22AIE204

Introduction to Computer Networks

A thesis on

Integrated Network Solutions to Tackle Real Life Issues

DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF THE UGC ACT, 1956

Submitted by:

Group 5, Batch B

Eshwanth Karti T R	- CB.EN.U4AIE22118
Nandhana Gireesh	- CB.EN.U4AIE22138
S Ananthasivan	- CB.EN.U4AIE22148
Snega Sri	- CB.EN.U4AIE22163

In partial fulfilment for the award of the degree of Bachelor of Technology in CSE-AIE

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641112 (INDIA)

December – 2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

AMRITA
VISHWA VIDYAPEETHAM
श्रद्धावान् लभते ज्ञानम्
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF THE UGC ACT, 1956

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "Integrated Network Solutions to Tackle Real Life Issues" submitted by Eshwanth Karti T R (CB.EN.U4AIE22118), Nandana Gireesh (CB.EN.U4AIE22138), S Ananthasivan(CB.EN.U4AIE22148), Snega Sri (CB.EN.U4AIE22163) for the award of the Degree of Bachelor of Technology in the "CSE(AI) " is a Bonafide record of the work carried out by them under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Ms Sruthi Guptha	**Dr K P Soman**
Project Guide	**Dean & HoD , Dept. AI**

Submitted for the university examination held on 20/12/2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

DECLARATION

We, Eshwanth Karti T R (CB.EN.U4AIE22118), Nandana Gireesh (CB.EN.U4AIE22138), S Ananthasivan (CB.EN.U4AIE22148), Snega Sri (CB.EN.U4AIE22163) at this moment declare that this thesis entitled "Implementation of SVM using ADMM & Image classification using Multiclass SVM", is the record of the original work done by us under the guidance of Mrs. Sruthi Guptha , Assistant Professor, Dept of Artificial Intelligence, Amrita School of Artificial Intelligence, Coimbatore. To the best of our knowledge, this work has not formed the basis for the award of any degree/diploma/ associateship /fellowship/or a similar award to any candidate in any University.

Eshwanth Karti T R	**CB.EN.U4AIE22118**
Nandhana Gireesh	**CB.EN.U4AIE22138**
S Ananthasivan	**CB.EN.U4AIE22148**
Snega Sri	**CB.EN.U4AIE22163**

Acknowledgement

We would like to express our special gratitude to our ICN professor, Ms Sruthi Guptha, who gave us the golden opportunity to do this wonderful project, which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given. We would also like to thank each other - the group members, as without every one of our cooperation, we would not have been able to complete the project within the prescribed time.

Contents

Abstract:

This semester project aims to address two distinct yet impactful aspects of computer networks, presenting a dual-pronged exploration into broadcast technology for inclusive learning environments and the development of an integrated smart home system. In the first subproject, we propose the creation of a broadcast tool designed to facilitate seamless communication in large classrooms or halls where the presenter's audio and video content can be disseminated to all participants connected to the same network. This initiative is particularly beneficial for individuals with special requirements, fostering inclusivity in educational settings.

The second subproject involves the implementation of a comprehensive smart home system within Cisco Packet Tracer. This system is meticulously crafted, with each room equipped with an array of interconnected smart devices. A central home gateway orchestrates the synchronisation of these devices, allowing real-time sensor data and smartphone controls to seamlessly integrate and enhance the overall smart living experience. The exploration extends beyond the practical implementation, delving into the intricate details of the protocols underpinning the Internet of Things (IoT) ecosystem.

Through these two subprojects, our team endeavours to showcase the practical applications of computer networks in fostering accessibility and convenience. By developing a broadcast tool for inclusive learning environments and a sophisticated smart home system, we aim to contribute to the evolution of technology that positively impacts various facets of daily life. The integration of protocols within the IoT infrastructure further enriches our understanding of networked solutions and their role in shaping the future of both educational and domestic landscapes.

Work Split

1. Video Audio Broadcast System:

For the development of the PreDy broadcasting system, the team has divided the responsibilities to efficiently implement the audio and video broadcasting programs along with screenshare functionality. Eshwanth takes the lead in crafting the audio broadcasting program, ensuring the seamless transmission of real-time audio data, and simultaneously, he spearheads the creation of the screenshare broadcasting program, allowing presenters to share their screens effortlessly. Eshwanth also contributes significantly to the documentation, providing in-depth insights into the functionalities of both broadcasting programs. On the other hand, Ananth is tasked with developing the video broadcasting program, focusing on the effective transmission of real-time video data. In addition, he oversees the screenshare program, ensuring clarity and precision in screen sharing during presentations. Ananth is actively involved in the documentation process, providing comprehensive explanations for both the video broadcasting and screenshare programs. This division of tasks ensures a systematic approach to the development of PreDy, with each team member playing a crucial role in shaping the audio, video, and screenshare components of the broadcasting system.

2. **IoT Systems Integration:**

Snega (Cisco Implementation): Leading the development of the integrated smart home system within Cisco Packet Tracer, configuring smart devices, and establishing a cohesive network infrastructure to support seamless communication.

Nandhana (Protocol Analysis): Responsible for analysing and understanding the protocols involved in the IoT system, ensuring a deep dive into the technical intricacies governing the communication between smart devices and the central home gateway.

Documentation (Combined Work): Collaborative effort to document the implementation details, protocols, and overall functioning of the IoT system. This documentation will serve as a comprehensive guide for understanding and replicating the smart home integration.

By dividing the tasks in this manner, each team member can contribute their expertise to their respective areas, fostering efficient progress in both subprojects. The combined effort in documentation ensures a cohesive and well-documented final report, presenting a holistic view of the undertaken endeavours.

Chapter 1:

Background

The idea for our project, PreDy (Presentation budDy), began one seemingly routine day when Ananth, one of our team members, noticed Eshwanth, another team member, taking a pleasant nap during a class period. This fascinating finding raised an important query: why do pupils frequently nod off in class? Investigating this further, we found an interesting National Institutes of Health data that indicates 38% of students give in to the temptation of napping during class. Although there are a number of reasons that contribute to this phenomenon, we hypothesise that one major trigger is the inherent issue of vast, silent, and visually hidden classrooms.

Understanding the significant influence of physical closeness on focus, we take our cue from the layout of conference and legislative rooms, where presenters use cameras and microphones to successfully interact with a distributed audience. We present PreDy, a cutting-edge tool created to improve learning and tackle the problems of big classes, in response to this discovery. Using the microphone and camera on the presenter's computer, our application broadcasts audio and video and if necessary, the screen in real time to all users connected to the PreDy platform.

Our goal is to create a welcoming and stimulating learning atmosphere for those inadvertent "sleeping beauties" and to ease their challenges by implementing PreDy on a university-wide basis. With this project, we hope to rethink the dynamics of classroom engagement and support the development of an attentive and participatory learning environment.

Chapter: 2

Theory

Protocols used:

Transmission Control Protocol (TCP):
Transmission Control Protocol (TCP) is a core communication protocol within computer networks that ensures reliable and ordered delivery of data between devices. It operates by establishing a connection between a sender (client) and a receiver (server) before data transmission begins. TCP's reliability stems from its ability to detect and retransmit lost or corrupted packets, ensuring that the data reaches its destination intact. The protocol guarantees the sequential delivery of information, crucial for applications where the correct order of data is essential. In the context of our project, PreDy, TCP plays a vital role in the transmission of control signals and data between the presenter's computer (server) and the connected clients. By prioritizing accuracy over speed, TCP ensures that audio and video content is delivered consistently, without loss or disorder. This reliability is paramount in maintaining the quality of real-time streaming, contributing to the seamless and effective functioning of our broadcasting tool in educational settings.

Internet Protocol (IP):
The Internet Protocol (IP) is a fundamental communication protocol that facilitates the routing and addressing of data packets across networks. IP provides a unique address for each device connected to the internet, allowing data to be directed to and from specific destinations. It operates at the network layer of the OSI model and is an essential component for the functioning of the internet. IP ensures the proper delivery of packets, managing the routing of information across interconnected networks. In the context of our project, PreDy, IP plays a pivotal role in establishing the end-to-end communication between the presenter's computer (server) and the connected clients. By assigning unique IP addresses to each device on the network, IP enables the seamless transmission of audio and video data. It is the backbone of our broadcasting tool, providing the necessary addressing and routing mechanisms to ensure that data reaches its intended recipients accurately and efficiently.

A socket is a software endpoint that establishes communication between two computers over a network. It is a fundamental networking concept that provides a standardized interface for processes (applications or services) on different devices to communicate with each other. Sockets are widely used in network programming to enable data exchange between client and server applications.

