



భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Assignment 1 - Lab Report

February 20, 2022

Computer Networks - 2

CS3543

Team

Roll No .	Name
CS19BTECH11012	Nisha M
ES19BTECH11007	Sravanthi Reddy M
MA19BTECH11003	Siddhant Chandorkar

Task 1:

We ran Task 1 on two laptops, both of which have two Ubuntu 20.04 server VMs. The first VM on one of the machines is called '*alice*' with an IP address of **10.0.0.253** and the second VM is named '*bob*' with an IP address of **10.0.0.254**. On the other laptop, the first and second VMs are called '*server1*' and '*server2*' and their IP addresses are **10.42.0.179** and **10.42.0.247** respectively.

The two VMs were connected via bridge0, a local bridge. The IP addresses of the two servers are shown in the following screenshots.

Laptop 1:

```
alice@alice:~$ ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc netem state UP group default qlen 1000
    link/ether 52:54:00:d5:76:1c brd ff:ff:ff:ff:ff:ff
    inet 192.168.36.13/24 brd 192.168.36.255 scope global dynamic enp1s0
        valid_lft 3295sec preferred_lft 3295sec
    inet 10.0.0.253/24 scope global enp1s0
        valid_lft forever preferred_lft forever
    inet6 2401:4900:4aa9:f2d8:5054:ff:fed5:761c/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 3296sec preferred_lft 3296sec
```

```
bob@bob:~$ ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc netem state UP group default qlen 1000
    link/ether 52:54:00:18:82:62 brd ff:ff:ff:ff:ff:ff
    inet 192.168.36.162/24 brd 192.168.36.255 scope global dynamic enp1s0
        valid_lft 3338sec preferred_lft 3338sec
    inet 10.0.0.254/24 scope global enp1s0
        valid_lft forever preferred_lft forever
    inet6 2401:4900:4aa9:f2d8:5054:ff:fe18:8262/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 3343sec preferred_lft 3343sec
```

Laptop 2:

```
sravanthi@server1:~$ ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:6e:21:e4 brd ff:ff:ff:ff:ff:ff
    inet 10.42.0.179/24 brd 10.42.0.255 scope global dynamic ens3
        valid_lft 3459sec preferred_lft 3459sec
    inet6 fe80::5054:ff:fe6e:21e4/64 scope link
        valid_lft forever preferred_lft forever

sravanthi@server2:~$ ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:a8:a2:73 brd ff:ff:ff:ff:ff:ff
    inet 10.42.0.247/24 brd 10.42.0.255 scope global dynamic ens3
        valid_lft 3445sec preferred_lft 3445sec
    inet6 fe80::5054:ff:fea8:a273/64 scope link
        valid_lft forever preferred_lft forever
```

sudo apt install vsftpd

is the command to install ftp on both VMs and the status of both the servers can be found using:

sudo service vsftpd status

sudo tc qdisc add dev enp1s0 root netem rate 100Mbit

is run on both the servers to configure 100Mbit link speed.

Later, on the second server, the command "ftp 10.0.0.254" is run to complete the data transfer. As seen in the accompanying screenshot, we start ftp between the VMs by entering the username and password of the other VM.

On the other laptop, 'server1' (10.42.0.179) is the FTP client and 'server2' (10.42.0.247) is the server.

We now attempt ten FTP transmissions of 100Mb of data without delay or packet loss. A couple of the trials are seen in the screenshots below.

Laptop 1

```
alice@alice:~$ ftp 10.0.0.254
Connected to 10.0.0.254.
220 (vsFTPd 3.0.3)
Name (10.0.0.254:alice): bob
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
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 8.49 secs (11.7839 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 (11.8015 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 (11.8006 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.
```

Laptop 2

```
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 (11.8072 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 8.51 secs (11.7463 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 (11.8031 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 (11.6898 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 (11.7989 MB/s)
```

Throughput:

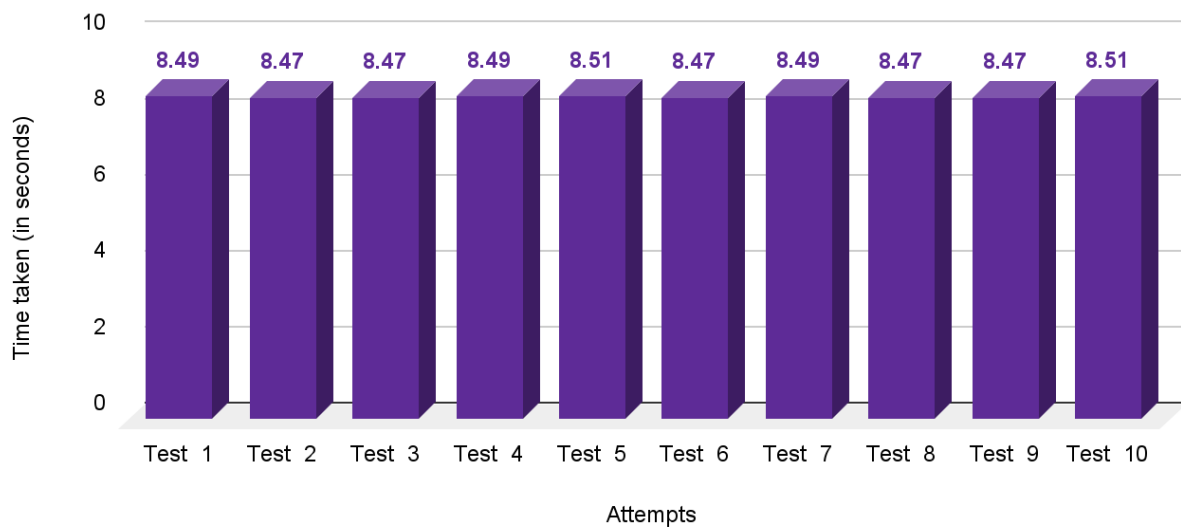
With a bandwidth of 100Mbps, the total time to transfer CS3543_100MB with full utilization would be given by

