MULTICAST STREAMING SYSTEM

A PROJECT REPORT

Submitted by

Sai Rishyanth Visinigiri [RA2111026010280]
Suhas Ganga [RA2111026010281]
Manisha Manoj [RA2111026010274]
Adithya Nandan Naidu Bandaru [RA2111026010278]
Shreeya Chauhan [RA2111026010276]

Under the Guidance of

Mrs. R Vidhya

Assistant Professor, Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Artificial Intelligence & Machine Learning

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**MULTICAST STREAMING SYSTEM**" is the bonafide work of Sai Rishyanth Visinigiri [RA2111026010280], Suhas Ganga [RA2111026010281], Manisha Manoj [RA2111026010274], Adithya Nandan Naidu Bandaru [RA2111026010278], Shreeya Chauhan [RA2111026010276] who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

FACULTY-IN-CHARGE
Mrs. R Vidhya
Assistant Professor,
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

HEAD OF THE DEPARTMENT
Dr. R Annie Uthra
Professor and Head,
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

SIGNATURE OF INTERNAL EXAMINER

SIGNATURE OF EXTERNAL EXAMINER

Department of Computational Intelligence

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course: B.Tech in Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning

Student Names: Sai Rishyanth Visinigiri, Suhas Ganga, Adithya Bandaru, Manisha Manoj, Shreeya Chauhan

Registration Number: RA2111026010280, RA2111026010281, RA2111026010278, RA2111026010274, RA2111026010276

Title of Work: Multicast Streaming System

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the UniversityWebsite, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc. that are not my own

- Not made any use of the report(s) or essay(s) of any other student(s) eitherpast or present
- Acknowledged in appropriate places any help that I have received fromothers (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Coursehandbook / University website

I understand that any false claim for this work will be penalized in accordance with theUniversity policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to the esteemed individuals who have been guiding lights throughout our academic journey and the completion of this course project.

First and foremost, we wish to express our deepest appreciation to our honorable Vice Chancellor, Dr. C. MUTHAMIZHCHELVAN, whose unwavering support has been a constant source of inspiration for all our endeavors.

Our thanks go out to Dr. S. Ponnusamy, our Registrar, for his continuous encouragement, which has motivated us to excel in our academic pursuits.

Dr. T. V. Gopal, Dean of the College of Engineering and Technology, deserves special mention for infusing innovation into every facet of our education and project execution.

We are indebted to Dr. Revathi Venkataraman, Chairperson of the School of Computing, whose mentorship instilled the confidence needed to successfully complete our course project.

Our sincere appreciation goes to our Course Project Faculty, Mrs. R Vidhya, Assistant Professor in the Department of Computational Intelligence, for his invaluable assistance, timely suggestions, and unwavering guidance throughout the duration of this project.

We also acknowledge the unwavering support of Dr. R. Annie Uthra, Professor and Head of the Department of Computational Intelligence, and our fellow departmental colleagues.

Last but not least, we extend our gratitude to our parents, friends, and all our near and dear ones, whose support, both direct and indirect, contributed significantly to the successful completion of our project.

Above all, we offer our thanks to the Almighty for bestowing His blessings upon us, enabling us to bring our course project to fruition.

I. LIST OF FIGURES

ABSTRACT

This project outlines the design and implementation of a multicast streaming system that emulates real-life broadcasting scenarios. The system operates with a server-client architecture, where clients can join a multicast group and access a variety of stations. Upon joining, clients receive a comprehensive station list and site information from the server over a TCP connection.

One distinctive feature of this system is that all stations continuously transmit data, regardless of whether a client is connected, closely resembling traditional broadcasting systems like television or radio. When a client selects a station from the provided list, it establishes a connection to that specific station, enabling live-streaming video reception.

The client-side application offers an intuitive graphical user interface (GUI) that allows users to interact with the system seamlessly. Clients have the flexibility to pause, resume, change stations, or terminate their connection at any time, utilizing threads to manage these dynamic actions.

This project aims to create a realistic and dynamic multicast streaming experience, similar to traditional broadcasting, while harnessing modern technology and user-friendly interfaces to enhance the viewing and interaction with the content.

What sets this project apart is its faithful replication of the dynamics of traditional broadcasting. Much like television and radio broadcasts, all stations within the system continually transmit data, irrespective of client connections. When a client selects a station, it forges a connection that enables real-time video streaming.

