INTERNET-RADIO-MULTICASTING-MULTIMEDIA-OVER-IP

A COURSE PROJECT REPORT

By

ADITI KHATRI
(RA1911030010984)
ANCHAL PORWAL
(RA1911003011002)

Under the guidance of

DR. KOWSIGAN M

In partial fulfillment for the Course

of

18CSC302J - COMPUTER NETWORKS

in

Computer Science & Engineering with Specialization in Cyber Security

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Kattankulathur, Chenpalpattu District
NOVEMBER 2021

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report " **internet-radio-multicasting-multimedia-over-ip**" is the bonafide work of **Aditi khatri (RA1911030010984), Anchal porwal (RA1911003011002),**

who carried out the project work under my supervision.

SIGNATURE

Course Coordinator
Associate Professor,
Data Science and Business Systems
SRM Institute of Science and Technology
Potheri, SRM Nagar, Kattankulathur,
Tamil Nadu 603203

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement. We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr.M.LAKSHMI, Professor and Head, Data Science and Business Systems,** and **Course Coordinator Dr.E. Sasikala, Associate Professor, Data Science and Business Systems** for their constant encouragement and support.

We are highly thankful for our Course project's Internal guide **Dr. C.N.S Vinoth Kumar, Assistant Professor, Computer Science and Engineering with specialization in Cyber Security,** for **his** assistance, timely suggestion, and guidance throughout the duration of this course project.

We extend my gratitude to the **Student, Dr. B. Amutha ma'am(HOD)** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

ABSTRACT

Multimedia—an integrated and interactive presentation of speech, audio, video, graphics, and text—has become a major driving force behind a multitude of applications. Increasingly, multimedia content is being accessed by a large number of diverse users and clients at anytime, and from anywhere, across various communication channels such as the Internet and wireless networks. As mobile cellular and wireless LAN networks are evolving to carry multimedia data, an all-IP-based system akin to the Internet is likely to be employed due to its cost efficiency, improved reliability, allowance of easy implementation of new services, independence of control and transport, and importantly, easy integration of multiple networks. In this project, we will design a internet-radio-multicasting-multimedia-over-ip .

INTRODUCTION

This project will work as mentioned below.First, Client will send a join request to the server to join the multicast group.After that Server will provide station list, site info to the client through TCP. Then whichever station it selects from the station list, it is connected to that station.All the stations are sending data, irrespective of client is connected or not. This functionality is incorporated to relate more with real life situation, e.g Tv/radio sends data even though there is no receiver connected.Whenever receiver connects to a particular station, it starts receiving live-streaming videos from that station. Receiver can pause, resume, change station or even terminate at any given time from GUI using thread.

REQUIREMENT ANALYSIS

C files

server.c

- gcc server.c
- ./a.out

station1.c

- gcc station1.c
- ./a.out 239.192.4.1

station2.c

- gcc station2.c
- ./a.out 239.192.4.2

client.c

- gcc `pkg-config --cflags gtk+-3.0` -o client client.c `pkg-config --libs gtk+-3.0`
- sudo ./client <IP-ADDRESS of the server>

receiver.c

- Compiled and executed by the client.

ARCHITECTURE AND DESIGN

1. 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
2. Sender to Receiver: UDP
    a. UDP is used to send multicast live-streaming videos from sender
    to all receivers who joined multicast group.
3. Implementation of GUI: using gtk .
4. For all the previous functionalities, we have implemented four different
functions which handles pause, resume, change the station, and
5. Terminate accordingly.

● Station Information

Station 1 name: F.R.I.E.N.D.S
Station 2 name: H.I.M.Y.M

Port Used for both stations: 5432

Multicast Address for station 1: 239.192.4.1
Multicast Address for station 2: 239.192.4.2

● Features:

- ❖ Receiver is receiving audio as well as video without any loss of data through UDP.

- ❖ Both the stations are sending data on the same port, but having different IP addresses

.

● Buffer calculation:

- ❖ Time (t seconds) is initially declared and bit-rate will depend on station. Thus, the size of the buffer should be large enough to hold received data of t seconds.

- ❖ Every time when the station is changed buffer-size will be recalculated according to t and bit-rate. For a particular station bit-rate is fixed and which is approximated.

- ❖ By calculation, we got the buffer size = 64000.

IMPLEMENTATION

SENDER

Sender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MC_PORT 5432
#define BUF_SIZE 64000


int main(int argc, char * argv[]){

  int s,s_tcp; /* socket descriptor */
  struct sockaddr_in sin,sin_t; /* socket struct */
  char buf[BUF_SIZE];
  int len;
  socklen_t sin_len;
  sin_len = sizeof(sin);
  /* Multicast specific */
  char *mcast_addr; /* multicast address */
  int tcp_ac;

  /* Add code to take port number from user */
  if (argc==2) {
    mcast_addr = argv[1];
  }
  else {
    fprintf(stderr, "usage: sender multicast_address\n");
    exit(1);
  }

        /* build address data structure for TCP*/
  bzero((char *)&sin_t, sizeof(sin_t));
  sin_t.sin_family = AF_INET;
  sin_t.sin_addr.s_addr = INADDR_ANY;
  sin_t.sin_port = htons(MC_PORT);
```

```
   /* Create a TCP socket */
  if ((s_tcp = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("server TCP: socket");
    exit(1);
  }
  else
  {
              printf("Socket created\n\n");
  }
  // binding server addr structure to s_tcp


  if ((bind(s_tcp, (struct sockaddr *) &sin_t, sizeof(sin_t))) < 0) {
    perror("receiver: bind()");
    close(s_tcp);
    exit(1);
  }
  else
  {
              printf("Binded\n\n");
  }

  if((listen(s_tcp, 512000))<0)
  {
              printf("Error in listening\n");
  }
  else
        printf("Listened\n\n");


  //if((tcp_ac = accept(s_tcp, (struct sockaddr*)&sin_t, &length))<0)
//Accept
  if((tcp_ac = accept(s_tcp, (struct sockaddr*)&sin_t, &sin_len))<0)  //Accept
  {
              printf("Error in accepting\n");
  }
  else
        printf("Accepted\n\n");


    int num;

        //usleep(5000000);
        //while(num==0){
        int n = recv(tcp_ac, &num, sizeof(num)+1, 0);
//}
    //read(s_tcp, &num, sizeof(num));
    printf("Station number is: %d\n\n",num);


  if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("server UDP: socket");
    exit(1);
```

```
  }

  // build address data structure
  memset((char *)&sin, 0, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = inet_addr(mcast_addr);
  sin.sin_port = htons(MC_PORT);

        int length = sizeof(mcast_addr);
  //printf("\nWrite messages below to multicast!\n\n");


  memset(buf, 0, sizeof(buf));