$$\begin{aligned} & \text{(Total file size in bits)/100Mb} \\ &= (104857600 \times 8) / (100 \times 1024 \times 1024) \\ &= 8\text{s} \end{aligned}$$

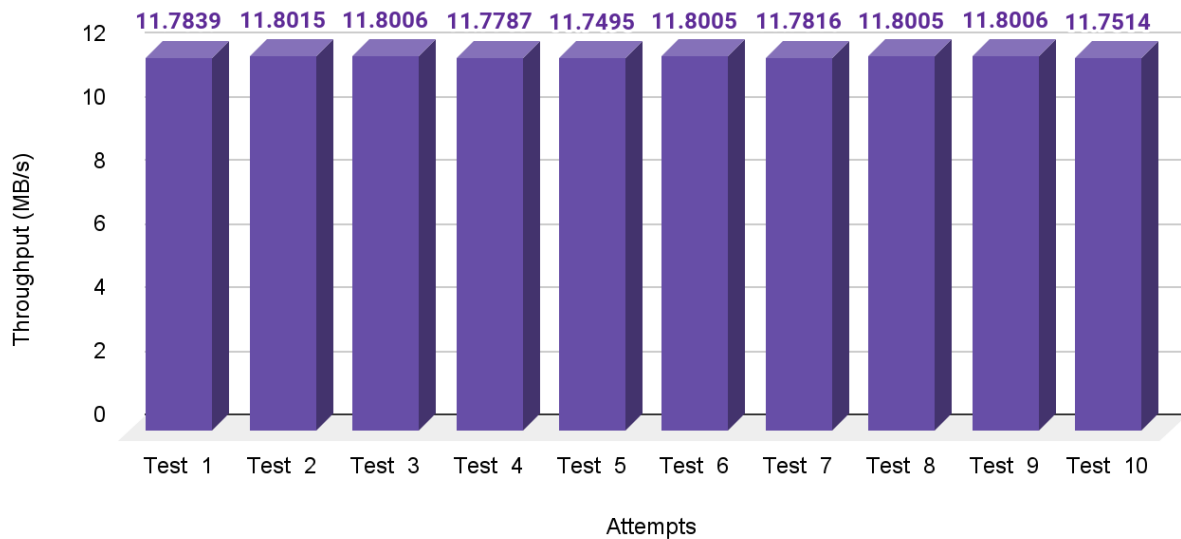
We can observe that this is very close to the output we observe on both machines. The graphs for time taken and throughput are provided below.

Graphs for Laptop 1:

Time taken to send the given data packet of size 100MB from Alice Server to Bob server