Key Characteristics of Sockets:

1. **Endpoint:** A socket is identified by an IP address and a port number, allowing data to be sent from one endpoint to another.

2. Communication: Sockets facilitate bidirectional communication between processes, allowing them to send and receive data.
3. Protocols: Sockets are typically associated with a specific network protocol, such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

Why Sockets are Used:
1. Inter-Process Communication (IPC): Sockets enable communication between processes running on the same machine or different machines connected to a network. This is crucial for distributed computing and networking scenarios.

2. Client-Server Architecture: Sockets are fundamental in client-server architectures, where a server process provides services and clients connect to it to request or receive those services. Examples include web servers serving web pages to browsers or chat servers communicating with chat clients.

3. Data Exchange: Sockets facilitate the exchange of various types of data, such as text, files, audio, and video, between applications.

4. Network Programming: Sockets are essential for network programming in various programming languages, allowing developers to create applications that communicate over the internet or a local network.

Layer Implementation:
Sockets operate at the transport layer of the OSI (Open Systems Interconnection) model. The transport layer is responsible for end-to-end communication and ensures the reliable and orderly delivery of data between two devices. TCP and UDP, the two main transport layer protocols, are commonly associated with sockets:

1. TCP (Transmission Control Protocol): TCP provides a reliable, connection-oriented communication service. Sockets using TCP offer features such as error checking, retransmission of lost packets, and in-order delivery. TCP is often used for applications where data integrity and reliability are crucial, such as file transfer and web browsing.

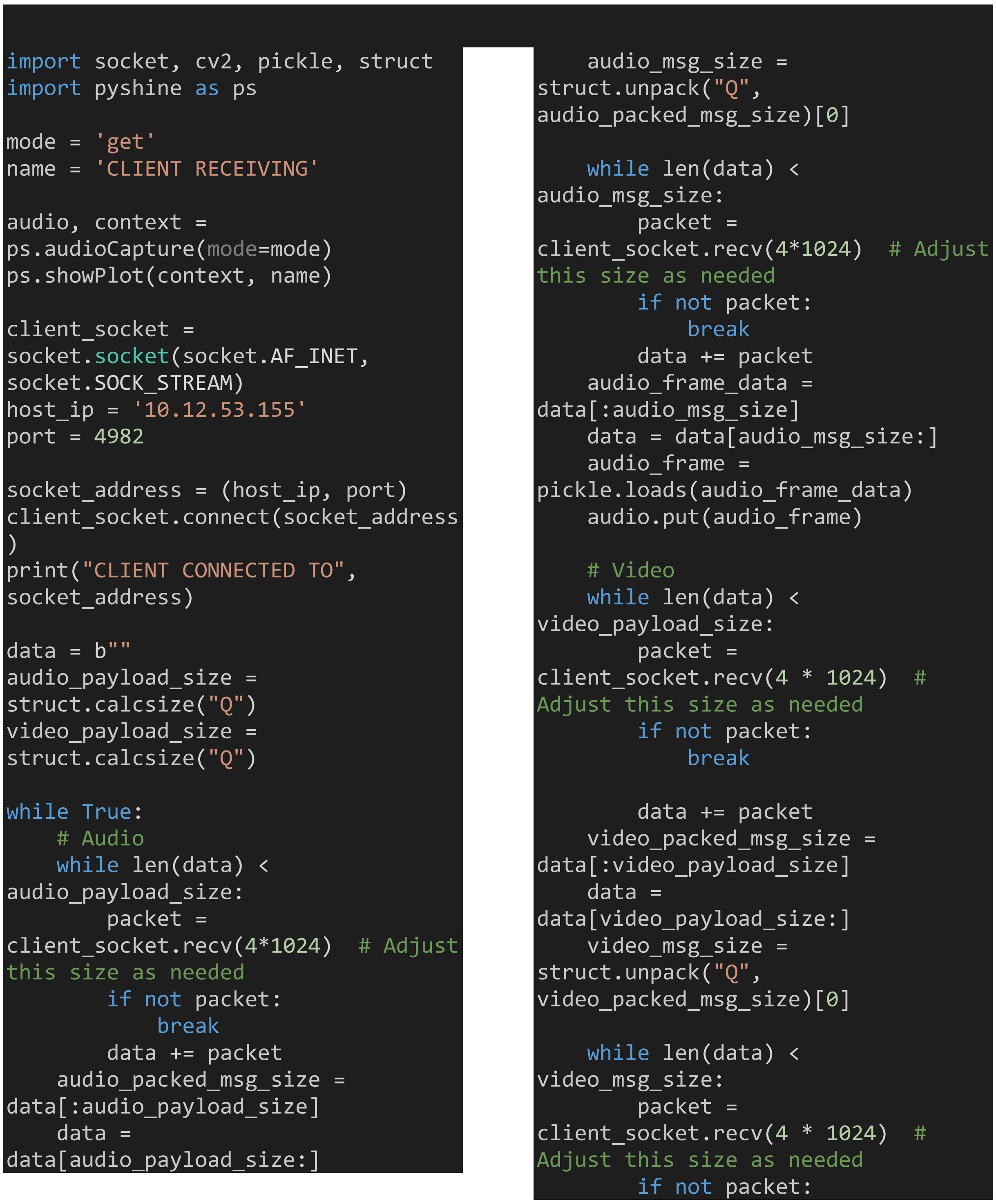
2. UDP (User Datagram Protocol): UDP provides a connectionless, unreliable communication service. Sockets using UDP offer lower overhead but do not guarantee reliable delivery or packet ordering. UDP is suitable for real-time applications like online gaming, streaming, and video conferencing, where low latency is more important than perfect data integrity.

In summary, sockets are a vital abstraction in network programming, allowing applications to communicate over a network using well-defined protocols at the transport layer. They play a central role in enabling the development of distributed and networked systems.

Chapter 3

Code Explanation

Client – Audio and Video Broadcast



```
import socket, cv2, pickle, struct
import pyshine as ps

mode = 'get'
name = 'CLIENT RECEIVING'

audio, context =
ps.audioCapture(mode=mode)
ps.showPlot(context, name)

client_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
host_ip = '10.12.53.155'
port = 4982

socket_address = (host_ip, port)
client_socket.connect(socket_address
)
print("CLIENT CONNECTED TO",
socket_address)

data = b""
audio_payload_size =
struct.calcsize("Q")
video_payload_size =
struct.calcsize("Q")

while True:
    # Audio
    while len(data) <
audio_payload_size:
        packet =
client_socket.recv(4*1024)  # Adjust
this size as needed
        if not packet:
            break
        data += packet
    audio_packed_msg_size =
data[:audio_payload_size]
    data =
data[audio_payload_size:]
```

```
    audio_msg_size =
struct.unpack("Q",
audio_packed_msg_size)[0]

    while len(data) <
audio_msg_size:
        packet =
client_socket.recv(4*1024)  # Adjust
this size as needed
        if not packet:
            break
        data += packet
    audio_frame_data =
data[:audio_msg_size]
    data = data[audio_msg_size:]
    audio_frame =
pickle.loads(audio_frame_data)
    audio.put(audio_frame)

    # Video
    while len(data) <
video_payload_size:
        packet =
client_socket.recv(4 * 1024)  #
Adjust this size as needed
        if not packet:
            break

        data += packet
    video_packed_msg_size =
data[:video_payload_size]
    data =
data[video_payload_size:]
    video_msg_size =
struct.unpack("Q",
video_packed_msg_size)[0]

    while len(data) <
video_msg_size:
        packet =
client_socket.recv(4 * 1024)  #
Adjust this size as needed
        if not packet:
            break
        data += packet
    video_frame_data =
data[:video_msg_size]
    data = data[video_msg_size:]
    video_frame =
pickle.loads(video_frame_data)
    # Process video frame as needed
```

```
    cv2.imshow("Received Video",
video_frame)
    if cv2.waitKey(1) & 0xFF ==
ord('q'):
        break
client_socket.close()
cv2.destroyAllWindows()
```

Explanation

This code represents a client program in a client-server architecture for streaming audio and video. The client connects to a server, receives audio and video frames, and displays the video frames using OpenCV. Here's a breakdown of the code:

1. Setting up Client Parameters:

- The client is configured to receive data (`mode = 'get'`).
- Audio capture is initiated using **`ps.audioCapture`**, and a plot for audio visualization is created using **`ps.showPlot`.**
- A TCP socket is created using **`socket.socket`** and connected to the server's IP address and port.

