King Saud University

College of Computer and Information Sciences

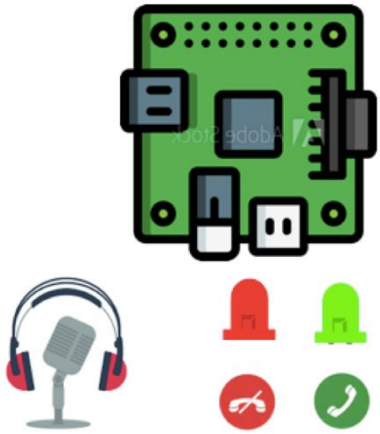
Department of Computer Engineering

CEN 493: Graduation Project (2)

First Semester 2019-2020

Making Landline Voice Calls Through Ethernet

Advisor: Dr. Ahmed Alhammad



Khalid Abdullah Aldoaij 435100736

Turki Abdullah Alawad 435102640

Ahmad Abdo Alomari 435107401

December 2019

It is submitted as partial fulfillment of the requirement for Degree of Bachelor of Science in Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Kingdom of Saudi Arabia.

Abstract

Current telecom providers offer the landline service as part of the fiber optic bundle. Normally, the customer's main router has a port to connect a phone and make landline calls. In this project, we propose to design an adapter to connect one of the Ethernet ports of the router to a landline port and make calls through the network. In this case, we save the customer from buying a landline phone set and make it easy to make calls through an app installed on a mobile phone from anywhere in the house.

Acknowledgment

First of all, we thank Allah for blessing us to reach this phase. We Also would like to extend the thanks to our family for the support they give to us since day one we joined King Saud University.
Moreover, a special "thank you letter" for our supervisor Dr. Ahmed Alhammad, for giving us this the great opportunity and the complete leading and guidance throughout the project. For his passion, support, and effort. We can reach to this level because of all the effort and caring he provided to us.

Contents List

Abstract **2**
Acknowledgment **3**
Contents List **4**
List of Tables **6**
List of Figures **7**
Chapter 1: Introduction **8**
Chapter 2: Background **11**
2.1 Telephone 12
2.1.1 Telephone Component 12
2.1.2 Telephone Line Wire 13
2.2 Ethernet 13
2.2.1 What is an Ethernet 13
2.2.2 The Ethernet Frame 13
2.2.3 The Ethernet Cables 14
2.2.4 Crossover Cable 14
2.3 IP and TCP/UDP Protocols 15
2.3.1 IP Protocol 15
2.3.2 TCP Protocol 16
2.3.3 UDP Protocol 16
2.4 Arduino 17
2.4.1 Types of Arduino boards 17
2.5 The Raspberry Pi 18
2.6 Streaming 18
2.6.1 QoS for Voice Streaming 18
Chapter 3: System Design **19**
3.1 System Overview 20
3.2 Hardware Component 21
3.2.1 Arduino 21
3.2.2 Arduino Ethernet Shield 22
3.2.3 The Raspberry Pi 22
3.3 The Libraries 23
3.3.1 Arduino Libraries 23
3.3.2 Raspberry Pi Libraries 24

Chapter 4: Implementation **25**
4.1 Arduino 26
4.1.1 Controlling LED ON/OFF using Push Button 26
4.1.2 Simplex Communication 26
4.1.3 Full Duplex Communication 27
4.1.4 Send text files 27
4.1.5 Send audio (mp3) file 28
4.2 The Raspberry Pi 29
4.2.1 Setup and Configuration 29
4.2.2 Controlling LED ON/OFF using Push Button 30
4.2.3 UDP Connection 30
4.2.4 Full Duplex Communication 31
4.2.5 Record and play audio 34
4.2.6 Streaming 35
4.2.7 Controlling the Streaming 36
Conclusion **38**
Future work **38**
Reference **39**
Appendix **41**
Appendix A (The Code) 41
Appendix B (The Survey) 56

List of Tables

Table 1: The Different Between Our Project And The Current Technology 10
Table 2: The Comparison of Some Arduino Board 17
Table 3: The Comparison Between The UNO, MKR Arduino And Raspberry Pi.....23
Table 4: The Some Of Arduino Libraries 23
Table 5: The Some Of Raspberry Pi Libraries.....24

List of Figures

Figure 1: Survey....9
Figure 2: The Public Switched Telephone Network (PSTN) 12
Figure 3: Telephone Component 12
Figure 4: OSI Model vs. TCP/IP Model 13
Figure 5: Ethernet Frame 13
Figure 6: Ethernet Crossover Cable 14
Figure 7: IPv4 Header 15
Figure 8: TCP Header 16
Figure 9: UDP Header....16
Figure 10: Difference Between TCP and UDP 16
Figure 11: Some Types of Arduino Boards 17
Figure 12: Some of Raspberry Pi models 18
Figure 13: The System Overview 20
Figure 14: Flow Chart of System....20
Figure 15: Arduino UNO 21
Figure 16: Arduino Mega....21
Figure 17: MKR 1010 Arduion....21
Figure 18: Arduino Ethernet Shield 22
Figure 19: The components of the Raspberry Pi 3 model B+ 22
Figure 20: LED ON/OFF Using Push Button (Arduino) 26
Figure 21: Simplex communication between two UNO Arduinos 26
Figure 22: Full Duplex Communication Between Two Arduinos 27
Figure 23: The Arduino Mega (Sender) And MKR1010 Arduino (Receiver) 27
Figure 24: The Three steps To Solve The Problem 28
Figure 25: Controlling LED ON/OFF (Raspberry Pi) 30
Figure 26: UDP connection between two Raspberry Pi 30
Figure 27: The process of creating UDP connection 31
Figure 28: Full Duplex communication 31
Figure 29:The state digram of system with connection control....32
Figure 30: Solve the block problem....33
Figure 31:Raspberry Pi with microphone and speaker 34
Figure 32: Streaming between Two Raspberry Pi 35
Figure 33: Controlling the streaming between two Raspberry Pi....36
Figure 34: How the system work 36

Chapter 1: Introduction

The landline telephone is one of many important devices that has ever been created. It is was invented in the 19th century, that was designed to transmit human voice from one node to another as a telecommunication system, a voice is converted to an electrical signal which transmitted over PSTN (Public Switched Telephone Network) and the receiver convert again that electrical signal into sound. There are many benefits of the landline telephone it is inexpensive, easy to use, available, and it is a plug and play device, but nowadays with massive growth of technology most people found that using cell phone gives you more freedom from limiting to stay within a small range where the telephone is, even though a making calls through cell phones will cost more, so the usage of landline telephone became few, even though current telecom providers offer the landline service as part of the fiber optic bundle.

We did a survey to know the current situation of using landline telephone in our community (kingdom of Saudi Arabia), and we found out that lots of people do not use the landline telephone by 66.7% from our population that is shown in **FIGURE 1**, 59% of people said the reason is the limitation of area (Cannot move it elsewhere), 32% said it is an old technology and is not used often between people. From the survey 73% said the internet service is provided with the telephone number is the reason to still using the landline telephone at home (see the result of the survey in Appendix B). This result supports the idea we are trying to provide it.

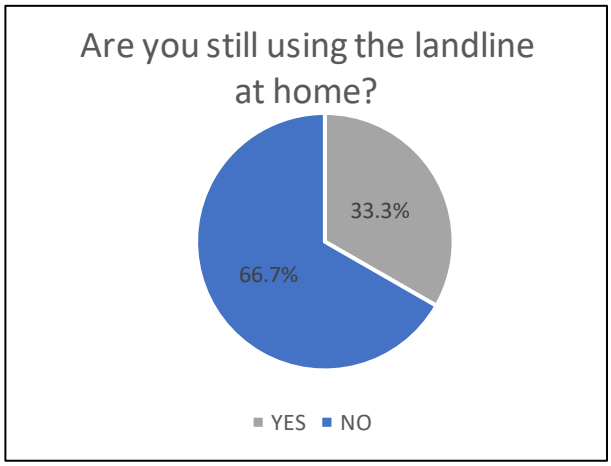


FIGURE 1: SURVEY

The idea of our project is that you can make Landline Calls Through an Ethernet, which allows you to use an application on your phone to make this call using the service was offered by the telecom providers.

Our project will be used in many areas such as:

- Business and Government area: it will increase the Work productivity; for example, it will help the call center employee to answer client calls even if the employee is not in his office.
- Entertainment area: it will allow people to make calls for a cheaper price. Also makes you answer your landline calls even if you are outside the home.

The difference between our project and the current technology that is shown in **TABLE 1**:

	Our project	**Current technology**
Use the same application		✓
Customer use his number	✓	
Use existing network	✓	✓
Need to create an account		✓
Can answer landline telephone calls	✓	
Use the stream concept	✓	✓
Need a centralized server		✓
Reduce the cost	✓	✓
Digitized the voice signal	✓	✓

TABLE 1: THE DIFFERENT BETWEEN OUR PROJECT AND THE CURRENT TECHNOLOGY

With reference to IEEE code of ethics, we will take these principles into our consideration. We have credited all sources that we used to take the knowledge on building our project. All the results shown in this report are collected from real experiments done by us on the lab and not Misrepresentation, Fabrication and Falsification them. accept responsibility in making decisions consistent with the safety, health and welfare of the public. Used the resource we were given in appropriate way.

In the following chapters, we will go deep and talk about software tools and hardware equipment that is used in our project. Also, we will show the problems we face and how we solved it. Our report will be written in the following way:
In **chapter two: Background**, we will show the background of the impotent concept that is needed in our project, **chapter three: System Design**, will describe our system and its component in more details (software and hardware), our work details with problems and how we deal whit it will be shown in **chapter four: Implementation.** Finally, we write the **Conclusion** and **Future Work**.

Chapter 2: Background

In this chapter, we will show the background of the impotent concept that is needed in our project as Telephone, Ethernet and Arduino.

2.1 Telephone

The Public Switched Telephone Network (PSTN) is now using to connect the landline telephones in the world, that provide to use digital technology between Central Offices (CO), but the end to end user line is an analog signal with bandwidth is 3000Hz, that is shown in **FIGURE 2**.

The analog system was the first telephone system established worldwide. Currently, telephone systems in a lot of countries are still completely analog. In time, however, these systems will become redundant as the world switches to digital telephony. At present, most telephone calls are analog from the telephone at home to the first Central Offices (CO), so the A/D and D/A conversion are made at this office. In the future, as telephone systems become all digital, this conversion from A/D and D/A will be made within the telephone set at home. Although the analog telephone system is gradually being converted to digital, the input and output of the system remains analog because the eventual use is for humans that are only able to process analog information.[1] [2] [3]

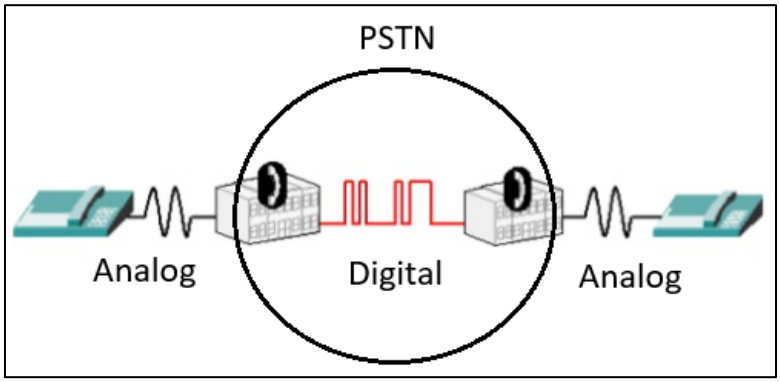


FIGURE 2: THE PUBLIC SWITCHED TELEPHONE NETWORK (PSTN)

2.1.1 Telephone Component

The modern telephone has three main parts:

- Hook Switch
- Speaker
- Microphone

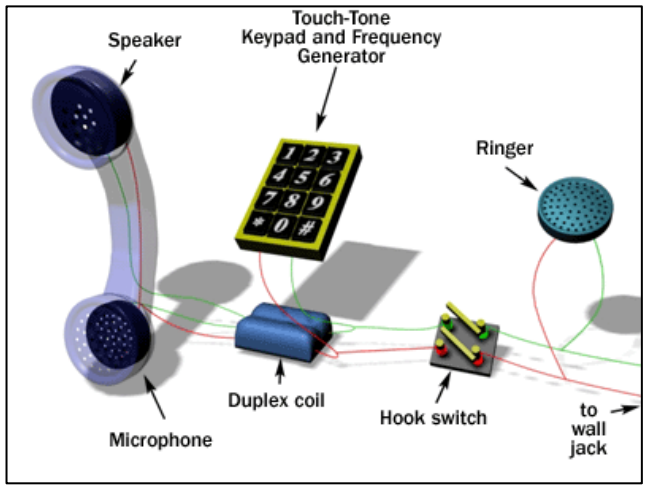


FIGURE 3: TELEPHONE COMPONENT

Hook Switch: It is a physical controller that used for answering and hanging up a call on a landline telephone. There are two cases: off-hook is the state of the landline telephone that gives the availability of dialing and transmission but prevents incoming calls, and on-hook is state of the landline telephone is idle (waiting for a call). The voltage on the on-hook state is approximately 50V, and it will be decreased to around 8V at off-hook state.

Speaker: It is an output device that converts the receiving electrical signal to sound wave.

Microphone: It is a device that converts sound wave to an electrical signal, includes an amplifier. The telephone component is shown in **FIGURE 3**.[4]

2.1.2 Telephone Line Wire

The telephone line wire carries the electrical signal and the ring voltage. In nowadays the CAT5 cable is commonly used, it contains four wires: green, red, black and yellow. Each pair consisted of a Tip (positive) and Ring (negative).