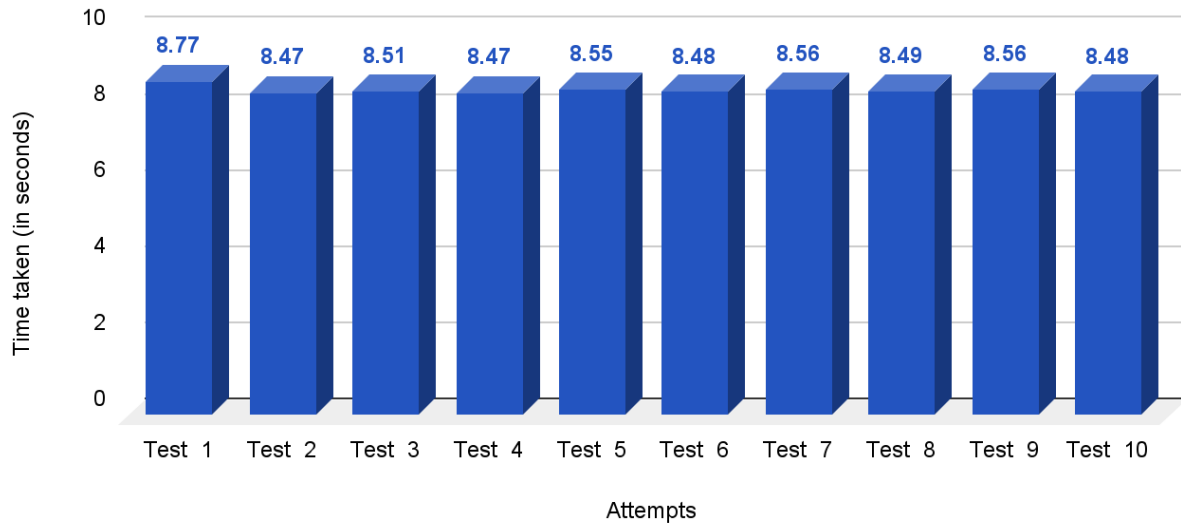


Throughput for each attempt to send 100MB data from Alice Server to Bob Server

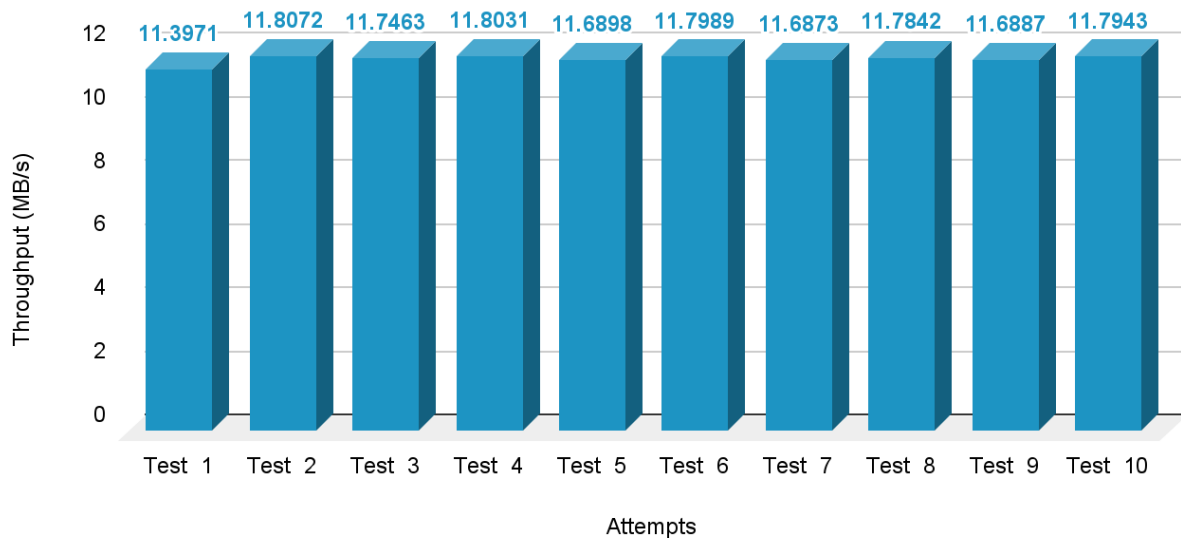


Graphs for Laptop 2:

Time taken to send the given data packet of size 100MB from sravanthi@server1 to sravanthi@server2



Throughput for each attempt to send 100MB data from sravanthi@server1 to sravanthi@server2



The second set of conditions given allows us to simulate bad network conditions. We use the following command on both VMs to introduce a 50ms delay and 5% packet loss rate in both directions:

```
sudo tc qdisc change dev enp1s0 root netem delay 50ms loss 5%
```

We perform FTP to observe the time taken for data transfer between the two VMs and the throughput under the second set of conditions.

Laptop 1

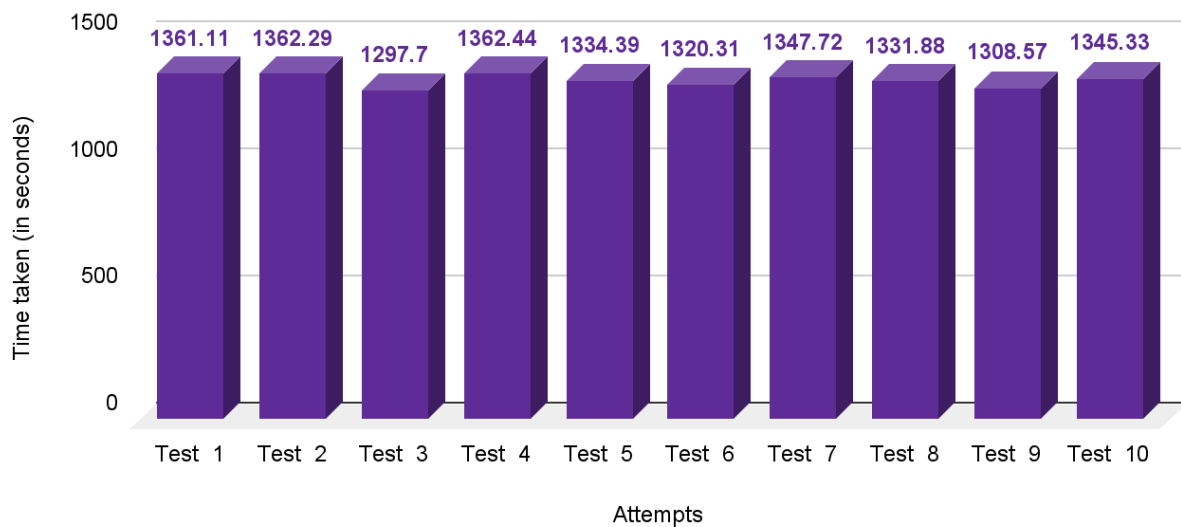
```
alice@alice:~$ ftp 10.0.0.254
Connected to 10.0.0.254.
220 (vsFTPD 3.0.3)
Name (10.0.0.254:alice): bob
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
```

