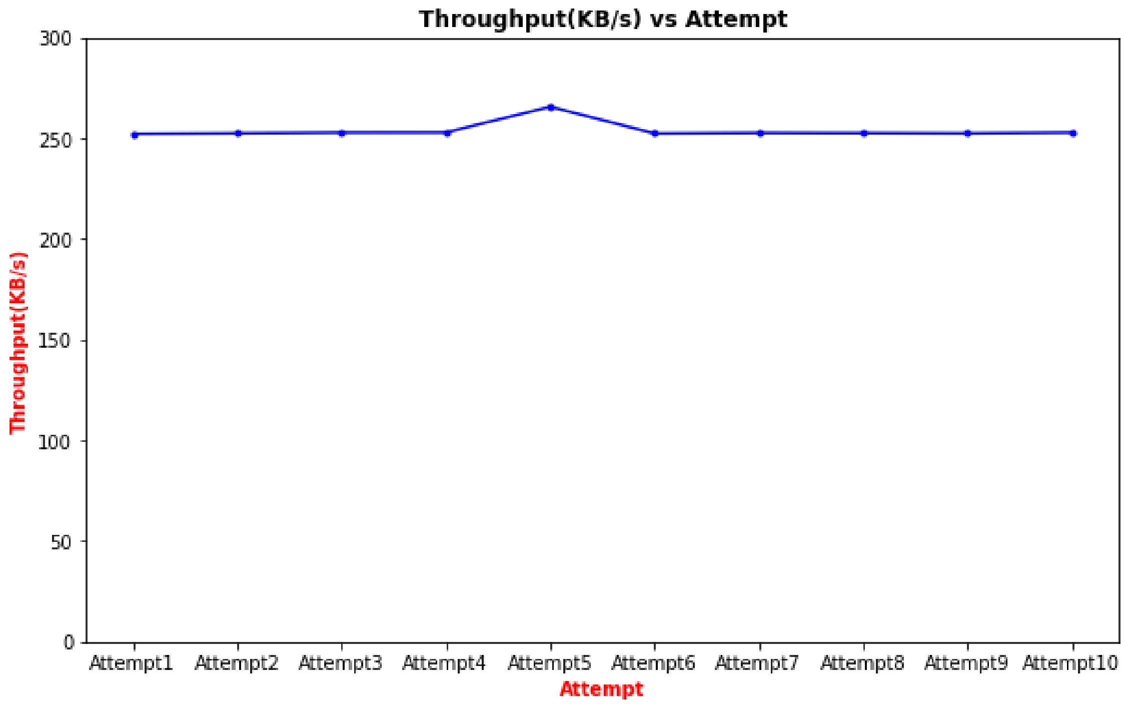
Find the attached screenshots of `OurUdpFtp` program running in vms with conditions delay 50ms and loss 5%



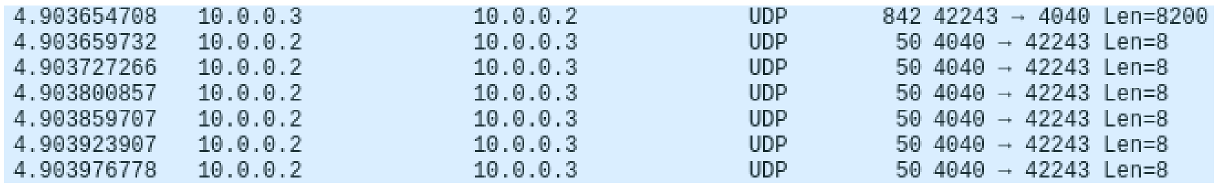
```
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 406.147 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 0.246216MBps - 252.125kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 405.674 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 0.246503MBps - 252.419kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 405.087 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 0.24686MBps - 252.785kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 405.024 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 0.246899MBps - 252.824kBps
Size of received_file : 104857600
cn_vm1@cnvm1:~$ ./server_final 4040
[Log] Socket created successfully
[Log] Binded socket successfully
Time taken for entire file transfer : 385.558 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 0.259365MBps - 265.589kBps
Size of received_file : 104857600
```

Attempt Number	Time Taken(sec)	Throughput(MB/s)	Throughput(KB/s)
1	406.147	0.2462	252.125
2	405.674	0.2465	252.419
3	405.087	0.2468	252.785
4	405.024	0.2469	252.824
5	385.558	0.2593	265.589
6	405.876	0.2466	252.316
7	406.135	0.2468	252.616
8	405.096	0.2469	252.519
9	405.539	0.2462	252.398
10	405.653	0.2465	252.789
Average	**403.578**	**0.2478**	**253.779**

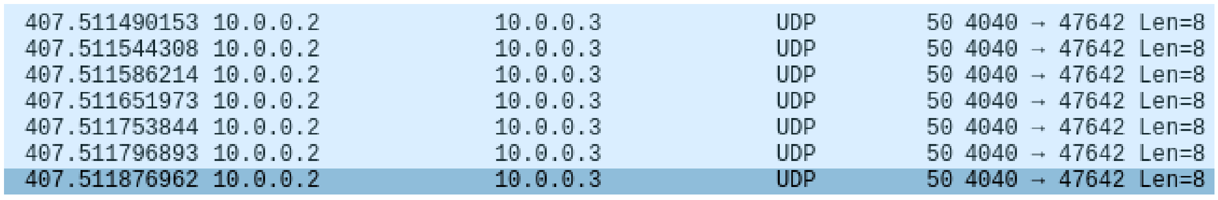




Wireshark screeshots:



```
4.903654708   10.0.0.3          10.0.0.2          UDP       842 42243 → 4040 Len=8200
4.903659732   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
4.903727266   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
4.903800857   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
4.903859707   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
4.903923907   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
4.903976778   10.0.0.2          10.0.0.3          UDP        50 4040 → 42243 Len=8
```



```
407.511490153 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511544308 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511586214 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511651973 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511753844 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511796893 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
407.511876962 10.0.0.2          10.0.0.3          UDP        50 4040 → 47642 Len=8
```

As one can see the first packet transferred as observed in the wireshark is at 4.903 sec and the last acknowledgements were transferred at 407.511. The diff being 402.608 which is close to the observed average time.

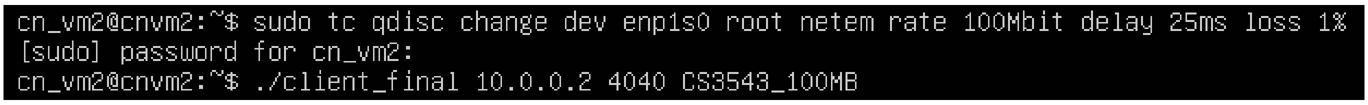
```
we know that,
    avg_throughput = file_size/avg_time_taken
    => 100(MB)/402.608(s) = 0.2483 Mbps = 254.341 kbps
```

Which is in par with the observed average throughput.

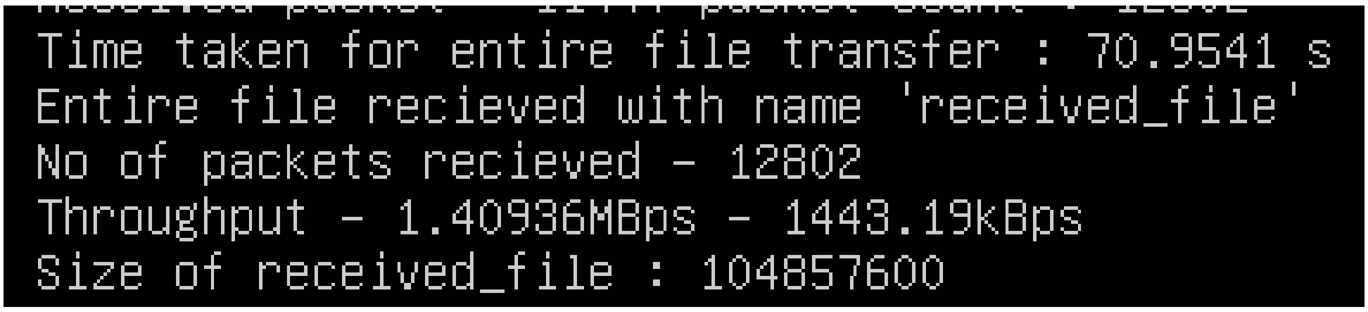
2.7.3 With delay 25ms loss 1%

Tried our program in different lossy environments. This time we set up delay of 25ms and loss 1%.

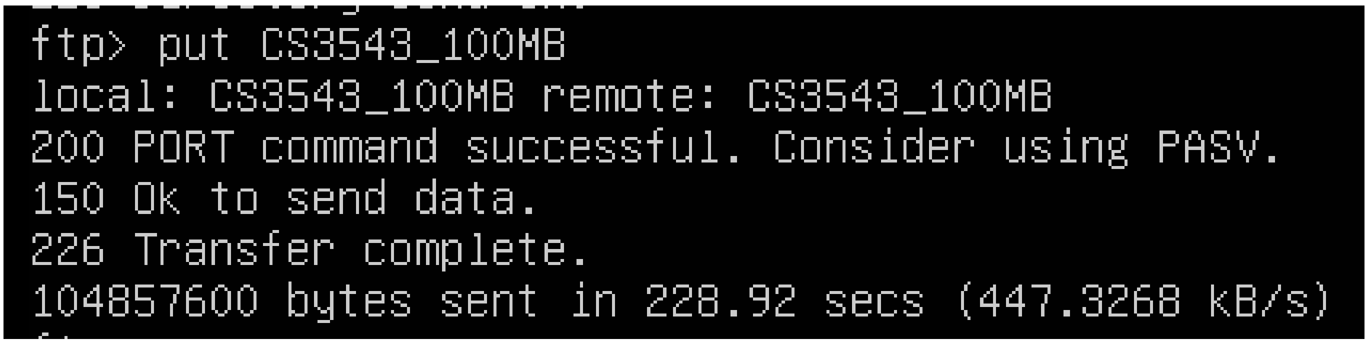
Find attached screenshots.



```
cn_vm2@cnvm2:~$ sudo tc qdisc change dev enp1s0 root netem rate 100Mbit delay 25ms loss 1%
[sudo] password for cn_vm2:
cn_vm2@cnvm2:~$ ./client_final 10.0.0.2 4040 CS3543_100MB
```



```
Time taken for entire file transfer : 70.9541 s
Entire file recieved with name 'received_file'
No of packets recieved - 12802
Throughput - 1.40936MBps - 1443.19kBps
Size of received_file : 104857600
```



```
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
104857600 bytes sent in 228.92 secs (447.3268 kB/s)
```

```
FTP took 229.42 seconds with an average of 447.1498 kB/s
whilst our program took 71.2 seconds with an average of 1441.9 kB/s
```

2.8 Observation:

We can see that the program is performing equally as much as it should with 100Mbit with the time gap being 1sec(possibly overhead) but is outperforming ftp(TCP) by nearly 3.3 times with delay 25ms loss 1% and nearly 4 times with delay 50ms loss 5%.
We are observing a high throughput in a loss and delay environments than the TCP

3 Side observation:

In the middle of the trails, out of curiosity we ran the UDP program over a network on two different hosts and tried to transfer the large file. However the entire network experienced a loss in bandwidth. We didn't encounter anything like this while running TCP. We found it interesting and looked into it. As a router places each incoming datagram in a queue in memory until it can be processed, the policy focuses on queue management. When datagrams arrive faster than they can be forwarded, the queue grows; when datagrams arrive slower than they can be forwarded, the queue shrinks. However, because memory is finite, the queue cannot grow without bound. At this particular point the queue is full of datagrams and TCP requests for entering the queue which eventually fails. As the internet mostly transfers TCP traffic(as HTTP uses TCP) the network experienced a tail drop.

4 Notes:

- For task 2 screenshots containing only 5 trail runs are updated in the report. As the report is being lengthy we decided to keep only limited number of screenshots and one can find the observations in the tabular and graph representation.
- We couldn't start wireshark right at the start of program, so we switched on the wireshark first and later started the programs on vms and monitored traffic in the bridge. Thus one finds the UDP packets transfer at a slight delay that is not exactly at 0 sec.
- The same program is running differently on different machines (possibly because of reading and writing the file is different between machines) and we submitted the programs in which the observations are best.
- The C++ program requires -lpthread flag as the program includes threading support and requires -std=c++17 flag as some c++ 17 standard filesystem support is used in the code.

```
$ g++ server.cpp -lpthread -std=c++17 -o server
$ g++ client.cpp -lpthread -std=c++17 -o client
```

THE END!
LaTeX generated document