Qos Directed Error Control for Video Multicasting over IP

With rapid advances in wireless packet networking technology, it is becoming possible to provide mobile users not only with data, voice, but also with video communication services. Many emerging mobile applications require the playback of video on the mobile hosts (MHs), and today's wireless networks are getting ready to provide sufficient bandwidth for support of compressed video transmission. Besides sufficient bandwidth, we need effective and efficient error control mechanism to deal with the following problem.

we need to achieve transmission efficiency. Automatic Repeat reQuest (ARQ) has been proposed as effective technique for reliable multicast in wired networks, as well as for reliable unicast in wireless networks. However, for wireless video multicast with real-time constraints, ARQ may have the following disadvantages:

(1) the sender may experience NACK (Negative ACK) implosion, and the sender may have to retransmit the same packet multiple times;
(2) the receivers may contend with each other to send NACKs uplink, causing unnecessary latency;

Objectives of this Project:

(1) To create a Multicast video Application using Socket Programming
(2) To simulate the multicast application on Mininet with the same topology.
(3) To implement FEC, FEC_retrans algorithm on the Multicast algorithm and simulate the same on mininet.

Steps in Multicast video applicationn

First of all, client will send a join request to the server to join the multicast group.

➔After that Server will provide list(the list containing the videos), site info to the client through TCP.

➔Then whichever station it selects from the list, it is connected to that station.

➔All the stations are sending data, irrespective of client is connected or not.

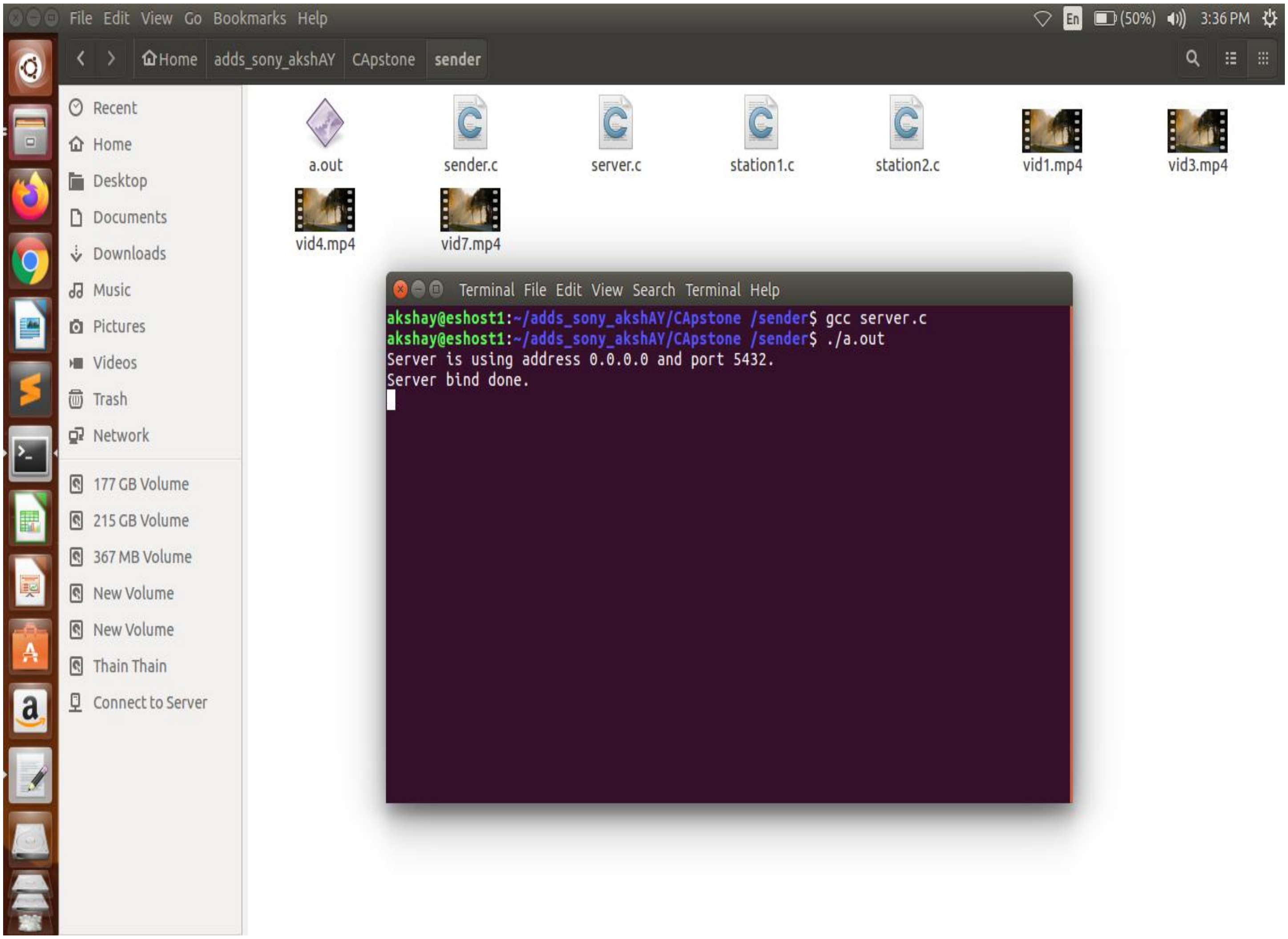
➔Whenever receiver connects to a particular station, it starts receiving live-streaming videos from that station.

➔Used Media player: ffplay . All videos at station side is converted using ffmpeg to make it streamable.

Execution of Server.c file

Client to Server: TCP a. TCP is used for one to one connection from client to server and it is used for station info and site info