Laptop 2

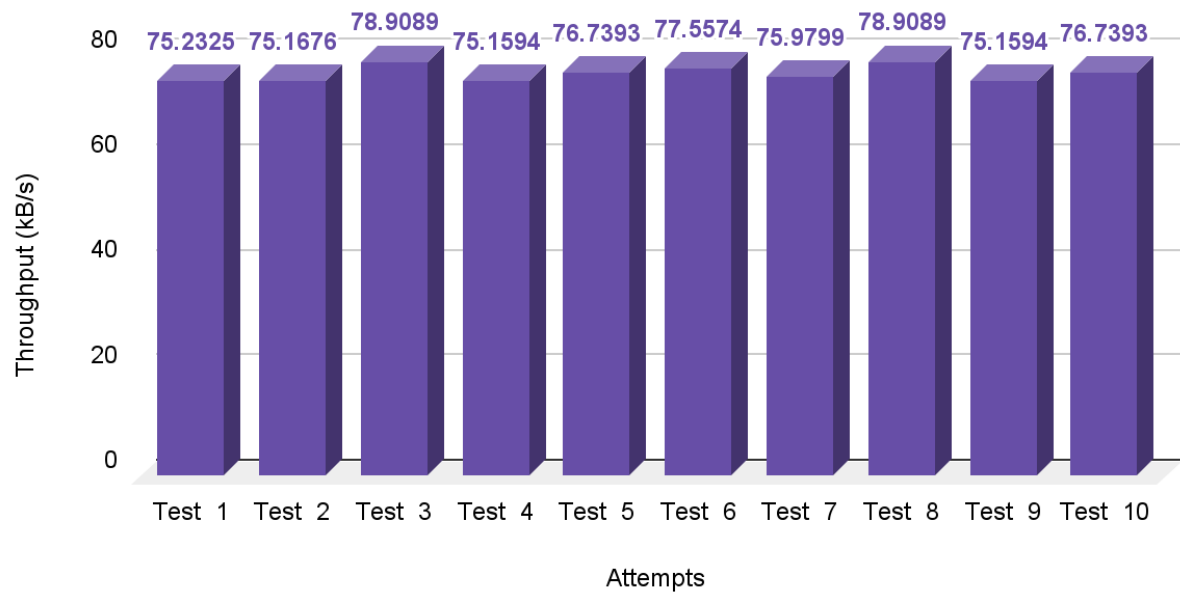
```
sravanthi@server1:~$ ftp 10.42.0.247
Connected to 10.42.0.247.
220 (vsFTPD 3.0.3)
Name (10.42.0.247:sravanthi): sravanthi
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put CS3543_100MB
local: CS3543_100MB remote: CS3543_100MB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
```

Graphs for Laptop 1:

Time taken to send the given data packet of size 100MB from Alice Server to Bob server with time delay and packet loss