2.2 Ethernet

In this section, we will shortly take about Ethernet, but first, we will discuss the TCP/IP model:
TCP/IP model developed by the US Defense Advanced Research Project Agency (DARPA). It consists of five layers:

- Application Layer: supporting network applications (e.g., FTP, SMTP, HTTP, …).
- Transport Layer: It provides transparent transfer of data between end systems (TCP/UDP).
- Network Layer: Routing the packet from source to destination (IP).
- Data link Layer: Responsible for data transfer between the point-to-point node (Ethernet).
- Physical Layer: Responsible for sending computer bits from one device to another.[5]

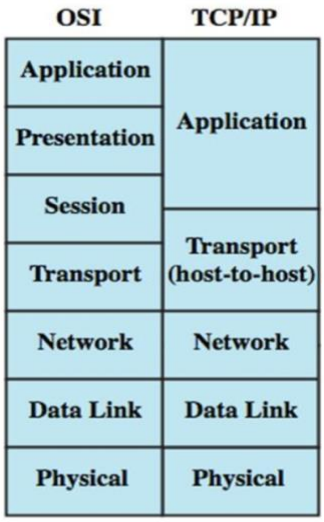


FIGURE 4: OSI MODEL VS. TCP/IP MODEL

2.2.1 What is an Ethernet

Ethernet is a technology used to connect devices by a wire to create a local area network (LANs); it allows devices to communicate with each other using a set of rules and protocols. It is a Data link Layer protocol in the TCP/IP model, Ethernet describes how to transmit frames based on the Media Access Control (MAC) Addres0s (IEEE 802), MAC address is linked to the hardware of network adapters Network Interface Card (NIC), which consist of six bytes that are divided into to two parts: Organizationally Unique Identifier (OUI) and Network Interface Controller (NIC). [6] [7]

2.2.2 The Ethernet Frame

The maximum size of the Ethernet frame is 1518 bytes; it contains the following parts:

- **Header:**
  Destination address: 6-Byte field which contains the MAC address of the destination.
  Source address: 6-Byte field which contains the MAC address of the source.
  Length or type: if 802.3, Length is a 2-Byte field, which indicates the length of entire Ethernet frame / Ethernet if type.
- **Data Load:**
  Data: This is the place where actual data is inserted, also known as **Payload.**
  Pad: if data < 46 byte.
- **Trailer:** FCS: Frame Check Sequence Field use Cyclic Redundancy Check (CRC-32).[8]

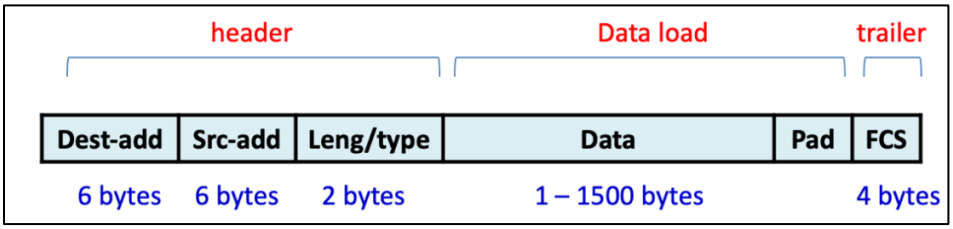


FIGURE 5: ETHERNET FRAME

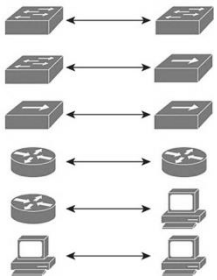
2.2.3 The Ethernet Cables

Ethernet cable is one of the most famous cables that has been used on wired networks. It has been used for connecting devices within a LAN; these cables have a distance limitation. The ethernet cable is like the landline telephone cable (RJ11), but it is wider and has more wires. They have similar shape and plug; the difference is that Ethernet contains eight wires and larger plug than the four wires which are in the landline telephone. Ethernet cables plug into Ethernet ports (RJ45), which are larger than phone cable ports. An Ethernet port on a computer is accessible through the Ethernet card on the motherboard. Ethernet cables support one or more industry standards including Category 5 and Category 6. [9]

2.2.4 Crossover Cable

A cross cable Ethernet cable is a type of Ethernet cable it does exactly as its name suggests, used to connect same type of devices:

- Switch to switch
- Switch to hub
- Hub to hub
- Router to router
- Router Ethernet port to PC NIC
- PC to PC



The inner wiring of Ethernet crossover cables reverses the transmit (TX) and receive (RX) signals, it crosses over the wire from pin 1 at one end to pin 3 at the other connector and crosses over the wire from pin 2 at one end to pin 6 at the other connector and remaining wires connect in the same place at both ends as is shown in **Figure 6**.

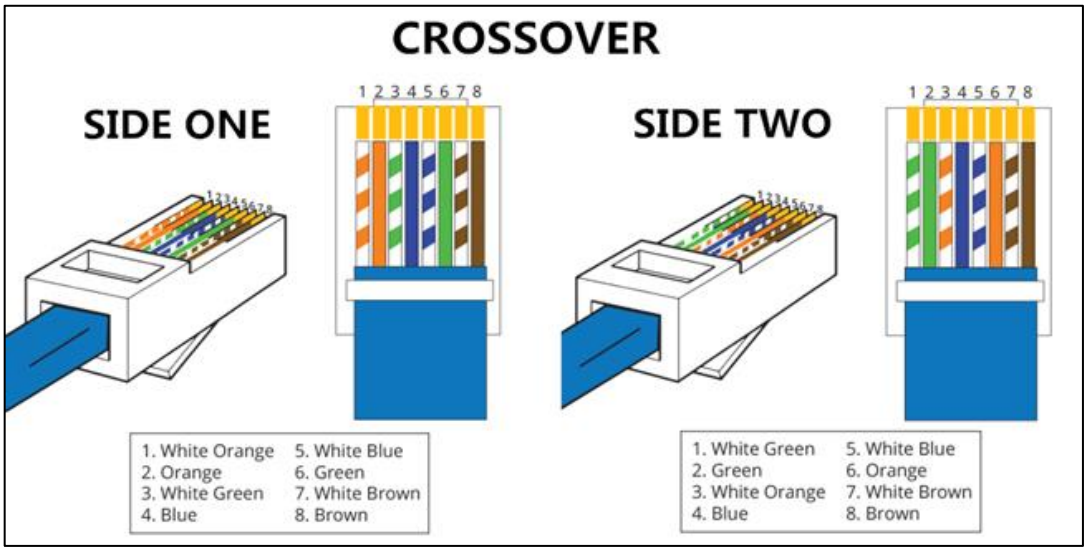


FIGURE 6: ETHERNET CROSSOVER CABLE

2.3 IP and TCP/UDP Protocols

In this section, we will shortly take about the IP and TCP/UDP protocols. The Internet Protocol (IP) is network layer protocol. The Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) both are Transport layer Protocols, and they are an End-to-End protocol each one used in the specific application.

2.3.1 IP Protocol

The IP Protocol is the one that is widely used on the routing in internet. Each device connect with internet has at least one IP address to differentiates it from other devices. Currently, there are two versions: IPv4 and IPv6.

At the beginning IPv4 originally designed for universities and government, but now different types of users, requirements. All network devices need a unique address whether watches, smartphones, tablet computer, smart TV, laptop etc. because of that, IPv4 addresses almost exhausted so IPv6 designed to solve this problem.

IPv4 address is 32 bits, consists of network and host portions, represented as decimal value four octets (bytes).

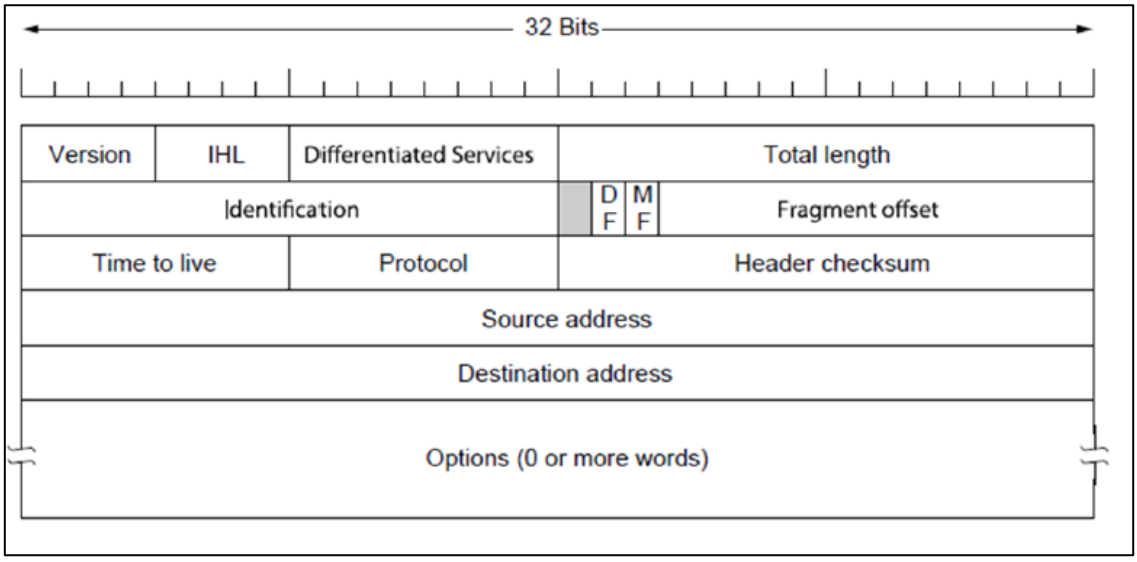


FIGURE 7: IPV4 HEADER

2.3.2 TCP Protocol

The TCP Protocol is the one that is widely used on the internet. TCP is a connection-oriented protocol, which means there must be a connection setup "handshake" before sending any data.[8]

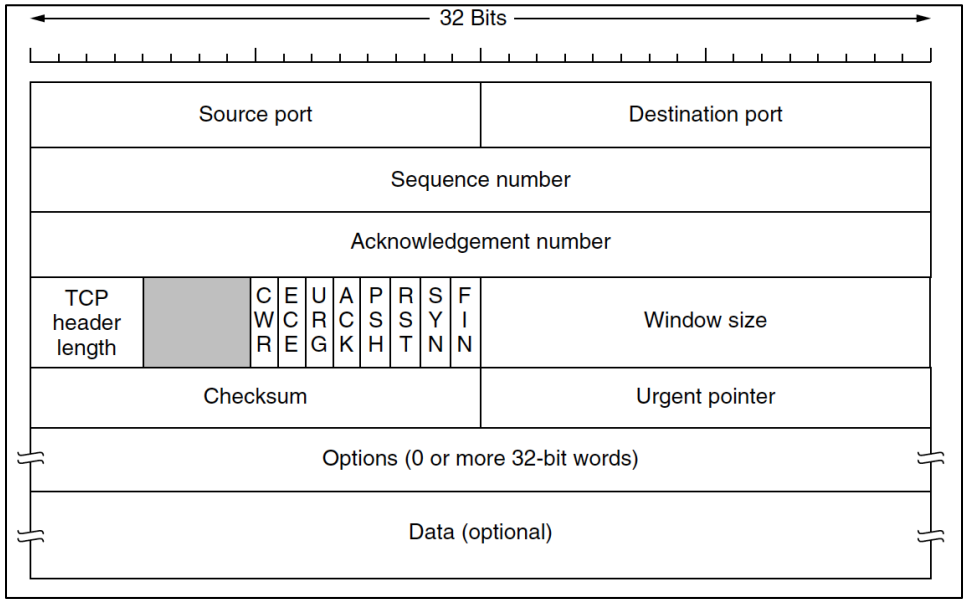


FIGURE 8: TCP HEADER

2.3.3 UDP Protocol

The UDP is an unreliable and connectionless protocol unlike the TCP protocol, the UDP does not provide flow control or error control, and there is no connection established needed, UDP is kind of a 'null' transport layer, the UDP is not used when important data is needed to be sent (e.g. webpages, database information), UDP used in Streaming applications like Audio and video because it offers speed since there is no much process.[8]. The difference between TCP and UDP is shows in **Figure 9**.[10]

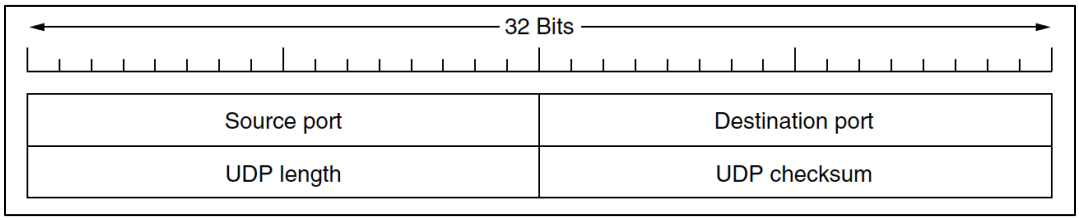


FIGURE 9: UDP HEADER

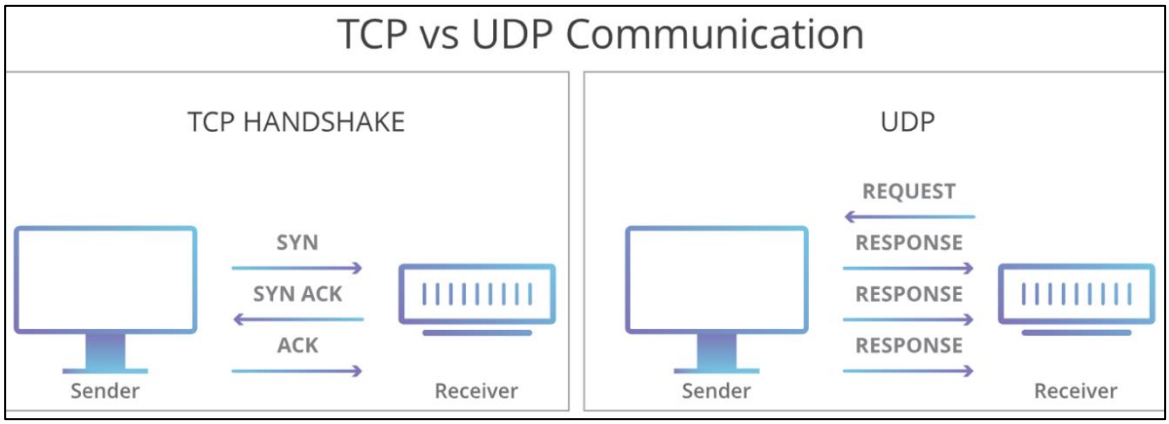


FIGURE 10: DIFFERENCE BETWEEN TCP AND UDP

2.4 Arduino

Arduino is an open-source program used for building simple projects, it is a single-board microcontroller, it consists of Software (IDE), and Hardware components designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM. It is designed to produce the application of interactive objects more accessible. Many electrical and computer engineering projects involve some kind of embedded system in which a microcontroller sits at the center as the primary source of control. There are a lot of versions of the Arduino board they are different in components, aim, and size, etc. Arduino is a great tool for developing interactive objects, it offers some advantage for teachers and students to prove chemistry and physics principles, or to get started with programming and robotics. The Arduino is cross-platform, inexpensive, the simple and clear programming environment and open source and extensible hardware/software [11]