Server is running on port no 5432 and having the ip address 0.0.0.0



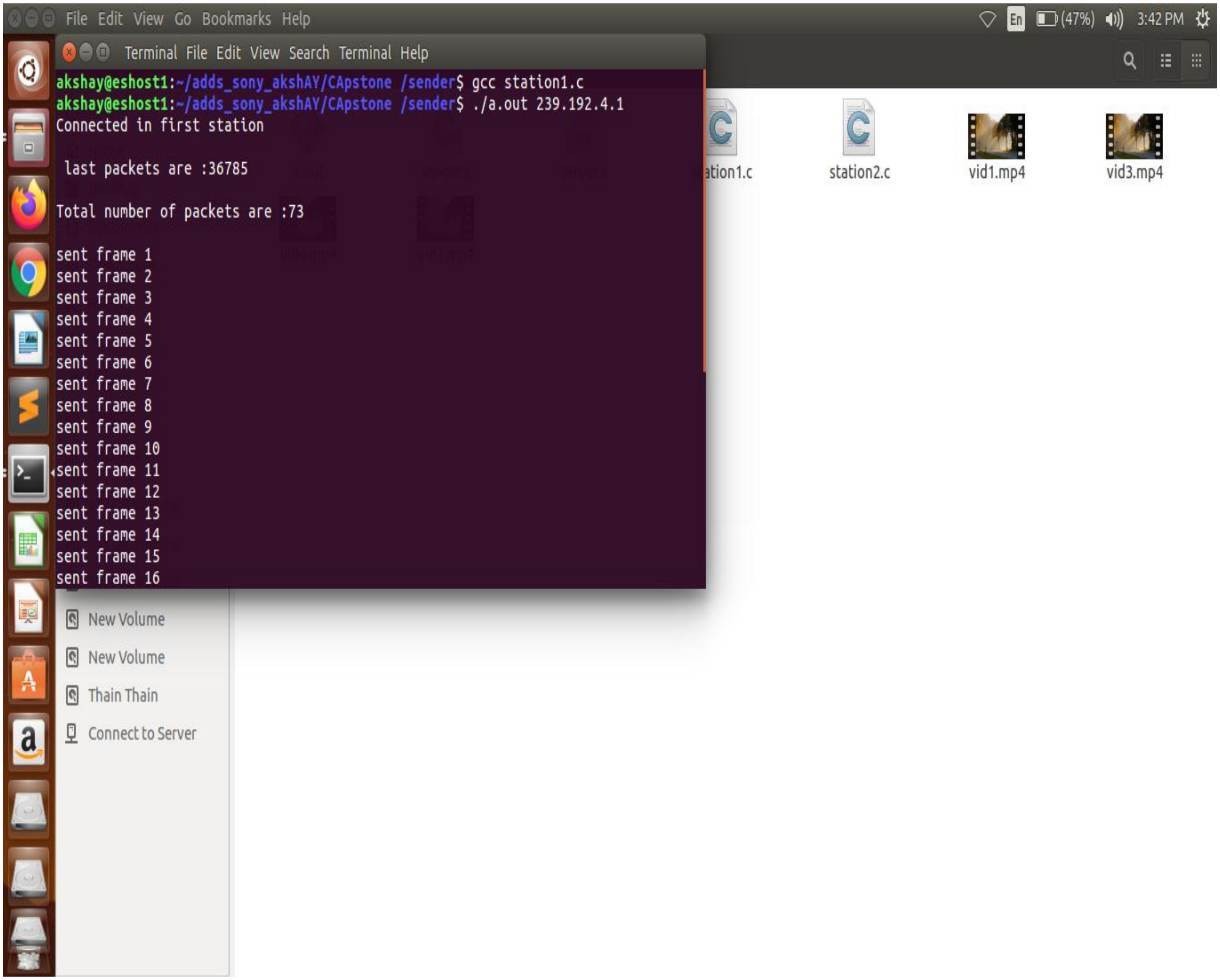
```
akshay@eshost1:~/adds_sony_akshAY/CApstone /sender$ gcc server.c
akshay@eshost1:~/adds_sony_akshAY/CApstone /sender$ ./a.out
Server is using address 0.0.0.0 and port 5432.
Server bind done.
```

\

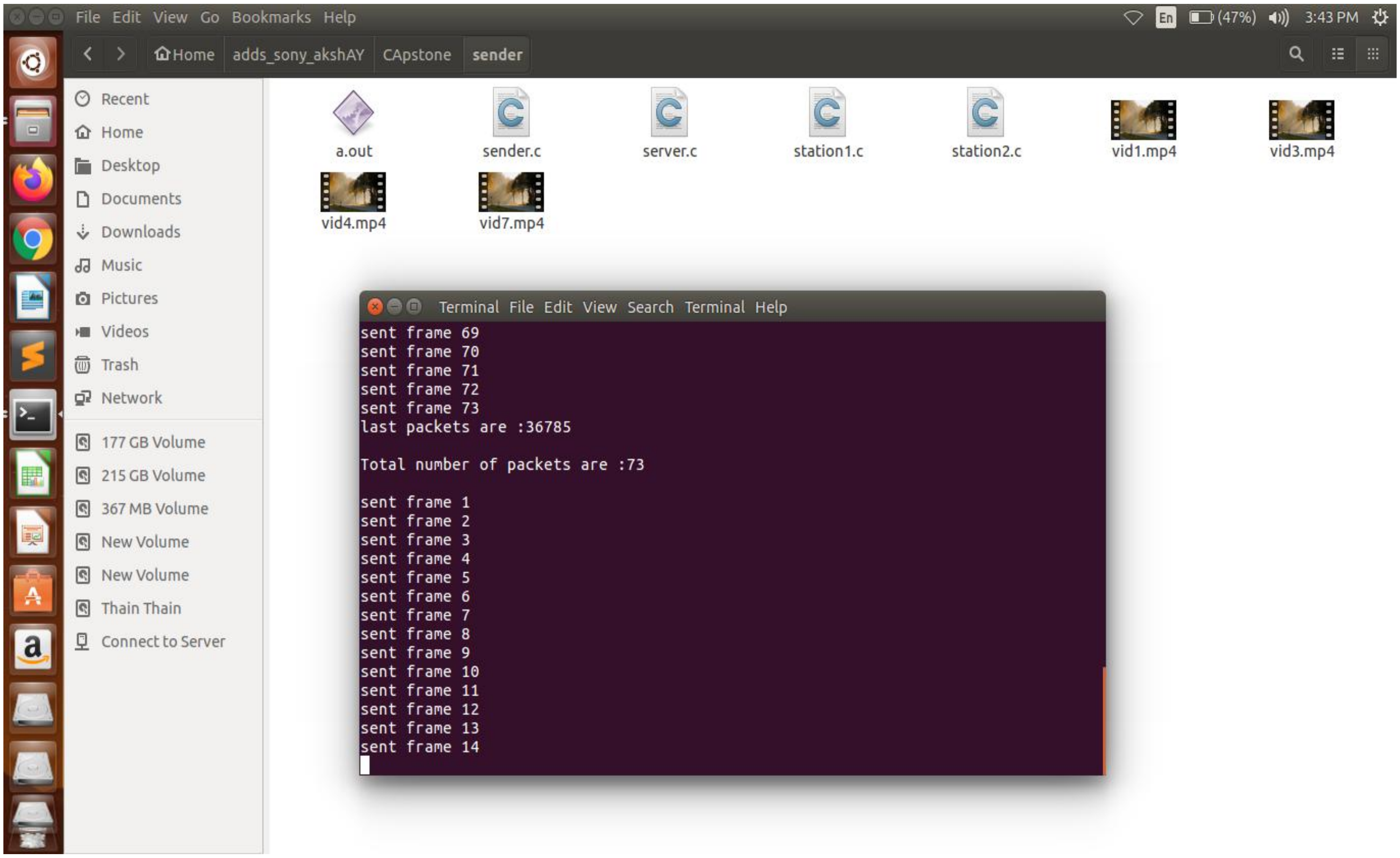
Station1.c

Here this file contains the videos to be sent and it sends the video frames one by one printing the total number of packets ,then it keeps on sending the frames until the connection is closed