2. Receiving Audio Data:

- The client enters a loop where it continuously receives audio data from the server.
- Audio data is received in chunks (packets) with a maximum size of 4*1024 bytes.
- The packed message size indicating the length of the audio data is extracted and unpacked using **`struct.unpack("Q", audio_packed_msg_size**)`.
- The client continues to receive data until it accumulates enough to reconstruct the audio frame.
- The reconstructed audio frame is deserialized using **`pickle.loads`** and then processed or displayed.

3. **Receiving Video Data:**

- Similar to audio, the client enters a loop to receive video data from the server.
- Packed message size and video data are received in chunks, and the process is repeated until enough data is received to reconstruct the video frame.
- The reconstructed video frame is deserialized using **`pickle.loads`** and displayed using OpenCV (**`cv2.imshow`**).

4. User Termination:

- The client continuously displays video frames until the user presses the **'q'** key.
- Upon detecting the **'q'** key press (`cv2.waitKey(1) & 0xFF == ord(**'q'**)`), the client breaks out of the loop and proceeds to close the client socket and destroy any remaining OpenCV windows.

5. Closing Connections:

- The client socket is closed using **`client_socket.close()`** to terminate the connection with the server.
- OpenCV windows are closed using **`cv2.destroyAllWindows()`**.

Example:
Suppose the server runs on IP address **'10.12.53.155'** and port **'4982'**. The client, when executed, connects to the server and starts receiving audio and video data. The received video frames are displayed in an OpenCV window, and the client can be terminated by pressing the **'q'** key, closing the connection gracefully.

Server – Audio and Video Broadcast

```
import socket, cv2, pickle, struct
import pyshine as ps

mode = 'send'
name = 'SERVER TRANSMITTING'

audio, context = ps.audioCapture(mode=mode)
cap = cv2.VideoCapture(0)  # Assuming you want to capture video from the default camera

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '10.12.53.155'
port = 4982
backlog = 5
socket_address = (host_ip, port)
print('STARTING SERVER AT', socket_address, '...')
server_socket.bind(socket_address)
server_socket.listen(backlog)

while True:
    client_socket, addr = server_socket.accept()
    print('GOT CONNECTION FROM:', addr)
    if client_socket:
        while True:
            # Audio
            audio_frame = audio.get()
            audio_data = pickle.dumps(audio_frame)
            audio_message = struct.pack("Q", len(audio_data)) + audio_data
            client_socket.sendall(audio_message)


            # Video
            ret, video_frame = cap.read()
            video_data = pickle.dumps(video_frame)
            video_message = struct.pack("Q", len(video_data)) + video_data
            client_socket.sendall(video_message)

            client_socket.close()
```

This code represents a server in a client-server architecture for streaming audio and video. The server continuously captures audio and video frames from the default microphone and camera, respectively, and sends them to connected clients. Let's break down the code with real-life examples:

1. Setting up Server Parameters:
- The server is configured to send data (`mode = 'send'`).
- Audio capture is initiated using `ps.audioCapture`, and a plot for audio visualization is created using `ps.showPlot`.
- Video capture is initiated using OpenCV (`cv2.VideoCapture(0)`), assuming the default camera is used.

2. Setting up Socket and Listening for Connections:

- A TCP socket is created using `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`.
- The server is bound to a specific IP address and port using `server_socket.bind(socket_address)` and starts listening for incoming connections using `server_socket.listen(backlog)`.

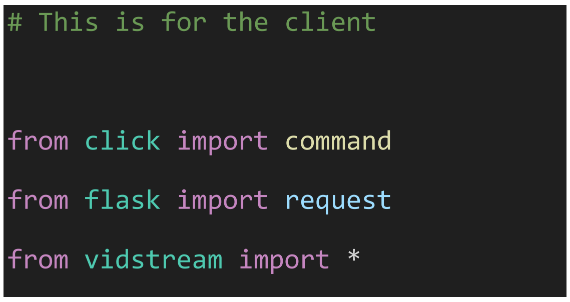
3. Accepting Client Connections:
- The server enters an infinite loop where it accepts client connections using `server_socket.accept()`.
- For each accepted connection, the server obtains the client socket (`client_socket`) and the client's address (`addr`).
- A real-life analogy would be a server in a video conferencing application waiting for clients (users) to join the conference.

4. Streaming Audio and Video to Clients:
- Within the inner loop for each connected client, the server continuously captures audio and video frames.
- The audio frame is obtained using **`audio.get()`**, serialized with **`pickle.dumps`**, and packed with its size before being sent to the client.
- Similarly, the video frame is captured using **`cap.read()`**, serialized with **`pickle.dumps`**, and packed with its size before being sent to the client.

5. Closing Connections:
- The server loop continues indefinitely, sending audio and video frames to connected clients.
- The server socket is not explicitly closed in this code snippet, but in a real-world scenario, you would want to include proper termination conditions and cleanup.

Real-Life Analogy:
Imagine a virtual classroom scenario where the server represents the teacher's computer. The teacher continuously captures and transmits their audio (voice) and video (live stream) to all connected students (clients) in the class. Each student, upon joining the virtual classroom, establishes a connection with the teacher's computer, and the teacher streams the audio and video content in real-time to provide an interactive learning experience.

Client – Screenshare

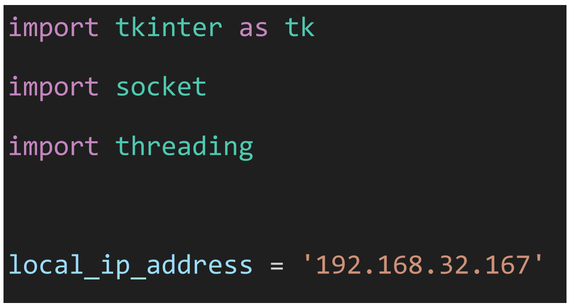


```
# This is for the client


from click import command

from flask import request

from vidstream import *
```



```
import tkinter as tk

import socket

import threading


local_ip_address = '192.168.32.167'
```

```python
############################ Functionality ########################

server = StreamingServer(local_ip_address,7777)

receiver = AudioReceiver(local_ip_address,6666)


def start_listening():

    t1 = threading.Thread(target=server.start_server)

    t2 = threading.Thread(target=receiver.start_server)

    t1.start()

    t2.start()



def start_screen_sharing():

    screen_client = ScreenShareClient(text_target_ip.get(1.0,'end-1c'),9999)

    t4 = threading.Thread(target=screen_client.start_stream)

    t4.start()




############################### GUI PART ##############################

window = tk.Tk()

window.title("Client")

window.geometry('500x200')


#set window color

window.configure(bg='yellow')


# adding elements to window

label_target_ip = tk.Label(window,text="Target IP:")

label_target_ip.pack()


text_target_ip = tk.Text(window,height=1)

text_target_ip.pack()


btn_listen = tk.Button(window,text="Start Listening",width=50,command=start_listening)

btn_listen.pack(anchor=tk.CENTER,expand=True)


btn_screen = tk.Button(window,text="Start Screen Sharing",width=50,command=start_screen_sharing)

btn_screen.pack(anchor=tk.CENTER,expand=True)



window.mainloop()


############################# GUI PART ##############################
```

Server Initialization:

The code initializes a StreamingServer object for video streaming on the client's local IP address (local_ip_address) and port 7777. An AudioReceiver object is also created for audio streaming on the local IP address and port 6666.

Listening Threads:

The start_listening function creates two separate threads (t1 and t2) to handle video and audio streaming concurrently.

t1 runs the start_server method of the StreamingServer object, initiating the video streaming server. t2 runs the start_server method of the AudioReceiver object, initiating the audio streaming server. This allows the client to listen for incoming video and audio streams simultaneously.

Screen Sharing:

The start_screen_sharing function initializes a ScreenShareClient object, specifying the target IP address obtained from the user input (text_target_ip).

A new thread (t4) is created to run the start_stream method of the ScreenShareClient object, enabling the client to start sharing its screen with the specified target IP address.

GUI (Graphical User Interface):

Tkinter Window:

The code creates a simple GUI window using the Tkinter library.

The window title is set to "Client," and its dimensions are set to 500x200 pixels.

The window background color is configured to yellow.

GUI Elements:

A label (label_target_ip) and a text box (text_target_ip) are added to the GUI to input the target IP address for streaming.

Two buttons (btn_listen and btn_screen) are provided for initiating the listening threads and starting screen sharing, respectively.

Button Commands:

The "Start Listening" button (btn_listen) triggers the start_listening function when clicked, initiating the video and audio streaming threads.

The "Start Screen Sharing" button (btn_screen) triggers the start_screen_sharing function when clicked, enabling the client to share its screen with the specified target IP address.