  /*while (fgets (buf, BUF_SIZE, stdin)) {
    if ((len = sendto(s, buf, sizeof(buf), 0,
                      (struct sockaddr *)&sin,
                     sizeof(sin))) == -1) {
      perror("sender: sendto");
      exit(1);
    }

    memset(buf, 0, sizeof(buf));

  }*/

  /*len = recvfrom(s, buf, sizeof(buf), 0,(struct sockaddr *)&sin, &sin_len);

        buf[len-2]='\0';*/
        FILE *fp=NULL;
        fp=fopen("vid1.mp4","rb");

                //printf("entered!");
                if(fp==NULL)
                {
                        sendto(s, "File Not Found\n", strlen("File Not Found\n")+1,
0,(struct sockaddr*)&sin, sin_len);
                }
                else
                {
                int tot_frame,i;
                fseek(fp, 0, SEEK_END);
                long fsize = ftell(fp);
                long p=(fsize % 64000);
                if ((fsize % BUF_SIZE) != 0)
                {
                        tot_frame = (fsize / BUF_SIZE) + 1;
                }
                else
                        tot_frame = (fsize / BUF_SIZE);

                printf("last packets are :%ld\n\n", p);
                printf("Total number of packets are :%d\n\n", tot_frame);
```

```
                fseek(fp, 0, SEEK_SET);

                if(sendto(s,&(tot_frame),sizeof(tot_frame),0,(struct sockaddr*)&sin,
sin_len)<0)
                    printf("Error in sending frame no:\n");

                if(tot_frame==0 || tot_frame==1)
                {
                      char *string = malloc(fsize + 1);
                              fread(string,1,fsize+1,fp);
                              fseek(fp, 0, SEEK_SET);
                              int x=sendto(s,string,fsize+1,0,(struct sockaddr*)&sin,
sin_len);
                      printf("%d\n",x);
                }
                else
                {
                      for(i=1;i<=tot_frame;i++)
                      {
                                      char *string = malloc(BUF_SIZE);
                                      len=fread(string,1,BUF_SIZE,fp);
                                      fseek(fp, 0, SEEK_CUR);
                                      int x=sendto(s,string,len,0,(struct
sockaddr*)&sin,sin_len);
                                      printf("sent frame %d\n",i);
                                            //usleep(2*100000);
                                  usleep(2*100000);
                      }

                }

               fclose(fp);
        }



                fp=fopen("va1.mp4","rb");

                printf("\tCurrent Stream: va1.mp4\n\n");
                printf("\tNext Stream: xyz.mp4\n\n");

                //printf("entered!\n");
                if(fp==NULL)
                {
                       //sendto(s, "File Not Found\n", strlen("File Not Found\n")+1,
0,(struct sockaddr*)&sin, sin_len);
                       sendto(s, "File Not Found\n", strlen("File Not Found\n")+1,
0,(struct sockaddr*)&sin, sin_len);
                }
                else
                {
                        int tot_frame,i;
```

```
                        fseek(fp, 0, SEEK_END);
                        long fsize = ftell(fp);
                        long p=(fsize % 64000);
                        if ((fsize % BUF_SIZE) != 0)
                        {
                                tot_frame = (fsize / BUF_SIZE) + 1;
                        }
                        else
                                tot_frame = (fsize / BUF_SIZE);

                        printf("last packets are :%ld\n\n", p);
                        printf("Total number of packets are :%d\n\n", tot_frame);

                        fseek(fp, 0, SEEK_SET);

                        if(sendto(s,&(tot_frame),sizeof(tot_frame),0,(struct
sockaddr*)&sin, sin_len)<0)
                            printf("Error in sending frame no:\n");

                        if(tot_frame==0 || tot_frame==1)
                        {
                              char *string = malloc(fsize + 1);
                                      fread(string,1,fsize+1,fp);
                                      fseek(fp, 0, SEEK_SET);
                                      int x=sendto(s,string,fsize+1,0,(struct
sockaddr*)&sin, sin_len);
                              printf("%d\n",x);
                        }
                        else
                        {
                              for(i=1;i<=tot_frame;i++)
                              {
                                            char *string = malloc(BUF_SIZE);
                                            len=fread(string,1,BUF_SIZE,fp);
                                            fseek(fp, 0, SEEK_CUR);
                                            int x=sendto(s,string,len,0,(struct
sockaddr*)&sin,sin_len);
                                            printf("sent frame %d\n",i);
                                      usleep(2*100000);
                              }

                        }

                        fclose(fp);
                }


                fp=fopen("cn.mp4","rb");

                //printf("entered!\n");
                if(fp==NULL)
                {
                       sendto(s, "File Not Found\n", strlen("File Not Found\n")+1,
```

```
0,(struct sockaddr*)&sin, sin_len);
                }
                else
                {
                        int tot_frame,i;
                        fseek(fp, 0, SEEK_END);
                        long fsize = ftell(fp);
                        long p=(fsize % 64000);
                        if ((fsize % BUF_SIZE) != 0)
                        {
                                tot_frame = (fsize / BUF_SIZE) + 1;
                        }
                        else
                                tot_frame = (fsize / BUF_SIZE);

                        printf("last packets are :%ld\n\n", p);
                        printf("Total number of packets are :%d\n\n", tot_frame);

                        fseek(fp, 0, SEEK_SET);

                        if(sendto(s,&(tot_frame),sizeof(tot_frame),0,(struct
sockaddr*)&sin, sin_len)<0)
                            printf("Error in sending frame no:\n");

                        if(tot_frame==0 || tot_frame==1)
                        {
                              char *string = malloc(fsize + 1);
                                      fread(string,1,fsize+1,fp);
                                      fseek(fp, 0, SEEK_SET);
                                      int x=sendto(s,string,fsize+1,0,(struct
sockaddr*)&sin, sin_len);
                              printf("%d\n",x);
                        }
                        else
                        {
                              for(i=1;i<=tot_frame;i++)
                              {
                                            char *string = malloc(BUF_SIZE);
                                            len=fread(string,1,BUF_SIZE,fp);
                                            fseek(fp, 0, SEEK_CUR);
                                            int x=sendto(s,string,len,0,(struct
sockaddr*)&sin,sin_len);
                                            printf("sent frame %d\n",i);
                                      usleep(10000);
                              }