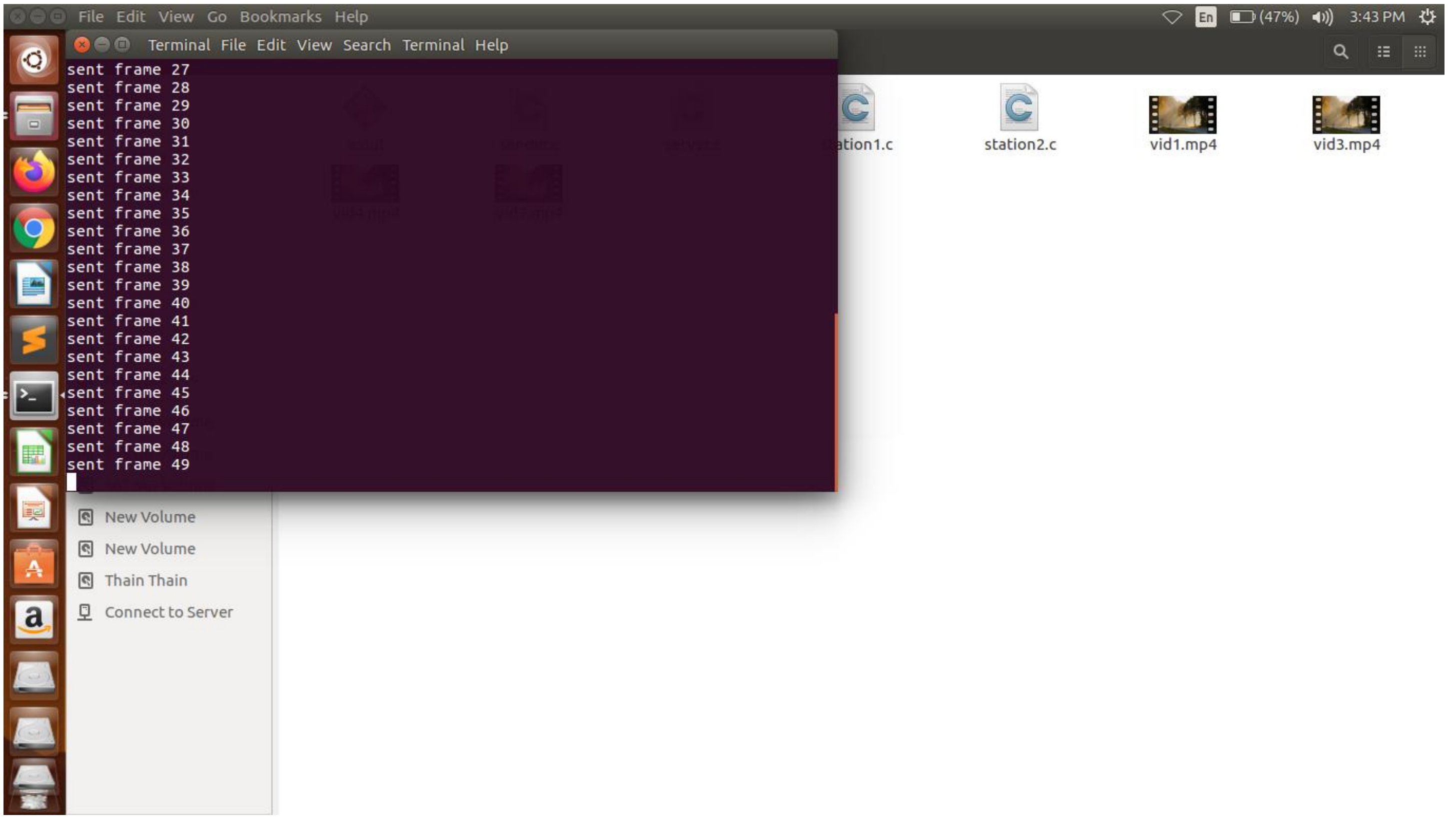
Ip address of station1 is 239.192.4.1



```
akshay@eshost1:~/adds_sony_akshAY/CApstone /sender$ gcc station1.c
akshay@eshost1:~/adds_sony_akshAY/CApstone /sender$ ./a.out 239.192.4.1
Connected in first station

 last packets are :36785

Total number of packets are :73

sent frame 1
sent frame 2
sent frame 3
sent frame 4
sent frame 5
sent frame 6
sent frame 7
sent frame 8
sent frame 9
sent frame 10
sent frame 11
sent frame 12
sent frame 13
sent frame 14
sent frame 15
sent frame 16
```