Server - Screenshare

```
from flask import request

from vidstream import *

import tkinter as tk

import socket

import threading


local_ip_address = socket.gethostbyname(socket.gethostname())


############################# Functionality #########################


server = StreamingServer(local_ip_address,9999)

receiver = AudioReceiver(local_ip_address,8888)


def start_listening():

    t1 = threading.Thread(target=server.start_server)

    t2 = threading.Thread(target=receiver.start_server)

    t1.start()

    t2.start()
```

```
def start_screen_sharing():

    screen_client = ScreenShareClient(local_ip_address,7777)

    t4 = threading.Thread(target=screen_client.start_stream)

    t4.start()



############################# GUI PART #############################

window = tk.Tk()

window.title("Server ")

window.geometry('500x200')


#set window color

window.configure(bg='green')


# adding elements to window

label_target_ip = tk.Label(window,text="The Server IP Address is equal to :")

label_target_ip.pack()


text_target_ip = tk.Label(window,text=local_ip_address)

text_target_ip.pack()
```

```
btn_listen = tk.Button(window,text="Start Server Listening to
Clients",width=50,command=start_listening)

btn_listen.pack(anchor=tk.CENTER,expand=True)

btn_screen = tk.Button(window,text="Start Screen Sharing From Server
Side",width=50,command=start_screen_sharing)

btn_screen.pack(anchor=tk.CENTER,expand=True)

window.mainloop()
```

Server Initialization:

The code initializes a StreamingServer object for video streaming on the server's local IP address (local_ip_address) and port 9999.An AudioReceiver object is created for audio streaming on the local IP address and port 8888.

Listening Threads:

The start_listening function creates two separate threads (t1 and t2) to handle video and audio streaming concurrently.t1 runs the start_server method of the StreamingServer object, initiating the video streaming server.

t2 runs the start_server method of the AudioReceiver object, initiating the audio streaming server. This allows the server to listen for incoming video and audio streams simultaneously.

Screen Sharing from Server:

The start_screen_sharing function initializes a ScreenShareClient object, specifying the local IP address of the server and port 7777. A new thread (t4) is created to run the start_stream method of the ScreenShareClient object, allowing the server to initiate screen sharing with client.

GUI (Graphical User Interface):

Tkinter Window:

The code creates a simple GUI window using the Tkinter library.

The window title is set to "Server," and its dimensions are set to 500x200 pixels. The window background color is configured to green.

GUI Elements:

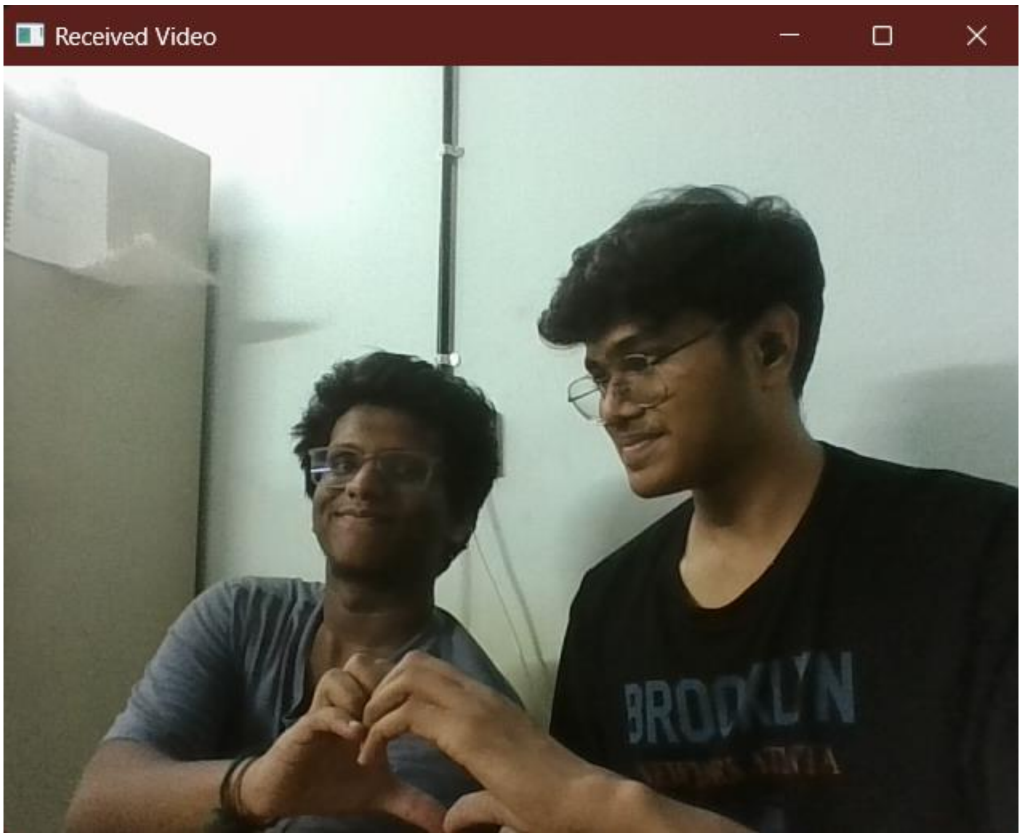
A label (label_target_ip) is added to the GUI to display the message "The Server IP Address is equal to."

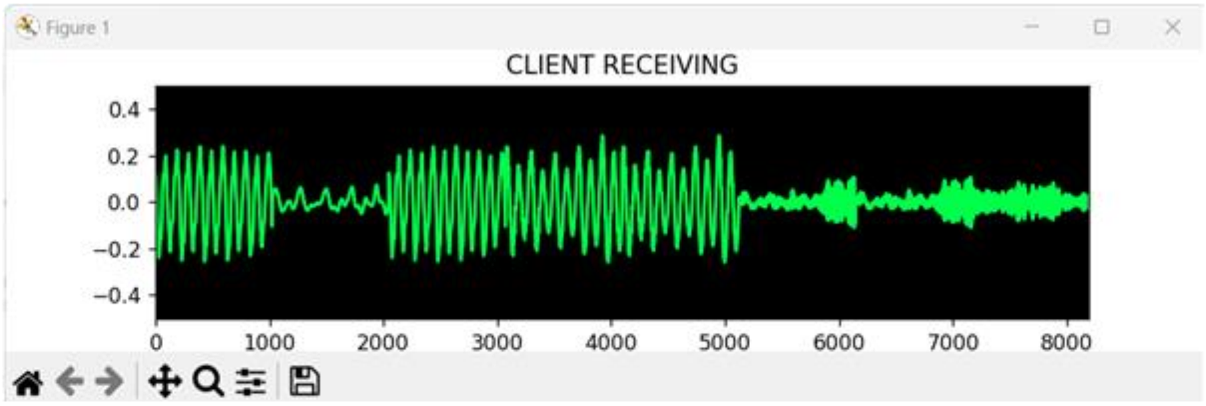
Another label (text_target_ip) displays the actual local IP address of the server.

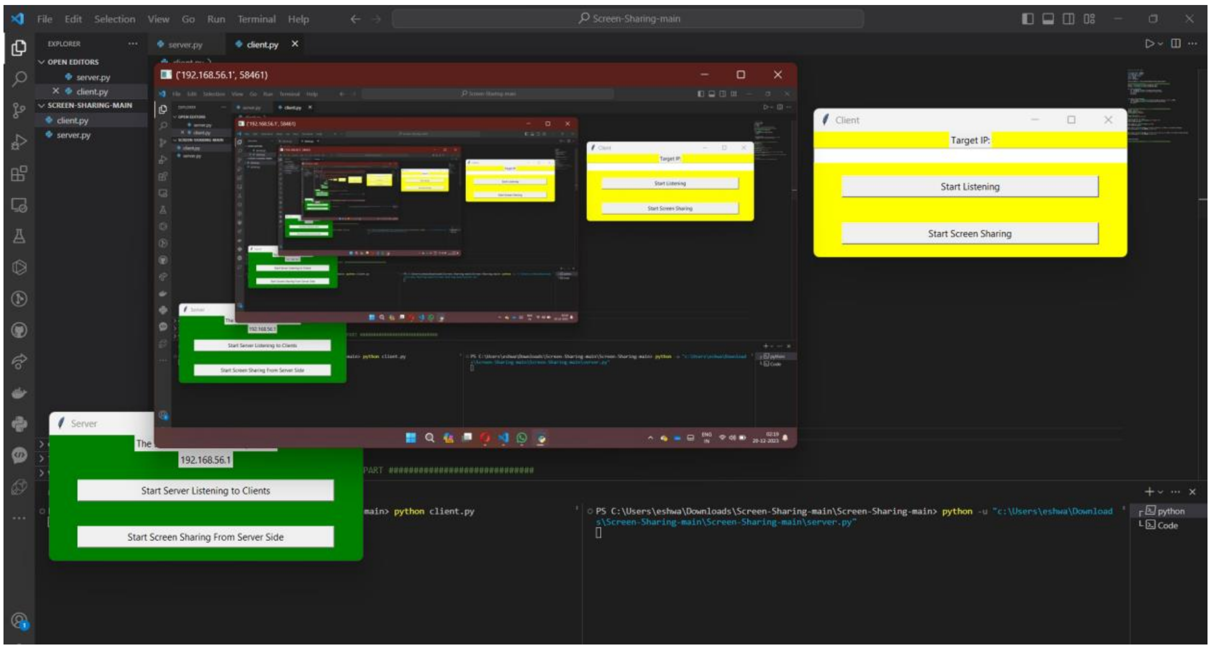
GUI Buttons:

Two buttons (btn_listen and btn_screen) are provided for initiating the listening threads and starting screen sharing from the server side, respectively.