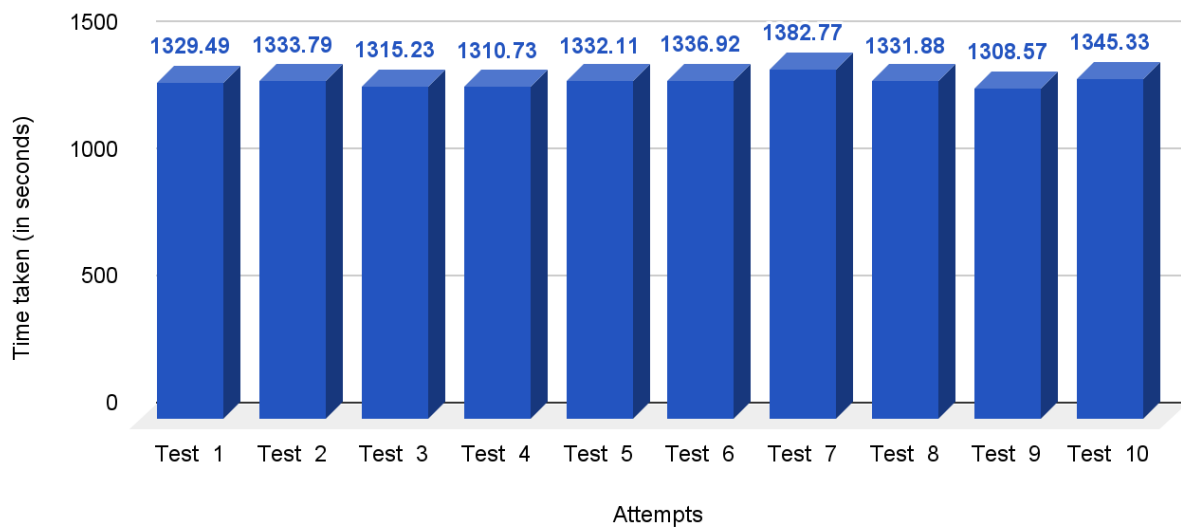


Throughput for each attempt to send 100MB data from Alice Server to Bob Server with delay and packet loss

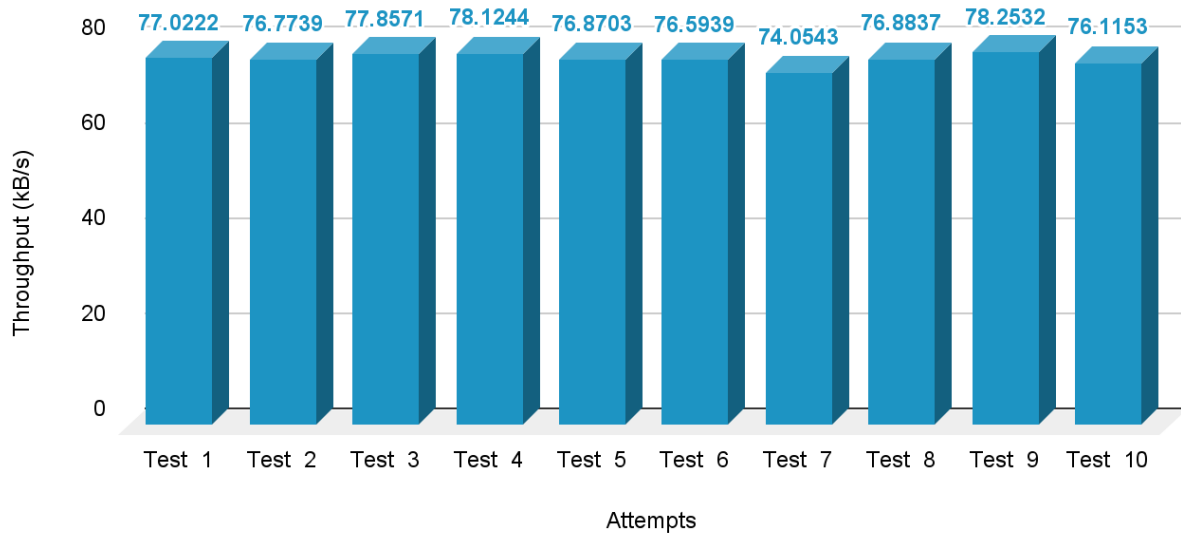


Graphs for Laptop 2:

Time taken to send the given data packet of size 100MB from server1 to server2 with time delay and packet loss



Throughput for each attempt to send 100MB data from server1 to server2 with delay and packet loss



Task 2:

Implementation of our-UDP-FTP

The setup we have used involves two VMs on the same host OS connected through a linux bridge. The given 100 MB data file is transferred using our-UDP-FTP from the client (sender) to the server (receiver), i.e., from Alice to Bob on Laptop1 and server1 to server2 on Laptop2.

The code implementation consists of two files one for client (sender.py) and another for server(receiver.py) both of which are written in python. Detailed explanation of each of the files is as follows:

sender.py implementation

This file takes 3 command line arguments (in the same order as mentioned):

1. Port Number of sender
2. IP Address of receiver
3. Port Number of receiver

The workflow of this program is as follows:

- The user is prompted for the name of the file. If any file is present with that filename or path, the program proceeds further.
- The filename string is then sent to the server (receiver) in bytes format.
- The next step is to calculate the total number of frames which is the size of the file divided by the size of the buffer.
- This number is sent to the receiver and the data sent back (as ACK) is checked. sender.py (client) retransmits until both numbers match, otherwise, until the maximum number of retransmissions are exceeded.

-
- Now the file is opened and read in binary mode, for every frame and the data is broken down into small packets and stored in the memory so that they can be transmitted later.
 - To send the packets stored in the memory to the server, we have implemented multi-threading in which every thread collects a specific number of data packets and transmits them to the server whereas the main thread receives the ACK.
 - Soon after the ACK of a packet is received by the client, the main thread sets the ACK field of that packet to True.