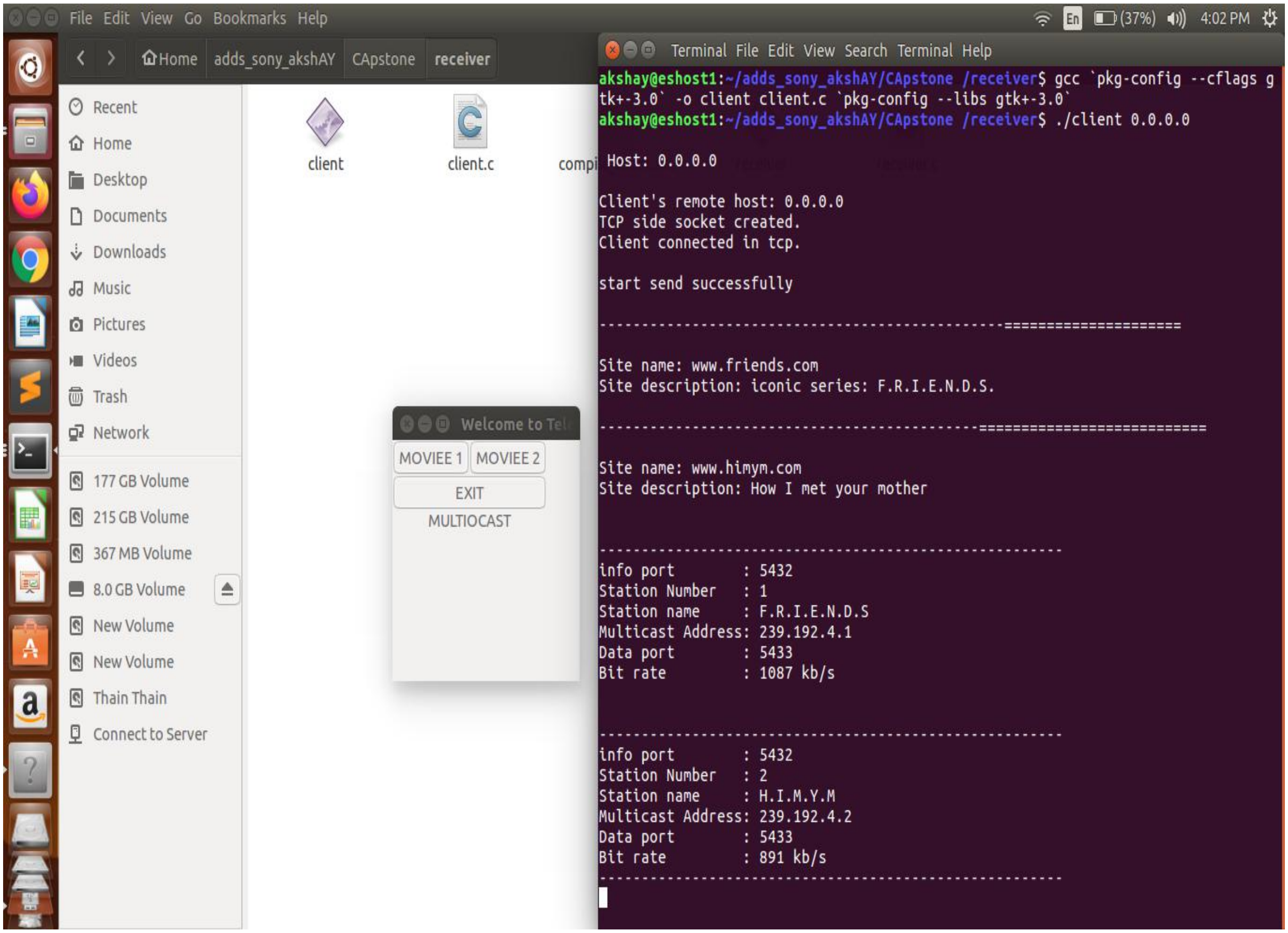
```
sent frame 69
sent frame 70
sent frame 71
sent frame 72
sent frame 73
last packets are :36785

Total number of packets are :73

sent frame 1
sent frame 2
sent frame 3
sent frame 4
sent frame 5
sent frame 6
sent frame 7
sent frame 8
sent frame 9
sent frame 10
sent frame 11
sent frame 12
sent frame 13
sent frame 14
```



```
sent frame 27
sent frame 28
sent frame 29
sent frame 30
sent frame 31
sent frame 32
sent frame 33
sent frame 34
sent frame 35
sent frame 36
sent frame 37
sent frame 38
sent frame 39
sent frame 40
sent frame 41
sent frame 42
sent frame 43
sent frame 44
sent frame 45
sent frame 46
sent frame 47
sent frame 48
sent frame 49
```

Client.c

Sender to Receiver: UDP a. UDP is used to send multicast live-streaming videos from sender to all receivers who joined multicast group. For the creartion of GUI we have used a inbuilt library called GTK with header file #include<gtk.h>

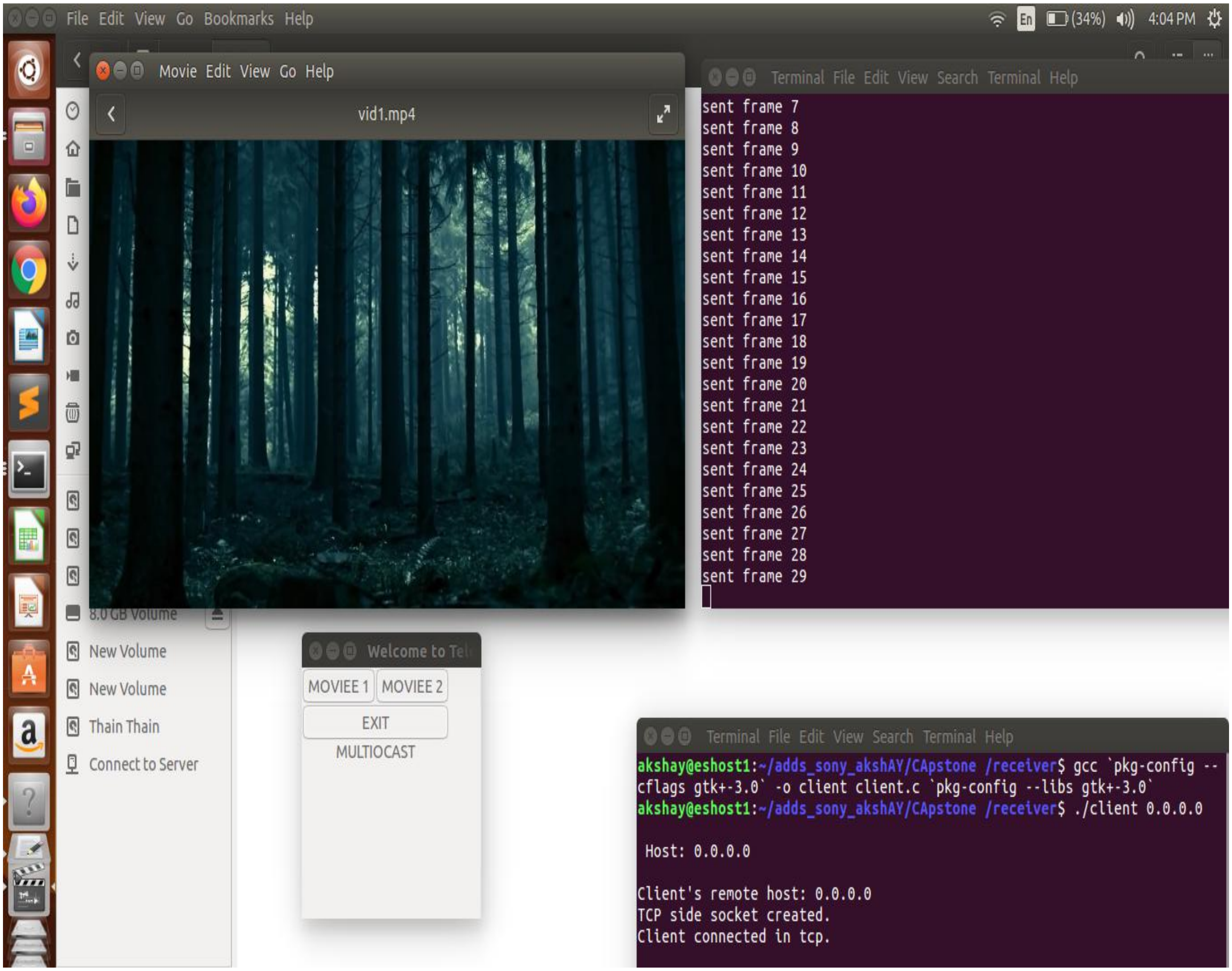


```
akshay@eshost1:~/adds_sony_akshAY/CApstone /receiver$ gcc `pkg-config --cflags g
tk+-3.0` -o client client.c `pkg-config --libs gtk+-3.0`
akshay@eshost1:~/adds_sony_akshAY/CApstone /receiver$ ./client 0.0.0.0

Host: 0.0.0.0

Client's remote host: 0.0.0.0
TCP side socket created.
Client connected in tcp.

start send successfully

------------------------------------------------=====================

Site name: www.friends.com
Site description: iconic series: F.R.I.E.N.D.S.

---------------------------------------------==========================

Site name: www.himym.com
Site description: How I met your mother


-------------------------------------------------------
info port        : 5432
Station Number   : 1
Station name     : F.R.I.E.N.D.S
Multicast Address: 239.192.4.1
Data port        : 5433
Bit rate         : 1087 kb/s


-------------------------------------------------------
info port        : 5432
Station Number   : 2
Station name     : H.I.M.Y.M
Multicast Address: 239.192.4.2
Data port        : 5433
Bit rate         : 891 kb/s
-------------------------------------------------------
```