The client-side application boasts an intuitive graphical user interface (GUI), placing the user at the helm of their multimedia experience. This interface empowers clients to exert fine-grained control over their viewing experience, including the ability to pause, resume, switch stations, or gracefully terminate their connection, all seamlessly coordinated through the use of threads.

This project not only strives to recreate the ambiance of traditional broadcasting but also harnesses the power of modern technology to enhance content accessibility and interaction. By offering an immersive and flexible multimedia experience, it addresses the evolving needs of today's interconnected audiences, bridging the gap between past and present broadcasting paradigms.

1. INTRODUCTION

1.1 GENERAL

The motivation for this project is rooted in the desire to create a multimedia streaming system that closely emulates real-life broadcasting practices. Traditional television and radio broadcasts continuously transmit content, regardless of whether there are receivers connected, and this project aims to replicate that behaviour. By doing so, it provides users with a more authentic and relatable multimedia experience, fostering a sense of immersion in the content.

Furthermore, the project is driven by a commitment to delivering a user-centric experience. In today's multimedia landscape, users expect more control over their content consumption. The ability to select stations, pause, resume, switch between channels, and terminate connections at will is a response to these evolving user demands. This approach ensures that users can tailor their multimedia experience to their preferences, enhancing their engagement and satisfaction.

Technological innovation is another key motivator. By integrating modern technology, network protocols, and advanced concepts in networking and multimedia streaming, the project serves as a practical platform for exploring and implementing innovative solutions. It offers an opportunity to tackle technical challenges and leverage cutting-edge tools to enhance the quality and efficiency of content delivery.

For students and developers, the project represents an educational opportunity. It provides hands-on experience in the realms of multicast communication, graphical user interface (GUI) design, thread management, and network programming. It enables individuals to gain practical skills and insights into these areas, making it a valuable learning endeavour.

Scalability and efficiency are also driving forces behind this project. Multicast streaming, as an efficient means of content distribution to multiple clients, has become increasingly relevant in scenarios where content needs to reach a large number of recipients. The project's scalability and efficiency make it a practical solution for such content distribution needs.

In conclusion, the motivation for this project is multifaceted, encompassing realism, user-centric design, technological exploration, educational opportunities, responsiveness to changing user expectations, scalability, and the potential for research contributions. This project aspires to offer an engaging and practical solution that bridges the gap between traditional broadcasting practices and the demands of the modern multimedia landscape.

1.2 PURPOSE

The project's objectives span a spectrum of critical elements in the development of a multicast streaming system. At its core, the project aims to create a functional system that emulates traditional broadcasting practices, enabling clients to join multicast groups and access real-time video streams from selected stations. Beyond this, the project focuses on enhancing user interaction through an intuitive and user-friendly graphical interface, allowing clients to choose stations, pause, resume, switch between channels, and terminate connections. Technological advancement is a key objective, with the implementation of state-of-the-art networking and multimedia streaming techniques to optimize content delivery efficiency and reliability.

This project also serves as an educational platform for students and developers, offering hands-on experience in various aspects of networking, GUI design, thread management, and network programming. Scalability and efficiency are integral goals, designed to make the system suitable for scenarios involving large numbers of clients, thus contributing to the optimization of data transmission. Moreover, the project fosters research potential, providing a foundation for the exploration of innovative multicast streaming techniques that could find broader applications in multimedia content delivery. Lastly, the project is dedicated to enhancing the user experience, offering flexible and interactive content consumption options to cater to the evolving expectations of modern audiences, ensuring a dynamic and engaging multimedia experience. In sum, these objectives collectively aim to create a comprehensive and impactful multicast streaming solution that not only replicates traditional broadcasting practices but also adapts to the contemporary landscape, embracing cutting-edge technology, education, and research.

1.3 PROBLEM STATEMENT

The project seeks to address the need for a robust multicast streaming system that replicates traditional broadcasting practices while leveraging modern technology for efficient content delivery. The system will allow multiple clients to connect to a multicast group and access real-time video streams from selected stations. This project specifically focuses on implementing this system using the C programming language and a client-server architecture with two stations, one acting as the sender and the other as the receiver.