The implementation of the thread function is as follows:

- At the start, the first and the last number of the packets to be transmitted are calculated.
- We create an aperture of defined size and initialize the starting window.
- Until this window becomes empty, we run the loop. The loop has the following logic :
 - If the ACK field of the packet is True, it means that the packet is already received and hence it will be dropped from the window further followed by the addition of the following packet to the window back.
 - If that current packet is being transmitted for the first time, then the SYN field of it will be checked . If the packet is timed out that can be checked by comparing the current time and expected recv time field of the packet. Then it is transmitted after updating the SYN field and `expec_recv_time` field.
- Parallely, the main thread collects all the ACKs and finally after receiving , we print the time and wait till all the worker threads join.
- The final step is to send the final packet that tells the server that all the ACKs are received by the client.

receiver.py implementation:

- At the start, the server receives the filename and the total number of frames that the client (sender.py) will send from the client.
- Once the packets are received from the client, the server stores them in the memory and sends back the ACK for every received packet .
- Then there is a loop that runs till all the frames are received and then the file is being written in the binary mode.
- If the total file is received before the sender has received all of the ACKs, re-receive the remaining packets and transmit the ACKs until the last frame is reached.

Features as part of RDT:

The RDT features that we have implemented include :

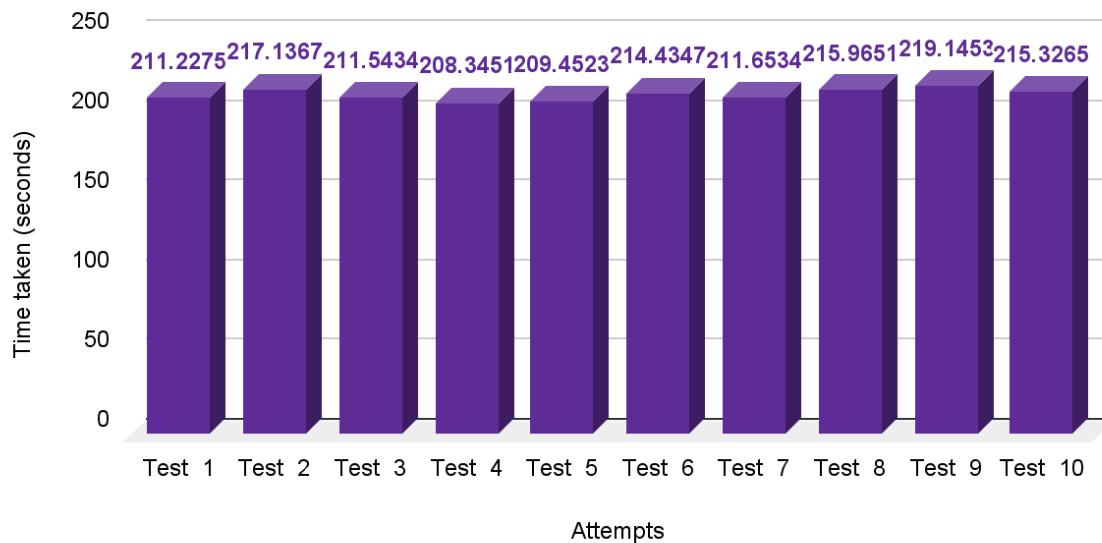
- Acknowledging the sent data and preventing packet loss -
 - Both the sender and the receiver will continue to run until all of the packets and their ACK's have been received by the receiver and sender respectively, guaranteeing that no packets are lost. If no ACKs are received during the timeout, the packets will be resent.
- Retransmission of packets -
 - If the packet that is sent is not acknowledged or the packet times out , our implementation makes sure that packet is sent. I.e the program continues to run till it gets acknowledged from the server.
- Fast Retransmission-
 - In our implementation, we are enabling the faster transmission of packets using multithreads. The exact implementation is the thread function where we are

maintaining the window for packets . In this way we could send numerous packets without waiting for the ACK of the previous packets. But ultimately we make sure that all the ACKs are received . The overall time without loss is less than 100 secs for our implementation.