Output







```
main> python client.py
```

```
PS C:\Users\eshwa\Downloads\Screen-Sharing-main\Screen-Sharing-main> python -u "c:\Users\eshwa\Download
s\Screen-Sharing-main\Screen-Sharing-main\server.py"
```

Chapter 5

Introduction to IoT

The Internet of Things (IoT) has emerged as a transformative force, reshaping the way we interact with the world around us. At its core, IoT represents the interconnectedness of devices, systems, and everyday objects through the power of the internet. This chapter serves as a foundational exploration into the key concepts, principles, and significance of IoT.

5.1 Definition and Scope of IoT:

IoT refers to the network of physical devices embedded with sensors, actuators, and connectivity capabilities that enable them to collect, exchange, and act upon data. These devices, ranging from household appliances to industrial machinery, form an intricate web of communication, facilitating a seamless exchange of information. The scope of IoT extends across various domains, encompassing smart homes, healthcare, agriculture, transportation, and industrial processes.

5.2 Core Components of IoT Systems:

Understanding the components that constitute an IoT system is crucial. Devices equipped with sensors and actuators form the 'Things,' which generate and receive data. Connectivity technologies, such as Wi-Fi, Bluetooth, or cellular networks, enable seamless communication. Cloud computing acts as the backbone, providing storage, processing power, and data analysis capabilities. Finally, user interfaces, often in the form of applications, allow users to interact with and control IoT devices.

5.3 The Pervasive Influence of IoT:

The influence of IoT is pervasive, permeating various aspects of our daily lives. In smart homes, IoT enables the synchronisation and automation of appliances, providing users with unprecedented control and efficiency. In agriculture, IoT sensors monitor soil conditions, weather patterns, and crop health, optimising farming practices. Healthcare benefits from wearable devices that track vital signs and enable remote patient monitoring, enhancing overall well-being.

5.4 Key Enablers and Technologies:

Several key technologies enable the proliferation of IoT. Low-cost sensors and actuators, coupled with miniaturised computing devices, contribute to the widespread adoption of IoT devices. Communication protocols, such as MQTT and CoAP, facilitate efficient data exchange between devices. Edge computing reduces latency by processing data closer to the source, enhancing real-time decision-making.

5.5 Challenges and Considerations:

While the potential of IoT is immense, it comes with challenges. Security and privacy concerns arise due to the massive amounts of data generated and transmitted. Standardisation and interoperability issues persist as the IoT landscape continues to evolve. Scalability and the need for sustainable power sources for IoT devices pose additional considerations.

5.6 Future Trends and Implications:

The future of IoT holds exciting possibilities. Advancements in artificial intelligence and machine learning will enhance the analytical capabilities of IoT systems, enabling more intelligent and context-aware decision-making. The integration of 5G networks will further propel the growth of IoT by providing faster and more reliable connectivity.
Chapter 6

6 Communication Protocols in IoT Systems

Now that we have a working idea of what IoT is and what it is capable of, let us now look into the magic that’s gonna make it all happen. In the real world, we all have to follow a set of rules set by a panel after thorough analysis of the country's needs that ensures fair chances to one and all, specific to the people under its control. Similarly, we have rules that guide components of a network to follow order - Enter Protocols.

6.1 Importance of Communication Protocols:

Communication protocols serve as the language and track that enables devices in an IoT network to interact with each other. The chapter begins by emphasising the pivotal role these protocols play in ensuring reliable, scalable, and interoperable communication within an IoT ecosystem.

6.2 Overview of Common IoT Protocols:

1. MQTT (Message Queuing Telemetry Transport): A lightweight, publish-subscribe protocol suitable for low-bandwidth, high-latency networks.
2. CoAP (Constrained Application Protocol): Designed for resource-constrained devices, CoAP is suitable for IoT applications and supports RESTful principles.
3. HTTP/HTTPS (Hypertext Transfer Protocol/Secure): Widely used in web applications, HTTP provides a familiar and accessible option for IoT communication.
4. TCP/IP (Transmission Control Protocol/Internet Protocol):
   The foundational TCP/IP protocol suite is essential for communication between the home gateway and all connected devices. It ensures reliable data transmission across the network and is the underlying protocol that supports higherlevel protocols like MQTT and HTTP/HTTPS.

6.3 Protocol Selection Considerations:

Choosing the right communication protocol is a critical decision in IoT system design. This section discusses key considerations such as bandwidth, latency, power consumption, and security when selecting a protocol. Understanding the specific requirements of the smart home automation project will guide the protocol selection process.

6.4 Implementation Challenges and Solutions:

While protocols facilitate communication, challenges such as data security, reliability, and scalability must be addressed. This section explores potential challenges and proposes solutions to ensure a robust communication framework for the smart home automation system.

6.5 Integration of Protocols in Smart Home Systems:

Building on the theoretical understanding of communication protocols, this section explores how these protocols are practically integrated into smart home automation systems. It sets the stage for the subsequent chapters where the selected protocols will be implemented within Cisco Packet Tracer to enable communication among simulated devices.

Chapter 7

MQTT (Message Queuing Telemetry Transport) in IoT

Here, we provide an in-depth exploration of MQTT, a widely adopted messaging protocol in the realm of the Internet of Things (IoT). Understanding MQTT is essential for implementing effective and efficient communication between devices in IoT systems. This chapter delves into the key features, principles, and applications of MQTT, setting the stage for its practical implementation in the subsequent chapters of the smart home automation project.

7.1 Introduction to MQTT:

The chapter begins with an introduction to MQTT, outlining its origins, development, and the specific challenges it addresses within the IoT landscape. Readers gain insights into the need for a lightweight and efficient messaging protocol, which MQTT fulfils by providing a reliable and scalable solution.

7.2 Core Principles of MQTT:

This section explores the fundamental principles that underpin MQTT's operation. Topics include the publish-subscribe architecture, Quality of Service (QoS) levels, and the use of topics as a means of categorising and organising messages. Understanding these principles lays the groundwork for configuring and optimising MQTT in practical scenarios.

7.3 MQTT Components and Architecture:

Readers are guided through the components that constitute an MQTT ecosystem. This includes MQTT brokers, clients, and the flow of messages between publishers and subscribers. The chapter provides clarity on the roles these components play in facilitating communication within an MQTT-enabled IoT network.

7.4 MQTT Topics and Messages:

A detailed exploration of MQTT topics and messages follows, elucidating how devices communicate by publishing messages to specific topics and subscribing to topics of interest. This section highlights the flexibility and versatility MQTT offers in tailoring communication patterns to the specific requirements of IoT applications.

7.5 Quality of Service Levels in MQTT:

MQTT supports different levels of Quality of Service to ensure message delivery reliability. This section dissects the QoS levels, providing insights into scenarios where each level is most appropriate. Understanding QoS levels is crucial for tailoring MQTT to meet the reliability requirements of diverse IoT applications.

7.6 Security Considerations in MQTT:

Security is paramount in IoT deployments, and this section explores the security features and considerations within the MQTT protocol. Topics include authentication, authorization, and encryption, ensuring that readers are equipped with the knowledge to implement secure MQTT communication channels.

7.7 MQTT Use Cases and Applications:

Practical applications of MQTT across various industries and IoT scenarios are presented, showcasing its versatility. Use cases include smart homes, industrial automation, healthcare, and more. By examining these real-world applications, readers gain a holistic understanding of how MQTT contributes to the success of IoT projects.

7.8 Challenges and Future Trends:

The chapter concludes by addressing potential challenges associated with MQTT implementation and offering insights into future trends and enhancements. This forwardlooking perspective ensures that readers are aware of the evolving landscape of MQTT within the dynamic IoT ecosystem.