1.4 CHALLENGES

Implementing a multicast streaming system with a client-server architecture, involving sender and receiver stations, can present several challenges. Some of the key challenges include:

Multicast Group Management: Managing a multicast group efficiently can be complex. Tracking connected clients, monitoring station selection, and ensuring seamless data distribution to all clients in the group requires careful design and synchronization.

Data Synchronization: Maintaining synchronization between sender and receiver stations can be challenging. Ensuring that the receiver receives data in real-time without disruptions, such as data loss or buffering delays, is crucial.

Network Stability: Multicast communication can be sensitive to network conditions. Challenges may arise when clients have varying network qualities or when the network infrastructure does not fully support multicast communication.

Scalability: Handling a growing number of clients within the multicast group without compromising performance is a significant challenge. The system should be designed to efficiently scale as the number of clients increases.

Error Handling: Handling errors in a multicast environment can be complex. The system needs to address issues such as lost packets, network interruptions, and client disconnects gracefully, without causing disruptions for other clients.

Security: Ensuring the security and privacy of the multicast content is critical. Unauthorized access to the multicast group or content interception should be prevented.

User Interface Design: Developing an intuitive and user-friendly GUI can be challenging, as it requires a good understanding of user experience (UX) design principles. Ensuring that the GUI works seamlessly with the underlying system is crucial for a positive user experience.

Thread Management: Implementing multithreading to handle multiple client connections and streaming can introduce complexities related to thread synchronization, resource management, and potential race conditions.

2. LITERATURE SURVEY

Resilient multicast support for continuous-media applications [1].

The IP multicast delivery mechanism provides a popular basis for delivery of continuous media to many participants in a conferencing application. However, the best-effort nature of multicast delivery results in poor playback quality in the presence of network congestion and packet loss.

Xu, X.R. & Myers, A.C. & Zhang, Hui & Yavatkar, Raj. (1997). Resilient multicast support for continuous-media applications. 183 - 194. 10.1109/NOSDAV.1997.629385.

Scalable video transport over wireless IP networks [2].

There has been great interest in transporting real-time video over wireless IP networks from both industry and academia. Real-time video applications have quality-of-service (QoS) requirements. However, the fluctuations of wireless channel conditions pose many challenges to providing QoS for video transmission over wireless IP networks. It has been shown that scalable video coding and adaptive services are viable solutions under a time-varying wireless environment. We propose an adaptive framework to support quality video communication over wireless IP networks.

Wu, Dapeng & Hou, Y.T. & Zhang, Ya-Qin. (2000). Scalable video transport over wireless IP networks. 1185 - 1191 vol.2. 10.1109/PIMRC.2000.881607.

3. REQUIREMENT ANALYSIS

1. Server Requirements:

- The server should be capable of managing and controlling the multicast group.
- It must provide a centralized point for clients to request entry into the multicast group.
- The server should maintain and update a list of available stations, including relevant site information, and deliver this information to clients.
- It must handle multiple client connections concurrently.

2. Client Requirements:

- Clients should be able to send join requests to the server to access the multicast group.
- Each client must be equipped with a user-friendly graphical user interface (GUI) that enables station selection, playback control, and connection management.
- Clients should be capable of initiating connections to the server and receiving streaming data from selected stations.
- The system should allow clients to interact with the GUI seamlessly, including options to pause, resume, switch stations, and terminate connections at will.

3. Sender Station Requirements:

- The sender station, as part of the multicast group, should continuously transmit data, simulating the behaviour of a traditional broadcasting station.
- It must efficiently send real-time video streams to the multicast group without interruption.
- The sender station should operate in coordination with the server and be capable of broadcasting data to multiple clients simultaneously.

4. Receiver Station Requirements:

- The receiver station should have the capability to connect to the multicast group established by the server.
- It must receive and display real-time video streams from the sender station.
- The receiver station should support functionalities like pausing, resuming, switching between stations, and gracefully terminating the connection through the user interface.

These requirements establish the roles and functionalities of the server, clients, sender, and receiver stations within the multicast streaming system.

4. ARCHITECTURE AND DESIGN

The proposed multicast streaming system involves a client-server architecture, encompassing sender and receiver stations. Below, we provide an overview of the architectural components, network protocols used, and the graphical user interface (GUI) implementation:

1. Client-Server Architecture:

Server: The server serves as the central control point, managing the multicast group and client interactions. It communicates with clients over TCP for station and site information. The server maintains a list of stations and clients, facilitating station selection and content delivery.