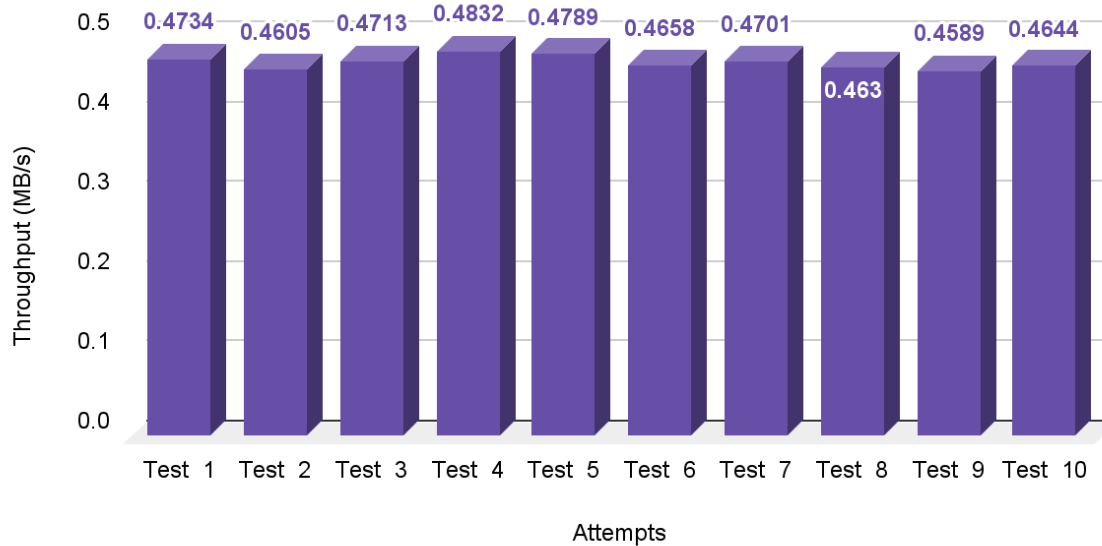
- Checksum -
 - This is to ensure the integrity of the data that is being transmitted. We have implemented the functionality of checksum where we calculate the checksum of the payload and embed it in the packet structure before transmitting. After we transmit the packet, at the receiver end, we unpack the structure and calculate the checksum again on the unpacked payload. Then we check if this checksum matches with the checksum embedded in the packet. If it does, then the data integrity is obeyed otherwise it might indicate data corruption.

Graphs:

Time taken to send 100MB data with our-UDP-FTP from Alice Server to Bob server with delay and packet loss

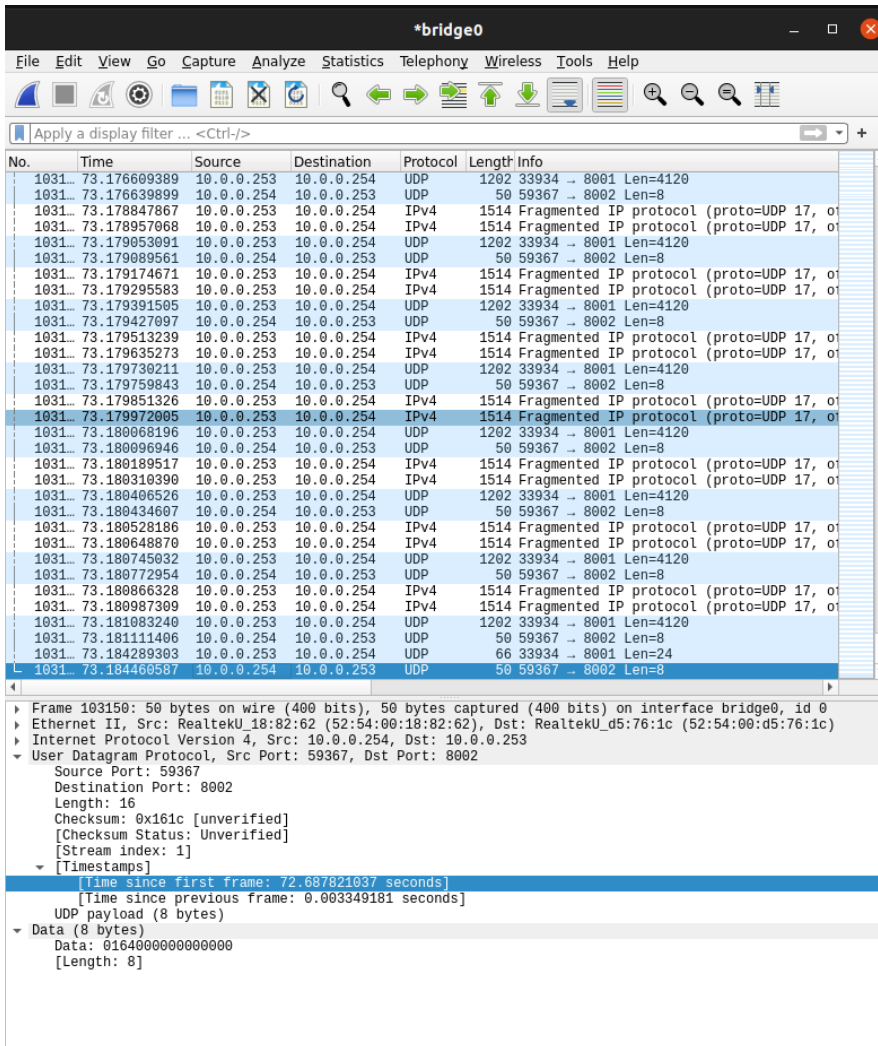


Throughput for sending 100MB data with our-UDP-FTP from Alice Server to Bob Server with delay and packet loss



Wireshark:

Without Loss:



The image shows a Wireshark packet capture on interface bridge0. The packet list pane displays a series of packets, including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol (UDP) packets. The packet details pane shows the structure of a UDP packet, including the source and destination ports, length, and checksum. The packet bytes pane shows the raw data of the packet, which is a file named CS3543_100MB.

