MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

COMPUTER NETWORKS LAB
CSE 3162

Mini Project Report

INTERNET RADIO MULTICASTING
MULTIMEDIA OVER IP

SUBMITTED BY:

Isha Sthanpati – 200905258

Daksh Soni - 200905206

ABSTRACT

In order to simulate TV, mobile networks, etc., we want to demonstrate concurrent TCP connections and how multimedia can be streamed across these. Multicast is a true broadcast. There is no direct relationship between the clients and server. This is similar to tuning into a station on a radio. Each client that listens to the multicast adds no additional overhead on the server. In fact, the server sends out only one stream. The same load is experienced on the server whether only one client or 1,000 clients are listening.
It's the job of the network switch or router to receive this single stream from the encoder and send it out to all the client

TABLE OF CONTENTS

1 Introduction

1.1 General Introduction to the topic

The Internet's protocol stack is significantly influenced by TCP, which offers a wrap of reliability over the underlying best-effort IP network. However, since the advent of multimedia streaming over the Internet, there has been contention that some of the same characteristics that make TCP a crucial component of the stability of the Internet also cause some unfavorable streaming conditions. To lessen these negative consequences of TCP, a number of solutions have been put forth, ranging from workarounds to alternate service models. Recent suggestions contend that employing techniques like client side buffering and receiver-driven bandwidth sharing, streaming can be implemented over TCP without encountering major problems, refuting the long-held belief that TCP is unfriendly to streaming applications.

1.2 Hardware and Software Requirements

The specific software and hardware involved vary widely as well in the real world. It is entirely possible to perform all functions in software (cheap but slow). It is also entirely possible to do all of them in hardware (fast but expensive). Dedicated network devices such as Cisco routers, etc. will perform more functions in hardware than say, a PC running Linux configured as a router.

Generally Layer 1 and mostly Layer 2 is guaranteed to be performed in hardware. Layer 3 is mostly done in hardware on an enterprise-level router such as those from Cisco. Many NICs come with a feature called "TCP Offload Engine" that can accelerate most of Layer 3 and 4 via hardware on the NIC. Your consumer-level routers from Walmart usually do the routing function entirely in software. Layer 5, 6, and 7 are usually not done in hardware, and if they are, the devices that do so are termed "accelerators".

For our project, we have used C programs and functions for establishing TCP and UDP connections. For the multicast radio, we have used GTK as the GUI. Videos have been saved as .mp4 files.

2 Problem Definition

The Internet's ubiquity has long made it an attractive platform for distributed multimedia applications. A particularly elusive goal has been effective streaming solutions. In streaming, both transferring and viewing or listening can be done simultaneously where as in downloading these two are temporally sequential; the transfer of the content must be completed before the playing started.

The real-time distribution of continuous audio and video data via streaming multimedia applications accounts for a significant, and expanding, portion of the Internet traffic. It is expected that real-time media-streaming traffic will increase rapidly, and will soon make up a significant portion of the total Internet bandwidth. The key characteristics of such real-time streaming applications are the potential for high data rates, and the need for low and predictable latency and latency variance, these problems can be minimized, if not fixed, using TCP connection as the bridge

3 Objectives

- To demonstrate how video can be broadcasted across concurrent TCP connections.
- To establish UDP connection between receiver and device, in order to display multimedia content.
- To simulate a smaller version of multimedia devices like Television and Radio.

4 Methodology

4.1 Execution Steps

- First of all, 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.
- Used Media player: ffplay. All videos at station side is converted using ffmpeg to make it stream able.
- Command used for conversion: ffmpeg -i inputfile.mp4 -f mpegts streamable_output.mp4
- Receiver can pause, resume, change station or even terminate at any given time from GUI using thread.
    - Change Station : Receiver can change it anytime.
        - Firstly it is disconnected from the station to whom it was connected earlier and then is connected to a new station as per receiver's choice and starts receiving respective live-streaming of data from that station.
    - Terminate : Whenever it is selected, it is disconnected from the station it was connected earlier. It will exit station, we have done this using 'pkill.
- Thread is used so that two processes can run in parallel
    - GUI
    - Socket programming to send and receive the data.

4.2 Design Configurations

- Client to Server: TCP
    - TCP is used for one to one connection from client to server and it is used for station info and site info
- Sender to Receiver: UDP
    - UDP is used to send multicast live-streaming videos from sender to all receivers who joined multicast group
- Implementation of GUI: using gtk
- For all the previous functionalities, we have implemented four different functions which handles pause, resume, change the station, and Terminate accordingly

4.3 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

4.4 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.