Clients: Clients connect to the server over TCP, receiving station information and site details. The GUI, implemented using GTK, provides an intuitive interface for station selection, playback control, and connection management.

Sender Station: The sender station continuously broadcasts live-streaming videos to the multicast group using UDP, ensuring that all connected clients receive the content in real-time.

Receiver Stations: Receiver stations, which are essentially the clients, use UDP to receive multicast video streams from the sender. The GUI allows users to interact with the content, enabling functions like pause, resume, station switching, and connection termination.

2. Network Protocols:

TCP (Client to Server): TCP is employed for client-server interactions, providing a reliable one-to-one connection. Clients use TCP to request entry into the multicast group, retrieve station information, and send control commands to the server.

UDP (Sender to Receiver): UDP is chosen for the efficient multicast delivery of live-streaming videos. The sender uses UDP to broadcast content to all clients who have joined the multicast group. UDP's low overhead and minimal latency make it suitable for real-time data transmission.

3. Graphical User Interface (GUI) Implementation:

The GUI is implemented using the GTK (GIMP Toolkit) framework. GTK provides a cross-platform toolkit for creating graphical user interfaces, offering widgets, theming, and event handling capabilities.

The GUI allows clients to interact with the system seamlessly. It provides options for station selection, playback control (pause and resume), station switching, and connection termination.

GTK's user-friendly design principles ensure that the GUI is aesthetically pleasing and intuitive, enhancing the user experience.

4. Functionalities:

Four key functions have been implemented to manage user interactions effectively:

Pause: Allows clients to temporarily halt video playback.

Resume: Enables clients to resume video playback after a pause.

Change Station: Permits clients to switch between different stations available in the multicast group.

Terminate: Allows clients to gracefully disconnect from the multicast group, ending their session.

The described architecture and design choices provide a foundation for a reliable and interactive multicast streaming system. By utilizing TCP and UDP for their respective purposes and incorporating an intuitive GUI, the system efficiently manages multimedia content delivery, ensuring that clients can access, control, and enjoy real-time video streams seamlessly.

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 l: 239.192.4. I

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.

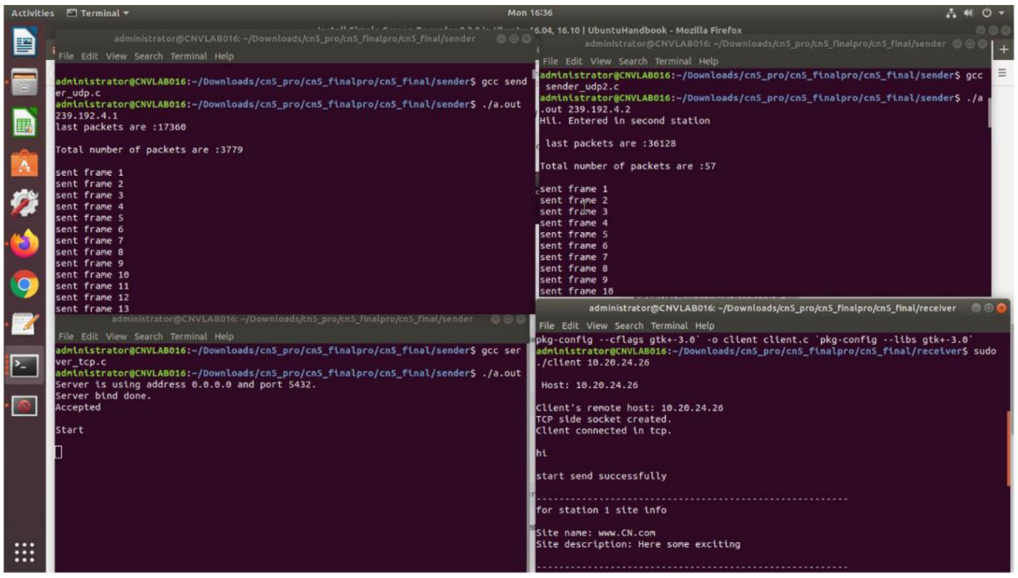
5. IMPLEMENTATION

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 streamable.
- Receiver can pause, resume, change station or even terminate at any given time from GUI using thread.
- Pause: It closes the multicast reception. This is implemented to relate with real life.
- When this button is pressed, it will generate an interrupt by changing flag value, which will temporarily stop receiving data from sender.
- Resume: resumes it, keeping the station the same.
- When this button is pressed, it will generate an interrupt by again changing flag value, which will start receiving data from sender.
- 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 rom the station it was connected earlier.