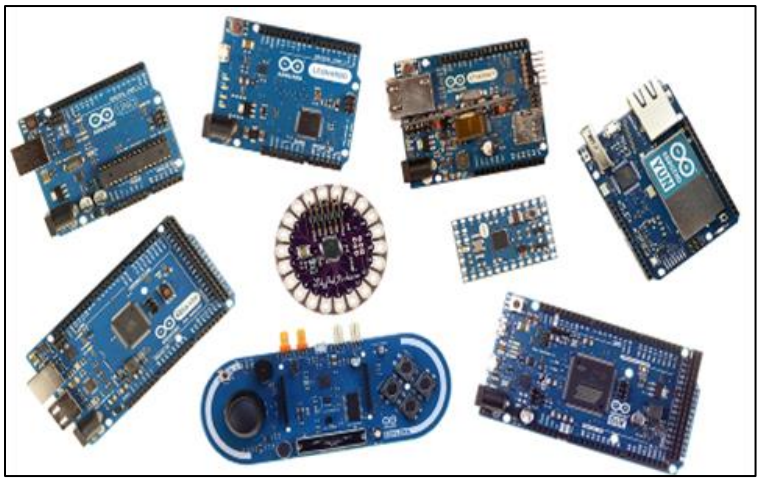


FIGURE 11: SOMETYPES OF ARDUINO BOARDS

2.4.1 Types of Arduino boards

The most widespread board is UNO, but all other types have existing support through online, because of similar architecture and the use of a similar language and IDE. There are different Arduino boards such as:

- Uno R3
- Leonardo
- Micro
- Nano
- Mini
- Mega
- Due
- Yun Mini
- MKR 1010 [12]

They are different in components, CPU power, Memory size, Physical dimensions, I/O capabilities, On-board peripherals, etc. The table below is a simple comparison between some Arduino Board:[13] [14] [15] [16]

Arduino Board	Processor	Memory	Digital IO/PWM	Anlage I/O
Arduino Uno	16Mhz ATmega328	2KB SRAM 32KB flash	14/6	6 input 0 output
Arduino Mega	16MHz ATmega2560	8KB SRAM 256KB flash	54/15	16 input 0 output
MKR 1010 Arduino	48MHz SAMD21	32 KB SRAM 256 KB flash	8/12	7 input 1 output
Arduino Due	84MHz AT91SAM3X8E	96KB SRAM 512KB flash	54 /12	12 input 2 output

TABLE 2: THE COMPARISON OF SOMEARDUINO BOARD

2.5 The Raspberry Pi

Raspberry Pi is the name of a series of single-board computers made by the Raspberry Pi Foundation, a UK charity that aims to educate people in computing and create easier access to computing education. The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT), also support mulit threading.[17]
Some people buy a Raspberry Pi to learn to code, and people who can already code use the Pi to learn to code electronics for physical projects. The Raspberry Pi can open opportunities for you to create your own home automation projects, which is popular among people in the open source community because it puts you in control, rather than using a proprietary closed system. From our side we tried to use it to help as to act like landline telephone.
There are different models of Raspberry Pi with different features:

- Model A+
- Model B+
- 3 Model B
- 3 Model B+
- 3 Model A+
- 4 Model B
- Zero
- Zero W
- Zero WH

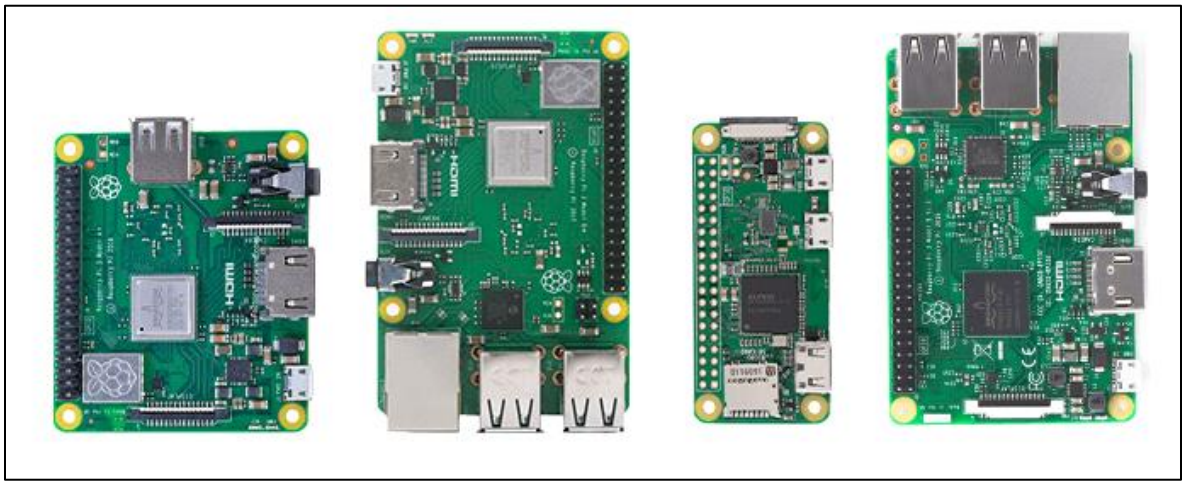


FIGURE 12: SOME OF RASPBERRY PI MODELS

2.6 Streaming

Streaming is a technology that can be used to send data to end Device (e.g., workstation, Phones) through the internet. When we talk about Streaming mostly, we mean transmitting audio or video. There are two methods to download content from the internet Progressive download and Streaming. [18]

2.6.1 QoS for Voice Streaming

Streaming need to have a fast internet connection, the required speed depends on the type of media is going to be streamed, in our case we care about a Streaming voice, so there is a QoS (Quality of Service) for this type of application, the most important QoS is Delay, Jitter. Voice streaming does not need a huge amount of bandwidth.[18]

In the next chapter, we talk in deep about the system design and the component of our system with which tools we were using.

Chapter 3: System Design

In this chapter, we introduce brief information about the devices and components that we used in our project. The first section will discuss System overview, the second section will be about the Hardware Component, and the last section will be about the Libraries we used.

3.1 System Overview

Our system is composed of one microcontroller, Telephone Circuits and software application, each will be described in the next sections of this chapter. Each user can make a call or answer a call through a router using the application. The router will pass the calls from the application to the adapter (our project) through RJ45 port. In this adapter the microcontroller that will accept or refuse the call (Controlled by the application), then retransmitted to a Telephone circuit, it will receive voice from microcontroller then retransmitted to the telecom provider (e.g., STC) through RJ11 port. The user from on other side do not know the source of the call, is it from an ordinary phone or our system **FIGURE 13** is show the system overview, the flow chart of system is shown in **FIGURE 14.**

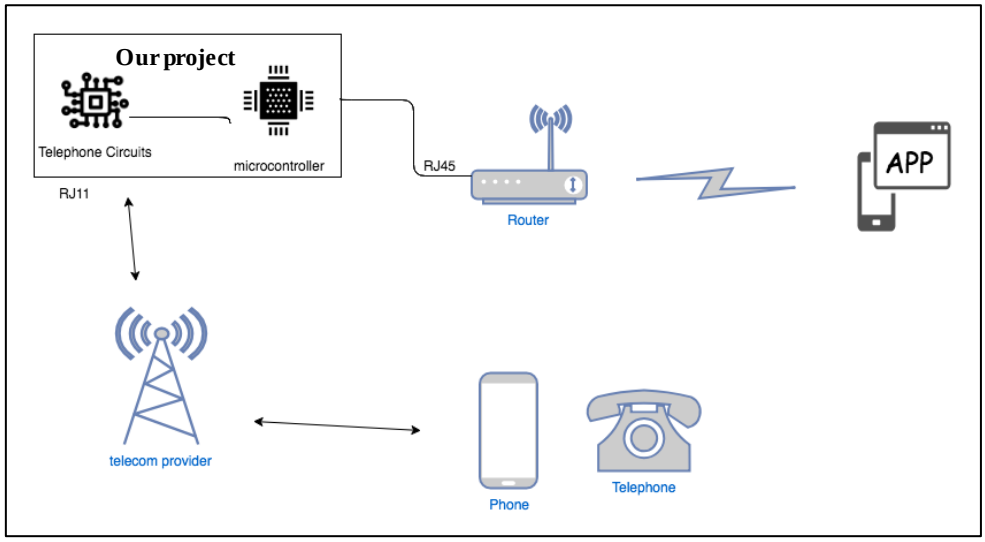


FIGURE 13: THE SYSTEM OVERVIEW

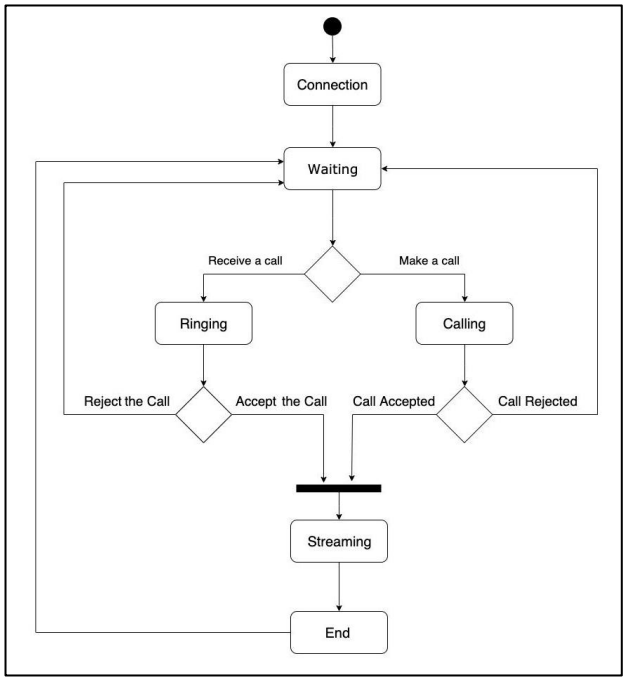


FIGURE 14: FLOW CHART OF SYSTEM

3.2 Hardware Component

In this section we will talk about the hardware component we used on our project. First, we will talk about Arduino and why we choose it and which Arduino type we used, then talk about the Raspberry Pi.

3.2.1 Arduino

When we started this project, we faced the commonly main problems: which microcontroller or microprocessor should be used? And Is it compatible with our project idea? In our project first we choice the Arduino UNO for many reasons, such that:

- Simple and easy to program.
- Easy to make UDP connection (by using Ethernet Shield).
- Has many resources.
- cheaper.

But when we try to send files, we faced a problem with the RAM limitation that was just 2KB that not allowed to send a large amount of data. So, we had to change the Arduino UNO with Arduino Mage and MKR 1010 Arduino with has better RAM as are shown in **TABLE 2** in previse chapter.

- **Arduino UNO:**

Arduino UNO is microcontroller board based on the ATmega328, with 16 MHz clock speed, it does not have the operating system, it can be powered via USB or with an external power supply, that is shown in **FIGURE 15**.

FIGURE 15: ARDUINO UNO

- **Arduino Mega:**

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, that is shown in **FIGURE 16**.

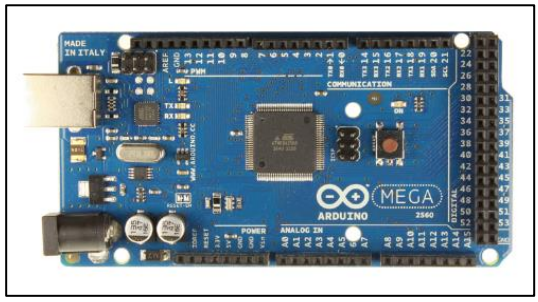


FIGURE 16: ARDUINO MEGA

- **MKR 1010 Arduino:**

The MKR 1010 Arduino is a microcontroller board based on the SAMD21 Cortex-M0+ 32bit Low Power ARM MCU. It has eight digital input/output pins, seven analog inputs, one analog output (of which 12 can be used as PWM outputs), with speed 32.768 kHz (RTC), 48 MHz, a USB connection, a power jack, that is shown in **FIGURE 17**.

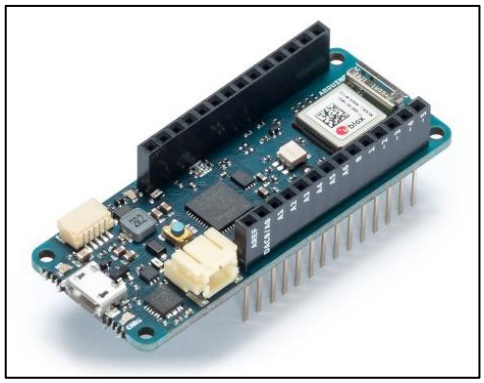


FIGURE 17: MKR 1010 ARDUION

3.2.2 Arduino Ethernet Shield

The Arduino Ethernet Shield V1 allows Arduino to connect to the internet. It provides a network (IP) stack capable of both TCP and UDP. It can support up to four simultaneous socket connections. It operates at 5V (supplied from the Arduino Board), it has an internal 16K buffer, the connection speed is 10/100Mb, and the connection between the Arduino and the Ethernet shield is on the SPI port. It also includes an SD card that can be used to store data. It is compatible with all Arduino/ Genuine Uno boards (e.g., Arduino Uno and Arduino Mega).[19]

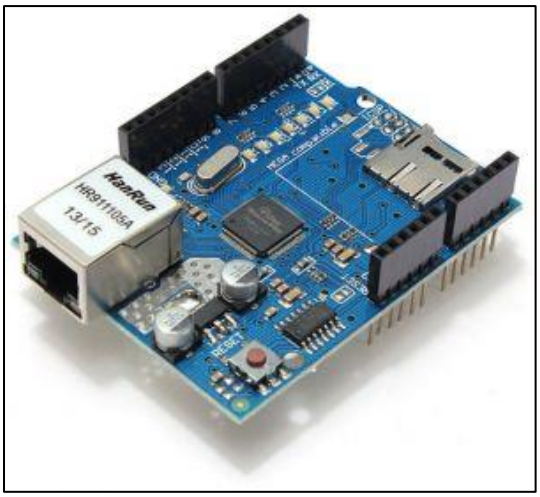


FIGURE 18: ARDUINO ETHERNET SHIELD

3.2.3 The Raspberry Pi

The Raspberry Pi 3 model B+ has Quad Core 1.4GHz Broadcom BCM2837 64bit CPU and 1GB RAM, BCM43438 wireless LAN WiFi built in and Bluetooth 4.2, Low Energy (BLE) on board, 100 Base Ethernet, 40-pin extended GPIO, with 4 USB 2 ports, 4 Pole stereo output and composite video port with Full size HDMI, CSI camera port for connecting a Raspberry Pi camera, DSI display port for connecting a Raspberry Pi touchscreen display and Micro SD port for loading your operating system and storing data. Upgraded switched Micro USB power source up to 2.5A. The **FIGURE 19** is show the components of the Raspberry Pi 3 model B+.[20]