                        }

                        fclose(fp);
                }

             close(s);
  return 0; }
```

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 512000

//Structure of site info
struct site_info
{
        uint8_t site_name_size;
        char site_name[ 20 ];
        uint8_t site_desc_size;
        char site_desc[ 100 ];
        uint8_t station_count;
};

//Structure of station info
struct station_info
{
        uint8_t station_number;
        char station_name[50];
        char multicast_address[32];
        uint16_t data_port;
        uint16_t info_port;
        uint32_t bit_rate;
};

int main()
{
  struct station_info stat1,stat2,stat3;
  struct site_info site1,site2,site3;
  struct sockaddr_in sin;

  char buf[MAX_LINE];
  int len;
  int s, new_s;
  char str[INET_ADDRSTRLEN];
  int num;

  /* build address data structure */
```

```
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);


  /* setup passive open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  inet_ntop(AF_INET, &(sin.sin_addr), str, INET_ADDRSTRLEN);
  printf("Server is using address %s and port %d.\n", str, SERVER_PORT);

  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
  }
  else
    printf("Server bind done.\n");

  listen(s, MAX_PENDING);      //Server listening

  while(1)
  {
                if((new_s = accept(s, (struct sockaddr*)&sin, &len))<0)
//Accept
                {
                        printf("Error in accepting\n");
                }
                else
                        printf("Accepted\n\n");

                 recv(new_s, buf, sizeof(buf), 0);     //Receive "Start"

                 printf("%s\n",buf);

                 /*Declaration of site info for station 1 */
                 bzero(&site1,sizeof(site1));
                 strcpy(site1.site_name,"www.friends.com");
                 strcpy(site1.site_desc,"iconic series: F.R.I.E.N.D.S.  ");

                 /*Declaration of station info for station 1 */
                 bzero(&stat1,sizeof(stat1));
                 stat1.station_number = 1;
                 strcpy(stat1.multicast_address,"239.192.4.1");
                 stat1.data_port = 5433;
                 stat1.info_port = 5432;
                 stat1.bit_rate = 1087;
                 strcpy(stat1.station_name,"F.R.I.E.N.D.S");

                 /*Declaration of site info for station 2 */
                 bzero(&site2,sizeof(site2));
```

```
                strcpy(site2.site_name,"www.himym.com");
                strcpy(site2.site_desc,"How I met your mother ");

                /*Declaration of station info for station 2 */
                bzero(&stat2,sizeof(stat2));
                stat2.station_number = 2;
                strcpy(stat2.multicast_address,"239.192.4.2");
        stat2.data_port = 5433; //for udp
        stat2.info_port = 5432; //for tcp
        stat2.bit_rate = 891;
                strcpy(stat2.station_name,"H.I.M.Y.M");

                /*Sending structure of site info */
                send(new_s, &(site1), sizeof(site1)+1, 0);
                send(new_s, &(site2), sizeof(site2)+1, 0);

                /*Sending structure of station info*/
                send(new_s, &(stat1), sizeof(stat1)+1, 0);
                send(new_s, &(stat2), sizeof(stat2)+1, 0);

            close(new_s);               //Close socket
    }
 return 0;
}
```

Station1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MC_PORT 5433
#define BUF_SIZE 64000

//structure of song info
struct song_info
{
          char song_name[ 50 ];
          uint16_t remaining_time_in_sec;
          char next_song_name[ 50 ];
};

int main(int argc, char * argv[])
{
          int s; // socket descriptor
```

```
struct sockaddr_in sin; // socket struct
char buf[BUF_SIZE];
int len;
socklen_t sin_len;
sin_len = sizeof(sin);

// Multicast specific
char *mcast_addr; // multicast address
char * video[5];

//Array of video names
video[0] = "vid7.mp4";
video[1] = "vid4.mp4";
video[2] = "vid1.mp4";
video[3] = "vid3.mp4";


// Add code to take port number from user
if (argc==2)
{
             mcast_addr = argv[1];
}
else
{
              fprintf(stderr, "usage: sender multicast_address\n");
              exit(1);
}

if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0)       //Create socket
{
              perror("server UDP: socket");
              exit(1);
}

// build address data structure
memset((char *)&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = inet_addr(mcast_addr);
sin.sin_port = htons(MC_PORT);

printf("Connected in first station\n\n ");
memset(buf, 0, sizeof(buf));

FILE *fp=NULL;
int i;
while(1)
{
        for(i=0; i<4; i++)           //Sending videos one by one
        {

             fp=fopen(video[i],"rb");

                    if(fp==NULL)      //Check if file exist
```

```
                                {
                                    printf("\nFile not found\n");
                                }
                                else
                                {
                                        int tot_frame,i;
                                        fseek(fp, 0, SEEK_END);
                                        long fsize = ftell(fp);
        //Calculate file size
                                        long p=(fsize % 64000);
                                        if ((fsize % BUF_SIZE) != 0)
                                        {
                                                tot_frame = (fsize / BUF_SIZE) + 1;
                                        }
                                        else
                                                tot_frame = (fsize / BUF_SIZE);

                                        printf("last packets are :%ld\n\n", p);
                                        printf("Total number of packets are :%d\n\n",
tot_frame);

                                        fseek(fp, 0, SEEK_SET);

                                        if(tot_frame==0 || tot_frame==1)
                                        {
                                                        char *string = malloc(fsize +
1);
                                                        fread(string,1,fsize+1,fp);
                                                        fseek(fp, 0, SEEK_SET);
                                                        int
x=sendto(s,string,fsize+1,0,(struct sockaddr*)&sin, sin_len); //Sending data to the
receiver
                                                        printf("%d\n",x);
                                        }
                                        else
                                        {

                                                        for(i=1;i<=tot_frame;i++)
                                                        {
                                                             char *string =
malloc(BUF_SIZE);

len=fread(string,1,BUF_SIZE,fp);
                                                             fseek(fp, 0, SEEK_CUR);
                                                             int
x=sendto(s,string,len,0,(struct sockaddr*)&sin,sin_len); //Sending data frame by
frame
                                                             printf("sent frame
%d\n",i);
                                                          //usleep(854400);
                                                          usleep(400000);
                                                        }
```