6. RESULT AND ANALYSIS

This project will work as mentioned above. 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.


Activities
Terminal
Mon 16:36
6.04, 16.10 | UbuntuHandbook - Mozilla Firefox
File Edit View Search Terminal Help

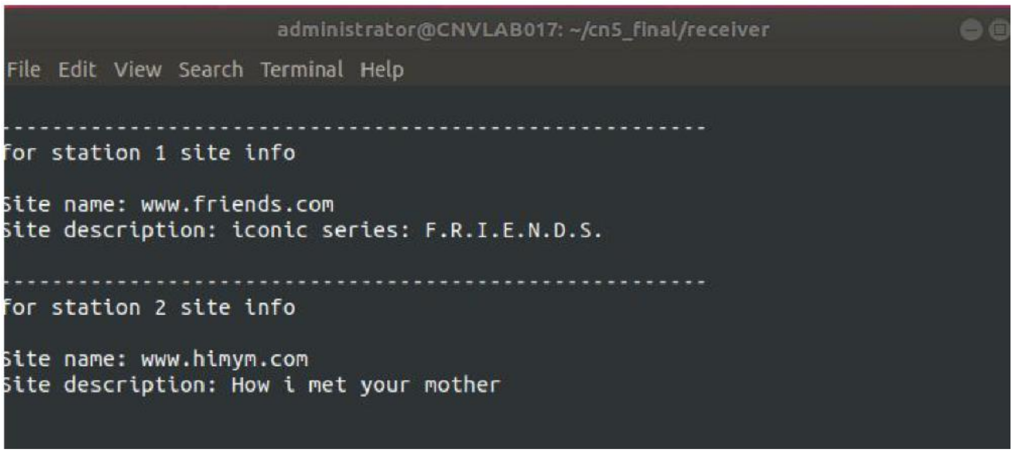
```
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc send
er_udp.c
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ ./a.out
239.192.4.1
last packets are :17360

Total number of packets are :3779

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
```

```
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc
 sender_udp2.c
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ ./a
.out 239.192.4.2
Hii. Entered in second station

 last packets are :36128

Total number of packets are :57

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
```

```
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc ser
ver_tcp.c
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ ./a.out
Server is using address 0.0.0.0 and port 5432.
Server bind done.
Accepted

Start
```

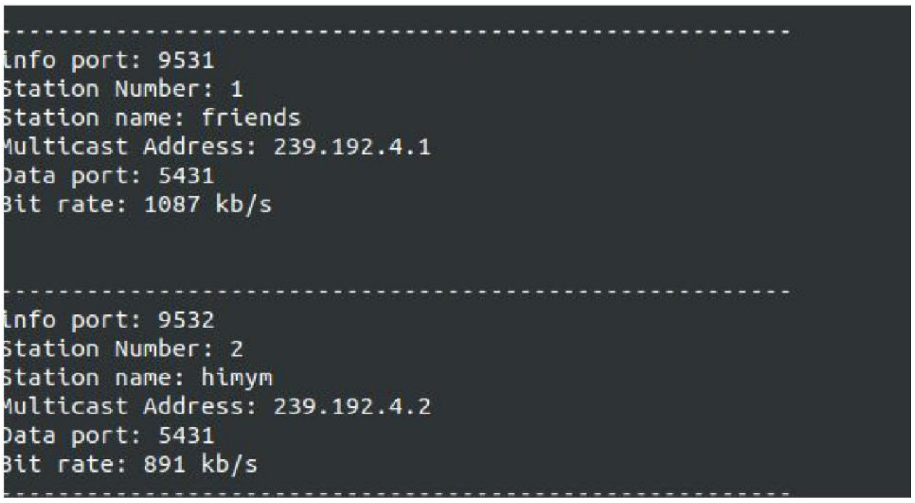
```
pkg-config --cflags gtk+-3.0` -o client client.c `pkg-config --libs gtk+-3.0`
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/receiver$ sudo
./client 10.20.24.26

 Host: 10.20.24.26

Client's remote host: 10.20.24.26
TCP side socket created.
Client connected in tcp.

hi

start send successfully

-------------------------------------------------------
for station 1 site info

Site name: www.CN.com
Site description: Here some exciting

-------------------------------------------------------
```