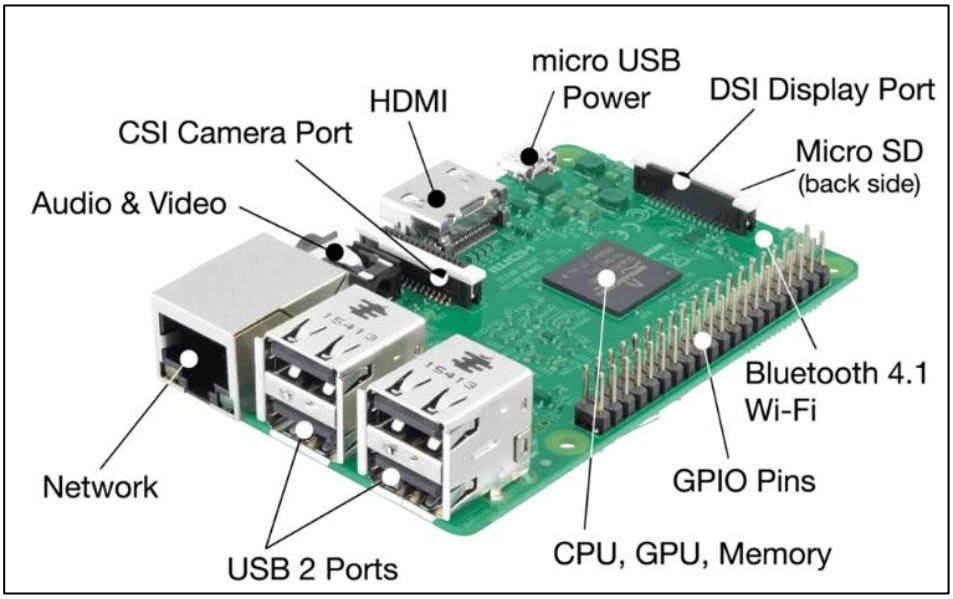


FIGURE 19: THE COMPONENTS OF THE RASPBERRY PI 3 MODEL B+

In **TABLE 3** below, we show the Comparison between the Arduino UNO, MKR1010 Arduino and the Raspberry Pi 3 model B+: [13] [14] [21]

	Arduino UNO	**MKR 1010 Arduino**	**Raspberry Pi 3 Model B+**
Processor	ATmega328P	SAMD21 Cortex-M0+ 32bit Low Power ARM MCU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit
Speed	16 MHz	32.768 kHz (RTC), 48 MHz	1.4 GHz
ROM	32 KB	256KB	SD Card
RAM	2 KB	32KB	1 GB
Operating System	No	No	Yes
Programing Languages	C or C++	C or C++	C, C++, Python, Java
Network	No	Wi-Fi	Gigabit Ethernet, Wireless LAN, Bluetooth 4.2
Number of I/O pins	20	16	40
Power Supply	5V USB or DC jack	5V USB or DC jack	5V USB
Price	22$	31.46$	35$

TABLE 3: THE COMPARISON BETWEEN THE UNO, MKR ARDUINO AND RASPBERRY PI

3.3 The Libraries

In this section, we will take about the libraries we used on our project. First, we will talk about the Arduino Libraries, then the Raspberry Pi Libraries.

3.3.1 Arduino Libraries

In the out project, we used the following libraries:[22]

Library name	**Designed for**	**Some example function**
SPI Library	allows to communicate with SPI devices, with the Arduino as the master device.	begin(), end()
Ethernet Library	work with the Arduino shield, that allow the Arduino board to connect to the internet	begin (), read (), write(), eginPacket(), endPacket(), parsePacket(), available(), stop(), remoteIP(), remotePort()
SD Library	allows reading from and writing to SD cards, supports FAT16 and FAT32 file systems on standard SD, can use with the Arduino Ethernet Shield	begin(), open(), exists(), close(), read(), peek(), position(), seek(), size(), available(), print(), println(), write(), mkdir(), rmdir(), flush(), isFolder()

TABLE 4: THE SOME OF ARDUINO LIBRARIES

3.3.2 Raspberry Pi Libraries

Raspberry Pi Libraries are files which are written using programing language such as C or Python,.. that are can give your Sketches more function. In the out project, we used the following libraries:[23] [24] [25]

Library name	Designed for	Some example function
RPi.GPIO Library	provides access to the General-Purpose Input/output pins	setup(), input(), output(), cleanup()
Socket Library	provides access to the Socket interface. It is available on all modern Unix systems, windows, MacOS, and probably additional platforms.	Sendto(), recvfrom(), close(),bind()
Threading Library	constructs higher-level threading interfaces on top of the lower level thread module.	start(), run() , join()
PyAudio Library	provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms, such as GNU/Linux, Microsoft Windows, and Apple Mac OS X / macOS.	Strem(), open(),
OS Library	provides a portable way of using operating system dependent functionality	ctermid(), getpid(), system()

TABLE 5: THE SOME OF RASPBERRY PI LIBRARIES

In the next chapter, we show how to implement our system with all the details and problems we faced and how we deal with it.

Chapter 4: Implementation

In this chapter, we introduce the implementation of our system. We are going to show our work step by step as we go through this chapter and show the problem we faced and how we solved it. Codes will be attached in Appendix A.
In first phase we used the Arduino then change to Raspberry Pi for some reasons, we well discuses in this chapter.

4.1 Arduino

In this Section, we are going to show how we used the Arduino by different scenario.

4.1.1 Controlling LED ON/OFF using Push Button

By using one UNO Arduino, we control a LED by using two push buttons: one to make the LED ON and then another to make it OFF, that is shown in **FIGURE 20**.

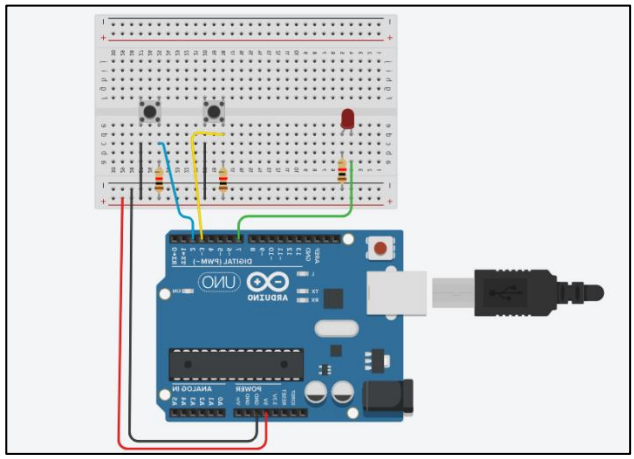


FIGURE 20: LED ON/OFF USING PUSH BUTTON (ARDUINO)

4.1.2 Simplex Communication

Now we have sender and receiver, so we need two UNO Arduinos that is shown in **FIGURE 21**, the sender can control the receiver LED's by sending a UDP message through the network using the Ethernet Shield. Also using the SPI protocol to connect the Ethernet Shield with the Arduino, for that, we should call some libraries as SPI Library and Ethernet Library. The sender can send (ON) or (OFF) message using one of two push buttons. The message is a Char array that will be send as a parameter of Udp.write() function, but it must be called between Udp.beginPacket() function and Udp.endPacket() function. In the receiver side, it should sense if there is an incoming packet by using the Udp.parsePacket() function and return the size of the packet. The receiver will read the packet using Udp.read() function it takes two parameters: packet buffer and buffers size. The packet buffer is a char type.

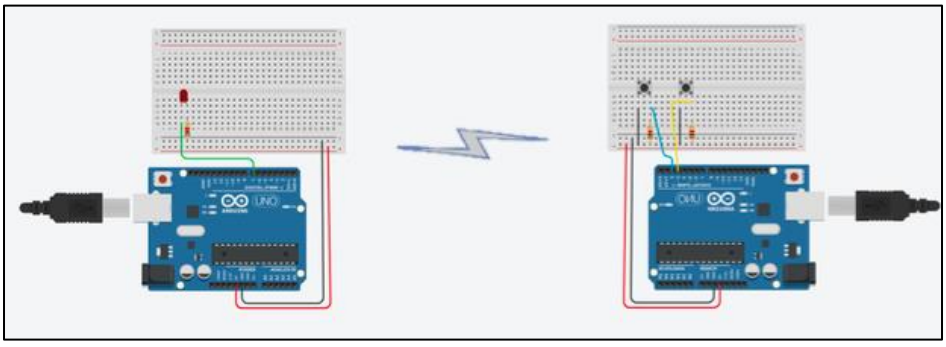


FIGURE 21: SIMPLEX COMMUNICATION BETWEEN TWO UNO ARDUINOS

4.1.3 Full Duplex Communication

In this section, we improve the previous section to make each Arduino control the other LED. By just merge the code of sender and receiver.

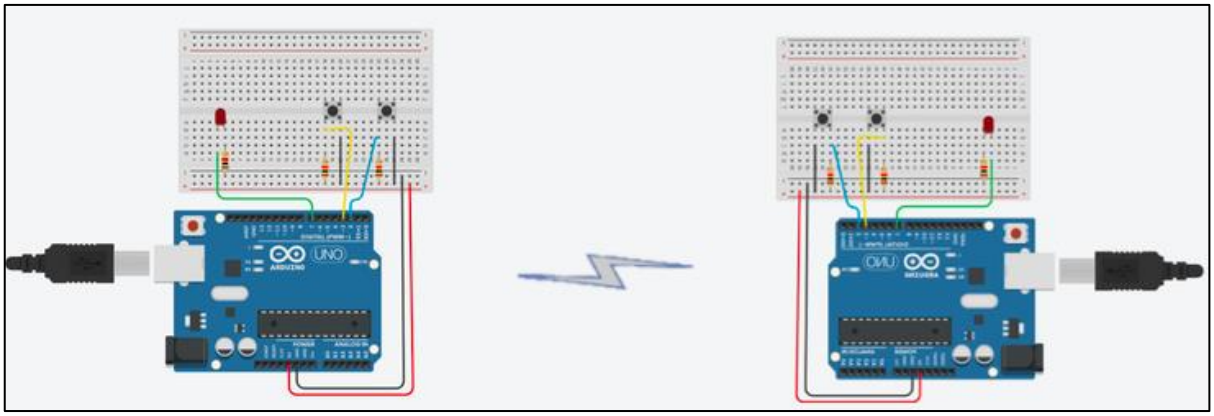


FIGURE 22: FULL DUPLEX COMMUNICATION BETWEEN TWO ARDUINOS

4.1.4 Send text files

In this section we are going to send (.txt and .mp3) files and show the problems we faced and how we solved it, here we replace the Arduino UNO with Arduino mega (sender) and MKR 1010 Arduino (receiver).

Now instead of just controlling LED we send text file (.txt) that is stored in the SD card that is included in the Ethernet Shield, we initialized the SD card using the SD Library. we use the Arduino mega (sender) and MKR 1010 Arduino (receiver) that is show in **FIGURE 23**.

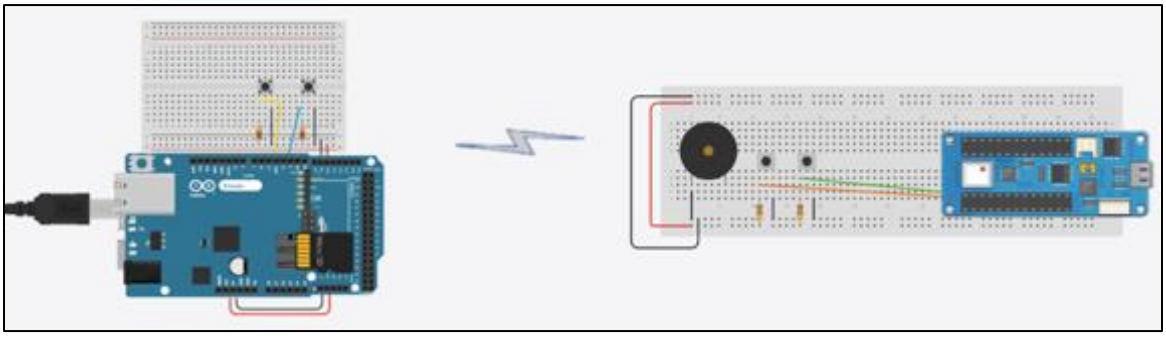


FIGURE 23: THE ARDUINO MEGA (SENDER) AND MKR1010 ARDUINO (RECEIVER)

When the file is sending we got the file with extra size because the last packet is stored with the same size of previse packets (same buffer size) even though it has less size than the buffer size, so we need to create new buffer to send the last packet, but we can not create a buffer without know the size of the buffer, we solve this problem by following these steps:

1. **Connection Request:** the sender sends the first packet that contains only the size of the file by divide the digits of the size of the file and stores each digit in one element of buffer (sizeOfData) with size 8 bytes, that packet is sent once at the beginning.

2. **Connection Response:** when receiving the packet from sender, the receiver start ring by using the buzzer with tone() function, and receiver know the size of file by extract each element from the buffer and calculate the size of file then find the remaining size that is used as size of buffer for the last packet. Then by using two push buttons, the receiver should do one of three cases:

a) Send YES message: by using the Answer push button the receiver sends one-byte packet contains (Y) to the sender to start the connection and send the file.
b) Send NO message: by using the No Answer push button the receiver sends one-byte packet contains (N) to the sender to refusal the connection.
c) Time out: if the time ends before select Answer push button or No Answer push button the receiver sends one-byte packet contains (N) to the sender to refusal the connection.

3. **Send Data:** the sender receives a connection response packet from the receiver then the sender decides if the receiver packet contains (Y) then start sending the file, or if the receiver packet contain (N) then the sender terminates the connection.These steps are shown in **FIGURE 24**.

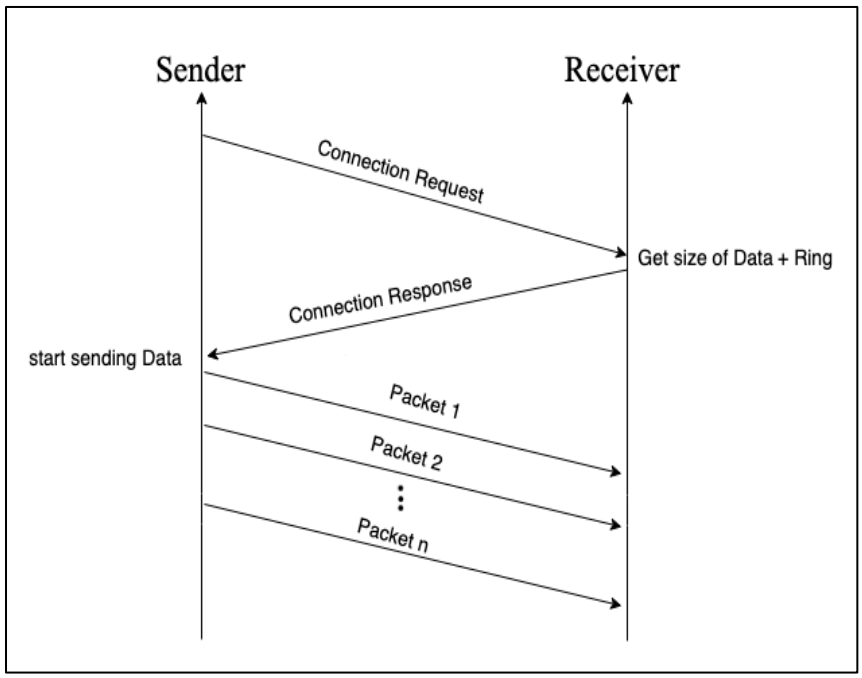


FIGURE 24: THE THREE STEPS TO SOLVE THE PROBLEM

4.1.5 Send audio (mp3) file

Now instead of sending text file (.txt), we send an audio file (.mp3) that is stored in the SD card that is included in the Ethernet Shield.
Our goal is making a call then start streaming voice, but we try to simplify the work by start sending the audio file, see the process of sending voice and the challenges and the requirements to get the knowledge to start do our goal which is voice streaming.