```
                                        }
                                        fclose(fp); //close file pointer
                                }
                        }
                }
                return 0;
}
```

Startion2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MC_PORT 5433
#define BUF_SIZE 64000

//structure of song info
struct song_info
{
        char song_name[ 50 ];
        uint16_t remaining_time_in_sec;
        char next_song_name[ 50 ];
};

int main(int argc, char * argv[])
{
        int s; // socket descriptor
        struct sockaddr_in sin; // socket struct
        char buf[BUF_SIZE];
        int len;
        socklen_t sin_len;
        sin_len = sizeof(sin);

        // Multicast specific
        char *mcast_addr; // multicast address
        char * video[5];

        //Array of video names
        video[0] = "vid5.mp4";
        video[1] = "vid6.mp4";
        video[2] = "vid7.mp4";
        video[3] = "vid8.mp4";
```

```
        // Add code to take port number from user
        if (argc==2)
        {
                    mcast_addr = argv[1];
        }
        else
        {
                     fprintf(stderr, "usage: sender multicast_address\n");
                     exit(1);
        }

        if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
        {
                     perror("server UDP: socket");
                     exit(1);
        }

        // build address data structure
        memset((char *)&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = inet_addr(mcast_addr);
        sin.sin_port = htons(MC_PORT);

        printf("Connected in second station\n\n ");
        memset(buf, 0, sizeof(buf));

        FILE *fp=NULL;
        int i;
        while(1)
        {
            for(i=0; i<4; i++)       //Sending videos one by one
            {

              fp=fopen(video[i],"rb");

              if(fp==NULL)           //Check if file exist
              {
                       printf("\nFile not found\n");
                }
              else
              {
                                      int tot_frame,i;
                                      fseek(fp, 0, SEEK_END);
                                      long fsize = ftell(fp);
    //Calculate file size
                                      long p=(fsize % 64000);
                                      if ((fsize % BUF_SIZE) != 0)
                                      {
                                              tot_frame = (fsize / BUF_SIZE) + 1;
                                      }
                                      else
                                              tot_frame = (fsize / BUF_SIZE);
```

```
                                        printf("last packets are :%ld\n\n", p);
                                        printf("Total number of packets are
:%d\n\n", tot_frame);

                                        fseek(fp, 0, SEEK_SET);

                                        if(tot_frame==0 || tot_frame==1)
                                        {
                                                        char *string = malloc(fsize +
1);
                                                        fread(string,1,fsize+1,fp);
                                                        fseek(fp, 0, SEEK_SET);
                                                        int
x=sendto(s,string,fsize+1,0,(struct sockaddr*)&sin, sin_len); //Sending data to the
receiver
                                                        printf("%d\n",x);
                                        }
                                        else
                                        {

                                                        for(i=1;i<=tot_frame;i++)
                                                        {
                                                                        char *string
= malloc(BUF_SIZE);

len=fread(string,1,BUF_SIZE,fp);
                                                                        fseek(fp, 0,
SEEK_CUR);
                                                                        int
x=sendto(s,string,len,0,(struct sockaddr*)&sin,sin_len); //Sending data frame by
frame
                                                                        printf("sent
frame %d\n",i);
                                                                       usleep(300000);
                                                        }

                                        }
                                        fclose(fp); //close file pointer
                        }
                }
        }
        return 0;
}
```

RECEIVER

Receiver.c

```
//Receiver with gtk
#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/if.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <pthread.h>

#define MC_PORT 5433
#define BUF_SIZE 64000

//structure of song info
struct song_info
{
          char song_name[ 50 ];
          uint16_t remaining_time_in_sec;
          char next_song_name[ 50 ];
};

pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;

// declaring mutex
//pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int done = 0;
int r=1;

/* Function for Pause */
int func1(GtkWidget *widget,gpointer   data)
{
   g_print ("video is pause...\n");
   done=1;
   return 0;
}

/* Function for Resume */
int func2(GtkWidget *widget,gpointer   data)
{
   g_print ("video resumed...\n");
```

```
    done=0;
         r=0;
         usleep(5000000);
    return 0;
}

/* Function for Change Station */
void func3(GtkWidget *widget,gpointer   data)
{
    g_print ("Request to change the station\n");
    system("pkill ffplay");
    remove("live_data.mp4");
    exit(0);
}

/* Function for Terminate */
int func4(GtkWidget *widget,gpointer   data)
{
  g_print ("Terminated from current station...\nBYE BYE...\n");
  system("pkill ffplay");
  remove("live_data.mp4");
  exit(0);
  return 0;
}

void* threadFunction(void* args)
{

  printf("in thread\n");

  int s,s_tcp; /* socket descriptor */
  struct hostent *hp;
  struct sockaddr_in sin,cliaddr; /* socket struct */
  char *if_name; /* name of interface */
  struct ifreq ifr; /* interface struct */


  char buf[BUF_SIZE],buf1[BUF_SIZE];
  int len;
  char str[500];
  /* Multicast specific */
  char *mcast_addr; /* multicast address */
  struct ip_mreq mcast_req;  /* multicast join struct */
  struct sockaddr_in mcast_saddr; /* multicast sender*/
  socklen_t mcast_saddr_len;
  char add[32];

  mcast_addr = args;
  if_name = "wlan0";

  /* create socket */
  if ((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
  {
```

```
                perror("receiver: socket");
                exit(1);
  }
  else
  printf("udp Socket created\n");

  int x=sizeof(sin);

  /* build address data structure */
  memset((char *)&sin, 0, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = htonl(INADDR_ANY);
  sin.sin_port = htons(MC_PORT);

  /*Use the interface specified */
  memset(&ifr, 0, sizeof(ifr));
  strncpy(ifr.ifr_name , if_name, sizeof(if_name)-1);


  if ((setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE, (void *)&ifr, sizeof(ifr))) < 0)
  {
      perror("receiver: setsockopt() error");
      close(s);
      exit(1);
  }
  else
                printf("setsockopt\n");

  /* bind the socket */

  if ((bind(s, (struct sockaddr *) &sin, sizeof(sin))) < 0)
  {
    perror("receiver: bind()");
    close(s);
    exit(1);
  }
  else
  printf("udp binded\n");
  /* Multicast specific code follows */

  /* build IGMP join message structure */
  mcast_req.imr_multiaddr.s_addr = inet_addr(mcast_addr);
  mcast_req.imr_interface.s_addr = htonl(INADDR_ANY);

  /* send multicast join message */
  if ((setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void*) &mcast_req,
sizeof(mcast_req))) < 0)
  {
    perror("mcast join receive: setsockopt()");
    exit(1);
  }

  /* receive multicast messages */
```

```
  printf("\nReady to listen!\n\n");
  int i=0;
  FILE *fp;
  fp=fopen("live_data.mp4","wb");

  /* reset sender struct */
  memset(&mcast_saddr, 0, sizeof(mcast_saddr));
  mcast_saddr_len = sizeof(mcast_saddr);

  while(1)
  {
        if(done==0)
       {
              memset(&buf, 0, sizeof(buf));
              int err,l;

              len = recvfrom(s, buf, sizeof(buf), 0,(struct sockaddr*)&mcast_saddr,
&mcast_saddr_len);

              if(len<0)
              {
                  printf("Error in receiving\n");
              }
              else
              {
                  printf("%d Receiving  %d\n",i,len);
                  fwrite(buf,1,len,fp);
                  if(i==8)
                   {
                                                          system("gnome-terminal -- sh -
c 'ffplay -i live_data.mp4;'");
                                              }
              }
              i++;
         }
  }

  fclose(fp);
  close(s);

}

int main(int argc, char *argv[])
{
    char *mcast_addr;

    if(argc==2)
                mcast_addr= argv[1];
         else
                printf("\nInvalid arguments");

    pthread_t id;
    pthread_create(&id,NULL,&threadFunction,mcast_addr);
```

```
    gtk_init (&argc, &argv);
    GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    GtkWidget *grid;
    GtkWidget *button;
    GtkWidget *label;
    GdkColor color;

    gtk_window_set_title (GTK_WINDOW (window), "Control!");
    gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);            //Set
window size
    gdk_color_parse ("light yellow", &color);        //Set color of GUI window
    g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);

    grid = gtk_grid_new ();

    gtk_container_add (GTK_CONTAINER (window), grid);
    gtk_widget_modify_bg ( GTK_WIDGET(window), GTK_STATE_NORMAL, &color);


    button = gtk_button_new_with_label ("Pause");

    g_signal_connect (button, "clicked", G_CALLBACK (func1), NULL);  //call function
1 for pause

    gtk_grid_attach (GTK_GRID (grid), button, 5, 5, 5, 5);

    button = gtk_button_new_with_label ("Resume");
    g_signal_connect (button, "clicked", G_CALLBACK (func2), NULL);  //call function
2 for resume

    gtk_grid_attach (GTK_GRID (grid), button, 5, 10, 5, 5);

    button = gtk_button_new_with_label ("Change station");
        g_signal_connect (button, "clicked", G_CALLBACK (func3), NULL);     //call
function 3 for change station

    gtk_grid_attach (GTK_GRID (grid), button, 5, 15, 5, 5);

    button = gtk_button_new_with_label ("Terminate");
    g_signal_connect (button, "clicked", G_CALLBACK (func4), NULL);  //call function
4 for teminate

    gtk_grid_attach (GTK_GRID (grid), button, 5, 20, 5, 5);


    gtk_widget_show_all (window);
    gtk_main ();


}
```