Figure 6.1 Successful Compilation of all C files


administrator@CNVLAB017: ~/cn5_final/receiver
File Edit View Search Terminal Help

```
-------------------------------------------------------
for station 1 site info

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

-------------------------------------------------------
for station 2 site info

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

Figure 6.2 Site information 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
-------------------------------------------------------
```

Figure 6.3 Station Information at Client Side

Station Selection GUI Window:

- Page I opens for the client whenever he is connected to the server.
- Page 2 opens after selecting any of the two buttons: F.R.I.E.N.D.S or H.I.M.Y.M

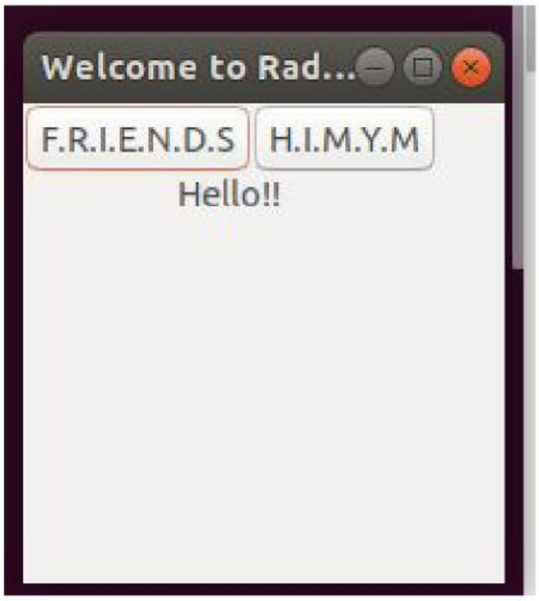
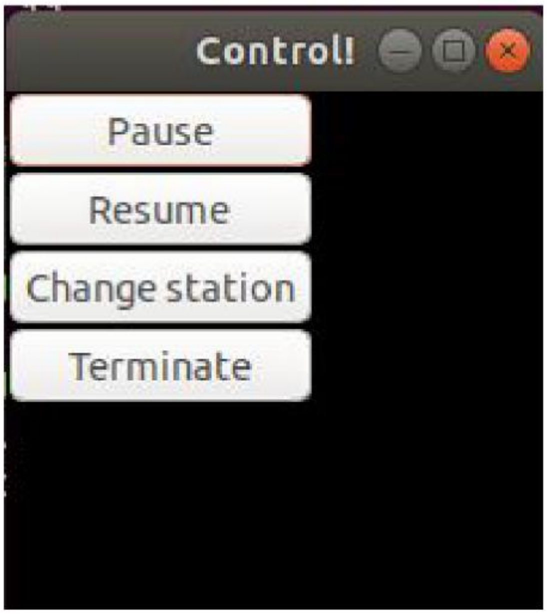
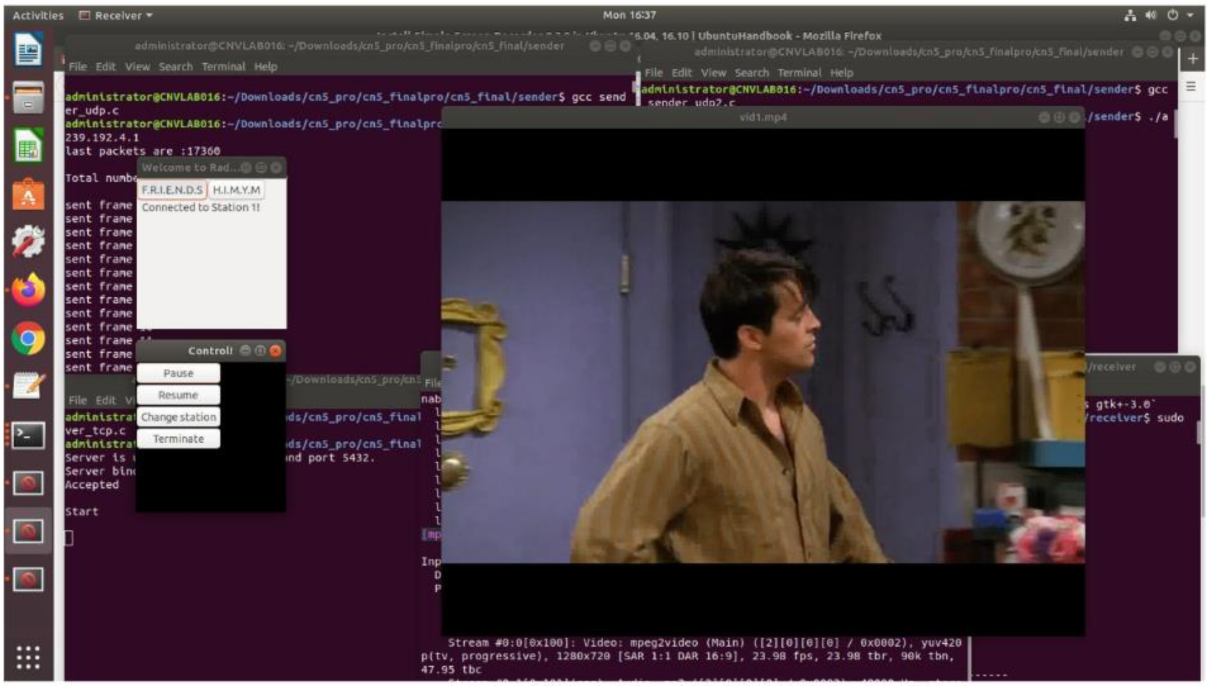
Page 1 of GUI	Page 2 of GUI
	