Through sending only audio file using Arduino we found it using it is not the perfect solution due the following problem:

- Lack of resource for streaming library
- Limitation of memory
- Limitation of other resource e.g. (Shields, ADC, DAC, Modules, …).
- Arduino does not support threading.

So, we deiced to change from using Arduino to Raspberry Pi which has built in ethernet, ADC and DAC and provide useful features and libraries as we mention in previos chapter that will help us to achieve our goal.

4.2 The Raspberry Pi

Before using it, we need to setup and learn the basic about that device which was more difficult than Arduino with different concept. Also, we changed the programming language from C/C++ to Python, which was new to us, so we spend time to learn the basic of Python. The reasons for changing are Python more compatible with the raspberry Pi than C/C++ and python was in Raspberry Pi by default, so we don't need to install it.

4.2.1 Setup and Configuration

First, we setup the Raspberry Pi and install the Raspbian operating system which is a free operating system based on Debian (Linux), optimised for the Raspberry Pi hardware, on the SD card whit the minimum recommended card size is 8GB, then we can run and use the Raspberry Pi.

We used Ethernet crossover cables between two Raspberry Pi to make UDP connecaion so, we set a static IP address for each the Raspberry Pi, that it dose not change. To do that we edit the configuration file: /etc/network/interfaces[26]

```
sudo nano /etc/network/interfaces
```

Then, modify the content of the file:

```
interface eth0
static ip_address=192.168.0.100/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```

4.2.2 Controlling LED ON/OFF using Push Button

By using Raspberry Pi, we control a LED by using two push buttons: one to turn ON the LED and other to turn it OFF, that is shown in **FIGURE 25**. We use a RPi.GPIO library so, we can deal with the General-Purpose Input/output pins. We used some functions such as:

- **GPIO.setmode(GPIO.BCM):** one way used to numbering the Input/output pins on a Raspberry Pi, it refers to the channel numbers on the Broadcom SOC.
- **GPIO.setwarnings(False):** used to disable any warning may be accrue when try to configure a script when have more than one script.
- **GPIO.setup(LED,GPIO.OUT):** used to set up every pin using as an input (GPIO.IN) or an output (GPIO.OUT).
- **GPIO.input(pushON):** used to read the value of a GPIO pin, it returns either 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.
- **GPIO.output(LED,1):** used to set the output state of a GPIO pin, state can be 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.
- **GPIO.cleanup():** used to clean up any resource you might have used.

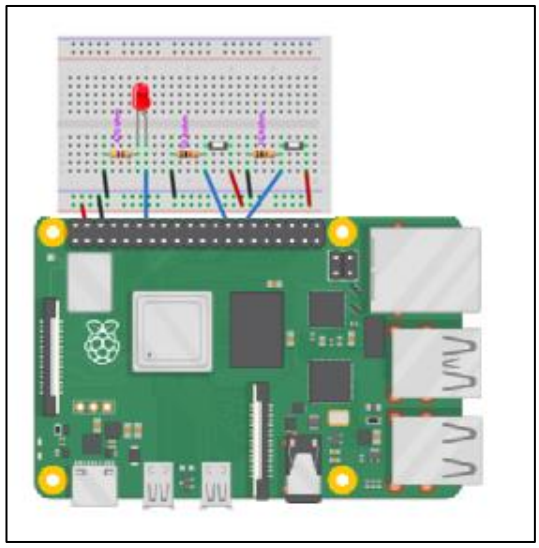


FIGURE 25: CONTROLLING LED ON/OFF (RASPBERRY PI)

4.2.3 UDP Connection

Now we have sender and receiver, so we need to have two Raspberry Pi, as is shown in **FIGURE 26**.

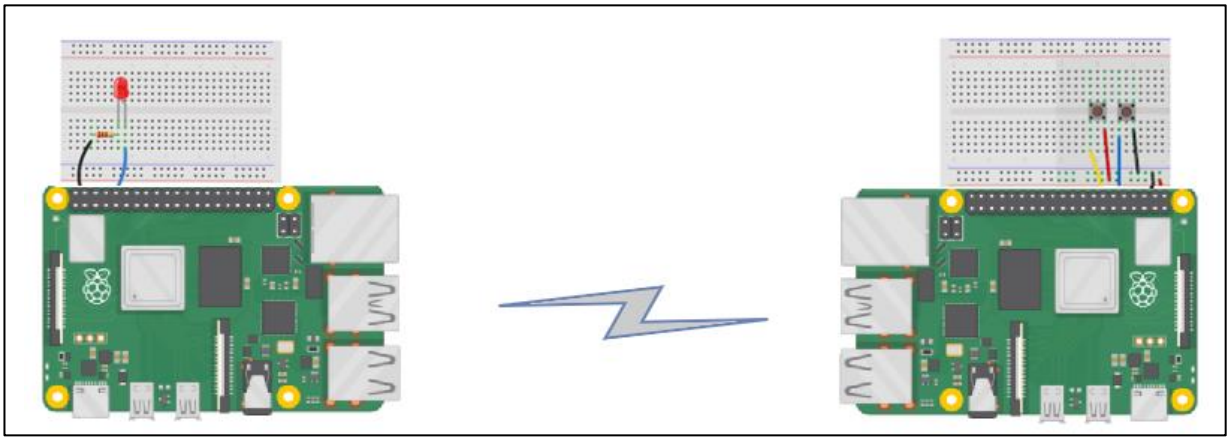


FIGURE 26: UDP CONNECTION BETWEEN TWO RASPBERRY PI

The sender (Client) can control the receiver (Server) LED's by sending a UDP message through the network. for that we should import the **socket** library. The sender can send **TurnOnLed** or **TurnOffLed** message using one of two push buttons. The message will be send using the function **sendto()** as will the receiver will receive the message using the function **recvfrom()**. But before that we need to create the UPD Connection and to do that the Server and Client must create a Socket using **socket.socket(socket.AF_INET, socket.SOCK_DGRAM)** and the Server must bind the Socket Pairs (IP address and port number) using **bind()** function, then the client should close the socket when the connection is done by using **close()** function**.** The **FIGURE 27** Shows the process of creating a UDP connection.

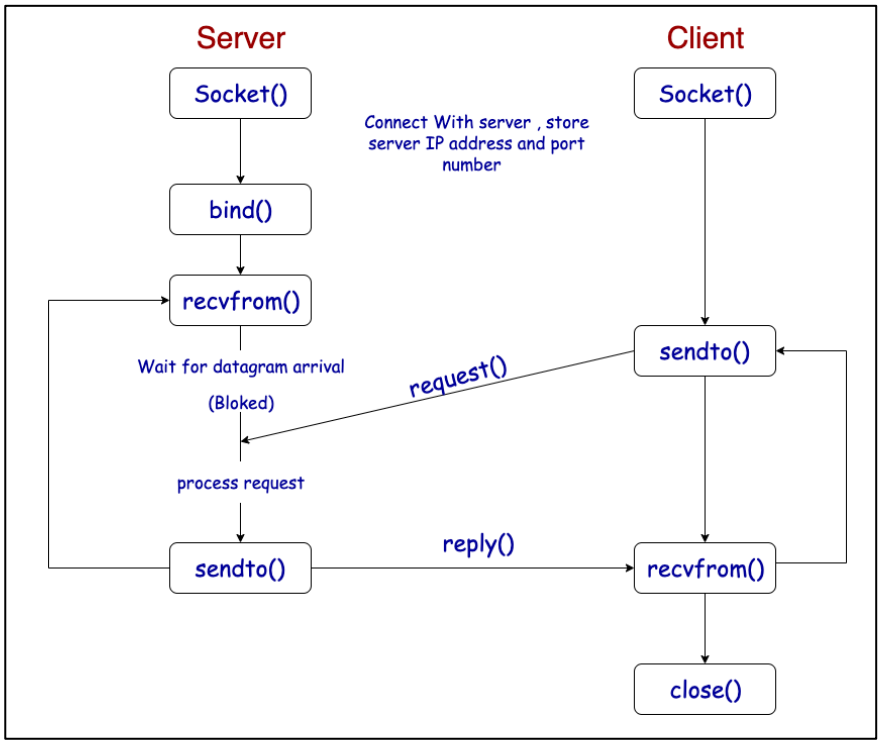


FIGURE 27: THE PROCESS OF CREATING UDP CONNECTION

4.2.4 Full Duplex Communication

In this section, we improve the previous section to make each Raspberry Pi control the other LED. Each node have two push button one to Turn On the other node LED and Buzzer and one to Turn it Off as is shown in **FIGURE 28**.at the being one of the node will try to call the other node by pressing the push button then the other node buzzer will ring and it must decide whether he answer or reject the call, when the call is answered the LED will turn on.

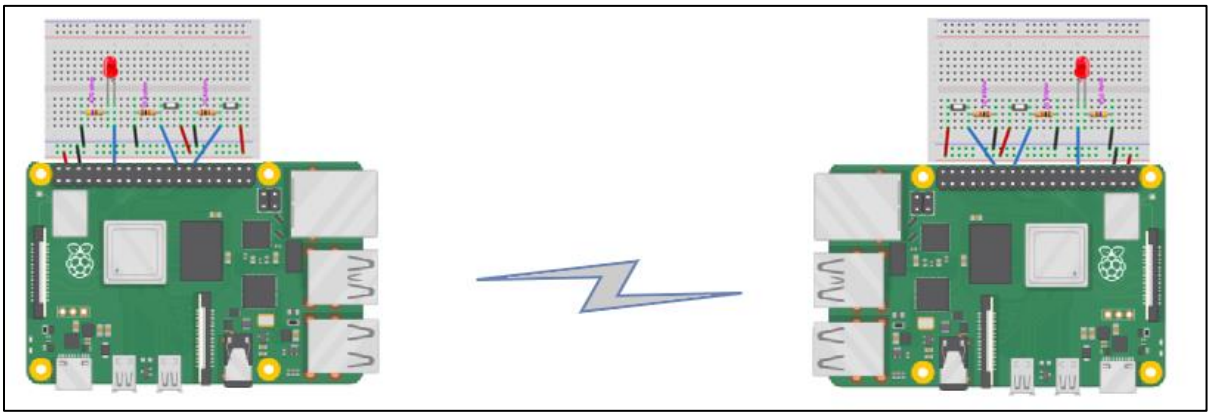


FIGURE 28: FULL DUPLEX COMMUNICATION

The same code will be run in the both node the only different is the Socket Pairs. Few problems showed up:

- **Deadlock**

Listen and send at the same time problem. Due to The function **recvfrom()** which stuck in a loop until it get a message. We had to use the **Threading** library, so the code can have two separate thread one will be for Listing **recvfrom ()** and one for sending **sendto()**.

- **Connection Control**

Who Control the call? To control the call, we had to identify four variables: Flag, Busy, Token and Confirm as is shown in **FIGURE 29**.

- **Flag**: a variable with initial value equal to two. Flag mean there's a connection.
- **Busy**: a variable with initial value equal to zero, which mean the receiver is not busy. When the sender starts the call the receiver Busy value will be equal to one which mean it can't accept any other call.
- **Token**: a variable with initial value equal to one, which mean every node can make the call. Once one node starts the call the other node Token will be equal to zero.
- **Confirm**: a variable with initial value equal to zero, which mean the streaming cannot start now, when the call is accepted by the receiver, the Confirm in bothsides will equal to 1 and the streaming will start.

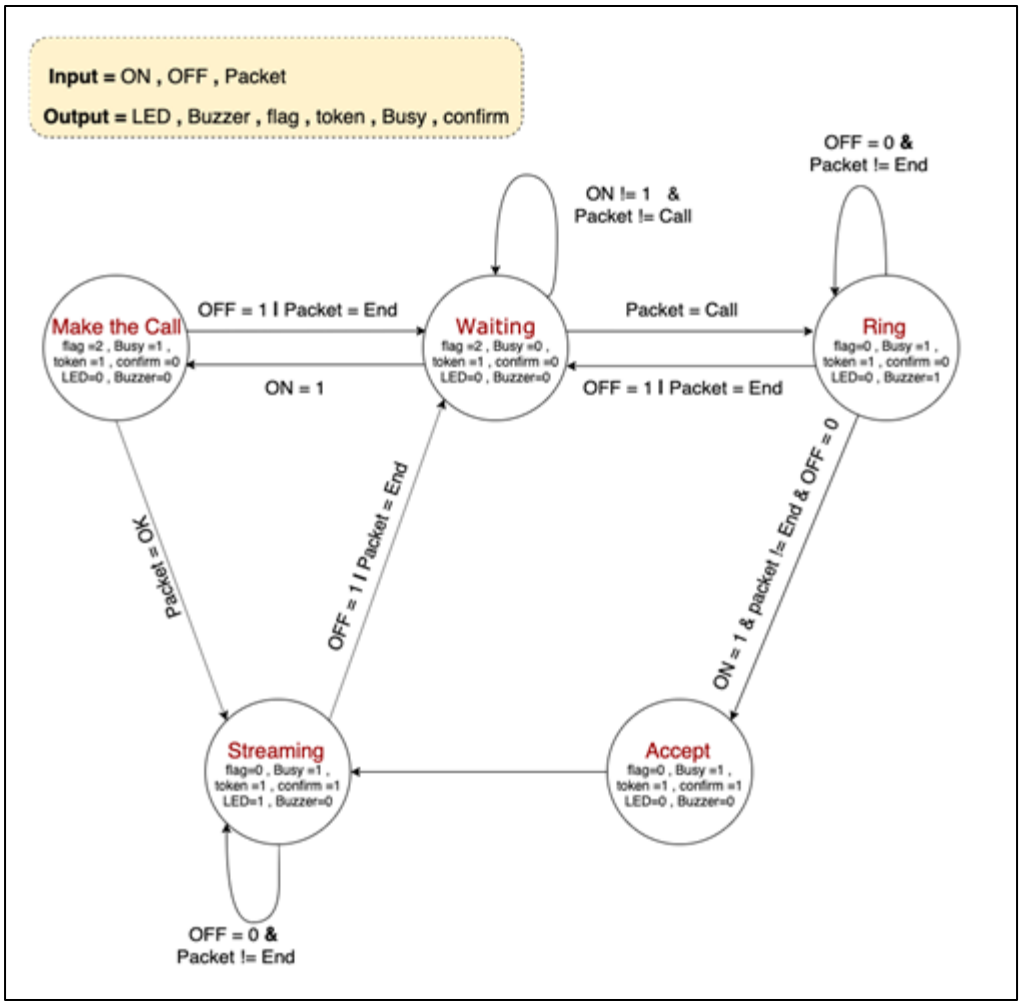


FIGURE 29: THE STATE DIGRAM OF SYSTEM WITH CONNECTION CONTROL

- **Who start the connection problem?**

As we mention before one of the node must act like server and use the **bind()** function to bind Socket Pairs and it must be the one run the code first. So, we need to allow each node to start first. we have two separate thread one will be for Listing **Lis()** and one for sending **SendD()** at each node. The below code solves the problem. it will happen at the begin of the code so each node will get the other node's Socket Pairs. both can start to send to each other no matter who start the code.