Client.c

```
//Client_with gtk

#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/if.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MC_PORT 5432
#define BUF_SIZE 64000

struct site_info
{
        uint8_t site_name_size;
        char site_name[ 20 ];
        uint8_t site_desc_size;
        char site_desc[ 100 ];
        uint8_t station_count;
};

struct station_info
{
uint8_t station_number;
char station_name[50];
char multicast_address[32];
uint16_t data_port;

uint16_t info_port;
uint32_t bit_rate;
};

int updateLabel(GtkLabel *lab, gchar *display)
{
        gtk_label_set_text (GTK_LABEL(lab), display); //set label to "display"
        return 0;
}

/* function for button 1 "station 1" */
int func1(GtkWidget *widget,gpointer   data, GtkLabel *lab)
{
        gchar *display;
        display="Connected to Station 1!";
        updateLabel(GTK_LABEL(lab), display);
```

```
        while(gtk_events_pending())
        gtk_main_iteration();
        system("gcc `pkg-config --cflags gtk+-3.0` -o receiver receiver.c  `pkg-
config --libs gtk+-3.0` ");
        char string[200] = "sudo ./receiver ";
        strcat(string,"239.192.4.1");
        system(string);
        return 0;
}

/* function for button 2 "station 2" */
int func2(GtkWidget *widget,gpointer   data, GtkLabel *lab)
{
        gchar *display;
        display="Connected to Station 2!";
        updateLabel(GTK_LABEL(lab), display);
        while(gtk_events_pending())
        gtk_main_iteration();
        system("gcc `pkg-config --cflags gtk+-3.0` -o receiver receiver.c  `pkg-
config --libs gtk+-3.0` ");
        char string1[200] = "sudo ./receiver ";
        strcat(string1,"239.192.4.2");
        system(string1);
        return 0;
}

int func3(GtkWidget *widget,gpointer   data, GtkLabel *lab)
{
        exit(0);
        return 0;
}


/* function for button 3*/
int main(int argc, char * argv[])
{
  char string[200]="./receiver ";
  char string1[200]="./temp ";
  FILE *fp;
  int s,s_tcp;                                                          /*
socket descriptor */
  struct hostent *hp;
  struct sockaddr_in sin,sin_t,cliaddr;   /* socket struct */
  char *if_name;
        /* name of interface */
  struct ifreq ifr;                                                     /*
interface struct */
    char *host;
  struct station_info stat1,stat2,stat3;
  struct site_info site1,site2,site3;
  //struct site_info site;
```

```
char buf[BUF_SIZE],buf1[BUF_SIZE];
int len;
char str[200];
/* Multicast specific */
char *mcast_addr;                                   /* multicast address */
struct ip_mreq mcast_req;             /* multicast join struct */
struct sockaddr_in mcast_saddr; /* multicast sender*/
socklen_t mcast_saddr_len;


if (argc==2) {
  host = argv[1];
}
else {
  fprintf(stderr, "usage: simplex-talk host\n");
  exit(1);
}
printf("\n Host: %s\n\n",host);
/* translate host name into peer's IP address */
hp = gethostbyname(host);
if (!hp) {
  fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
  exit(1);
}
else
  printf("Client's remote host: %s\n", argv[1]);



/* build address data structure for TCP*/
memset((char *)&sin_t, 0, sizeof(sin_t));
sin_t.sin_family = AF_INET;
bcopy(hp->h_addr, (char *)&sin_t.sin_addr,hp->h_length);
//sin_t.sin_addr.s_addr = INADDR_ANY;
sin_t.sin_port = htons(MC_PORT);

// Create a TCP socket
if ((s_tcp = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
  perror("server TCP: socket");
  exit(1);
}
else
  printf("TCP side socket created.\n");

if (connect(s_tcp, (struct sockaddr *)&sin_t, sizeof(sin_t)) < 0)
  {
   // perror("simplex-talk: connect");
   printf("\ntcp not connected\n");
    close(s_tcp);
    exit(1);
  }
else
```