5 Implementation Details

5.1 server.c

```
#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 SERVER_PORT 5435
#define MAX_PENDING 5
#define MAX_LINE 512000

//Structure of site info
struct site_info
{
uint8_t site_name_size;  //unsigned single-byte
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;  //address family
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) { //protocol family
perror("simplex-talk: socket");
exit(1);
}

//converts from an Internet address in binary format, specified by src, to standard text format
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; //number of bits that are conveyed or processed per unit of time
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.gg.com");
strcpy(site2.site_desc,"feel good series: Gilmore Girls ");

/*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,"Gilmore Girls");

/*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;
}
```

5.2 client.c

```
#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 5435
#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;   //store info on host
```

```
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("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! Please make your choice.");
  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 ("Gilmore Girls");
  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 func3 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;
}
```

5.3 station.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 5435
#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] = "vid1.mp4";
     video[1] = "vid2.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));

     while(1)
     {
               len=strlen(video[0]);
```

```
                int x=sendto(s,video[0],len,0,(struct sockaddr*)&sin,sin_len);
    }
    return 0;
}
```

5.4 receiver.c

```
#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 5435
#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;

/* Function for Change Station */
void func1(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 func2(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 = "enp0s3";

  /* 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);
```

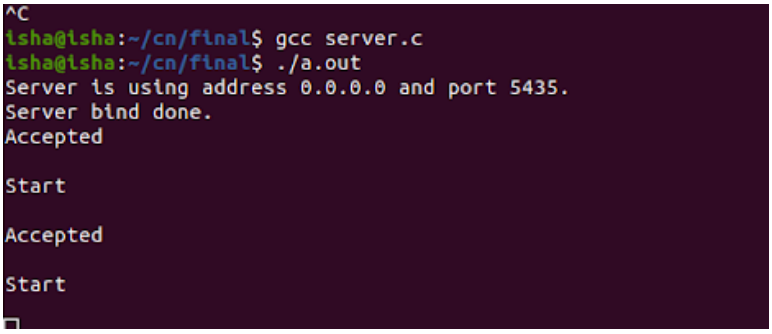
```
if ((setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE, (void *)&ifr, sizeof(ifr))) < 0)
 {
      perror("receiver: setsockopt() error");
      close(s);
      exit(1);
 }
 else
       printf("setsockopt\n");

 /* 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");

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

 while(1)
      {
                memset(&buf, 0, sizeof(buf));

                len = recvfrom(s, buf, sizeof(buf), 0,(struct sockaddr*)&mcast_saddr, &mcast_saddr_len);
                buf[8]='\0';
                if(len<8)
                {
                printf("Error in receiving\n");
                }
                else
                {
```

```
printf("%d Receiving  %d\n",i,len);
                printf("buffer %s\n",buf);
                char sr[100]="ffplay -i ";
                strcat(sr,buf);
                system(sr);
                sr[0]='\0';
                }
  }

  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 ("Change station");
     g_signal_connect (button, "clicked", G_CALLBACK (func1), 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 (func2), NULL);    //call function 4 for teminate
```

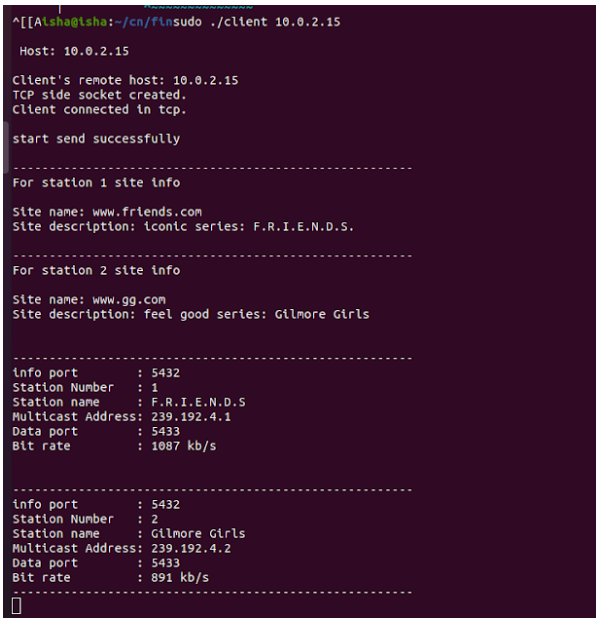
```
gtk_grid_attach (GTK_GRID (grid), button, 5, 20, 5, 5);


        gtk_widget_show_all (window);
        gtk_main ();


}
```

6. OUTPUT

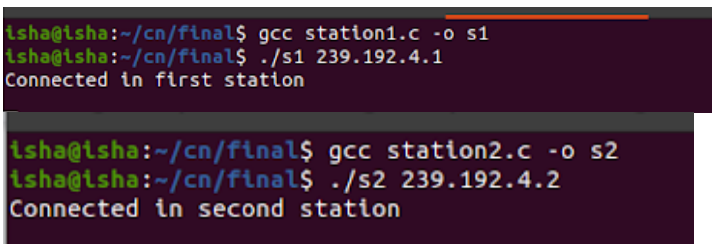
Server.c :



Client.c



Station1.c and Station2.c



Receiver.c:

When ‘F.R.I.E.N.D.S’ button was clicked:

When ‘Gilmore Girls’ button was clicked:

7. CONTRIBUTIONS

Sender(server.c , station1.c , station2.c) - Isha Sthanpati
Receiver(receiver.c , client.c) - Daksh Soni
Report Content and Editing- Isha Sthanpati, Daksh Soni

8. REFERENCES

https://www.gtk.org/docs/installations/linux
https://formulae.brew.sh/formula/ffmpeg#default
https://ffmpeg.org/ffplay.html
https://www.rtautomation.com/rtas-blog/multicast/#:~:text=Unicast%20uses%20TCP%20(Transmission%20Control,remember%2C%20is%20an%20acknowledged%20protocol