```
while ( HandShake == 1):
    def Lis():
        global HandShake, ad, da,cm
        da , ad = servSock.recvfrom(buff)
        cm = da
        ip = ad[0]
        port = ad[1]
        HandShake = 0

    def SendD():
        global HandShake
        msgFromClient       = "Hi Do You Hear Me ?"
        bytesToSend         = str.encode(msgFromClient)
        cliSock.sendto(bytesToSend, (addressRecive) )
        HandShake = 0

    if (First_Thread == 1):
        t1 = threading.Thread(target=SendD)
        t2 = threading.Thread(target=Lis)
        t1.start()
        t2.start()
        t1.join()
        t2.join()
        First_Thread = 0
```

But still there was a another problem which is one of the node (A) will get the other node's (B) socket pairs, but the node (B) won't get node's (A) socket pairs and will be waiting (blocked) in the **Lis()** function to get node's (A) socket pairs. To solve this problem, who first get the information of another node will send another message (Yes, I Do) as shown as is shown in **FIGURE 30**.

```
if (da[:].decode("utf-8") == "Hi Do You Hear Me ?"):
    msgFromClient       = "Yes I DO"
    bytesToSend         = str.encode(msgFromClient)
    cliSock.sendto(bytesToSend, (addressRecive) )
```

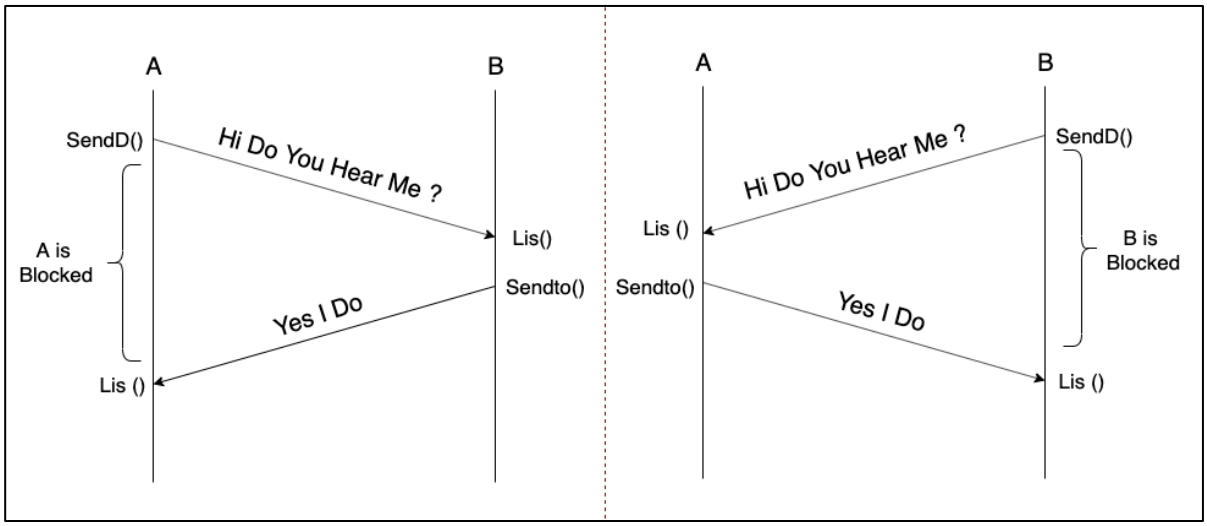


FIGURE 30: SOLVE THE BLOCK PROBLEM

4.2.5 Record and play audio

Before start Streaming through the network, we tried to record an audio and play it on one Raspberry Pi. So, we need microphone and speaker as is shown in **FIGURE 31**.

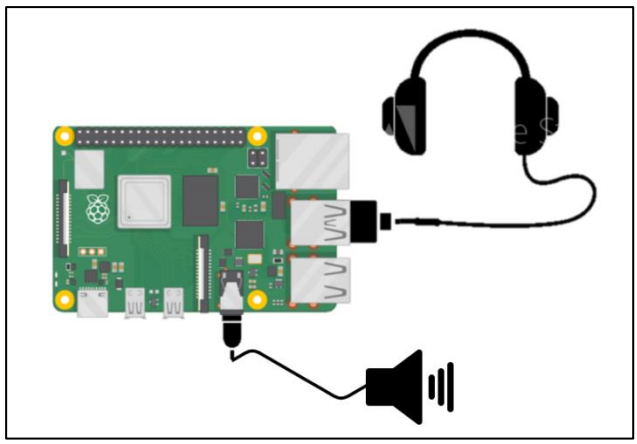


FIGURE 31: RASPBERRY PI WITH MICROPHONE AND SPEAKER

We faced the problem, microphone and speaker are not just plug and use directly we must set up and configure it before use it. We used the USB microphone and speaker in jack port:

- Set up the speaker in jack port

The Raspberry Pi has three audio output modes: **HDMI** and **headphone jack** and **USB**. we can switch between these modes at any time. we switched the audio output to headphone jack:

```
amixer cset numid=3 1
```

The output is being set to 2, which is HDMI. Setting the output to 1 switch to analogue (headphone jack). The default setting is 0 which is automatic.[27]

- Set up the USB microphone

Step1: Open the Terminal of the Raspberry Pi then check and see the USB devices connected by the Raspberry Pi to Check If the Device is Detected by Raspberry:

```
lsusb
```

Step2: to locate the correct recording device (Microphone) card and the device number. use the command to list all the playback devices available on our Raspberry Pi

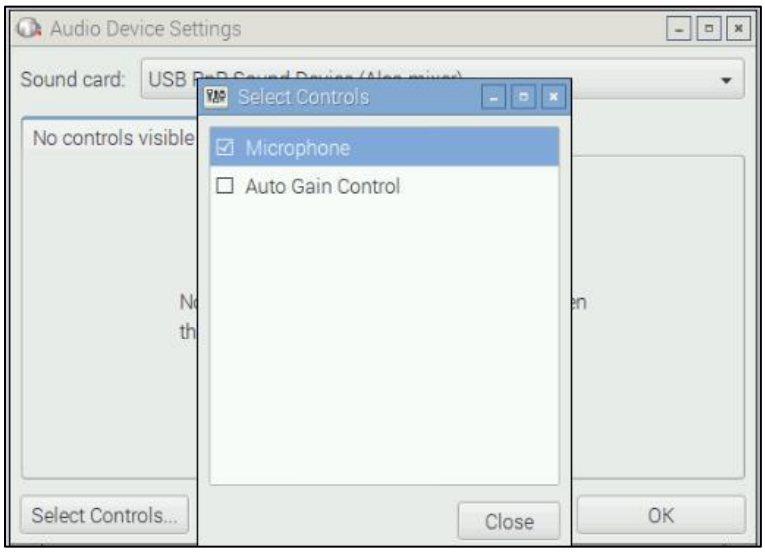
```
arecord -l
```

Step3: to Records testing Record for three seconds and play it:

```
arecord -D plughw:1,0 -d 3 test.wav && aplay test.wav
```

Step4: Adjust the gain of the microphone:

Open Audio Device Settings → Menu in the top left → Preferences → Audio Device Settings.[28]



Also, we used **PyAudio** library which provides Python bindings for PortAudio, initialize and terminate PortAudio, open and close streams, query and inspect the available PortAudio Host APIs, query and inspect the available PortAudio audio devices and can easily use Python to play and record audio on a variety of platforms. Use the package manager to install PyAudio library:[23]