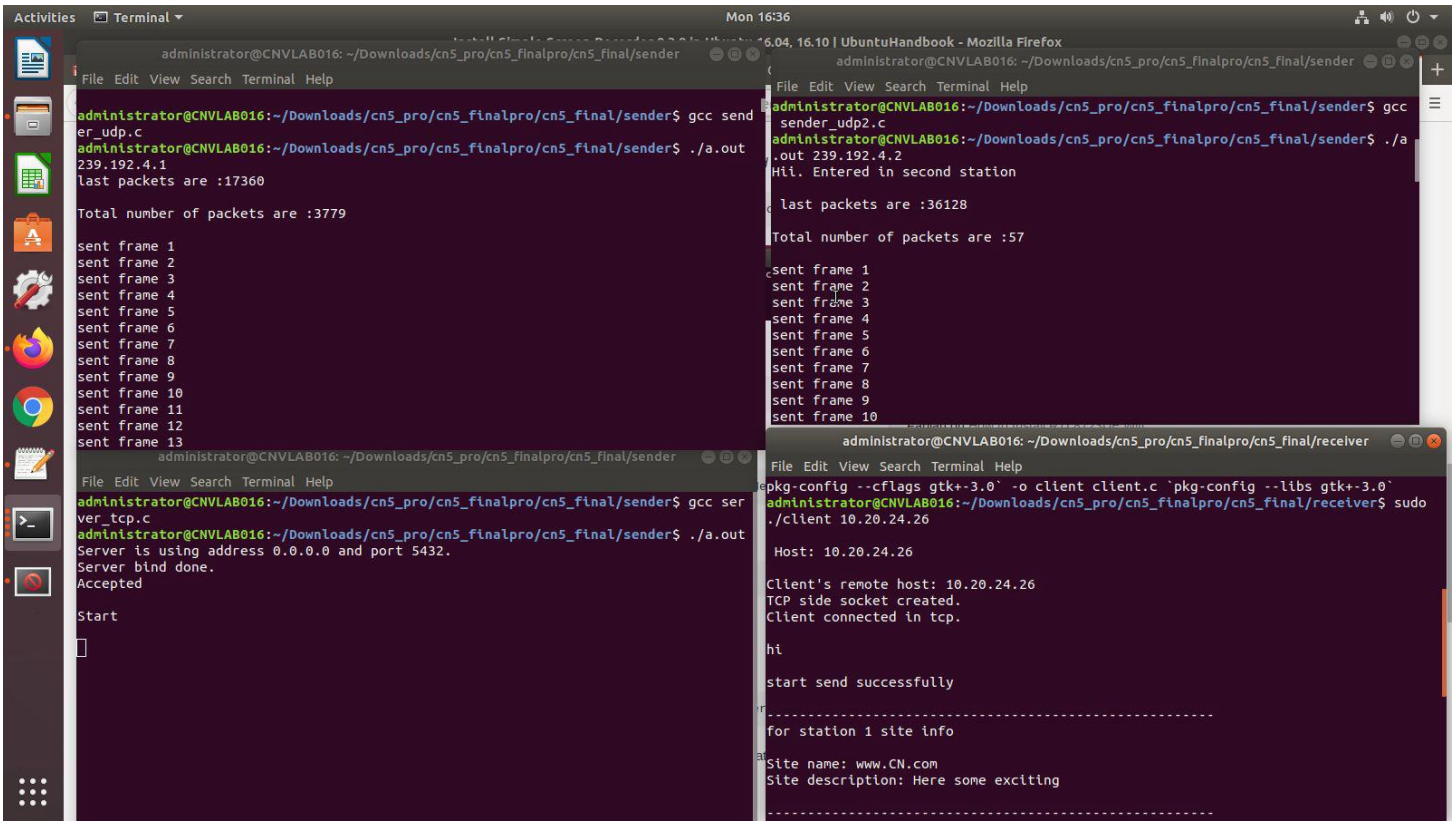
```
    printf("Client connected in tcp.\n");


 int num;

if((send(s_tcp, "Start\n", strlen("Start\n")+1, 0)) <0)
 printf("\nclient not ready to receive\n");

else
 printf("\nstart send successfully\n");

 //Reset
 bzero(&stat1,sizeof(stat1));
 bzero(&stat2,sizeof(stat2));
 bzero(&site1,sizeof(site1));
 bzero(&site2,sizeof(site2));


 //Receive site info for station 1
 recv(s_tcp, &(site1), sizeof(site1)+1, 0);

 printf("\n-----------------------------------------------------\n");
 printf("For station 1 site info\n");
 printf("\nSite name: %s",site1.site_name);
 printf("\nSite description: %s\n",site1.site_desc);

 //Receive site info for station 2
 recv(s_tcp, &(site2), sizeof(site2)+1, 0);

 printf("\n-----------------------------------------------------\n");
 printf("For station 2 site info\n");
 printf("\nSite name: %s",site2.site_name);
 printf("\nSite description: %s\n",site2.site_desc);


 //Receive station info for station 1
 recv(s_tcp, &(stat1), sizeof(stat1)+1, 0);

 printf("\n\n-----------------------------------------------------\n");
 printf("info port         : %d\n", stat1.info_port);
 printf("Station Number    : %d\n",stat1.station_number);
 printf("Station name      : %s\n",stat1.station_name);
 printf("Multicast Address: %s\n",stat1.multicast_address);
 printf("Data port         : %d\n", stat1.data_port);
 printf("Bit rate          : %d kb/s\n",stat1.bit_rate);

 //Receive station info for station 2
 recv(s_tcp, &(stat2), sizeof(stat2)+1, 0);

 printf("\n\n-----------------------------------------------------\n");
 printf("info port         : %d\n", stat2.info_port);
 printf("Station Number    : %d\n",stat2.station_number);
 printf("Station name      : %s\n",stat2.station_name);
```

```
  printf("Multicast Address: %s\n",stat2.multicast_address);
  printf("Data port        : %d\n", stat2.data_port);
  printf("Bit rate         : %d kb/s",stat2.bit_rate);
  printf("\n----------------------------------------------------------\n");


  //GUI for station list
  gtk_init (&argc, &argv);
  GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  GtkWidget *grid;
  GtkWidget *button;
  GtkWidget *label;
  GtkWidget *lab;
  GdkColor color;

  gtk_window_set_title (GTK_WINDOW (window), "Welcome to Television!");
  gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);
  gdk_color_parse ("light yellow", &color);           //Set color of GUI window
  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);


  lab = gtk_label_new ("Hello!!");
  grid = gtk_grid_new ();

  gtk_container_add (GTK_CONTAINER (window), grid);


  button = gtk_button_new_with_label ("F.R.I.E.N.D.S");
  g_signal_connect (button, "clicked", G_CALLBACK (func1), lab);     //Call func1
for station 1

  gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);

  button = gtk_button_new_with_label ("H.I.M.Y.M");
  g_signal_connect (button, "clicked", G_CALLBACK (func2), lab);     //Call func2
for station 2

  gtk_grid_attach (GTK_GRID (grid), button, 1, 0, 1, 1);

  button = gtk_button_new_with_label ("EXIT");
  g_signal_connect (button, "clicked", G_CALLBACK (func3), lab);     //Call func2
for exit

  gtk_grid_attach (GTK_GRID (grid), button, 0, 2, 2, 1);


  gtk_grid_attach (GTK_GRID(grid), lab, 0, 4, 4, 1);

  gtk_widget_show_all (window);
  gtk_main ();

  close(s_tcp);        //close socket
  return 0;}
```