Figure 6.4 GUI Window of Page 1 and 2



```
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc send
er_udp.c
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro
239.192.4.1
last packets are :17360

Total numbe

sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
```

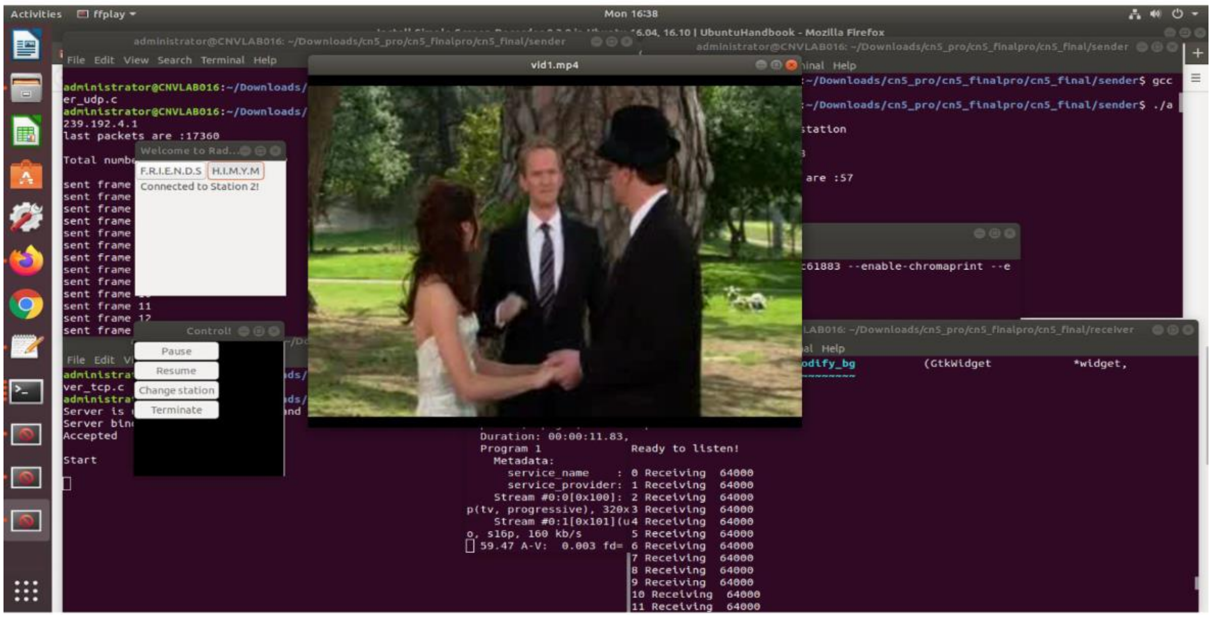
```
administrat                          ds/cn5_pro/cn5_final
ver_tcp.c
administrat                          ds/cn5_pro/cn5_final
Server is                            nd port 5432.
Server bin
Accepted

Start
```

```
administrator@CNVLAB016:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc
 sender_udp2.c
                                                 /sender$ ./a
```

```
s gtk+-3.0`
/receiver$ sudo
```

```
  Stream #0:0[0x100]: Video: mpeg2video (Main) ([2][0][0][0] / 0x0002), yuv420
p(tv, progressive), 1280x720 [SAR 1:1 DAR 16:9], 23.98 fps, 23.98 tbr, 90k tbn,
47.95 tbc
```