```
sudo apt-get install python-pyaudio python3-pyaudio
```

4.2.6 Streaming

In this section, we start streaming the voice between two Raspberry Pi, sender and receiver each has microphone and speaker as is shown in **FIGURE 32**.

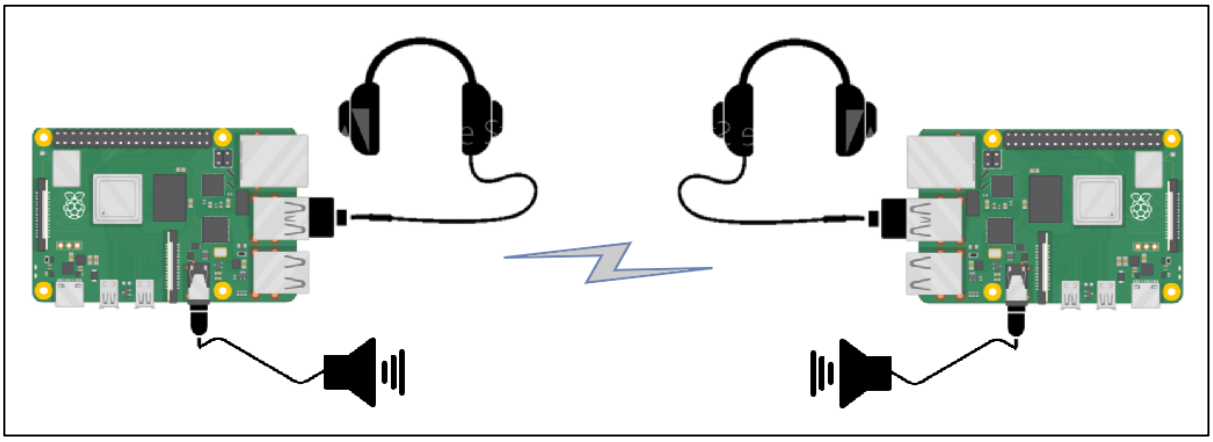


FIGURE 32: STREAMING BETWEEN TWO RASPBERRY PI

First we need to open streaming channel **audio.open()** by using PyAudio library, after we created the socket. Each node has two thread: one thread for **send ()** and the other for **receive ()**.
The sender will talk through microphone, then the PyAudio will call a specified function: **callback ()** which run in separate thread, and it's used to send the audio. The other node will receive it using **recvfrom()** function at receiver's thread, then we write the data by using **streamr.write()** function which actually plays the audio in speaker. Each node can send and receive the audio simultaneous.

4.2.7 Controlling the Streaming

Now for the final result we need to control to the call: make the call, accept or reject and cancel the call, we need to integrate the pervious code (Full Duplex Communication and Streaming). The **FIGURE 33** shows the system.

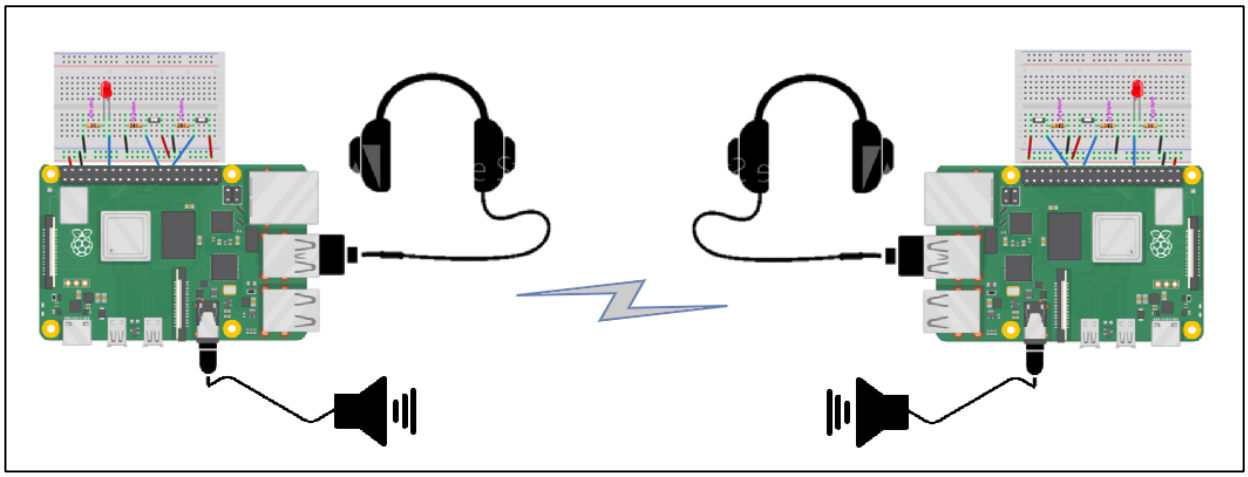


FIGURE 33: CONTROLLING THE STREAMING BETWEEN TWO RASPBERRY PI

We deiced to use the principle divide and conquer instead of integrates the pervious code. we will have two separate code working in series. The reason we did that is to avoid a few problems such as:

- Quality of the audio will degrade.
- Complexity of the code.
- Delay.
- Other problem.

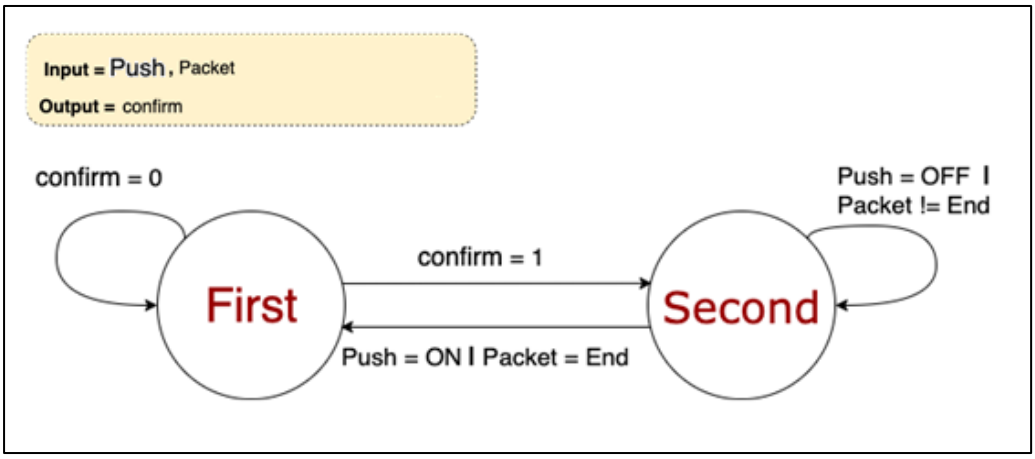


FIGURE 34: HOW THE SYSTEM WORK

The **FIGURE 34** shows how the system work, we have two separate code First and Second:

- **First:** consider only about managing the process of initiation the call (Full Duplex Communication).
- **Second:** consider only about maintaining and ending the call (Streaming).

Only one work at a time and each one should kill the other when it run. To achieve that we had to kill the process for each code once it done it is job. We had to use the OS Library which allow you to control the operating system by writing python code:

- To run a new process, we use:

  os.system("python3 first.py")

- To kill the running process, we use:

  os.system("pkill -f first.py")

so far, we make the streaming voice calls through Ethernet which is main goal of our project, the remaining work is conect the our work with a telephone circuit.

Conclusion

At the end of the project, we were able to stream the voice through the Ethernet which is was an essential part of the project. the remains part was to interface the landline telephone with the microprocessor, but due the challenge we had such as changing from the micro-controller from Arduino to Raspberry pi. We were not able to achieve this part; however, we learn more than we thought we will as: Learning two different programing language (C, Python), working and see the different between micro-controller Arduino and Raspberry pi, learning how to work with threading and multiple connections, learning the basic of the landline telephone and achieving important skills e.g. (Teamwork, effective search, Time management, ...).

Future work

In the future as we passion about what we did, we will keep improving this project and try to interface the landline telephone with the Raspberry pi. Also, we wish that our work can be a reference and starting point for anyone would like to develop or continue this project or any similar project.

Reference

[1] "1 1.POTS= Plain Old Telephone Service 2.PSTN=Public Switched Telephone Network 3.Each pair of communications wires consist of a tip and ring 4.Tip is positive, - ppt download." [Online]. Available: https://slideplayer.com/slide/4492192/. [Accessed: 09-Apr-2019].

[2] "Chapter Eight: The Telephone System - ppt video online download." [Online]. Available: https://slideplayer.com/slide/1484984/. [Accessed: 09-Apr-2019].

[3] "SIMS-201 The Telephone System Wired and Wireless. - ppt video online download." [Online]. Available: https://slideplayer.com/slide/1484980/. [Accessed: 09-Apr-2019].

[4] "Inside a Telephone - Telephone Design | HowStuffWorks." [Online]. Available: https://electronics.howstuffworks.com/telephone1.htm. [Accessed: 09-Apr-2019].

[5] W. Stallings, *Data and computer communications*, 10. ed. Boston, Mass.: Pearson, 2014.

[6] "Computer Network | Ethernet Frame Format - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/computer-network-ethernet-frame-format/. [Accessed: 09-Apr-2019].

[7] "What is Ethernet? - Definition from WhatIs.com," *SearchNetworking*. [Online]. Available: https://searchnetworking.techtarget.com/definition/Ethernet. [Accessed: 09-Apr-2019].

[8] A. S. Tanenbaum and D. Wetherall, *Computer networks*, 5. ed., internat. ed. Boston, Mass.: Pearson, 2011.

[9] B. M. A. M. graduate who brings years of technical experience to articles on SEO, computers, and W. Networking, "Types of Ethernet Cables and What They Do," *Lifewire*. [Online]. Available: https://www.lifewire.com/what-is-an-ethernet-cable-817548. [Accessed: 09-Apr-2019].

[10] "What is UDP/IP," *Cloudflare*. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/. [Accessed: 10-Apr-2019].

[11] "Arduino - Introduction." [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed: 09-Apr-2019].

[12] "Different types of Arduino Boards Used By Engineering Stundents," *ElProCus - Electronic Projects for Engineering Students*, 06-Sep-2016. [Online]. Available: https://www.elprocus.com/different-types-of-arduino-boards/. [Accessed: 10-Apr-2019].

[13] "Arduino MKR WiFi 1010 - MKR Family - Arduino." [Online]. Available: https://store.arduino.cc/usa/mkr-wifi-1010. [Accessed: 09-Apr-2019].

[14] "Arduino Uno Rev3." [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3. [Accessed: 09-Apr-2019].

[15] "Arduino Due." [Online]. Available: https://store.arduino.cc/usa/due. [Accessed: 10-Apr-2019].

[16] "Arduino Mega 2560 Rev3." [Online]. Available: https://store.arduino.cc/usa/mega-2560-r3. [Accessed: 10-Apr-2019].

[17] "What is a Raspberry Pi? | Opensource.com." [Online]. Available: https://opensource.com/resources/raspberry-pi. [Accessed: 09-Dec-2019].

[18] S. C. S. C. has been writing about tech since 2000 H. writing has appeared in publications such as CNN.com, P. C. World, InfoWord, and M. Others, "What Is Streaming and When Do You Use It?," *Lifewire*. [Online]. Available:

https://www.lifewire.com/internet-streaming-how-it-works-1999513. [Accessed: 09-Apr-2019].
[19] "Arduino - ArduinoEthernetShieldV1." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1. [Accessed: 10-Apr-2019].
[20] "Raspberry Pi 3 Model B+ on sale now at $35 - Raspberry Pi." [Online]. Available: https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/. [Accessed: 09-Dec-2019].
[21] "Raspberry Pi 3 Model B," *Raspberry Pi.* .
[22] J. A. Langbridge, *Arduino sketches tools and techniques for programming wizardry*. Indianapolis, IN: John Wiley & Sons, 2015.
[23] "PyAudio: PortAudio v19 Python Bindings." [Online]. Available: http://people.csail.mit.edu/hubert/pyaudio/. [Accessed: 09-Dec-2019].
[24] "raspberry-gpio-python / Wiki / BasicUsage." [Online]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/. [Accessed: 09-Dec-2019].
[25] "socket — Low-level networking interface — Python 3.8.0 documentation." [Online]. Available: https://docs.python.org/3/library/socket.html. [Accessed: 09-Dec-2019].
[26] S. Monk, *Raspberry Pi Cookbook*, Second. 2016.
[27] "Audio configuration - Raspberry Pi Documentation." [Online]. Available: https://www.raspberrypi.org/documentation/configuration/audio-config.md. [Accessed: 09-Dec-2019].
[28] "To use USB mini microphone on Raspbian - Wiki." [Online]. Available: http://wiki.sunfounder.cc/index.php?title=To_use_USB_mini_microphone_on_Raspbian. [Accessed: 09-Dec-2019].

Appendix

Appendix A (The Code)

```python
# 4.2.1 Controlling LED ON/OFF using Push Button
import RPi.GPIO as GPIO

pushON=16
pushOFF=20
LED=23
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(pushOFF, GPIO.IN)#PushOff
GPIO.setup(pushON, GPIO.IN)#PushON

while True:
    if (GPIO.input(pushON)==True):
        GPIO.output(LED,1)
    if (GPIO.input(pushOFF)==True):
        GPIO.output(LED,0)
GPIO.cleanup()
```

```python
#4.2.3 Simplex Communication/Client side
import RPi.GPIO as GPIO
import socket

pushON=16
pushOFF=20
LED=23
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(pushOFF, GPIO.IN)#PushOff
GPIO.setup(pushON, GPIO.IN)#PushON

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

host = '192.168.1.100'
port = 9999

while True:

    if (GPIO.input(pushON)==True):
        data="Y"
        bytesToSend =str.encode(data)
        s.sendto(bytesToSend,(host, port))
    if (GPIO.input(pushOFF)==True):
        data="N"
        bytesToSend =str.encode(data)
        s.sendto(bytesToSend,(host, port))

GPIO.cleanup()
```

```python
#4.2.3 Simplex Communication/Server side

import RPi.GPIO as GPIO
import socket
import os


pushON=16
pushOFF=20
LED=23
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(pushOFF, GPIO.IN)#PushOff
GPIO.setup(pushON, GPIO.IN)#PushON

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
address=("",5001)
sock.bind((address))

while True:
    data,add = sock.recvfrom(1024)
    print(data[:].decode("utf-8"))
    if (data[:].decode("utf-8") == "Y"):
        GPIO.output(LED,1)
    if (data[:].decode("utf-8") == "N"):
        GPIO.output(LED,0)

GPIO.cleanup()
```

```python
# 4.2.4 Full Duplex Communication
import RPi.GPIO as GPIO
import socket
import threading
import socket
pushON=16
pushOFF=20
LED=23
tone = 12
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(tone, GPIO.OUT)#Tone
GPIO.setup(pushOFF, GPIO.IN)#PushOff
GPIO.setup(pushON, GPIO.IN)#PushON
```

```
# Varibals for M/D
global flag, token, Confirm, busy
Confirm = 0
flag = 2
token = 1
busy = 0
buff = 8192
##configuration for Server Socket##
addressSend = ("0.0.0.0" , 8888)
servSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSock.bind(addressSend)
##configuration for Client Socket##
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
addressRecive=('192.168.1.100',9999)
HandShake = 1 # so we do the HandShake only at the begin
First_Thread = 1 # so we only open the thread once .
# here one of the node will get the Message first then will leave the while
while ( HandShake == 1):
    def Lis():
        global HandShake, ad, da,cm
        da , ad = servSock.recvfrom(buff)
        cm = da
        ip = ad[0]
        port = ad[1]
        HandShake = 0

    def SendD():
        global HandShake
        msgFromClient       = "Hi Do You Hear Me ?"
        bytesToSend         = str.encode(msgFromClient)
        cliSock.sendto(bytesToSend, (addressRecive) )
        HandShake = 0
    if (First_Thread == 1):
        t1 = threading.Thread(target=SendD)
        t2 = threading.Thread(target=Lis)
        t1.start()
        t2.start()
        t1.join()
        t2.join()
        First_Thread = 0
if (da[:].decode("utf-8") == "Hi Do You Hear Me ?"):
    msgFromClient       = "Yes I am"
    bytesToSend         = str.encode(msgFromClient)
    cliSock.sendto(bytesToSend, (addressRecive) )
print ("WORK !")
# Here one of the node will start the call by the push buttom.
def MakeTheCall():
    global flag, token, Confirm
    while True:
        if (GPIO.input(pushON)==True and token == 1):
            sdata="CAN I CALL YOU"
            bytesToSend =str.encode(sdata)
            servSock.sendto(bytesToSend,(ad))
        if (GPIO.input(pushOFF)==True and token == 1):
            sdata="I WANT TO FINSH THE CALL"
            bytesToSend =str.encode(sdata)
            servSock.sendto(bytesToSend,(ad))
```

```
def DecisionToMake():
    def AnsOrRjc():
        global flag,busy, token , Confirm
        while True:
            if (flag == 0 ):
                if (GPIO.input(pushON)==True and flag==0):
                        GPIO.output(tone,0)
                        GPIO.output(LED,1)
                        Confirm = 1
                        token = 1
                        ConfirmMessage = "OK"
                        bytesToSend =str.encode(ConfirmMessage)
                        servSock.sendto(bytesToSend,(ad))
                if (GPIO.input(pushOFF)==True and flag==0):
                        GPIO.output(tone,0)
                        GPIO.output(LED,0)
                        flag=2
                        busy = 0
                        token = 1
                        ConfirmMessage = "NO"
                        bytesToSend =str.encode(ConfirmMessage)
                        servSock.sendto(bytesToSend,(ad))
                if(flag==1):
                    GPIO.output(tone,0)
                    GPIO.output(LED,0)
                    flag=2
                    token = 1
    def Ring():
         global flag, busy, token , Confirm
         while True:
            data,add = cliSock.recvfrom(buff)
            if (data[:].decode("utf-8") == "CAN I CALL YOU" and busy == 0):
                flag=0 #accept the call
                GPIO.output(tone,1)
                busy=1
                token = 0
            if(data[:].decode("utf-8") == "I WANT TO FINSH THE CALL"):
                flag=2
                busy = 0
                GPIO.output(LED,0)
                GPIO.output(tone,0)
                token = 1
            if (data[:].decode("utf-8") == "OK"):
                    global Confirm
                    Confirm = 1
                    busy=1
                    flag=0
            if (data[:].decode("utf-8") == "NO"):
                    Confirm = 0
    t5 = threading.Thread(target=Ring)
    t6 = threading.Thread(target=AnsOrRjc)
    t5.start()
    t6.start()
    t5.join()
    t6.join()
```

```
t3 = threading.Thread(target=MakeTheCall)
t4 = threading.Thread(target=DecisionToMake)
t3.start()
t4.start()
t3.join()
t4.join()
```

```
#4.2.5 record audio
import pyaudio
import wave

form_1 = pyaudio.paInt16 # 16-bit resolution
chans = 1 # 1 channel
samp_rate = 44100 # 44.1kHz sampling rate
chunk = 4096 # 2^12 samples for buffer
record_secs = 5 # seconds to record
dev_index = 2 # device index found by p.get_device_info_by_index(ii)
wav_output_filename = 'GP2.wav' # name of .wav file

audio = pyaudio.PyAudio() # create pyaudio instantiation

# create pyaudio stream
stream = audio.open(format = form_1,rate = samp_rate,channels = chans, \
                    input_device_index = dev_index,input = True, \
                    frames_per_buffer=chunk)
print("recording")
frames = []

# loop through stream and append audio chunks to frame array
for ii in range(0,int((samp_rate/chunk)*record_secs)):
    data = stream.read(chunk)
    frames.append(data)

print("finished recording")

# stop the stream, close it, and terminate the pyaudio instantiation
stream.stop_stream()
stream.close()
audio.terminate()

# save the audio frames as .wav file
wavefile = wave.open(wav_output_filename,'wb')
wavefile.setnchannels(chans)
wavefile.setsampwidth(audio.get_sample_size(form_1))
wavefile.setframerate(samp_rate)
wavefile.writeframes(b''.join(frames))
wavefile.close()
```