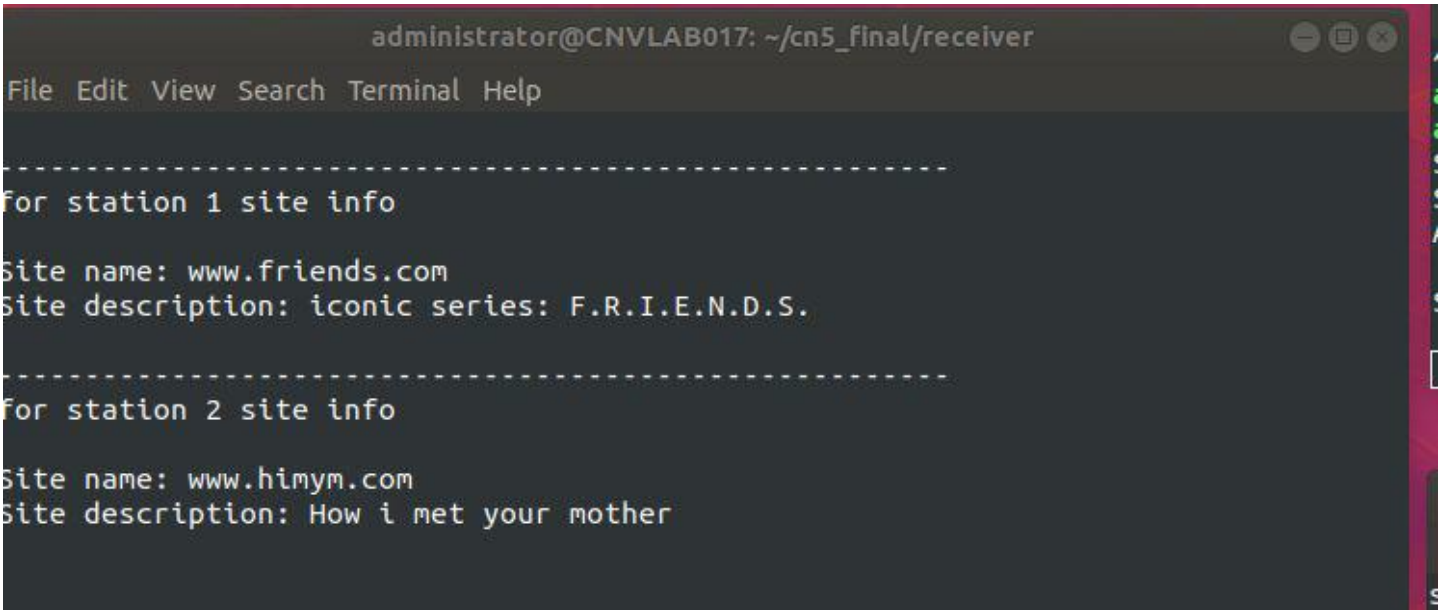
EXPERIMENT RESULT & ANALYSIS

6.1. RESULTS:

1.Successful compilation of all files.



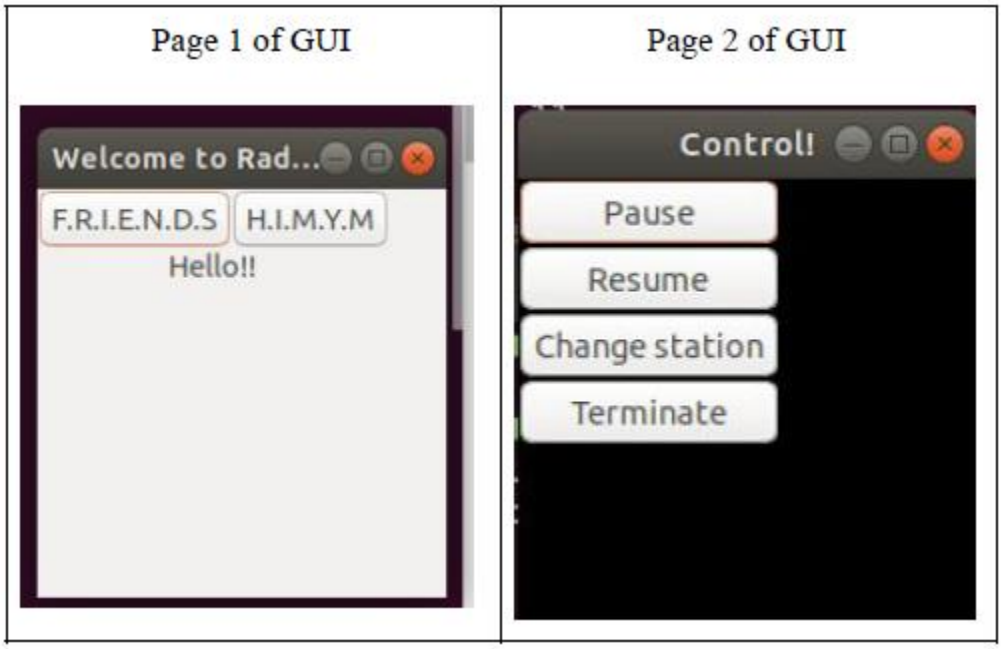
2. Site info and station info at client side



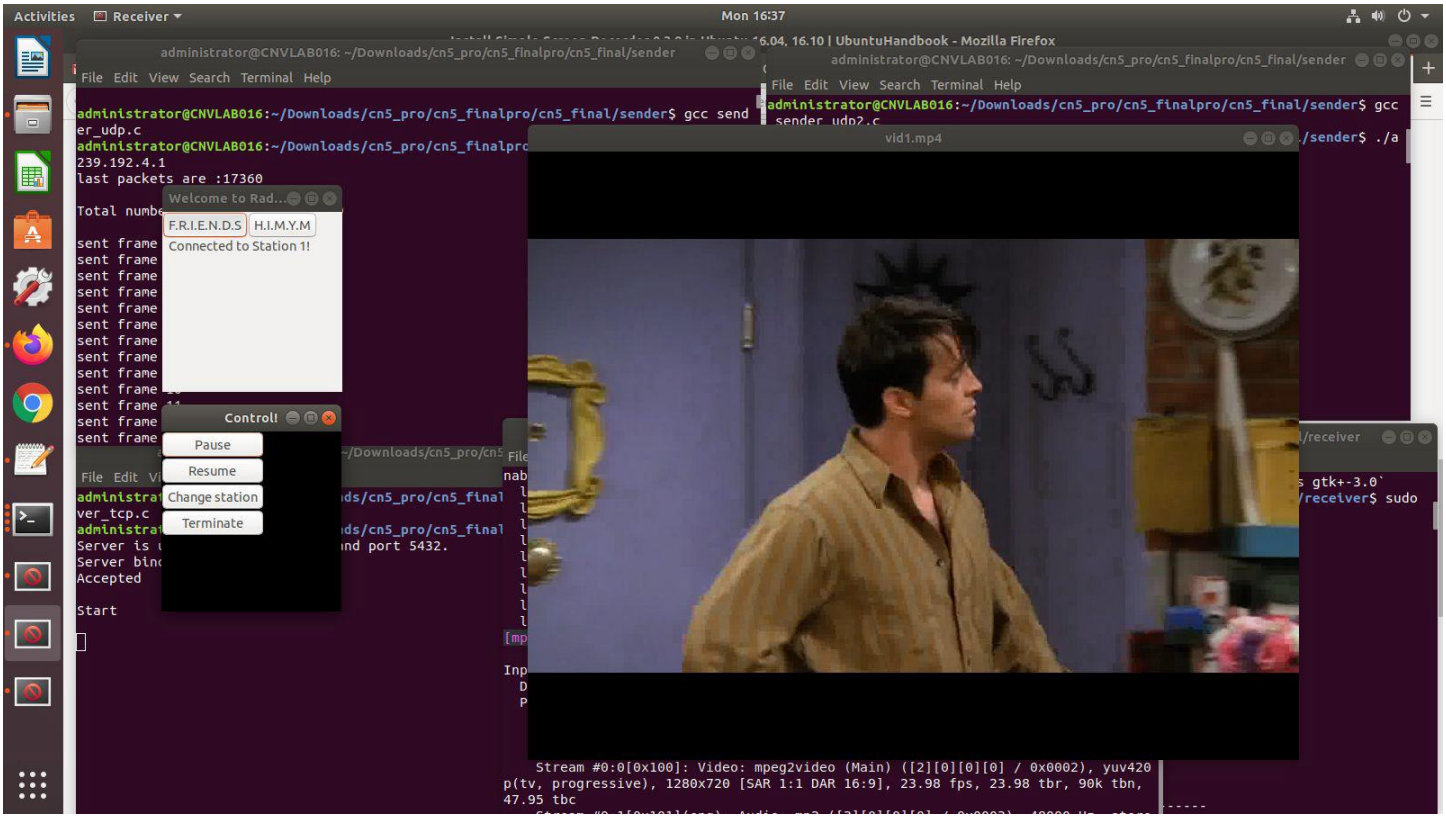
Station info at client side

```
----------------------------------------------------------
info port: 9531
Station Number: 1
Station name: friends
Multicast Address: 239.192.4.1
Data port: 5431
Bit rate: 1087 kb/s


----------------------------------------------------------
info port: 9532
Station Number: 2
Station name: himym
Multicast Address: 239.192.4.2
Data port: 5431
Bit rate: 891 kb/s
----------------------------------------------------------
```