reciever.c file is executed by the client.c file

the reciever.c file contains command to convert the video file which is in mp4 to mpeg format

command : ffmpeg -i inputfile.mp4 -f mpegts streamable_output.mp4



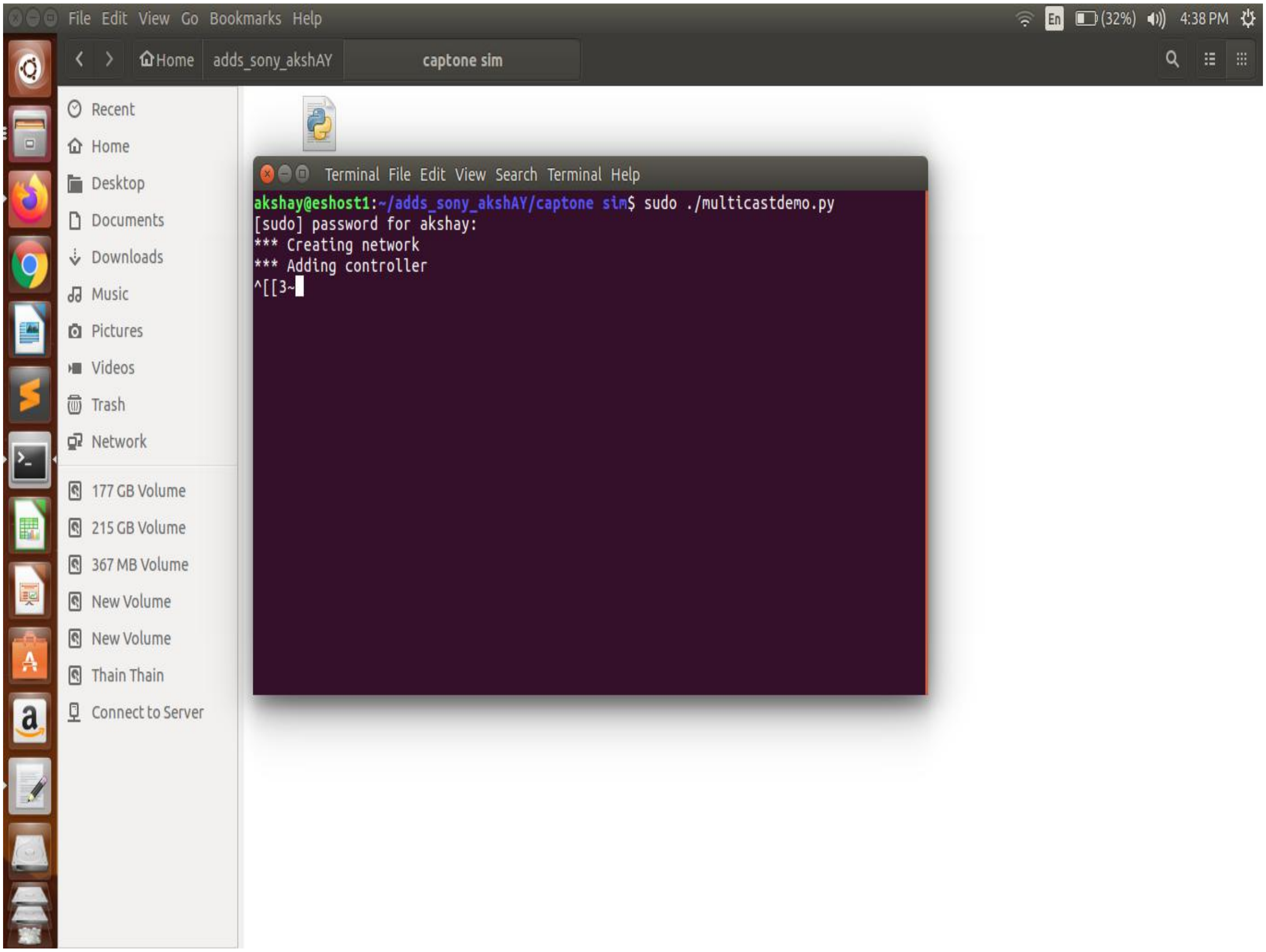
```
sent frame 7
sent frame 8
sent frame 9
sent frame 10
sent frame 11
sent frame 12
sent frame 13
sent frame 14
sent frame 15
sent frame 16
sent frame 17
sent frame 18
sent frame 19
sent frame 20
sent frame 21
sent frame 22
sent frame 23
sent frame 24
sent frame 25
sent frame 26
sent frame 27
sent frame 28
sent frame 29
```

```
akshay@eshost1:~/adds_sony_akshAY/CApstone /receiver$ gcc `pkg-config --
cflags gtk+-3.0` -o client client.c `pkg-config --libs gtk+-3.0`
akshay@eshost1:~/adds_sony_akshAY/CApstone /receiver$ ./client 0.0.0.0

 Host: 0.0.0.0

Client's remote host: 0.0.0.0
TCP side socket created.
Client connected in tcp.
```