Chapter 8

HTTP/HTTPS in IoT Communication

Now we shall focus on the HTTP/HTTPS protocols and their relevance in the context of Internet of Things (IoT) communication. As foundational protocols for web-based interactions, understanding HTTP and its secure variant, HTTPS, is crucial for implementing effective and accessible communication within IoT systems. This chapter delves into the principles, use cases, security considerations, and practical applications of HTTP/HTTPS, laying the groundwork for their implementation in the smart home automation project using Cisco Packet Tracer.

8.1 Introduction to HTTP/HTTPS:

This chapter commences with an overview of HTTP (Hypertext Transfer Protocol) and its secure counterpart, HTTPS. Readers gain insights into the historical development and evolution of these protocols, highlighting their significance in facilitating communication over the World Wide Web and their adaptation for IoT use cases.

8.2 Core Principles of HTTP:

This section explores the core principles that govern HTTP's request-response model. Topics include methods (GET, POST, etc.), status codes, and headers. Understanding these principles is fundamental to configuring HTTP-based communication within IoT applications.

8.3 Secure Communication with HTTPS:

The chapter provides an in-depth examination of HTTPS, emphasising the importance of secure communication in IoT deployments. Encryption, digital certificates, and the SSL/TLS protocols are explored, offering readers insights into the mechanisms that underpin HTTPS and contribute to data security.

8.4 HTTP/HTTPS Components and Architecture:

Readers are guided through the components that form the architecture of HTTP/HTTPS-based IoT communication. This includes clients, servers, and the role of intermediaries such as proxies. The chapter elucidates the flow of requests and responses, establishing a foundation for practical configurations in IoT scenarios.

8.5 RESTful Principles in IoT:

This section introduces the Representational State Transfer (REST) architectural style, emphasising its alignment with HTTP principles. Readers gain an understanding of RESTful APIs and how they contribute to creating scalable and interoperable IoT systems.

8.6 HTTP/HTTPS in IoT Use Cases:

Practical applications of HTTP/HTTPS in diverse IoT use cases are explored. This includes scenarios such as remote device control, data retrieval, and communication with cloud-based services. The chapter showcases the adaptability and ubiquity of HTTP/HTTPS in various IoT deployments.

8.7 Security Considerations in HTTP/HTTPS:

Security considerations within the HTTP/HTTPS protocols are thoroughly examined. Topics include authentication mechanisms, secure coding practices, and the mitigation of common security threats. This section equips readers with the knowledge to implement secure HTTP/HTTPS communication channels in their IoT projects.

8.8 Challenges and Future Trends:

The chapter concludes by addressing challenges associated with HTTP/HTTPS in the context of IoT, such as scalability and resource consumption. It also provides insights into emerging trends and enhancements, ensuring readers are attuned to the evolving landscape of HTTP/HTTPS in IoT deployments.

Chapter 9

TCP (Transmission Control Protocol) in IoT Communication

Chapter 6 explores the role of TCP (Transmission Control Protocol) in the context of Internet of Things (IoT) communication. As a foundational protocol in networking, TCP plays a crucial role in ensuring reliable and connection-oriented data transmission. This chapter delves into the principles, use cases, considerations, and practical applications of TCP within an IoT environment, preparing readers for its implementation in the smart home automation project using Cisco Packet Tracer.

9.1 Introduction to TCP:

The chapter begins with an introduction to TCP, providing an overview of its development, principles, and its significance in reliable and ordered data delivery. Readers gain insights into how TCP establishes and maintains connections between devices, a fundamental aspect of IoT communication.

9.2 TCP Components and Connection Establishment:

Readers are guided through the components that constitute the TCP protocol suite and the process of connection establishment. This includes the role of TCP clients and servers, as well as the sequence of events involved in establishing a reliable connection between devices in an IoT network.

9.3 Reliable Data Transfer with TCP:

TCP ensures reliable data transfer. This includes acknowledgment mechanisms, retransmission strategies, and the use of sequence numbers to maintain order in transmitted data. The reliability of TCP makes it suitable for scenarios where accurate data delivery is critical.

9.4 Flow Control and Congestion Avoidance:

TCP incorporates flow control and congestion avoidance mechanisms to optimise data transmission in varying network conditions.

9.5 Security Considerations in TCP:

Security is paramount in IoT deployments, and this protocol addresses security considerations within the IOT system. It includes services like encryption, secure socket layers (SSL), and considerations for ensuring the confidentiality and integrity of data transmitted over TCP connections.

Chapter 10

Plans for packet tracer implementation

In this chapter, we explore the planned expansion of features within the smart home simulation, detailing the addition of new devices and their interconnections. The objective is to enhance the overall functionality and user experience, ensuring a comprehensive and seamlessly interconnected smart home environment within the confines of Cisco Packet Tracer.

Steps Involved :

Part 1: Connecting Devices to the Home Gateway

Step 1: Configure Home Gateway.

a Click Home to expand the cluster.
b. Click Home Gateway. Click Config tab.
c. Click Internet. Select DHCP for IP Configuration.
d. Click Wireless. Enter MyHomeGateway as the SSID. Select WPA2-PSK as the authentication. Enter CiscoIoT as the PSK Pass Phrase.
e. Click Home PC. Click Desktop. Click IP Configuration. Select DHCP for IP Configuration.

Step 2: Verify Home Gateway connectivity to registration server.

a. From Home PC, enter www.register.pka in the Web Browser. Enter admin as the Username and admin as the Password to log in to the remote registration server.
Note: It may take a few minutes for all the IoT devices to be listed.
b. From Home PC, enter 192.168.25.1 in the Web Browser. Enter admin as the Username and admin as the Password to log into the local IoT server hosted on Home Gateway.

Part 2: Interacting with IoT Devices

Step 1: Operating air conditioning through smartphones

Currently the Thermostat is off. Set the Thermostat to automatically adjust to the desired setting.Expand the Thermostatstatus bar. Click Auto. Enter the desired auto heating and auto cooling temperatures and click Set.

Thermostat is installed with its condition and ac is connected to thermostat and ac program is set in thermostat itself ,because ac is connected to thermostat and home gateway of thermostat.

Also the temperature monitor and humidity device have also been connected.Which comes to know that after the ac is on, what is the temperature of the room and the humidity is also known.

Step 2: Smoke detection with fire prevention

a. Place the smoke detector in the kitchen, bedroom,garage of the house
b. Place the fire prevention tools (fire sprinkler, window and door, garage door) in the kitchen , bedroom and garage of the house. c. place the smart device for access the data.
d. Connect and configure all components with home gateway. Take 3 switch to connect all fire prevention tools of kitchen, bedroom and garage of the house.Connect switch with home gateway.
e. Check them whether all are connected (access from smart device) then write the conditions for automation of fire prevention in IoT monitor.
f. At last, generate some smoke using car (Alt + click), It will open all window, doors and sprinkler.

Step 3: Theif Catching System:

Creating a thief catcher system using Cisco Packet Tracer involves simulating the integration of motion detectors, CCTV cameras, and a siren within a networked environment

a. Place the Motion Detector,Webcam(CCTV) and Siren in the entrance of the house
b. Connect and configure all components with home gateway and smartphone Take 3 switch to connect all fire prevention tools of kitchen, bedroom and garage of the house.
c. Check them whether all are connected (access from smart device) then write the conditions for thief catching system in IoT monitor.
d. At last, (Alt + click) to move with cursor.The CCTV will on and Siren will ring.