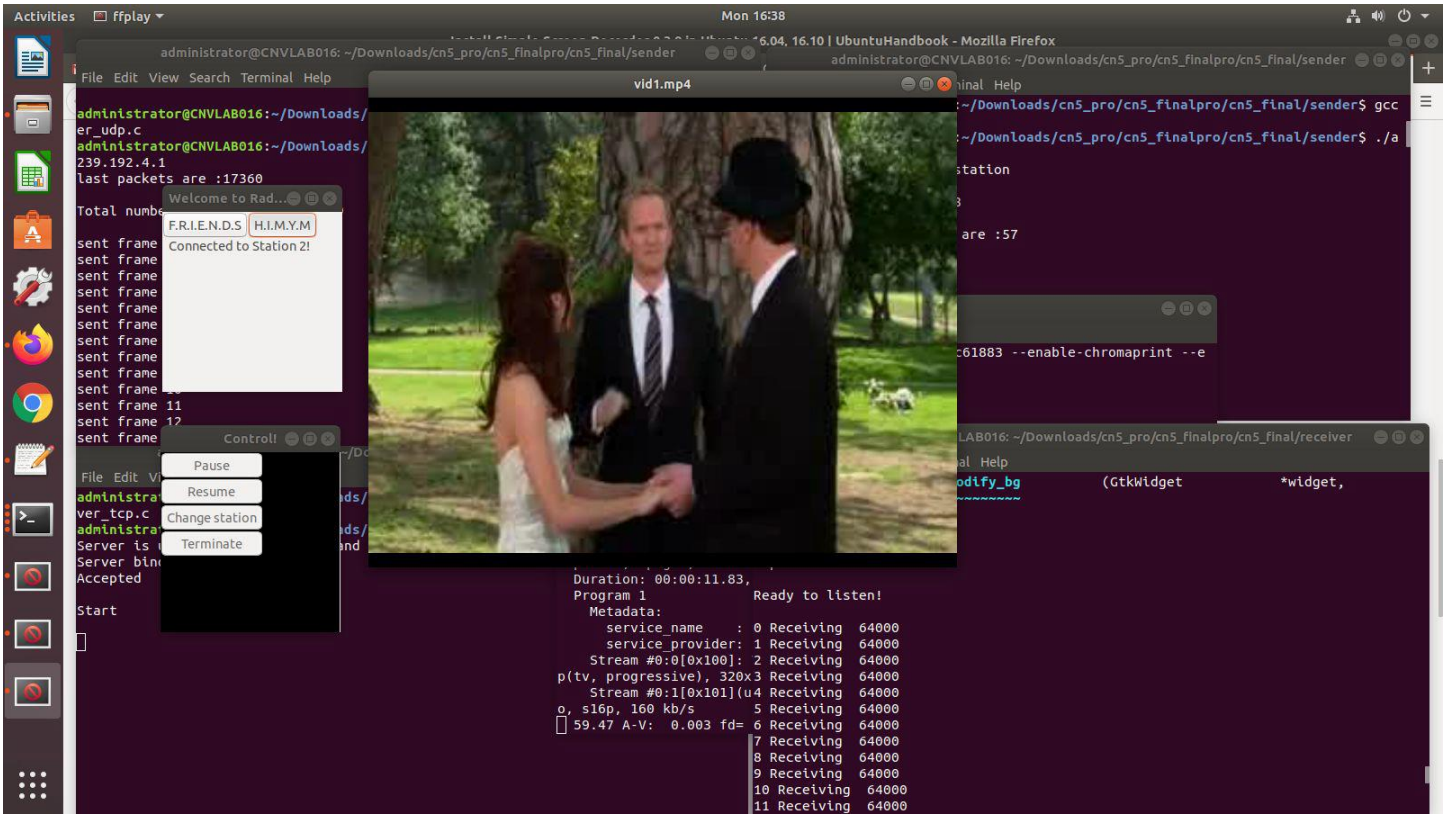
3. Station selection GUI window



4.Station 1 :F.R.I.E.N.D.S



6. Station 2: H.I.M.Y.M



6.2. INDIVIDUAL CONTRIBUTION:

TASK	MEMBER
✓ Server, Sender code:	Aditi
✓ Client, Receiver code:	Anchal
✓ GUI creation:	Aditi
✓ GUI connection with socket programming:	Anchal
✓ Debugging of code, Report creation	Both

REFERENCES

1. http://mcl.usc.edu/wp-content/uploads/2014/01/200402-Editorial-for-the-Special-Issue-on-Multimedia-over-IP-and-Wireless-Networks.pdf
2. https://en.wikipedia.org/wiki/Multicast
3. https://www.cse.wustl.edu/~jain/talks/ftp/netsem5.pdf