Figure 6.4: Station 1 F.R.I.E.N.D.S



```
administrator@CNVLAB016:~/Downloads/
er_udp.c
administrator@CNVLAB016:~/Downloads/
239.192.4.1
last packets are :17360

Total numbe

sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame
sent frame 11
sent frame 12
sent frame
```

```
administra
ver_tcp.c
administra
Server is
Server bin
Accepted

Start
```

```
:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ gcc
:~/Downloads/cn5_pro/cn5_finalpro/cn5_final/sender$ ./a
station

 are :57
```

```
:61883 --enable-chromaprint --e
```

```
odify_bg            (GtkWidget            *widget,
```

```
  Duration: 00:00:11.83,
  Program 1             Ready to listen!
    Metadata:
      service_name    : 0 Receiving  64000
      service_provider: 1 Receiving  64000
    Stream #0:0[0x100]: 2 Receiving  64000
p(tv, progressive), 320x3 Receiving  64000
    Stream #0:1[0x101](u4 Receiving  64000
o, s16p, 160 kb/s       5 Receiving  64000
  59.47 A-V:  0.003 fd= 6 Receiving  64000
                        7 Receiving  64000
                        8 Receiving  64000
                        9 Receiving  64000
                        10 Receiving  64000
                        11 Receiving  64000
```

Figure 6.5: Station 2 H.I.M.Y.M

7. CONCLUSION

In conclusion, the multicast streaming system with a client-server architecture, sender-receiver stations, and dynamic buffer management represents a robust and flexible solution for replicating traditional broadcasting practices in a modern context. This project successfully combines elements of networking, multimedia streaming, user interface design, and dynamic buffer allocation to create an engaging and authentic broadcasting experience.

The client-server architecture efficiently manages client interactions, station information, and content distribution. Clients connect to the server to access station data through a user-friendly graphical user interface (GUI) implemented using GTK. This GUI empowers users to select stations, control playback, and manage their connections with ease, catering to evolving user expectations for interactive multimedia experiences.

The project leverages TCP for reliable client-server communication and UDP for efficient multicast content delivery from sender to multiple receiver stations. The sender station continuously broadcasts live-streaming videos, mirroring traditional broadcasting practices, while receiver stations engage with the content, offering functions such as pausing, resuming, station switching, and graceful termination.

The dynamic buffer management ensures that the system adapts to varying bit-rates and station changes, ensuring the buffer can hold data for the specified time, enhancing the system's efficiency and performance.

This project not only provides a platform for immersive multimedia consumption but also serves as an educational tool for students and developers, offering hands-on experience in network programming, GUI design, and thread management. Additionally, it has research potential in optimizing multicast streaming techniques.

In a world where multimedia content delivery continues to evolve, this project bridges the gap between traditional broadcasting and modern user expectations, showcasing the adaptability and relevance of multicast streaming systems. It emphasizes realism, user-centric design, technological innovation, scalability, and research contributions, making it a comprehensive and impactful solution in the field of multimedia content distribution.

8. REFERENCES

[1] Xu, X.R. & Myers, A.C. & Zhang, Hui & Yavatkar, Raj. (1997). Resilient multicast support for continuous-media applications. 183 - 194. 10.1109/NOSDAV.1997.629385.

[2] Wu, Dapeng & Hou, Y.T. & Zhang, Ya-Qin. (2000). Scalable video transport over wireless IP networks. 1185 - 1191 vol.2. 10.1109/PIMRC.2000.881607.

9. APPENDIX (CODE)

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;
}
```

<u>**Reciever.c:**</u>

```
//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 ();



}
```

Server.c:

```
//server tcp  with gtk

#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;
}
```