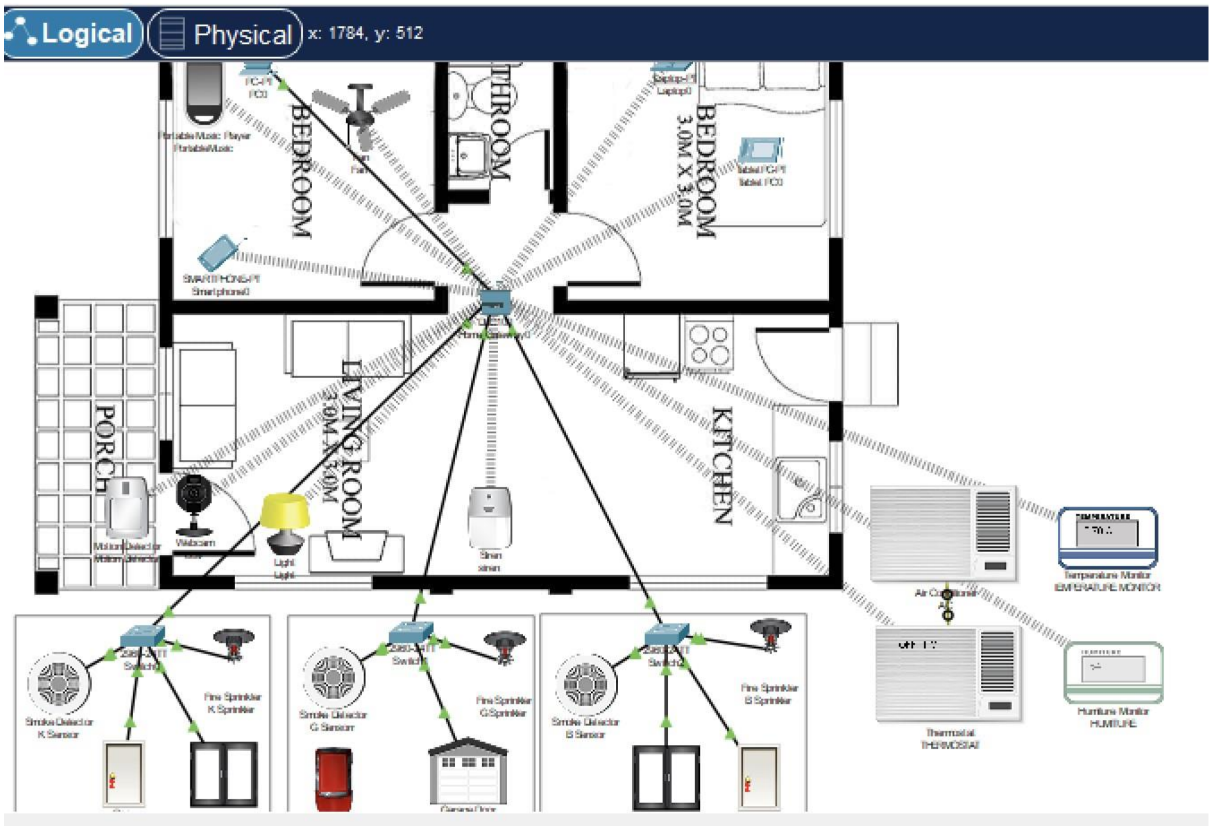
Devices used in the setup:

No.	**Device or Function Machine**	
1	Smart Phone, Tablet, Laptop	Connect to home getaway to access smart object
2	Home Gateway	Used to register smart object and give IP address to it, then you can remotely managed through a web interface hosted
3	MicroController (MCU-PT) Board	Microcontroller board is used to interconnect different smart object and provide programming environment with different language those are JavaScript, python and visual basic, to control the connected smart object
4	Fire monitor	Detect IR in the range of fire
5	Fire Sprinkler	A Sprinkler that puts out fire
6	Fire	Used to simulate different scenario in home design since it affect fire
7	Outdoor Security Camera	Monitoring what's happening outside , it's work when the motion outdoor detector detect motion
8	Motion Outdoor Detector	Connect to home getaway and provide Detection of motion outside the home
9	Trip Sensor	Is a type of motion detector that can detect movement across a laser beam, used for security
10	Siren	Provide sound for event on the trip sensor
11	RFID Reader	Read the ID of an RFID Card, Transmit that ID to the registration server
12	RFID Card	Interacts with the RFID Reader based on some conditions
13	Smart door	Connect to home getaway and provide Function based event
12	Carbon Monoxide Detector	Detects the level of the carbon monoxide, alarm will turn on when the level > 20%
14	Old car	Used to change the Carbon Monoxide level, Carbon Dioxide and smoke level
15	Motion Car Detector	Connect to home getaway and it senses the movement of the car
16	Garage Door	Automatically opens when there is a car coming, it remains open for 120 second

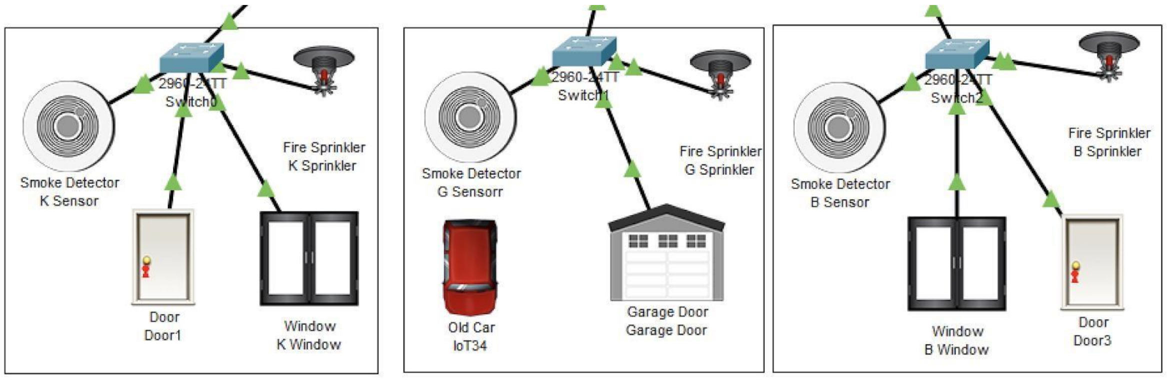
No.	Device or Machine	Function
17	Smart window	Used to control the window remotely Affects Argon, Carbon Monoxide, Carbon Dioxide, Hydrogen, Helium, Methane, Nitrogen, O2, Ozone, Propane, and Smoke, wind speed
18	Wind Speed Sensor	It measures wind speed and provides data reporting average wind speed
19	Smart light in home corridor	Used to give light in the corridor when there is movement
20	Motion Detector in Corridor	Connect to home getaway and provide Detection of motion in the home corridor
21	Smart thermostat	Used to sense the temperature of the home
22	Air conditioner	Reduce temperature if temp >=22 °C
23	Furnace	Increase temperature if temp <=15 °C
24	Appliance	Instrument / Device uses at home
25	Solar panel	Generates power based on the amount of SUNLIGHT in the environment Sends generated power to another device such as Battery
26	Battery	Send power to other devices
27	Ceiling fan	Used to ventilate the home environment
28	Portable Music Player	Plays music through Bluetooth capabilities
29	Bluetooth Speaker	Plays sound through Bluetooth from a Portable Music Player

Outputs:

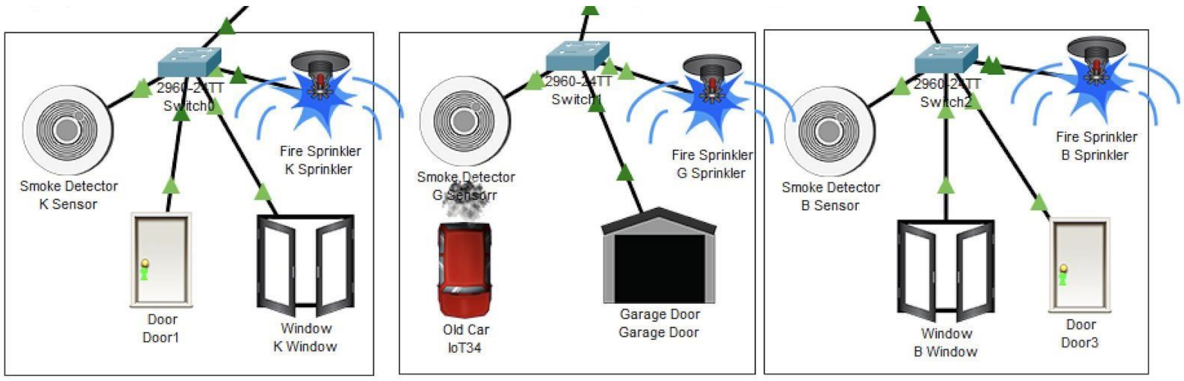
1. Home -IoT Aplication



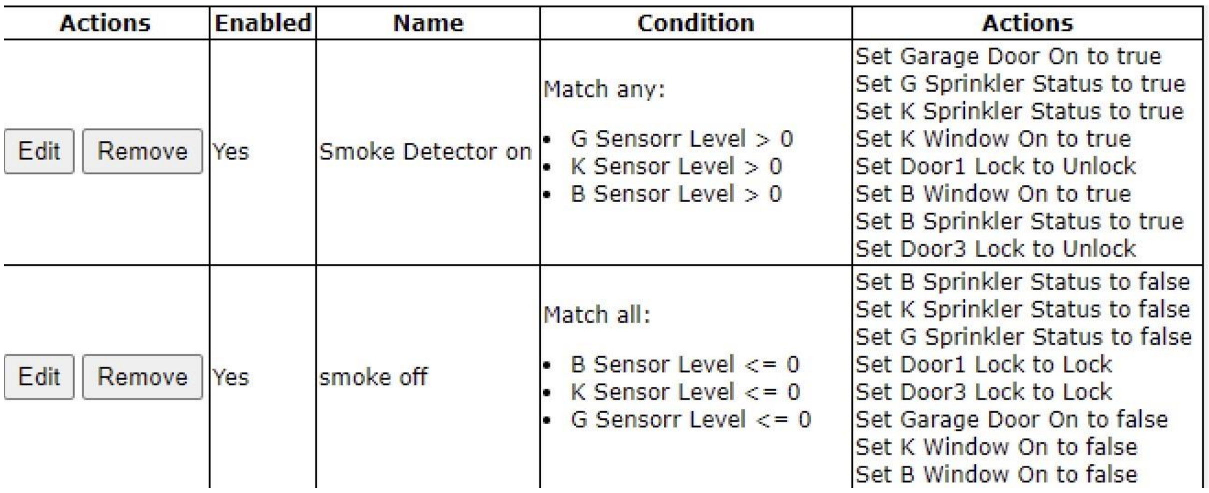
2.Smoke detection with fire prevention when there is no smoke