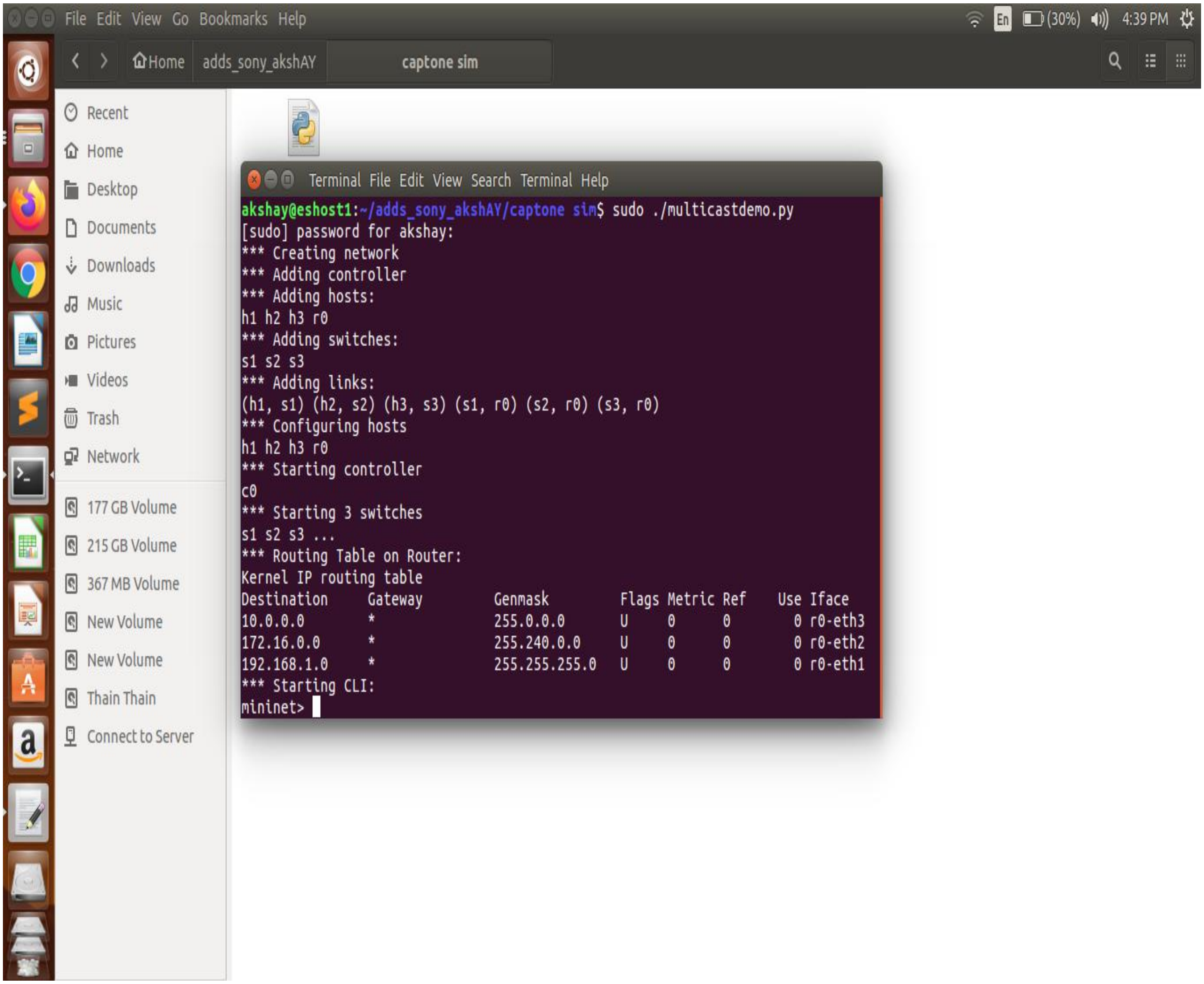
Simulation in Mininet

We used mininet to simulate the multast application where in we had r0 h1 h2 h3 as the nodes.

We had a custom topology of 3 nodes h1 h2 h3



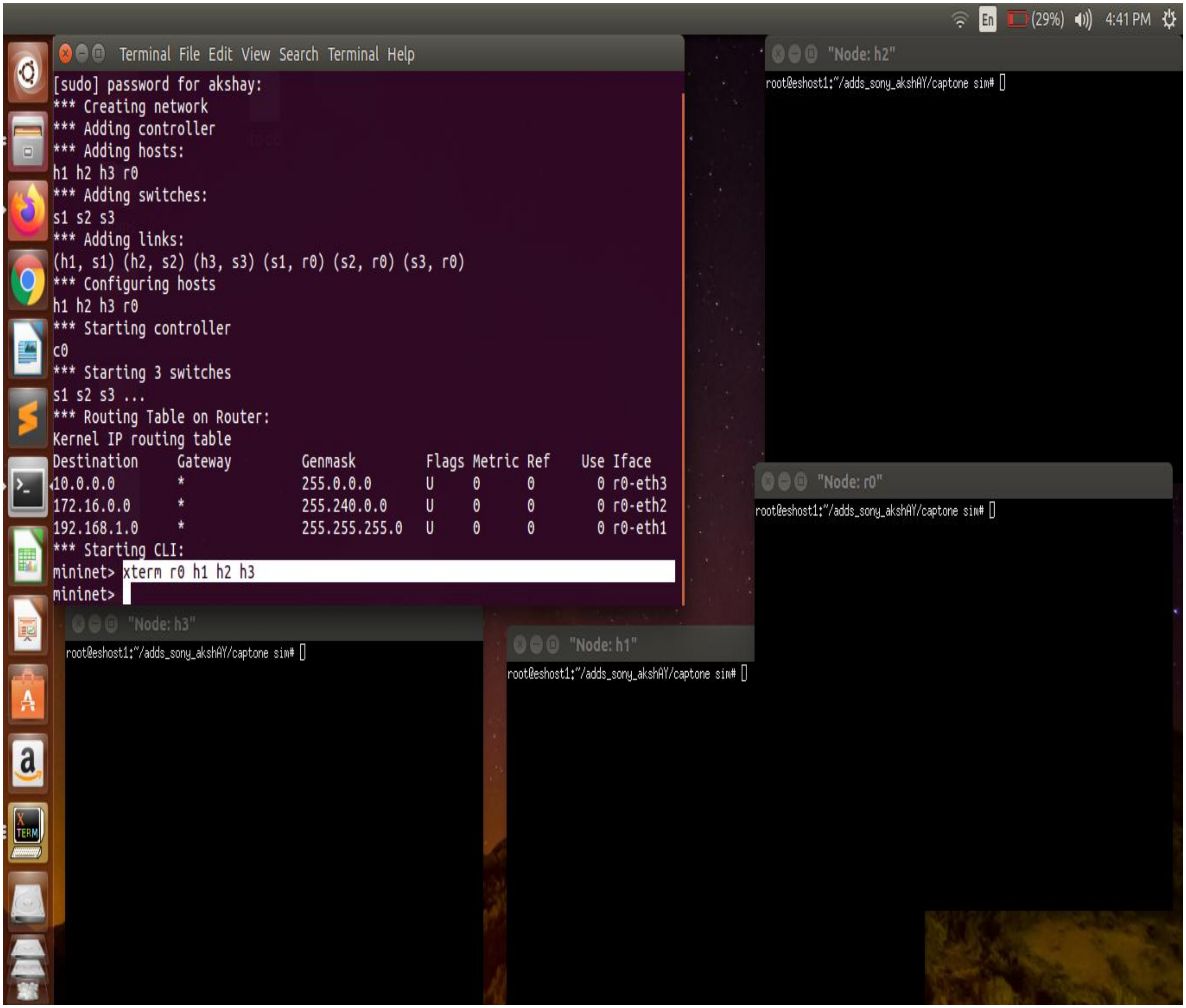
```
akshay@eshost1:~/adds_sony_akshAY/captone sim$ sudo ./multicastdemo.py
[sudo] password for akshay:
*** Creating network
*** Adding controller
^[[3~
```



```
akshay@eshost1:~/adds_sony_akshAY/captone sim$ sudo ./multicastdemo.py
[sudo] password for akshay:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 r0
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, r0) (s2, r0) (s3, r0)
*** Configuring hosts
h1 h2 h3 r0
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Routing Table on Router:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        *               255.0.0.0       U     0      0        0 r0-eth3
172.16.0.0      *               255.240.0.0     U     0      0        0 r0-eth2
192.168.1.0     *               255.255.255.0   U     0      0        0 r0-eth1
*** Starting CLI:
mininet>
```