```python
#4.2.5 Play audio
import time
import RPi.GPIO as GPIO
from pygame import mixer
# Pins definitions
btn_pin = 16
# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)
# Initialize pygame mixer
mixer.init()
# Remember the current and previous button states
current_state = True
prev_state = True
# Load the sounds
sound = mixer.Sound('GP2.wav')
# If button is pushed, light up LED
try:
    while True:
        current_state = GPIO.input(btn_pin)
        if (current_state == False) and (prev_state == True):
            sound.play()
        prev_state = current_state
# When you press ctrl+c, this will be called
finally:
    GPIO.cleanup()
```

```python
#4.2.6 Streaming/Client
import socket
from time import ctime
import threading
import sys
import select
import pyaudio

##configuration to Server Socket##
MYADDR = ("0.0.0.0", 45003)
buff = 8192
servSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSock.bind(MYADDR)
##configuration to Client Socket##
add = '192.168.1.100'
port =12345
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
CHUNK = 4096
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
audio = pyaudio.PyAudio()
```

```
25
26  def callback(in_data, frame_count, time_info, status):
27      for s in read_list[1:]:
28          servSock.sendto(in_data , (x,y))
29      return (None, pyaudio.paContinue)
30
31  stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
32      input=True, frames_per_buffer=CHUNK, stream_callback=callback)
33  streamr = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
34          output=True, frames_per_buffer=CHUNK)
35  read_list = [servSock]
36  print ("Waiting for a connection...")
37  ## handShake to get the ip from each other..
38  # send message so the other side get your IP .
39  msgFromClient       = "Hello UDP Server"
40  bytesToSend         = str.encode(msgFromClient)
41  cliSock.sendto(bytesToSend, (add,port))
42  # get the client IP
43  wait_packet = servSock.recvfrom(buff)
44  Cadd= wait_packet[1]
45  Cm=wait_packet[0]
46  y = Cadd[1]
47  x = Cadd[0]
48  print ("...Connection made with {0}".format(wait_packet[1]))
49
50  def receive():
51      try:
52          while True:
53              rMessage = cliSock.recvfrom(buff)
54
55              streamr.write(rMessage[0])
56              #cliSock.close()
57      except KeyboardInterrupt:
58          pass
59
60
61  def send():
62      try:
63          while True:
64              for s in read_list:
65                  if s is servSock:
66                      read_list.append(Cm)
```

```
                else:
                    data = servSock.recvfrom(buff)

                    if not data:
                        read_list.remove(servSock)
    except KeyboardInterrupt:
        pass
    servSock.close()


t1 = threading.Thread(target=send, name=1)
t2 = threading.Thread(target=receive, name=2)

t1.start()
t2.start()

t1.join()
t2.join()
```

```
#4.2.6 Streaming/Server
import socket
from time import ctime
import threading
import sys
import select
import pyaudio
##configuration to Server Socket##
MYADDR = ("0.0.0.0" , 12345)
buff = 8192
servSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSock.bind(MYADDR)
##configuration to Client Socket##
rADDR = ('192.168.1.100')
rPOTT = 45003
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
CHUNK = 4096
audio = pyaudio.PyAudio()

def callback(in_data, frame_count, time_info, status):
    for s in read_list[1:]:
        servSock.sendto(in_data , (ip,RandmoPortC))
    return (None, pyaudio.paContinue)

stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
            input=True,frames_per_buffer=CHUNK,stream_callback=callback)
streamr = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,output=True
                     frames_per_buffer=CHUNK)
```

```
# stream.start_stream()
read_list = [servSock]

print ("Waiting for a connection...")
## handShake to get the ip from each other..
# get the client IP
wait_packet = servSock.recvfrom(buff)
Cm = wait_packet[0]
Cadd= wait_packet[1]
ip = Cadd[0]
RandmoPortC= Cadd[1]
#print ("...Connection made with {0}".format(wait_packet[1]))
# send message so the other side get your IP .
msgFromClient       = "Hello UDP Server"
bytesToSend         = str.encode(msgFromClient)
cliSock.sendto(bytesToSend, (rADDR ,rPOTT) )

def receive():
    try:
        while True:
            rMessage = cliSock.recvfrom(buff)
            streamr.write(rMessage[0])
    except KeyboardInterrupt:
        pass

def send():

    try:
         while True:
             for s in read_list:
                 if s is servSock:
                     read_list.append(Cm)
                 else:
                     data = servSock.recvfrom(buff)
                     if not data:
                         read_list.remove(servSock)
     except KeyboardInterrupt:
         pass
     servSock.close()

t1 = threading.Thread(target=send )
t2 = threading.Thread(target=receive  )
t1.start()
t2.start()
t1.join()
t2.join()
```

```python
#4.2.7 Controlling the Streaming
import RPi.GPIO as GPIO
import socket
import threading
import os
print(os.getpid())
os.system("pkill -f second.py")
##configuration for the GPIO##
pushON=16
pushOFF=20
LED=23
tone = 12
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(tone, GPIO.OUT)#Tone
GPIO.setup(pushOFF, GPIO.IN)#PushOff
GPIO.setup(pushON, GPIO.IN)#PushON
# Varibals for M/D
global flag, token, Confirm, busy
flag = 2
busy = 0
token = 1
Confirm = 0
buff = 8192
##configuration for Server Socket##
addressSend = ("0.0.0.0" , 2225)
servSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSock.bind(addressSend)
##configuration for Client Socket##
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
addressRecive=('192.168.1.100',1114)

HandShake = 1 # so we do the HandShake only at the begin
First_Thread = 1 # so we only open the thread once .
# here one of the node will get the Message first then will leave the while
#
while ( HandShake == 1):
    def Lis():
        global HandShake, ad, da,cm
        da , ad = servSock.recvfrom(buff)
        cm = da
        ip = ad[0]
        port = ad[1]
        HandShake = 0

    def SendD():
        global HandShake
        msgFromClient       = "Hi Do You Hear Me ?"
        bytesToSend         = str.encode(msgFromClient)
        cliSock.sendto(bytesToSend, (addressRecive) )
        HandShake = 0

```

```
    if (First_Thread == 1):
        t1 = threading.Thread(target=SendD)
        t2 = threading.Thread(target=Lis)
        t1.start()
        t2.start()
        t1.join()
        t2.join()
        First_Thread = 0

if (da[:].decode("utf-8") == "Hi Do You Hear Me ?"):
    msgFromClient       = "Yes I DO"
    bytesToSend         = str.encode(msgFromClient)
    cliSock.sendto(bytesToSend, (addressRecive) )
print ("WORK !")
# Here one of the node will start the call by the push buttom.
def MakeTheCall():
    global flag, token, Confirm , busy
    while True:
        if (GPIO.input(pushON)==True and token == 1):
            sdata="CAN I CALL YOU"
            bytesToSend =str.encode(sdata)
            servSock.sendto(bytesToSend,(ad))
            busy = 1
        if (GPIO.input(pushOFF)==True and token == 1):
            sdata="I WANT TO FINSH THE CALL"
            bytesToSend =str.encode(sdata)
            servSock.sendto(bytesToSend,(ad))
def DecisionToMake():
    def AnsOrRjc():
        global flag,busy, token , Confirm
        while True:
            if (flag == 0 ):
                if (GPIO.input(pushON)==True and flag==0):
                        GPIO.output(tone,0)
                        GPIO.output(LED,1)
                        Confirm = 1
                        ConfirmMessage = "OK"
                        bytesToSend =str.encode(ConfirmMessage)
                        servSock.sendto(bytesToSend,(ad))
                        if(Confirm == 1):
                            GPIO.cleanup()
                            cliSock.close()
                            servSock.close()
                            os.system("python3 sec.py")
                if (GPIO.input(pushOFF)==True and flag==0):
                        GPIO.output(tone,0)
                        GPIO.output(LED,0)
                        flag=2
                        busy = 0
                        token = 1
                        ConfirmMessage = "NO"
                        bytesToSend =str.encode(ConfirmMessage)
                        servSock.sendto(bytesToSend,(ad))
```

```
                if(flag==1):
                    GPIO.output(tone,0)
                    GPIO.output(LED,0)
                    flag=2
                    token = 1
    def Ring():
        global flag, busy, token , Confirm
        while True:
           data,add = cliSock.recvfrom(buff)
           if (data[:].decode("utf-8") == "CAN I CALL YOU" and busy == 0):
               flag=0 #accept the call
               GPIO.output(tone,1)
               busy=1
               token = 0
           if(data[:].decode("utf-8") == "I WANT TO FINSH THE CALL"):
               flag=2
               busy = 0
               GPIO.output(LED,0)
               GPIO.output(tone,0)
               token = 1
           if (data[:].decode("utf-8") == "OK"):

                    Confirm = 1
                    flag=0
                    GPIO.output(LED,1)
                    if(Confirm == 1):
                        GPIO.cleanup()
                        cliSock.close()
                        servSock.close()
                        os.system("python3 sec.py")
           if (data[:].decode("utf-8") == "NO"):
                   Confirm = 0
                   GPIO.output(LED,0)
    t5 = threading.Thread(target=Ring)
    t6 = threading.Thread(target=AnsOrRjc)
    t5.start()
    t6.start()
    t5.join()
    t6.join()
t3 = threading.Thread(target=MakeTheCall)
t4 = threading.Thread(target=DecisionToMake)
t3.start()
t4.start()
t3.join()
t4.join()
```

```
#4.2.7 Controlling the Streaming
import RPi.GPIO as GPIO
import socket
import threading
import pyaudio
import os

print(os.getpid())
os.system("pkill -f first.py")
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
CHUNK = 4096
buff = 8192
```

```
##configuration for the GPIO##
pushOFF=20
LED=23
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)#LED
GPIO.setup(pushOFF, GPIO.IN)#PushOff
##configuration to Server Socket##
addressSend = ("0.0.0.0" , 8888)
servSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSock.bind(addressSend)
##configuration to Client Socket##
cliSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
addressRecive=('192.168.1.100',9999)
##configuration to Server Socket##
addressSendV = ("0.0.0.0" , 6666)
servSockV = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servSockV.bind(addressSendV)
##configuration to Client Socket##
cliSockV = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
addressReciveV=('192.168.1.100',7777)
audio = pyaudio.PyAudio()
def callback(in_data, frame_count, time_info, status):
    for s in read_list[1:]:
        servSockV.sendto(in_data , (adV))
    return (None, pyaudio.paContinue)
stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
                input=True, frames_per_buffer=CHUNK, stream_callback=callback)
streamr = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,
                    output=True, frames_per_buffer=CHUNK)
# stream.start_stream()
read_list = [servSockV]
global HandShake ,First_Thread,Secand_Thread
HandShake = 1 # so we do the inination only at the begin
First_Thread = 1 # so we only open the thread once .
Secand_Thread = 1 # so we only open the thread once .
# here one of the node will get the Message first then will leave the while
while ( HandShake == 1):
    def Lis():
        global HandShake
        global ad,da , cm , cmV,adV
        da , ad = servSock.recvfrom(buff)
        daV , adV = servSockV.recvfrom(buff)
        cmV = daV
        cm = da
        ip = ad[0]
        port = ad[1]
        print(ad)
        print(adV)
        print(da[:].decode("utf-8"))
        HandShake = 0
    def SendD():
```

```
        global HandShake
        msgFromClient       = "Hi Do You Hear Me ?"
        bytesToSend         = str.encode(msgFromClient)
        cliSock.sendto(bytesToSend, (addressRecive) )
        cliSockV.sendto(bytesToSend, (addressReciveV) )
        HandShake = 0
        print ("Waiting for a connection...")


    if (First_Thread == 1):
        t1 = threading.Thread(target=SendD)
        t2 = threading.Thread(target=Lis)
        t1.start()
        t2.start()
        t1.join()
        t2.join()
        First_Thread = 0

if (da[:].decode("utf-8") == "Hi Do You Hear Me ?"):
    msgFromClient       = "Yes I DO"
    bytesToSend         = str.encode(msgFromClient)
    cliSock.sendto(bytesToSend, (addressRecive) )
    cliSockV.sendto(bytesToSend, (addressReciveV) )

print ("WORK aduio !")

import RPi.GPIO as GPIO
def receive():
    try:
        while True:
            rMessage = cliSockV.recvfrom(buff)
            streamr.write(rMessage[0])

            if (GPIO.input(pushOFF)==True ):
                GPIO.output(LED,0)
                msgFromClient       = "END THE CALL"
                bytesToSend         = str.encode(msgFromClient)
                cliSock.sendto(bytesToSend, (addressRecive) )
                GPIO.cleanup()
                os.system("python3 first.py")

    except KeyboardInterrupt:
        pass
            #cliSock.close()

def send():

    try:
        while True:


            for s in read_list:
                if s is servSockV:
                    read_list.append(cmV)
```

```
121
122                 else:
123                     data = servSockV.recvfrom(buff)
124
125                     if not data:
126                         read_list.remove(servSockV)
127     except KeyboardInterrupt:
128         pass
129     servSockV.close()
130
131 def wait_for_end():
132     da , ad = servSock.recvfrom(buff)
133      if (da[:].decode("utf-8") == "END THE CALL"):
134          GPIO.output(LED,0)
135          GPIO.cleanup()
136          os.system("python3 first.py")
137
138
139
140
141  t3 = threading.Thread(target=send)
142  t4 = threading.Thread(target=receive)
143  t5 = threading.Thread(target=hopework)
144
145  t3.start()
146  t4.start()
147  t5.start()
148  t3.join()
149  t4.join()
150  t5.join()
```

Appendix B (The Survey)

Landline Calls

قياس مدى أهمية الهاتف الأرضي بالوقت الحالي (نتيجة هذا الاستبيان ستكون مضمنة في التقرير النهائي نرجوا تحري الدقة في الاجابة)

*مطلوب

هل مازلت تستخدم الهاتف الأرضي بالمنزل ؟ *

- نعم
- لا

التالي

Landline Calls

*مطلوب

Landline Calls

قياس مدى أهمية الهاتف الأرضي بالوقت الحالي

ماهي العوامل التي تجعل الهاتف موجود بالمنزل ؟ *

- خدمة الانترنت مرتبطه بالهاتف (DSL & fiber optic)
- تتواصل الجهات الحكومية والشركات من خلال الهاتف
- يقتصر التواصل مع بعض الاقارب من خلال الهاتف فقط
- التواصل مع الموجودين في البيت من خلال الهاتف (عن طريق التحويلة)
- أخرى:

رجوع إرسال

Landline Calls

*مطلوب

Landline Calls

قياس مدى أهمية الهاتف الأرضي بالوقت الحالي

ما هي أبرز الأسباب اللي تجعل استخدام الهاتف الأرضي قليلة ؟ *

- ما فيه مميزات كثير مثل الجوال
- مرتبط بمكان محدد (ما أقدر أنقله بأكثر من مكان)
- قل انتشاره بين الناس
- تقنية قديمة
- أخرى:

رجوع التالي

- **The result of the Survey:**