3. Smoke detection with fire prevention when there is smoke

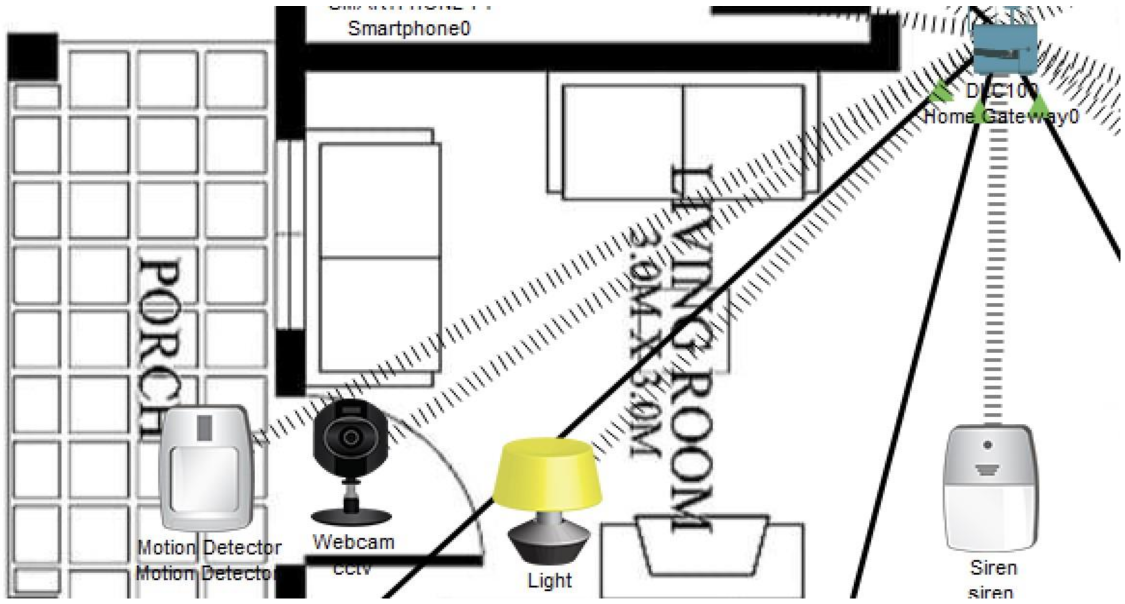


4.The conditions used for the Smoke detection with fire prevention

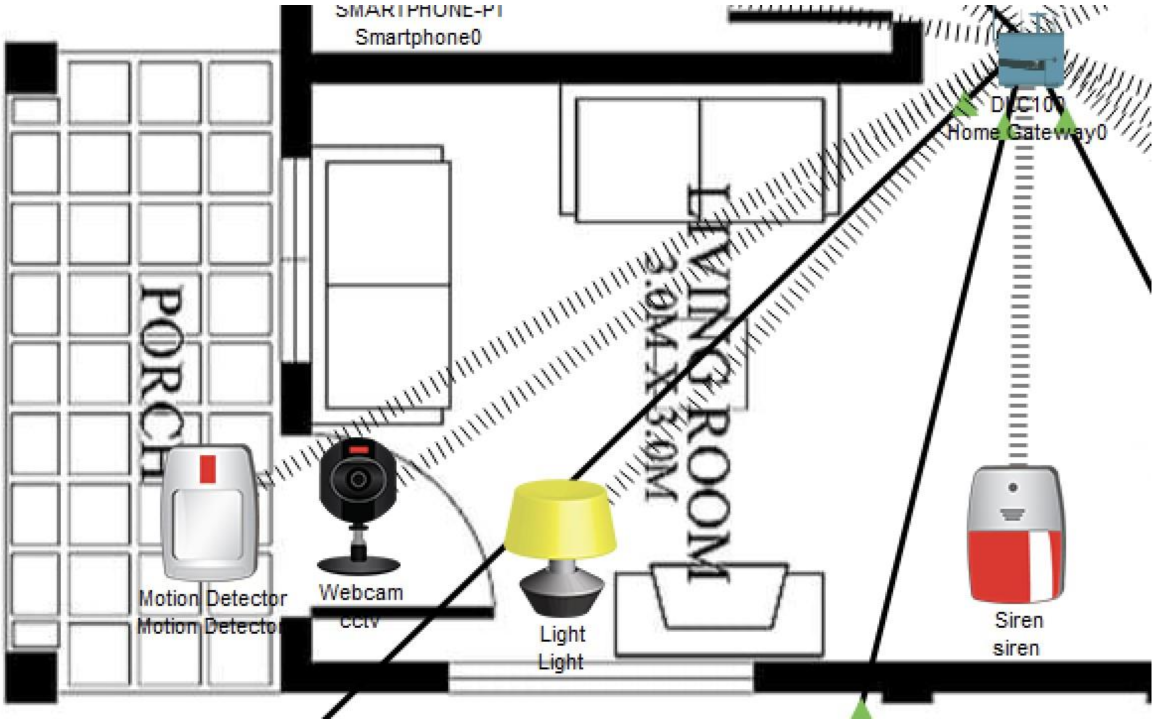


Actions	Enabled	Name	Condition	Actions
Edit Remove	Yes	Smoke Detector on	Match any: • G Sensorr Level > 0 • K Sensor Level > 0 • B Sensor Level > 0	Set Garage Door On to true Set G Sprinkler Status to true Set K Sprinkler Status to true Set K Window On to true Set Door1 Lock to Unlock Set B Window On to true Set B Sprinkler Status to true Set Door3 Lock to Unlock
Edit Remove	Yes	smoke off	Match all: • B Sensor Level <= 0 • K Sensor Level <= 0 • G Sensorr Level <= 0	Set B Sprinkler Status to false Set K Sprinkler Status to false Set G Sprinkler Status to false Set Door1 Lock to Lock Set Door3 Lock to Lock Set Garage Door On to false Set K Window On to false Set B Window On to false

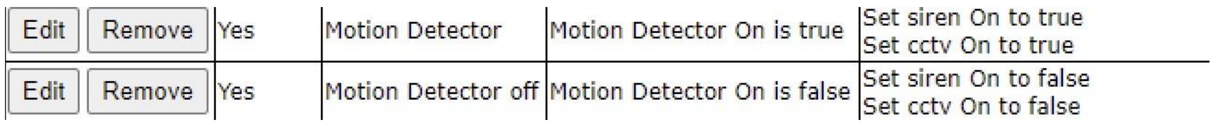
5.Thief Catching System-Motion Detector is off



6. Thief Catching System-Motion Detector is on



7.The conditions used in Thief Catching System



Edit	Remove	Yes	Motion Detector	Motion Detector On is true	Set siren On to true Set cctv On to true
Edit	Remove	Yes	Motion Detector off	Motion Detector On is false	Set siren On to false Set cctv On to false

Conclusion:

In conclusion, our semester-long project has successfully brought to fruition two impactful initiatives in the realms of computer networks: a cutting-edge broadcast tool for inclusive learning environments and a meticulously crafted smart home system. Through these endeavors, we have demonstrated the tangible benefits of technology in fostering accessibility, inclusivity, and convenience.

The broadcast tool addresses the unique needs of large educational settings, ensuring seamless communication for all participants and emphasizing inclusivity. On the other hand, the smart home system, implemented within Cisco Packet Tracer, showcases the integration of interconnected devices through a central gateway, offering a glimpse into the future of intelligent living.

As we reflect on these achievements, it is evident that our exploration not only contributes to the evolution of technology but also underscores the transformative potential of computer networks in shaping a more connected and accessible world for education and daily life alike.