Using xtern r0 h1 h2 h3 we create three terminal nodes ,h1 h2 h3 acting as video requesting nodes.

The routing table is also shown below



```
[sudo] password for akshay:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 r0
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, r0) (s2, r0) (s3, r0)
*** Configuring hosts
h1 h2 h3 r0
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Routing Table on Router:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        *               255.0.0.0       U     0      0        0 r0-eth3
172.16.0.0      *               255.240.0.0     U     0      0        0 r0-eth2
192.168.1.0     *               255.255.255.0   U     0      0        0 r0-eth1
*** Starting CLI:
mininet> xterm r0 h1 h2 h3
mininet>
```

"Node: h2"

```
root@eshost1:~/adds_sony_akshAY/captone sim#
```

"Node: r0"

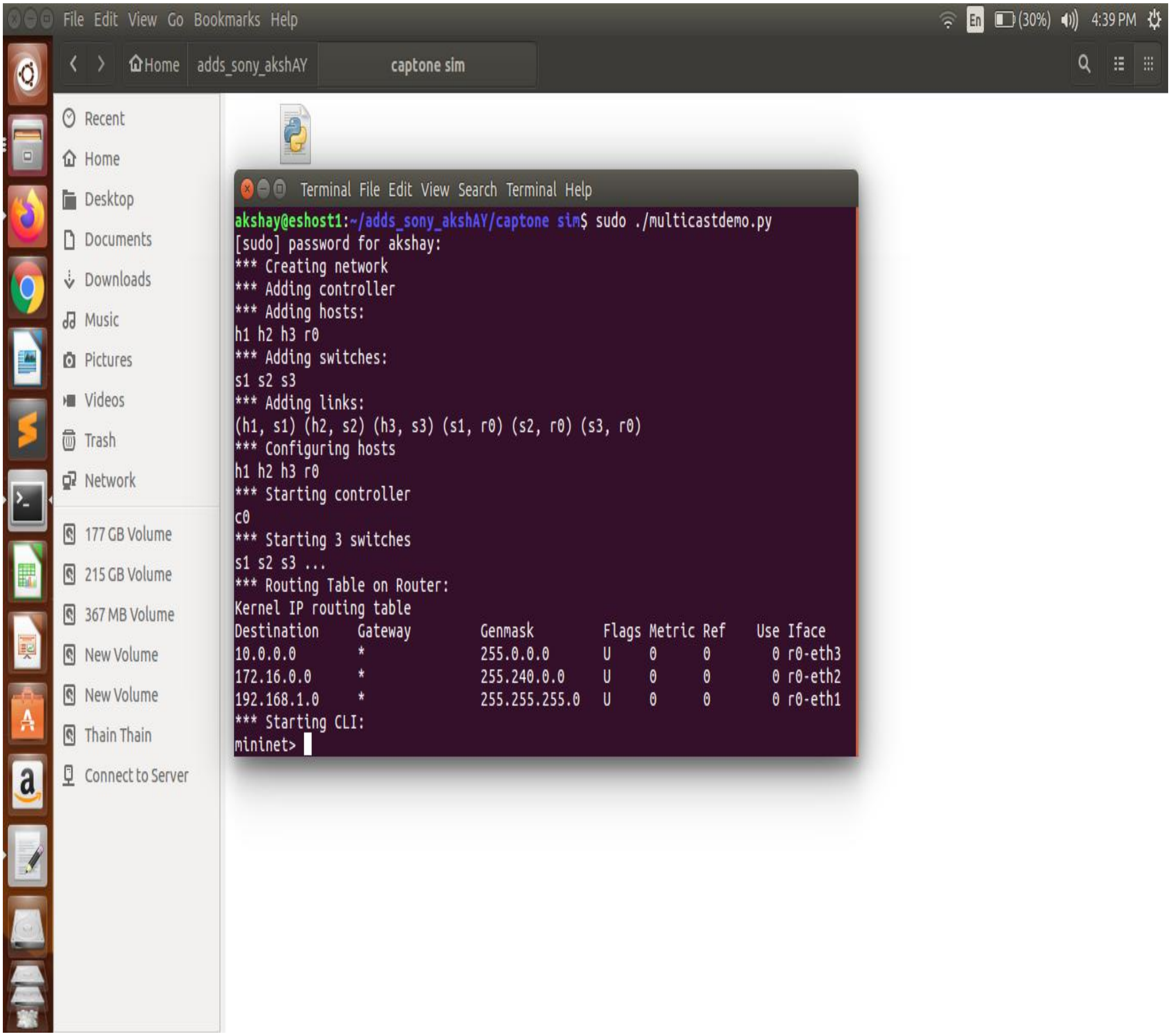
```
root@eshost1:~/adds_sony_akshAY/captone sim#
```

"Node: h3"

```
root@eshost1:~/adds_sony_akshAY/captone sim#
```

"Node: h1"

```
root@eshost1:~/adds_sony_akshAY/captone sim#
```

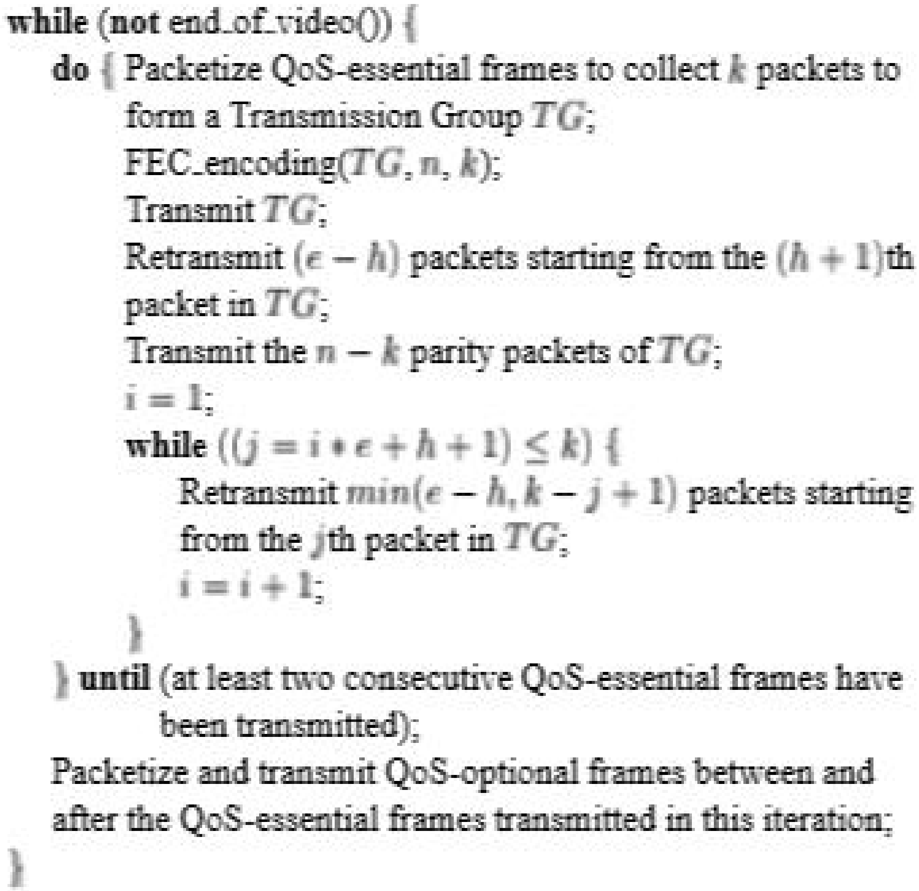


```
akshay@eshost1:~/adds_sony_akshAY/captone sim$ sudo ./multicastdemo.py
[sudo] password for akshay:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 r0
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, r0) (s2, r0) (s3, r0)
*** Configuring hosts
h1 h2 h3 r0
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Routing Table on Router:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        *               255.0.0.0       U     0      0        0 r0-eth3
172.16.0.0      *               255.240.0.0     U     0      0        0 r0-eth2
192.168.1.0     *               255.255.255.0   U     0      0        0 r0-eth1
*** Starting CLI:
mininet> 
```

The next thing to implement is the FEC and FEC_retrans algorithm in Mininet .

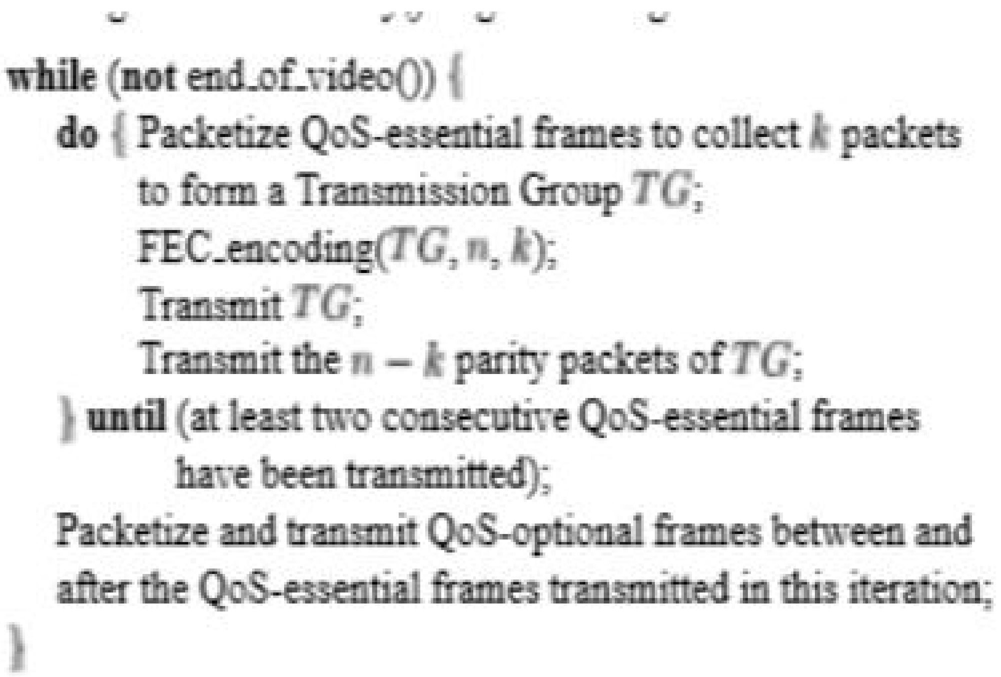
The sudo code for the algorithm is as follows

1. FEC Algorithm



```
while (not end_of_video()) {
    do { Packetize QoS-essential frames to collect $k$ packets to
         form a Transmission Group $TG$;
         FEC_encoding($TG, n, k$);
         Transmit $TG$;
         Retransmit $(\epsilon - h)$ packets starting from the $(h + 1)$th
         packet in $TG$;
         Transmit the $n - k$ parity packets of $TG$;
         $i = 1$;
         while $((j = i * \epsilon + h + 1) \leq k)$ {
             Retransmit $min(\epsilon - h, k - j + 1)$ packets starting
             from the $j$th packet in $TG$;
             $i = i + 1$;
         }
    } until (at least two consecutive QoS-essential frames have
             been transmitted);
    Packetize and transmit QoS-optional frames between and
    after the QoS-essential frames transmitted in this iteration;
}
```

2. FEC_retrans



```
while (not end_of_video()) {
    do { Packetize QoS-essential frames to collect $k$ packets
            to form a Transmission Group $TG$;
            FEC_encoding($TG, n, k$);
            Transmit $TG$;
            Transmit the $n - k$ parity packets of $TG$;
    } until (at least two consecutive QoS-essential frames
                have been transmitted);
    Packetize and transmit QoS-optional frames between and
    after the QoS-essential frames transmitted in this iteration;
}
```

The remaining task is try to add these in the code and simulate this algorithm which works on the principle of video differentiation which segregates the video into Qos Essential Frames and Qos Optional Frames and plot graphs for the analysis of error control.