No.	Time	Source	Destination	Protocol	Length	Info
1031	73.176609389	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.176639899	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.178847867	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.178957068	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179053091	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.179089561	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.179174671	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179295583	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179391505	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.179427097	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.179513239	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179635273	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179730211	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.179759843	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.179851326	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.179972005	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180066196	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.180096946	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.180189517	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180310390	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180406526	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.180434607	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.180528186	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180648870	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180745032	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.180772954	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.180866328	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.180987309	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1031	73.181083240	10.0.0.253	10.0.0.254	UDP	1202	33934 → 8001 Len=4120
1031	73.181111406	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8
1031	73.184289303	10.0.0.253	10.0.0.254	UDP	66	33934 → 8001 Len=24
1031	73.184460587	10.0.0.254	10.0.0.253	UDP	50	59367 → 8002 Len=8

Frame 103150: 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface bridge0, id 0
Ethernet II, Src: RealtekU_18:82:62 (52:54:00:18:82:62), Dst: RealtekU_d5:76:1c (52:54:00:d5:76:1c)
Internet Protocol Version 4, Src: 10.0.0.254, Dst: 10.0.0.253
User Datagram Protocol, Src Port: 59367, Dst Port: 8002
Source Port: 59367
Destination Port: 8002
Length: 16
Checksum: 0x161c [unverified]
[Checksum Status: Unverified]
[Stream index: 1]
[Timestamps]
[Time since first frame: 72.687821037 seconds]
[Time since previous frame: 0.003349181 seconds]
UDP payload (8 bytes)
Data (8 bytes)
Data: 0164000000000000
[Length: 8]



The terminal window shows the execution of a file transfer script. The script prompts the user for the filename of the file to send, the total packets to send, and the file transmission time. The output shows that the file was successfully transferred without loss.

```
alice@alice:~$ python3 sender.py 8002 10.0.0.254 8001
Enter filename of file to send (type "exit" to quit): CS3543_100MB
Size of file = 104857600 bytes
Total packets to send = 25600
File transmission time = 72.689129966 secs
Throughput = 1.3757215149882045 MB/s
alice@alice:~$ _
```

With Loss and Delay:

The image displays two side-by-side windows. The left window is Wireshark, titled '*bridge0', showing a packet capture. The right window is a terminal titled 'ubuntu20.04 on QEMU/KVM' showing the execution of a file transfer test.

Wireshark Packet Capture:

No.	Time	Source	Destination	Protocol	Length	Info
1174	210.385409927	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.385493181	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.385539081	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	210.399584266	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.408680998	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.408767007	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.408850565	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	210.409966559	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.435730746	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.459150336	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.485850277	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.485877986	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.486006498	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	210.510298164	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.510238890	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.536316299	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.590392701	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.598482297	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.598509598	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	210.648917353	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	210.735402158	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.735490202	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	210.735529884	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	210.785783046	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1174	211.010180419	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	211.010346115	10.0.0.253	10.0.0.254	IPv4	1514	Fragmented IP protocol (proto=UDP 17, o
1174	211.010356966	10.0.0.253	10.0.0.254	UDP	1178	60017 → 8001 Len=4096
1174	211.060688161	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1175	211.116129328	10.0.0.253	10.0.0.254	UDP	66	60017 → 8001 Len=24
1175	211.166463826	10.0.0.254	10.0.0.253	UDP	50	45458 → 8002 Len=8
1175	211.171527945	2404:6800::...	2401:4900:4...	UDP	174	443 → 59677 Len=112
1175	211.171592134	2404:6800::...	2401:4900:4...	UDP	123	443 → 59677 Len=61

Terminal Output:

```
alice@alice:~$ sudo tc qdisc change dev enp1s0 root netem delay 50ms loss 5%
alice@alice:~$ python3 sender.py 8002 10.0.0.254 8001
Enter filename of file to send (type "exit" to quit): CS3543_100MB
Size of file = 104857600 bytes
Total packets to send = 25600
File transmission time = 211.227572966 secs
Throughput = 0.47342304130008817 MB/s
alice@alice:~$
```

Wireshark Details:

- Frame 117501: 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface bridge0, id 0
- Ethernet II, Src: RealtekU_18:82:62 (52:54:00:18:82:62), Dst: RealtekU_d5:76:1c (52:54:00:d5:76:1c)
- Internet Protocol Version 4, Src: 10.0.0.254, Dst: 10.0.0.253
- User Datagram Protocol, Src Port: 45458, Dst Port: 8002
- Source Port: 45458
- Destination Port: 8002
- Length: 16
- Checksum: 0x161c [unverified]
- [Checksum Status: Unverified]
- [Stream index: 1]
- [Timestamps]
- [Time since first frame: 211.115890099 seconds]
- [Time since previous frame: 0.105775665 seconds]
- UDP payload (8 bytes)
- Data (8 bytes)
- Data: 0164000000000000
- [Length: 8]

Application Header:

