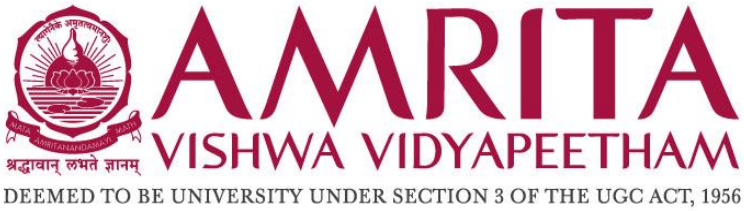


Bachelor of Technology (B. Tech.)

Computer and Communication Engineering (CCE)

Amrita School of Engineering, Coimbatore – 641 112

Lab Record

19CCE304 – Computer Networks

Fifth Semester

Name: Santosh

Roll Number: CB.EN.U4CCE20053

Academic Year – 2022-23

Faculty In-charge – Dr Gandhiraj R

TEAM MEMBERS

1. **B Ambareesh – [CB.EN.U4CCE20006]**
2. **Narendran S – [CB.EN.U4CCE20036]**
3. **Narun T – [CB.EN.U4CCE20037]**
4. **Pabbathi Greeshma – [CB.EN.U4CCE20040]**
5. **Santosh – [CB.EN.U4CCE20053]**

TABLE OF CONTENTS

OneDrive Link: 19CCE304_CN_Experiments_053

EXPERIMENT 1 - SIMPLE CLIENT-SERVER APPLICATION DEVELOPMENT USING PYTHON

AIM:

Develop a simple client-server application using Python language protocols to be verified with User Datagram Protocol (UDP) and Transmission Control Protocol (TCP).

SOFTWARE:

1. Oracle VM VirtualBox 6.1.38, Oracle Corporation
2. Ubuntu 22.04 (64-bit) Operating System
3. Microsoft Visual Studio Code, Microsoft Corporation
4. Python 3.10.7 (64-bit), Python Software Foundation

THEORY:

I. Key Terminologies:

1. Server – It is simply a computer that can be used to share resources with other computers based on request.
2. Client – It is the other computer(s) that request resources from the server.
3. Client-Server network: Clients can access resources and services from a central computer using a client-server network, which can be either a local area network (LAN) or a wide-area network (WAN), like the Internet.
4. Protocol – An established set of guidelines that govern how data is transferred between various devices connected to the same network is known as a network protocol. In essence, it enables interconnected devices to interact with one another despite any variations in their internal workings, organisational structures, or aesthetics.
5. IP Address – A device on the internet or a local network can be identified by its IP address, which is a special address. The rules defining the format of data delivered over the internet or a local network are known as "Internet Protocol," or IP.
6. Packet Fragmentation – For the resulting fragmentation to fit across a link with a smaller maximum transmission unit (MTU) than the original packet size, IP fragmentation is a technique that divides packets into smaller units of the Internet Protocol. The receiving host puts the pieces back together. Both the specifics of the technique for fragmentation and the broader architectural strategy for fragmentation differ between IPv4 and IPv6.
7. Socket – One endpoint of a two-way communication channel between two network-running programmes is a socket. For the TCP layer to recognise the application that data is intended to be transferred to, a socket is tied to a port number. A port number and an IP address make up an endpoint.

II. What is User Datagram Protocol (UDP)?

UDP decided to use the rudimentary method of marking each UDP packet with a pair of unsigned 16-bit port numbers in the range of 0 to 65,536 to distinguish between distinct talks.

The application at the destination IP address to which the communication should be sent is specified by the destination port, whereas the source port identifies the specific process or programme that submitted the packet from the source system.

III. How does UDP work?

A straightforward stateless protocol is UDP. Although this simplicity is excellent for straightforward applications, dependability issues arise. Delivering UDP packets across Network Address Translator (NAT) devices in particular is challenging. Let's review what NAT devices truly do so that you can see why.

As the Internet grew in use, it became impracticable to assign each host a different IP address. The NAT protocol was developed to address this issue by enabling a device on the edge of a network to advertise a publicly available, globally unique IP address for other hosts to connect to, and mapping that to an internal IP address that is distinct from the internal network. This makes it possible for numerous networks to share the same local IP address space.

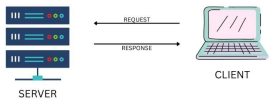


Figure 1 – Block Diagram of User Datagram Protocol (UDP)

NAT devices function by adhering to TCP's connection (and teardown) protocol and keeping a routing table for TCP-based connections. A new NAT routing entry is established when an application creates a TCP connection, and the NAT routing entry can be erased once the TCP connection has been terminated.

Since UDP has no connection protocol and no state, it is very difficult for NAT devices to maintain an accurate routing table for UDP datagrams. Any NAT device used between two hosts must be explicitly configured to port forward UDP traffic to connect to devices inside the network.

IV. Applications of UDP:

1. Used for straightforward request-response communication when data size is smaller and flow and error control are less of a concern. Because UDP provides packet switching, it is an appropriate protocol for multicasting.
2. Some routing update protocols, such as RIP, employ UDP (Routing Information Protocol). Usually applied to real-time applications that can't stand inconsistent delays in a message's different parts.
3. Following implementations use UDP as a transport layer protocol -
    - NTP (Network Time Protocol)

- DNS (Domain Name Service)
- BOOTP, DHCP
- NNP (Network News Protocol)
- Quote of the Day Protocol
- TFTP, RTSP, RIP.

4. The application layer can do some of the tasks through UDP –
    - Trace Route
    - Record Route
    - Timestamp
5. UDP takes a datagram from Network Layer, attaches its header, and sends it to the user. So, it works fast.
6. UDP is a null protocol if you remove the checksum field. For example,
    - Reduce the requirement for computer resources.
    - When using the Multicast or Broadcast to transfer.
    - The transmission of Real-time packets, mainly in multimedia applications.

V. Advantages of UDP:

1. It employs a tiny packet size and a tiny header (8 bytes). The UDP protocol requires less memory and less processing time since there are fewer bytes in the overhead.
2. It does not need a connection to be made and kept up. The absence of an acknowledgement field in UDP speeds up communication because there is no need to wait for an acknowledgement (ACK) or store data in memory until it is received.
3. For error detection, it employs a checksum on each packet. It can be applied to situations in which only one packet of data needs to be transmitted between the hosts.

VI. Disadvantages of UDP:

1. It is an unreliable, connectionless transport protocol. No windowing or mechanism exists to guarantee that data is received in the same sequence as it was transmitted.
2. It makes no use of error checking. As a result, if UDP finds a mistake in the received packet, it discreetly discards it.
3. There is no traffic management. As a result, congestion can be caused by several users sending enormous amounts of data via UDP, and nothing can be done to stop it.
4. Error recovery is only dealt with by the application layer. Application developers can therefore just ask the user to send the message again.
5. Routers may use UDP carelessly. They frequently discard UDP packets before TCP packets and do not retransmit a UDP datagram following a collision. There is no flow control and no acknowledgement for received data.

VII. What is Transmission Control Protocol (TCP)?

Transmission Control Protocol, or TCP, is a communications standard that enables computer hardware and software to exchange messages over a network. It is made to send packets across the internet and make sure that data and messages are successfully sent through networks.

TCP is one of the fundamental standards that the Internet Engineering Task Force (IETF) has specified as the guidelines for the internet (IETF). It provides end-to-end data delivery and is one of the most widely utilised protocols in digital network communications. Data is organised by TCP before being sent between a server and a client. It ensures the accuracy of the data transmitted via a network, before data transmission.

VIII. How does TCP work?

The TCP/IP paradigm divides the data into little bundles and then reassembles the bundles into the original message on the other end to ensure at each message reaches its target place intact. Instead of transmitting everything at once, delivering the information in smaller bundles of information makes it easier to retain efficiency.

When a message is divided into bundles, if one route is congested but the destination is the same, the bundles may go along other routes. For instance, when a user requests a web page on the internet, the server processes the request and delivers the user a sense in the form of an HTML page. The HTTP Protocol is the one that the server uses.

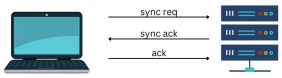


Figure 1 – Block Diagram of Transmission Control Protocol (TCP)

The TCP layer is then asked by HTTP to establish the necessary connection and transfer the HTML file. The data is now divided into individual packets by the TCP, which then sends them to the IP layer. The packets are subsequently transmitted via several pathways to their destination. When the transmission is complete and all packets have been received, the TCP layer in the user's system acknowledges.

IX. Applications of TCP:

1. Segment Numbering System - TCP keeps track of the segments being transmitted or received by allocating numbers to every one of them, which is one of its most noticeable features. Data bytes that need to be sent are given a specific byte number, and segments are given sequence numbers. Segments that have been received are given acknowledgement numbers.
2. Flow Control – The rate at which a sender delivers data is constrained by flow control. To guarantee dependable delivery, this is done. The sender is constantly informed by the receiver how much data can be received (using a sliding window)
3. Error Control – To ensure dependable data transport, TCP uses an error control technique. Byte-Oriented Error Control and Segment-Based Error Detection

and Control include, Corrupted Segment & Lost Segment Management, Out-of-Order Segments, Duplicate Segments, etc.

4. Congestion Control – TCP takes into account the level of congestion in the network. Congestion level is determined by the amount of data sent by a sender. It assigns an IP address to each computer on the network and a domain name to each site, making each device site distinct over the network. Open Protocol, is not owned by any organisation or individual.

X. Advantages of TCP:

1. This protocol is dependent; it offers an error checking system and a recovery mechanism; it controls flow; it ensures that data reaches its destination in the precise sequence in which it was sent and it proves flow control.
2. TCP is designed for wide-area networks, therefore tiny networks with limited resources may have trouble with their scale. Additionally, because TCP runs on multiple layers, the network's speed may be slowed down. This means that it can only represent the TCP/IP suite of protocols.
3. Its nature is not generic. This means that it can only represent the TCP/IP suite of protocols. For instance, a Bluetooth connection is not compatible. Since their creation, around 30 years ago, there have been no changes.

XI. Disadvantages of TCP:

1. TCP is a complicated model to set up and manage and control. The shallow/overhead of TCP is higher than IPX. Replacing protocol in TCP is not easy.
2. It does not separate the concept of service, interface, and protocols. So, it is not suitable to describe new technologies in the new network. In this model, the transport layer does not guarantee the delivery of packets.
3. It has no clear operations from its services, interfaces, and protocols. It is not generic in nature. So, it fails or loses to represent any protocol stack other than the TCP/IP suite.
4. It was originally designed and implemented for a wide area network. It is not optimized for small networks like LAN and PAN. TCP can't be used for broadcast and multicast connections.
5. All the work is being done by your OS, so if there are bugs in your OS, then you will face many problems like problems surfing and downloading from the net.

SOURCE CODE:

UDP Server

```
import socket # Low-level networking interface

if __name__ == "__main__":

    localIP = "127.0.0.1" # IP address of the local host
    localPort = 20001 # Port Number
    bufferSize  = 1024 # Byte Size

    UDPServerSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM) # Create a datagram socket, belonging to IPv4 family
and following connection-less UDP protocol
    UDPServerSocket.bind((localIP, localPort)) # Bind the socket with IP
address and port number
    print("\nUDP Server Up & Listening...")

    # Listen for incoming datagrams:-
    while(True):
        data, address = UDPServerSocket.recvfrom(bufferSize) # Receive the
address to send the data back
        data = data.decode("UTF-8")
        print(f"Client: {data}")

        data = data.upper()
        data = data.encode("UTF-8")
        UDPServerSocket.sendto(data, address) # Sending a reply to client
```

UDP Client

```
import socket # Low-level networking interface

if __name__ == "__main__":

    localIP = "127.0.0.1" # IP address of the local host
    localPort = 20001 # Port Number
    serverAddressPort = (localIP, localPort)
    bufferSize = 1024 # Byte Size

    UDPClientSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM) # Create a UDP datagram socket at the client side,
belonging to IPv4 family and following connection-less UDP protocol

    while(True):
        data = input("\nEnter data: ")
        data = data.encode("UTF-8") # Encode the string
```

```
        UDPClientSocket.sendto(data, serverAddressPort) # Sending a reply to
server

        data, address = UDPClientSocket.recvfrom(bufferSize) # Receive the
address to send the data back
        data = data.decode("UTF-8") # Decode the bytes
        print(f"Server: {data}")
```

TCP Server

```
import socketserver # Simplifies the task of writing network servers
bufferSize = 1024 # Byte Size

# The TCP Server class for demonstration:-
class Handler_TCPServer(socketserver.BaseRequestHandler):

    # We need to implement the Handle method to exchange data with TCP
client:-
    def handle(self):

        self.data = self.request.recv(bufferSize).strip() # TCP socket
connected to the client for receiving data from the socket
        print("\n{} sent: ".format(self.client_address[0]) + str(self.data))
        self.request.sendall("Acknowledgment from TCP Server!\n".encode()) #
Send back ACK for data arrival confirmation

if __name__ == "__main__":

    HOST, PORT = "127.0.0.1", 9999 # IP Address of the local host and port
number
    tcp_server = socketserver.TCPServer((HOST, PORT), Handler_TCPServer) #
Initialize the TCP server object and bind it to the localhost on the 9999 port
    tcp_server.serve_forever() # Activate the TCP server

    # To abort the TCP server, press Ctrl + C.
```

TCP Client

```
import socket # Low-level networking interface

host_ip, server_port = "127.0.0.1", 9999 # IP Address of the local host and
port number
bufferSize = 1024 # Byte Size
data = input("\nEnter data: ")
tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a UDP
datagram socket at the client side, belonging to IPv4 family and following TCP
protocol
```

```
try: # Some Code...

    # Establish connection to TCP server and exchange data:-
    tcp_client.connect((host_ip, server_port)) # Connect to a remote socket
address
    tcp_client.sendall(data.encode()) # Send data to the socket
    received = tcp_client.recv(bufferSize) # Receive data from the socket

finally: # Some Code... (ALWAYS EXECUTED)
    tcp_client.close() # Mark the socket closed

print ("Bytes Sent: {}".format(data))
print ("Bytes Received: {}\n".format(received.decode()))
```

SCREENSHOTS OF INPUTS AND OUTPUTS:

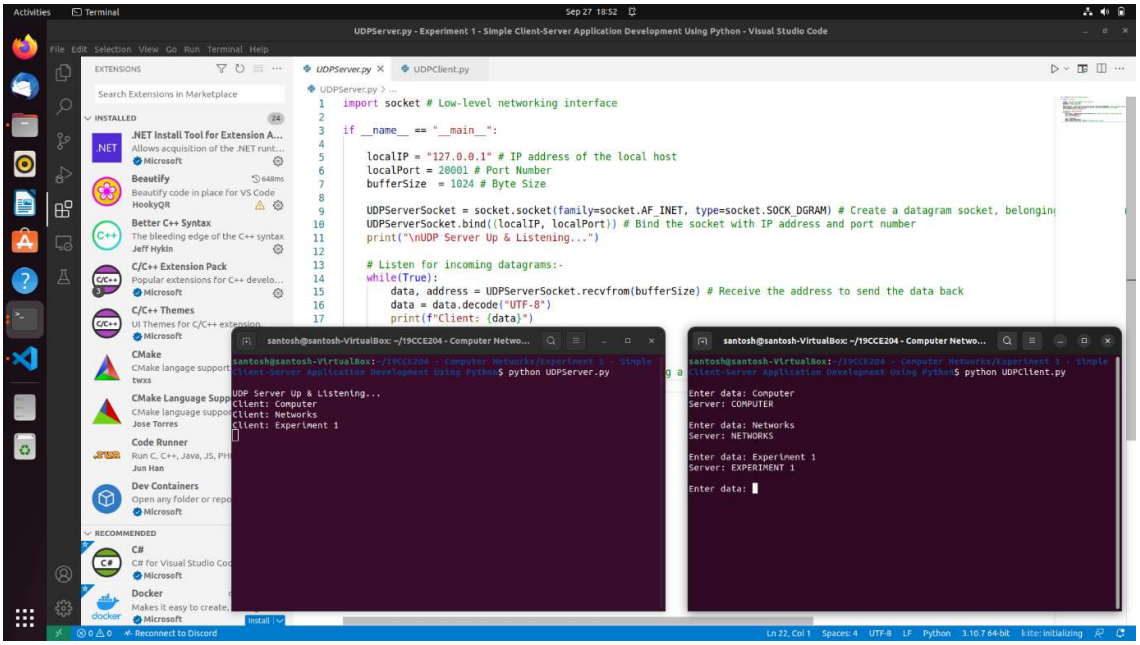


Figure 1 - User Datagram Protocol (UDP)

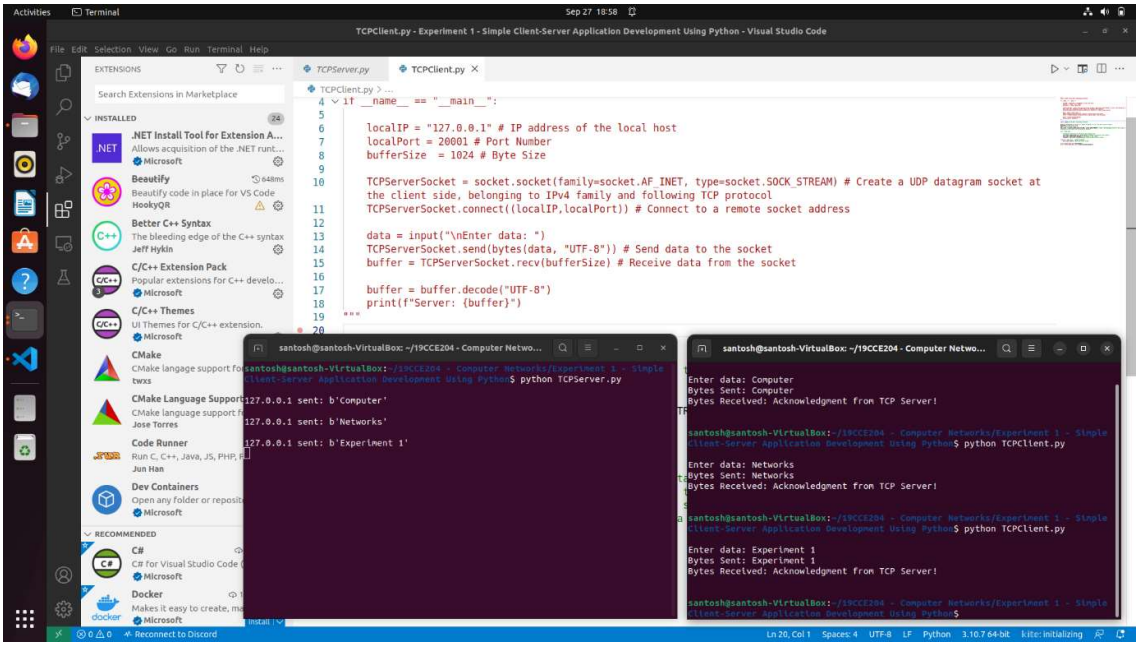


Figure 2 - Transmission Control Protocol (TCP)

CONCLUSION:

Thus, developed a simple client-server application using Python language protocols to be verified with UDP and TCP and all simulation results were verified successfully.

REFERENCES:

1. Foundations of Python Network Programming, Brandon Rhodes, John Goerzen, Apress Berkeley, CA, Third Edition, 2014 (Softcover ISBN: 978-1-4302-5854-4, eBook ISBN: 978-1-4302-5855-1)
2. Socket Module Python Documentation – https://docs.python.org/3/library/socket.html
3. Socket Server Module Python Documentation – https://docs.python.org/3/library/socketserver.html
4. UDP - Client and Server Example Programs In Python – https://pythontic.com/modules/socket/udp-client-server-example

CONTRIBUTION:

1. Narendran S [CB.EN.U4CCE20036] – Theory Documentation + Code Review
2. T Narun [CB.EN.U4CCE20037] – Topic Research + Code Debugging/Testing
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Code Work for UDP
4. Santosh [CB.EN.U4CCE20053] – Code Work for TCP

EXPERIMENT 2 – BUILDING A WEB SERVER - A HTTP CLIENT

Aim:

To develop a web server and client application using HTTP.

Software:

1. Oracle VM VirtualBox 6.1.38, Oracle Corporation
2. Ubuntu 22.04 (64-bit) Operating System
3. Microsoft Visual Studio Code, Microsoft Corporation
4. Python 3.10.7 (64-bit), Python Software Foundation

Theory:

I. HTTP:

Hyper Text Transfer Protocol, or HTTP, is a set of guidelines that servers must adhere to send all types of files, including images, text, audio, and video, across the internet (www). Clients and servers make up the internet. Consequently, if you visit the internet via a web client like Google Chrome, Mozilla, Internet Explorer, etc.

You request a web server whenever you type in the name of any website you choose to visit. Assume that when you type amazon.in, your browser—the web client—is asking the web server and frequently a large number of pages from it. It might be JSON, HTML, CSS, pictures, or video.

The fundamental interaction between you and the server is one of request and response. The HTTP protocol is being used to submit this request. A protocol is just a set of guidelines or standards that have been accepted by all users of the internet. The client-server request-response model is used here. When communicating with the server, HTTP, an application-layer protocol, often uses the Transmission Control Protocol (TCP).

II. Fundamentals of HTTP:

1. Data URLs – They are a particular class of URI that contains the resource they refer to. Although there are certain limitations, data URLs are quite practical.
2. Resource URLs – URLs of sources Although some websites the browser connects to can also access non-standard Resource URLs, those prefixed with the resource scheme are used by Firefox and Firefox browser extensions to load resources internally.
3. Multiple Input Multiple Execution (MIME) Formats – Different kinds of content can now be exchanged thanks to HTTP/1.0. This article explains how the MIME standard and the Content-Type header are used to accomplish this.

4. Choosing between URLs with and without www – This article offers advice on how to decide whether or not to use a www-prefixed name and discusses the implications of that decision.
5. Website Messages – The structure of HTTP Messages sent during requests or answers is pretty obvious. HTTP/2's binary frame and message structures encapsulate and represent HTTP/1.x messages.
6. HTTP/1.x Connection Management – The first HTTP version to offer persistent connections and pipelining was HTTP/1.1. HTTP/2 connection management - How connections are established and maintained has been extensively rethought in HTTP/2.
7. Negotiating the Connect – HTTP provides a series of headers beginning with Except for a browser to specify the format, language, or encoding it prefers. The process of this advertising, how the server should respond, and how it selects the best response are all explained in this article.
8. HTTP Server – HTTP server is an easy, static command-line program. It is robust enough for production use but also easy to use and hack for testing, local development, and learning.

III. Working:

Websites' domain names are used to access web server software, which ensures that the site's content is sent to the user who requests it. There are various parts to the software side, including at least one HTTP server. Both HTTP and URLs can be understood by the HTTP server. A web server is a piece of hardware that houses web server software and other website-related assets like HTML texts, pictures, and JavaScript files.

A web browser, such as Google Chrome or Firefox, will use HTTP to request a file that is stored on a web server. The HTTP server will accept the request when the web server receives it. Web server software, which ensures that the content of the site is transferred to the person who requests it, can be accessed through domain names for websites.

The software side is made up of several components, including at least one HTTP server. The HTTP server can comprehend both HTTP and URLs. A web server is a piece of hardware that contains JavaScript files, HTML text, and other website-related resources. It also houses web server software.

Using HTTP, a web browser like Google Chrome or Firefox will ask for a file that is kept on a web server. The HTTP server will accept the request when the web server receives it.

There are two possible ways for a web server to reply to a client request:

1. Transfer of the file to the client connected to the requested URL.

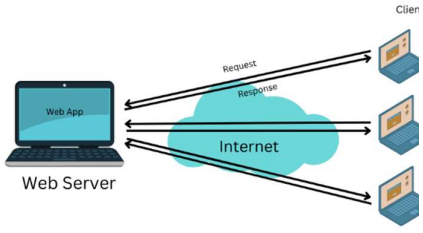


Figure 1 – Working of Web Server and Client

2. Response creation with the use of a script and database communication.

   (i) When a client requests a web page, the web server looks for the requested page and, if it is found, returns an HTTP response to the client.

   (ii) An HTTP response will be sent by the web server if the requested web page cannot be found, Error 404 No such page.

   (iii) If a client requests additional resources, the web server will get in touch with the application server and data store to build the HTTP response.

Source Code:

19CCE304_Expt_Code.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
"""
http.server - Defines classes for implementing HTTP servers.
HTTPServer - This class builds on the TCPServer class by storing the server
address as instance variables named server_name and server_port.
BaseHTTPRequestHandler - This class is used to handle the HTTP requests that
arrive at the server.
"""

import time # Provides various time-related functions

HOST = "192.168.109.13" # IP address of server
PORT = 5000 # Port Number

class requestHandler(BaseHTTPRequestHandler):

    # GET method is used to appends form data to the URL in name or value
pair:-
    def do_GET(self):
        self.send_response(200) # Sends the response header only
        self.send_header("Content-type", "text/html") # Adds the HTTP header
to an internal buffer which will be written to the output stream when either
end_headers() or flush_headers() is invoked.
        self.end_headers() # Adds a blank line (indicating the end of the HTTP
headers in the response) to the headers buffer and calls flush_headers().

        self.wfile.write(bytes("<html><body><h1>HTTP_client_server</h1></body>
</html>", "utf-8")) # Contains the output strean for writing a response back
to the client.

    # POST is a method that is supported by HTTP and depicts that a web server
accepts the data included in the body of the message:-
    def do_POST(self):
        self.send_response(200)
        self.send_header("Content-type", "application/json")
```

```
        self.end_headers()

        date = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time()))
# Convert a tuple or struct_time representing a time as returned by gmtime()
or localtime() to a string as specified by the format argument.
        self.wfile.write(bytes('{"time": "'+ date +'"}', "utf-8"))

server = HTTPServer((HOST, PORT), requestHandler)
print("The Server is Running...")

server.serve_forever() # Handle requests until an explicit shutdown() request.
server.server_close() # Clean up the server. May be overridden.

# Client URL (cURL) is a computer software project providing a library and
command-line tool for transferring data using various network protocols.
```

Screenshots of Input and Output:

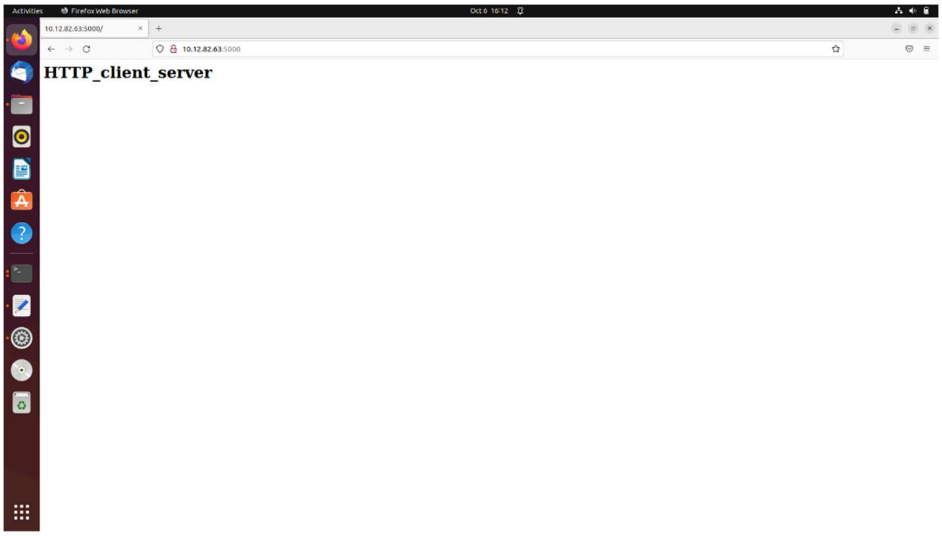


Figure 2 – HTTP Server Formed by HTML Code

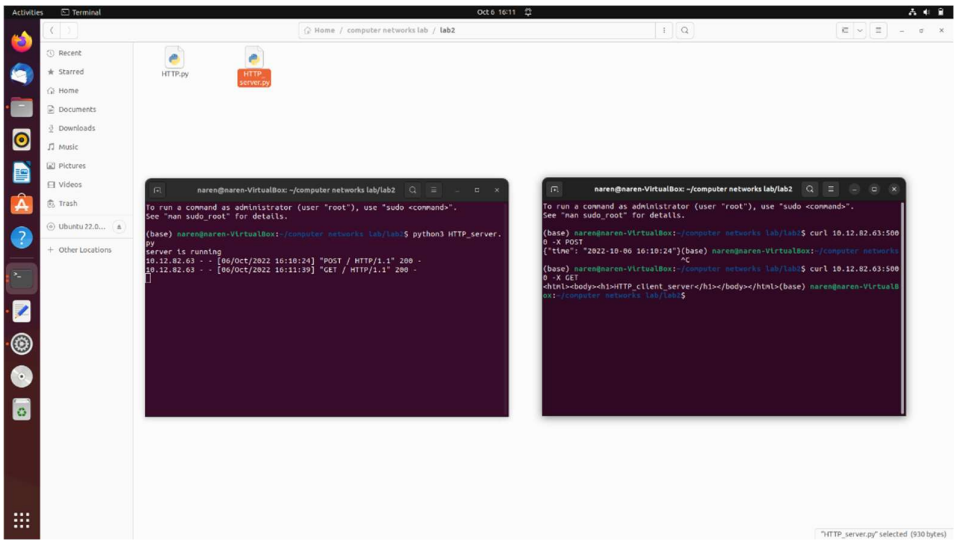


Figure 3 – Client and Server Communication

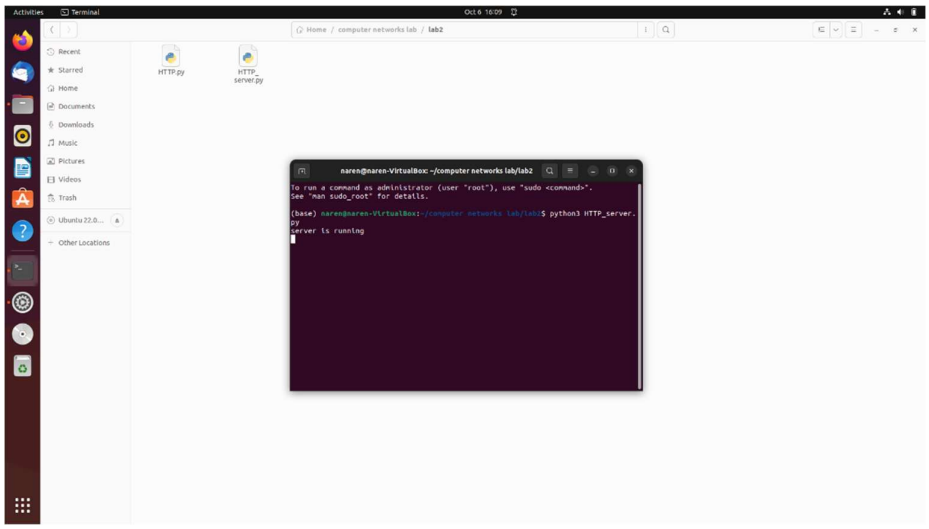


Figure 4 – Active Server

Conclusion:

Thus, the web server has been created and the required HTML script is displayed in the web browser under HTTP. Similarly, the client can access the server and get the necessary response from the server. Hence, all the simulation results were verified successfully.

References:

1. Foundations of Python network programming by Beaulne, Alexandre Goerzen, John Membrey, Peter Rhodes, Brandon 2014.
2. HTTP Server Python Documentation - http.server — HTTP servers — Python 3.10.7 documentation

Contribution:

1. Narendran S [CB.EN.U4CCE20036] – Development of Driver Code + Theory Documentation
2. Narun T [CB.EN.U4CCE20037] – Development of HTTP Server + Testing
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Development of HTTP Client + Topic Research
4. Santosh [CB.EN.U4CCE20053] – Code Debugger and Reviewer

EXPERIMENT 3: DEVELOPING AN EMAIL CLIENT USING PYTHON

(TEAM 9)

AIM:

Develop an email client using python:

1. Compose parsing email;
2. Compose Multiple Input Multiple Execution (MIME) attachment(s) and their decoding; and
3. Decode the email header.

SOFTWARES REQUIRED:

1. Microsoft Visual Studio Code, Microsoft Corporation
2. Oracle VM VirtualBox 6.1.38, Oracle Corporation
3. Python 3.10.7 (64-bit), Python Software Foundation
4. Termux Mobile Application (Available for Free on Play Store), Fredrik Fornwall
5. Ubuntu 22.04 (64-bit) Operating System

THEORY:

Q. What is meant by an email client?

A Mail User Agent is another name for an email client. Actually, sending and receiving emails is done through a programme or application that a person has installed on his or her computer. Emails are written, sent, and viewed using an email client. It is mostly utilised for managing emails, including their creation, transmission, receipt, and deletion.

Every email address has three components, which are required for sending and receiving emails:

1. Username – Anything is acceptable as long as it complies with the email provider's rules.
2. Domain Name (or) Host – This is the mail server that will store the message. It might be @outlook, @gmail, etc.
3. Domain Kind – This information indicates if an email address is for commercial use (.com), educational use (.edu), or any other special type.

Q. What does an email client contain?

1. Through the use of a user-friendly interface, it conceals the complexity of sending and receiving emails. Any user, regardless of experience level, may send and receive emails with ease by utilising this interface.

2. The majority of email client servers have some sort of backup, so if a message is deleted, it just moves to the trash can and may be recovered from there.
3. Each email client offers a high level of encryption to the messages as they travel over the internet to the server, protecting the communications' confidentiality or privacy.
4. Accessibility, which refers to the ability to read messages from any computer or mobile device from anywhere in the globe via an internet connection, is one of the most crucial characteristics.
5. There are no memory or storage issues because email clients offer more than enough RAM on email servers.
6. As a to-do list for establishing crucial reminders, email clients may also be utilised.

Microsoft Outlook, A Well-Known Email Client:

It is one of the greatest email programmes since it contains a tonne of options that people can use to handle their personal information. It is a component of the Office 365 programme. Both standalone and multi-user applications are possible with this programme. Microsoft Outlook has a number of functions, including:

- Managing Tasks
- Management of Contact
- Organizing Reminders
- Calendar management and taking notes

Q. What are some of the applications of an email client?

1. Internet Communication with Others – By sending and receiving messages, email clients facilitate safe online communication.
2. Expansion and Development of Business – In the current digital era, it is almost difficult to avoid interacting with an email client for any reason. It aids in business growth by offering a digital platform that makes it simple to carry out branding or advertising.
3. Transfer of Various File Types – The usage of email clients makes it very simple to exchange files with distant customers. Different sorts of files and media may be transmitted with ease utilising email clients.
4. Data Backup – Historically, backing up useful files has been a major challenge for everyone, but thanks to email, it is now simple to save useful material that can be retrieved at any time and from any location.
5. Record Keeping – It is also used to retain written records of client transactions, ensuring that everyone's assertions are true and that no one can refute them.

Q. What are some of the issues with an email client?

1. System Updates – Webmail provides updates every few weeks, however, email clients may not release any changes for many years.
2. Access Email on Several Computers – IMAP is a popular email protocol that certain email providers employ and is great for synchronising between devices. Other POP-enabled

email clients don't have reliable synchronisation. Users will be left without many options and in a bind when trying to access their email from other machines.

3. Data Backup – Because email clients save all sent and received emails on the computer, if there is a software or hardware fault and the emails have not been backed up, then there is a chance that all emails will be lost.

Key Difference Between Email and Web Client:

They both have the same function: sending emails, downloading files, using a calendar, and storing contact information. The methods for accessing them and retrieving deleted emails do, however, differ significantly. Webmail can only be accessed using web browsers, whereas email clients may be accessed through desktop programmes.

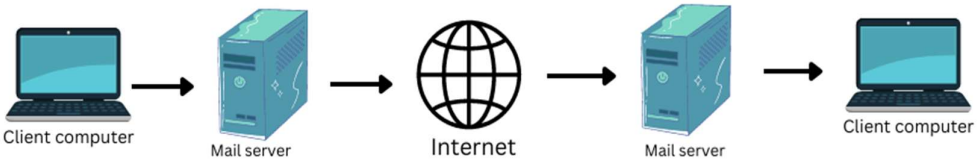


Figure 1 – Demonstration of Email Client Process

SOURCE CODE:

Email.py

```
import smtplib # Defines an SMTP client session object that can be used to
send mail to any internet machine with an SMTP or ESMTP listener daemon.

"""
The email package provides some convenient encoders in its encoders module.
These encoders are actually used by the MIMEAudio and MIMEImage class
constructors to provide default encodings.
"""
from email import encoders

from email.mime.text import MIMEText # A subclass of MIMENonMultipart, the
MIMEText class is used to create MIME objects of major type text.
from email.mime.base import MIMEBase # This is the base class for all the
MIME-specific subclasses of Message.
from email.mime.multipart import MIMEMultipart # A subclass of MIMEBase, this
is an intermediate base class for MIME messages that are multipart.

server = smtplib.SMTP('smtp.gmail.com', 587) # Forms an email client with SMTP
listener module
# server.connect("smtp.gmail.com", 587)
```

```
# server.ehlo()
# with open('password.txt' , 'r') as f:

server.starttls()
server.login('group9cn@gmail.com', 'fbyj klbk yfsf zapk') # Log in with mail
credentials
msg = MIMEMultipart() # Secure Multipurpose Internet Mail Extensions (S/MIME)
is used to send attachments

msg['From'] = 'Santosh' # Sender Name
msg['To'] = 'group9cn@gmail.com' # Receiver Email Address
msg['Subject'] = 'Test File - Experiment 3' # Subject of the Mail

# with open('attachment.txt', 'r') as f:
# message = f.read()
message  = 'Hi! This is Santosh (53) here. I have sent an email sent from
Python.'
msg.attach(MIMEText(message , 'plain')) # Text Message Attachment

filename = 'Email.py' # File Name of Attachment
attachment = open(filename, 'rb') # Opens a file, and returns it as a file
object
p = MIMEBase('application', 'octet-stream') # File Attachment
p.set_payload(attachment.read())

encoders.encode_base64(p) # Encodes the payload into base64 form and sets the
Content-Transfer-Encoding header to base64.
p.add_header('content-Disposition', f'attachment; filename = {filename}') #
Header name is the same as file name
msg.attach(p)

text = msg.as_string() # Allows str(msg) to produce a string containing the
serialized message in a readable format.
server.sendmail('group9cn@gmail.com' , 'group9cn@gmail.com', text) # Send
mail, from sender to receiver with text message
print("\nThe mail along with the attachment(s) has been sent to your
inbox.\n")
```

Email_Decode.py

```
import imaplib # This module defines three classes, IMAP4, IMAP4_SSL and
IMAP4_stream, which encapsulate a connection to an IMAP4 server and implement
a large subset of the IMAP4rev1 client protocol as defined in RFC 2060.

import email # Library for managing email messages
imap_server = 'imap.gmail.com' # IMAP server with the IMAP listener module
email_address = 'group9cn@gmail.com'
password = 'fbyj klbk yfsf zapk'
```

```
imap = imaplib.IMAP4_SSL(imap_server) # This is a subclass derived from IMAP4
that connects over an SSL encrypted socket
imap.login(email_address, password) # Log in with mail credentials
imap.select("Inbox") # Item selection for decoding
_, msgnum = imap.search(None, "ALL") # Return the count of mails in 'Inbox'
print("\nCount of Mails in Inbox: " + str(msgnum) + "\n")

# Print Decoded Details:-
for msgnum in msgnum[0].split(): # Splits a string into a list
    _, data = imap.fetch(msgnum, ("RFC822")) # Fetch (parts of) messages

    message = email.message_from_bytes(data[0][1]) # Return a message object
structure from a bytes-like object.
    print(f"Message Index Number: {msgnum}")
    print(f"From : {message.get('From')}")
    print(f"To : {message.get('To')}")

    print(f"BCC: {message.get('BCC')}")
    print(f"Date: {message.get('Date')}")
    print(f"Subject: {message.get('Subject')}")
    print("Content: ")

    # The walk() method is an all-purpose generator which can be used to
iterate over all the parts and subparts of a message object tree, in depth-
first traversal order.
    for part in message.walk():
        if part.get_content_type() == "text/plain": # Return the message's
content type, coerced to lower case of the form maintype/subtype.
            print(part.as_string()) # Allows str(msg) to produce a string
containing the serialized message in a readable format.

    print("\n")
```

SCREENSHOTS OF INPUTS & OUTPUTS:

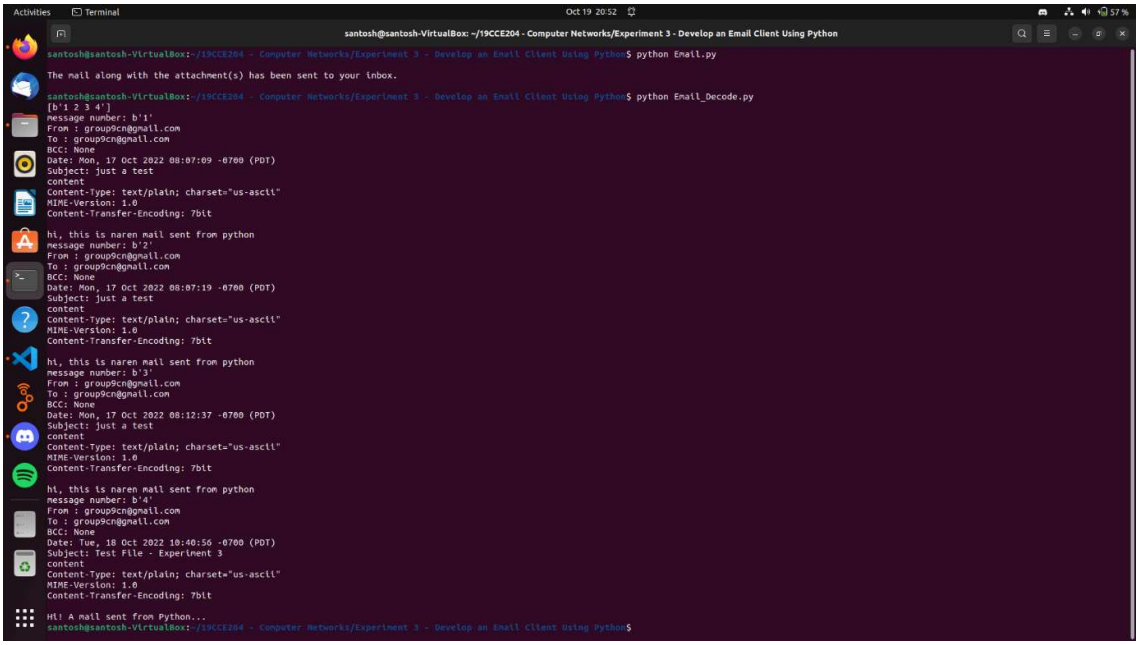


Figure 2 – Program Output Via Terminal

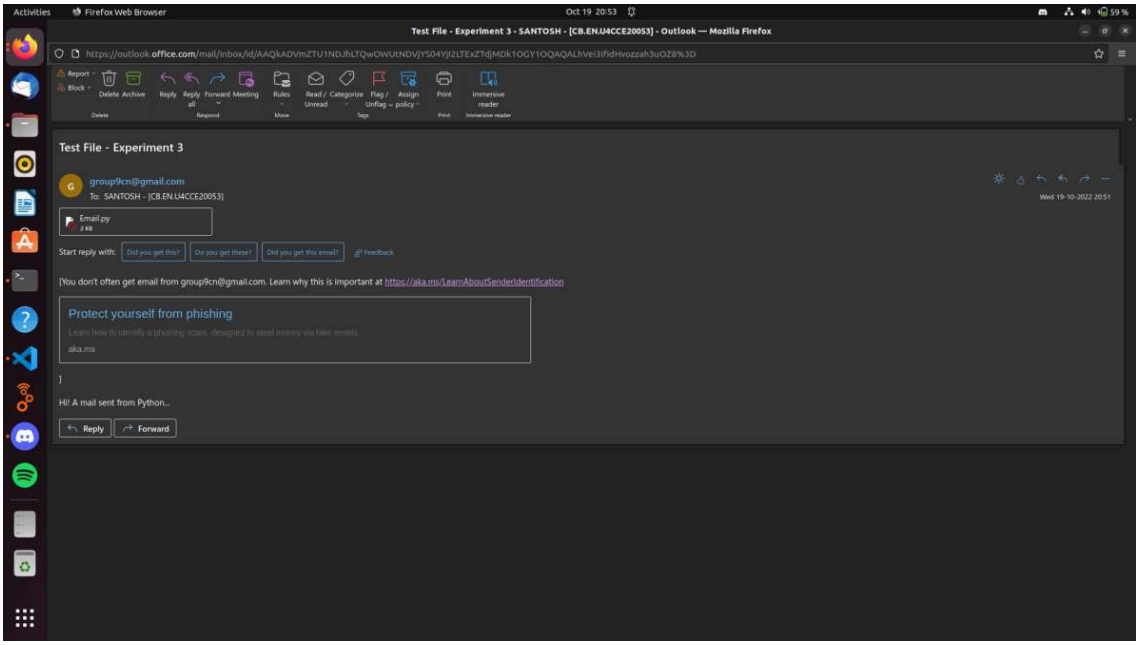


Figure 3 – Email Along with Attachment

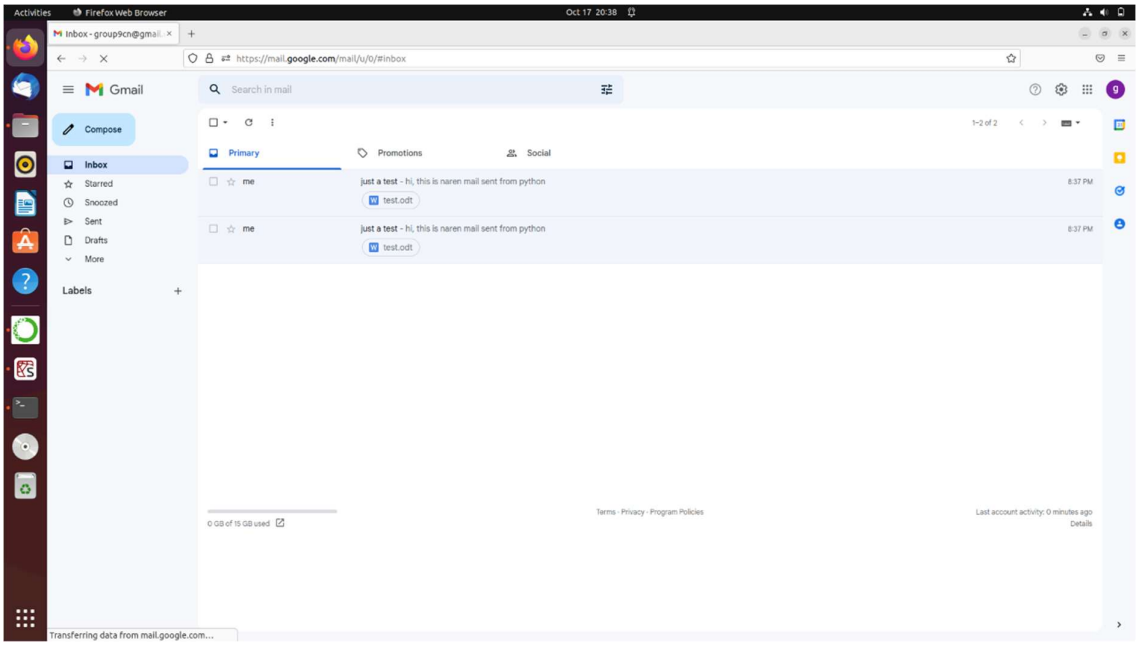


Figure 4 – Mails in Inbox

CONCLUSION:

Thus, developed an email client using Python that composes parsing email and MIME attachments. Decoding of the same along with the email header has been performed too. All the simulation results were verified successfully.

REFERENCES:

1. Foundations of Python 3 Network Programming, John Goerzen, Brandon Rhodes, Second Edition, Apress Publisher, ISBN-10: 1430258543, ISBN-13: 978-1430258544
2. Foundations of Python Network Programming by Beaulne, Alexandre Goerzen, John Membrey, Peter Rhodes, Brandon 2014.

CONTRIBUTION:

1. Narendran S [CB.EN.U4CCE20036] – Compose parsing email + Code Debugging
2. Narun T [CB.EN.U4CCE20037] – Compose Multiple Input Multiple Execution (MIME) attachments + Code Reviewing
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Decode attachments and email header + Code Testing
4. Santosh [CB.EN.U4CCE20053] – Topic Research + Documentation

EXPERIMENT 4: SHORTEST PATH ALGORITHMS USING PYTHON

(TEAM 9 – LINK STATE ROUTING)

AIM:

To demonstrate the link state routing algorithm and protocols using Python.

SOFTWARES REQUIRED:

1. Microsoft Visual Studio Code, Microsoft Corporation
2. Oracle VM VirtualBox 6.1.38, Oracle Corporation
3. Python 3.10.7 (64-bit), Python Software Foundation
4. Ubuntu 22.04 (64-bit) Operating System
5. Cisco Packet Tracer 8.2.0

THEORY:

Link State Routing Algorithm is a routing technique which is used by routers to share information/knowledge about their neighbouring routers with the rest of the routers on their network. The link state algorithm helps each router in the network to make its routing table, with the help of the information about the network topology.

Routing:

1. Routing is a process in which the path each data packet has to follow is established. In this process, a routing table is created.
2. This table contains all the information on the path a data packet has to follow to reach its appropriate destination.
3. Various algorithms are used to find the optimized path for the transmission of a data packet.
4. An optimal route is a path in which the connection cost is the least from the source to the destination.
5. Here we use an algorithm called Dijkstra's algorithm for the calculation of the path with the least cost, after which the routing table is created.
6. This table created by each router is interchanged with other routers in the network.
7. This in turn helps in the speedy and reliable transmission of data.

Dijkstra's Algorithm: It solves the single-source shortest-path problem for graphs with nonnegative edge costs.

Declare all nodes unscanned and initialize *d* and parent.

while there is an unscanner node with a tentative distance < $+\infty$ do

u: = the unscanned node with minimal tentative distance

relax all edges *(u, v)* out of u and declare *u* scanner

Flooding:

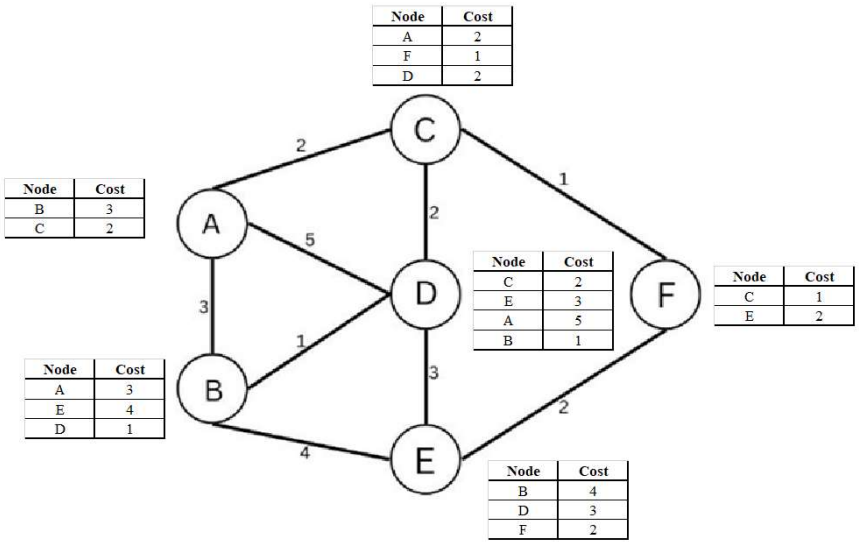
1. It is a process in which every router (except its neighbours) transmits information to other routers about its neighbouring routers. This process is called flooding.
2. Every router receiving the information sends copies of that information to all its neighbouring routers.
3. Finally, all routers in the network have a copy of the same information. This information is only exchanged when there is a modification in the information, which makes the link state routing algorithm effective.

Phases of Link State Routing:

There are two phases in link state routing: –

1. Reliable Flooding – In this phase, the information about neighbouring routers is shared and gathered through the process of flooding. This phase is again divided into two phases,
    - Initial State – In this state, every router becomes aware of the cost of transmission of its neighbours.
    - Final State – In this state, every router knows the information about the entire graph.
2. Route Calculation – In this phase all routers in the network use algorithms like Dijkstra's algorithm to find the shortest or optimal path to all the other routers in the network.

Example of Link State Routing:



Iteration	Tree	B	C	D	E	F
Initial	{A}	3	2	5	∞	∞
1	{A, C}	3	-	4	∞	3
2	{A, B, C}	-	-	4	7	3
3	{A, B, C, F}	-	-	4	5	-
4	{A, B, C, D, F}	-	-	-	5	-
5	{A, B, C, D, E, F}	-	-	-	-	-

SOURCE CODE:

Node.py

```
#!/usr/bin/python3
class Node:
    def __init__(self,element):
        self._element = element

    def __str__(self):
        return str(self._element)
    def __lt__(self,v):
        return self._element < v.element()
    def element(self):
        return self._element
```

Link.py

```
#!/usr/bin/python3
class Link:

    def __init__(self,v,w,element):
        #create link between node v and w
        self._nodes=(v,w)
        self._element = element

    def __str__(self):
        return("\n" + str(self._nodes[0])+"-"+
               str(self._nodes[1])+":"+str(self._element))

    def nodes(self):
        return self._nodes

    def start(self):
        return self._nodes[0]

    def end(self):
        return self._nodes[1]

    def opposite(self,v):
        if self._nodes[0]==v:
```

```
            return self._nodes[1]
        elif self._nodes[1]==v:
            return self._nodes[0]
        else:
            return None

    def element(self):
        return self._element
```

APQ_Heap.py

```
#!/usr/bin/python3
class APQHeap:
    """ Maintain an collection of items, popping by lowest key.

        This implementation maintains the collection using a binary heap.
        Keys and or values can be updated.
        Items can be arbitrarily removed.
    """
    class Element:
        """ An element with a key and value and an index to the heap-array.
"""

        def __init__(self, k, v, index):
            self._key = k
            self._value = v
            self._index = index

        def __eq__(self, other):
            """ Return True if this key equals the other key. """
            return self._key == other._key

        def __lt__(self, other):
            """ Return True if this key is less than the other key. """
            return self._key < other._key

        def _wipe(self):
            """ Set the instance variables to None. """
            self._key = None
            self._value = None
            self._index = None

    def __init__(self):
        """ Create an APQ with no elements. """
        self._heap = []
        self._size = 0

    def __str__(self):
        """ Return a breadth-first string of the values. """
```

```
        outstr = '<-'
        for elt in self._heap:
            outstr += str(elt._value) + ':' + str(elt._key) + '-'
        return outstr + '<'

    def add(self, key, value):
        """ Add Element(key,value,size) to the heap. """
        e = APQHeap.Element(key, value, self._size)
        self._heap.append(e)
        self._upheap(self._size)
        self._size += 1
        return e

    def min(self):
        """ Return the min priority key,value. """
        if self._size:
            return self._heap[0]._key, self._heap[0]._value
        return None, None

    def _swap_last_into_place(self, topos):
        """ Swap the last item into position topos. """
        if self._size > 1: #if other items, restructure
            self._heap[topos] = self._heap[self._size - 1]
            self._heap[topos]._index = topos
            self._heap.pop()
            self._size -= 1
            parent = self._parent(topos)
            if parent > -1 and self._heap[topos]._key <
self._heap[parent]._key:
                self._upheap(topos)
            else:
                self._downheap(topos)
        else:
            self._heap.pop()
            self._size -= 1

    def remove_min(self):
        """ Remove and return the min priority key,value. """
        returnvalue = None
        returnkey = None
        if self._size:
            returnkey = self._heap[0]._key
            returnvalue = self._heap[0]._value
            self._heap[0]._wipe()
            self._swap_last_into_place(0)
        return returnkey, returnvalue

    def length(self):
```

```
        """ Return the number of items in the heap. """
        return self._size

    def is_empty(self):
        """ Return True if the heap is empty. """
        return self._size == 0

    def _left(self, posn):
        """ Return the index of the left child of elt at index posn. """
        return 1 + 2*posn

    def _right(self, posn):
        """ Return the index of the right child of elt at index posn. """
        return 2 + 2*posn

    def _parent(self, posn):
        """ Return the index of the parent of elt at index posn. """
        return (posn - 1)//2

    def _upheap(self, posn):
        """ Bubble the item in posn in the heap up to its correct place. """
        if posn > 0 and self._upswap(posn, self._parent(posn)):
            self._upheap(self._parent(posn))

    def _upswap(self, posn, parent):
        """ If healp elt at posn has lower key than parent, swap. """
        if self._heap[posn] < self._heap[parent]:
            self._heap[posn], self._heap[parent] = self._heap[parent],
self._heap[posn]
            self._heap[posn]._index = posn
            self._heap[parent]._index = parent
            return True
        return False

    def _downheap(self, posn):
        """ Bubble the item in posn in the heap down to its correct place. """
        #find minchild position
        #if minchild is in the heap
        #    if downswap with minchild is true
        #        downheap minchild
        minchild = self._left(posn)
        if minchild < self._size:
            if (minchild + 1 < self._size and
                self._heap[minchild]._key > self._heap[minchild + 1]._key):
                minchild +=1
            if self._downswap(posn, minchild):
                self._downheap(minchild)
```

```
    def _downswap(self, posn, child):
        """ If healp elt at posn has lower key than child, swap. """
        #Note: this could be merged with _upswap to provide a general
        #heapswap(first, second) method, which swaps if the element
        #first has lower key than the element second
        if self._heap[posn]._key > self._heap[child]._key:
            self._heap[posn], self._heap[child] = self._heap[child],
self._heap[posn]
            self._heap[posn]._index = posn
            self._heap[child]._index = child
            return True
        return False

    def update_key(self, location, key):
        """ Update the key of the item in location. """
        #location is actually an Element
        #updating the key may require us to rebalance the heap
        location._key = key
        parent = self._parent(location._index)
        if parent > -1 and location._key < self._heap[parent]._key:
            self._upheap(location._index)
        else:
            self._downheap(location._index)

    def update_value(self, location, value):
        """ Update the value of the item in location. """
        #location is actually an Element
        location._value = value

    def get_key(self, location):
        """ Return the current key (priority value) for item in location. """
        #location is actually an Element
        return location._key

    def remove(self, location):
        """ Remove the item in location from the APQ, and return key,value.
"""
        #location is actually an Element
        returnkey = location._key
        returnvalue = location._value
        pos = location._index
        self._swap_last_into_place(pos)
        location._wipe()
        location = None
        return returnkey, returnvalue

    def _printstructure(self):
        """ Print out the elements one to a line. """
```

```
        for elt in self._heap:
            if elt is not None:
                print('(', elt._key, ',', elt._value, ',', elt._index, ')')
            else:
                print('*')
```

Graph.py

```
#!/usr/bin/python3
from node import Node
from link import Link
from apq_heap import APQHeap

class Graph:
    #the keys are the nodes and values are the link
    def __init__(self):
        #create an initial empty graph
        self._structure = dict()
    def __str__(self):
        strnodes="\nNodes"
        for v in self._structure:
            strnodes +="\t"+str(v)
        links=self.links()
        strlink="\nLinks:"
        for w in links:
            strlink +=str(w)
        return ("Total Nodes: " +str(self.num_nodes())+"\n"+"Total Links: "+
                                str(self.num_links())+ strnodes+strlink)
    #ADT methods to query the graph

    def num_nodes(self):
        return len(self._structure)

    def num_links(self):
        num = 0
        for v in self._structure:
            num +=len(self._structure[v])
        return num // 2

    def nodes(self):
        return [key for key in self._structure]

    def get_node_by_label(self, element):
        for v in self._structure:
            if v.element()== element:
                #print(v)
                return v
        return None
    def links(self):
```

```
        linklist = []
        for v in self._structure:
            for w in self._structure[v]:
                #to avoid duplicates,only return if v in the first nodes
                if self._structure[v][w].start() ==v:
                    linklist.append(self._structure[v][w])
        return linklist

    def get_links(self,v):
        #list of all links
        if v in self._structure:
            linklist = []
            for w in self._structure[v]:
                linklist.append(self._structure[v][w])
            return linklist
        return None

    def get_link(self,v,w):
        #link between v and w
        if(self._structure != None and v in self._structure
           and w in self._structure[v]):
            return self._structure[v][w]
        return None

    def degree(self, v):
        #return degree of node v
        return len(self._structure[v])

    def get_weight(self,v,w):
        #Cost between v and w
        if(self._structure !=None and v in self._structure
            and w in self._structure[v]):
            return len(self._structure[v][w])

    def add_node(self,element):
        v=Node(element)
        self._structure[v]=dict()
        return v


    def add_node_if_new(self, element):
        for v in self._structure:
            if v.element()== element:
                #print("already there")
                return v
        return self.add_node(element)

    def add_link(self, v, w,element):
```

```
        if not v in self._structure or not w in self._structure:
            return None
        e=Link(v,w,element)
        self._structure[v][w]=e
        self._structure[w][v]=e
        return e

    def highestdegreenode(self):
        #return the vertex with highet degree
        hd=-1
        hdv=None
        for v in self._structure:
            if self.degree(v)>hd:
                hd=self.degree(v)
                hdv = v
        return hdv

    def linkState(self, src):
        closed = dict()
        locs = dict()
        pred = {src:None}
        apq = APQHeap()#empty apq
        locs[src]=apq.add(0,src)
        while not apq.is_empty():
            key, u = apq.remove_min()
            del locs[u]
            closed[u]=(key,pred[u])
            for e in self.get_links(u):
                v = e.opposite(u)
                if v not in closed:
                    newcost = e.element() + key
                    if v not in locs:
                        pred[v] = u
                        locs[v] = apq.add(newcost,v)
                    elif newcost< apq.get_key(locs[v]):
                        pred[v] = u
                        apq.update_key(locs[v],newcost)
        return closed

    def graphreader(filename):

        """ Read and return the route map in filename. """
        graph = Graph()
        file = open(filename, 'r')
        entry = file.readline() #either 'Node' or 'Edge'
        num = 0
        print("**********Welcome to Link State Topology*************")
        while entry == 'Node\n':
```

```
            num += 1
            nodeid = int(file.readline().split()[1])
            node = graph.add_node(nodeid)
            entry = file.readline() #either 'Node' or 'Edge'
        print("\tFound", num, 'Nodes and added into the graph')
        num = 0
        while entry == 'Edge\n':
            num += 1
            source = int(file.readline().split()[1])
            sv = graph.get_node_by_label(source)
            target = int(file.readline().split()[1])
            tv = graph.get_node_by_label(target)
            length = float(file.readline().split()[1])
            edge = graph.add_link(sv, tv, length)
            file.readline() #read the one-way data
            entry = file.readline() #either 'Node' or 'Edge'
        print("\tFound", num, 'Links and added into the graph')
        print(graph)
        return graph
```

Test.py

```
#!/usr/bin/python3
from  Graph import Graph
import networkx as nx
import matplotlib.pyplot as plt
#Test methods
def test1():
    graph = Graph.graphreader("graph1.txt")
    for i in range(1,6):
        src = graph.get_node_by_label(i)
        d = graph.linkState(src)
        for element in sorted(d):
            print("Path from " + str(src) + " to Node " +str(element) +
                  " : Length :" + str(d[element][0]) + ". Preceding : " +
str(d[element][1]) )
        for element in sorted(d):
            cost,pred = d[element]
            print("From Node:" + "Cost")
            print(element,'       :',cost, '(', pred ,')')
def test2():
    graph = Graph.graphreader("simplegraph2.txt")
    src = graph.get_node_by_label(14)

    d = graph.linkState(src)
    for element in d:
        print("Path from "+str(src) + "  to Node " + str(element)
            + ". Length :" + str(d[element][0]) + ". Preceding : " +
str(d[element][1]))
```

```
        for element in sorted(d):
            cost,pred = d[element]
            print("From Node:" + "Cost")
            print(element,'       :',cost, '(', pred ,')')

if __name__=="__main__":
    test1()
    #test2()
```

Node_Info.txt

Node	to: 3	from: 3
id: 1	length: 2	to: 4
Node	oneway: false	length: 2
id: 2	Edge	oneway: false
Node	from: 1	Edge
id: 3	to: 4	from: 3
Node	length: 5	to: 6
id: 4	oneway: false	length: 1
Node	Edge	oneway: false
id: 5	from: 2	Edge
Node	to: 4	from: 4
id: 6	length: 1	to: 5
Edge	oneway: false	length: 3
from: 1	Edge	oneway: false
to: 2	from: 2	Edge
length: 3	to: 5	from: 5
oneway: false	length: 4	to: 6
Edge	oneway: false	length: 2
from: 1	Edge	oneway: false

SCREENSHOTS OF INPUTS & OUTPUTS:

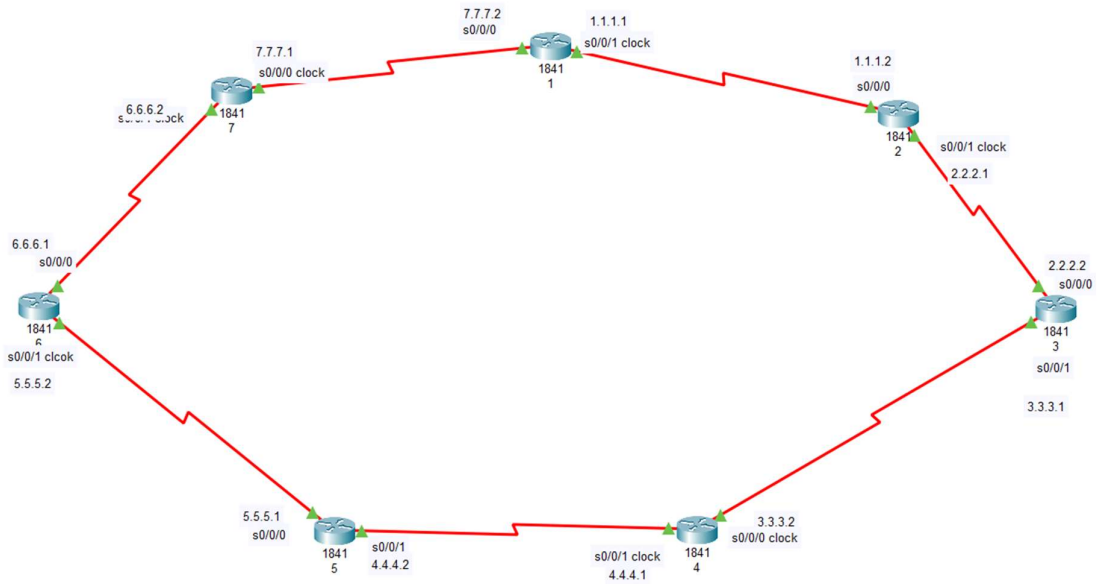
```
Total Nodes: 6
Total Links: 9
Nodes   1   2   3   4   5   6
Links:
1-2:3.0
1-3:2.0
1-4:5.0
2-4:1.0
2-5:4.0
3-4:2.0
3-6:1.0
4-5:3.0
5-6:2.0
Path from 1 to Node 1 : Length :0. Preceding : None
Path from 1 to Node 2 : Length :3.0. Preceding : 1
Path from 1 to Node 3 : Length :2.0. Preceding : 1
Path from 1 to Node 4 : Length :4.0. Preceding : 3
Path from 1 to Node 5 : Length :5.0. Preceding : 6
Path from 1 to Node 6 : Length :3.0. Preceding : 3
From Node:Cost
1        : 0 ( None )
From Node:Cost
2        : 3.0 ( 1 )
From Node:Cost
3        : 2.0 ( 1 )
From Node:Cost
4        : 4.0 ( 3 )
From Node:Cost
5        : 5.0 ( 6 )
From Node:Cost
6        : 3.0 ( 3 )
```

```
Total Nodes: 6
Total Links: 9
Nodes   1   2   3   4   5   6
Links:
1-2:3.0
1-3:2.0
1-4:5.0
2-4:1.0
2-5:4.0
3-4:2.0
3-6:1.0
4-5:3.0
5-6:2.0
Path from 1 to Node 1 : Length :0. Preceding : None
Path from 1 to Node 2 : Length :3.0. Preceding : 1
Path from 1 to Node 3 : Length :2.0. Preceding : 1
Path from 1 to Node 4 : Length :4.0. Preceding : 3
Path from 1 to Node 5 : Length :5.0. Preceding : 6
Path from 1 to Node 6 : Length :3.0. Preceding : 3
From Node:Cost
1        : 0 ( None )
From Node:Cost
2        : 3.0 ( 1 )
From Node:Cost
3        : 2.0 ( 1 )
From Node:Cost
4        : 4.0 ( 3 )
From Node:Cost
5        : 5.0 ( 6 )
From Node:Cost
6        : 3.0 ( 3 )
```

```
Total Nodes: 6
Total Links: 9
Nodes   1   2   3   4   5   6
Links:
1-2:3.0
1-3:2.0
1-4:5.0
2-4:1.0
2-5:4.0
3-4:2.0
3-6:1.0
4-5:3.0
5-6:2.0
Path from 1 to Node 1 : Length :0. Preceding : None
Path from 1 to Node 2 : Length :3.0. Preceding : 1
Path from 1 to Node 3 : Length :2.0. Preceding : 1
Path from 1 to Node 4 : Length :4.0. Preceding : 3
Path from 1 to Node 5 : Length :5.0. Preceding : 6
Path from 1 to Node 6 : Length :3.0. Preceding : 3
From Node:Cost
1        : 0 ( None )
From Node:Cost
2        : 3.0 ( 1 )
From Node:Cost
3        : 2.0 ( 1 )
From Node:Cost
4        : 4.0 ( 3 )
From Node:Cost
5        : 5.0 ( 6 )
From Node:Cost
6        : 3.0 ( 3 )
```


```
Total Nodes: 6
Total Links: 9
Nodes   1   2   3   4   5   6
Links:
1-2:3.0
1-3:2.0
1-4:5.0
2-4:1.0
2-5:4.0
3-4:2.0
3-6:1.0
4-5:3.0
5-6:2.0
Path from 1 to Node 1 : Length :0. Preceding : None
Path from 1 to Node 2 : Length :3.0. Preceding : 1
Path from 1 to Node 3 : Length :2.0. Preceding : 1
Path from 1 to Node 4 : Length :4.0. Preceding : 3
Path from 1 to Node 5 : Length :5.0. Preceding : 6
Path from 1 to Node 6 : Length :3.0. Preceding : 3
From Node:Cost
1        : 0 ( None )
From Node:Cost
2        : 3.0 ( 1 )
From Node:Cost
3        : 2.0 ( 1 )
From Node:Cost
4        : 4.0 ( 3 )
From Node:Cost
5        : 5.0 ( 6 )
From Node:Cost
6        : 3.0 ( 3 )
```

```
Total Nodes: 6
Total Links: 9
Nodes   1   2   3   4   5   6
Links:
1-2:3.0
1-3:2.0
1-4:5.0
2-4:1.0
2-5:4.0
3-4:2.0
3-6:1.0
4-5:3.0
5-6:2.0
Path from 1 to Node 1 : Length :0. Preceding : None
Path from 1 to Node 2 : Length :3.0. Preceding : 1
Path from 1 to Node 3 : Length :2.0. Preceding : 1
Path from 1 to Node 4 : Length :4.0. Preceding : 3
Path from 1 to Node 5 : Length :5.0. Preceding : 6
Path from 1 to Node 6 : Length :3.0. Preceding : 3
From Node:Cost
1        : 0 ( None )
From Node:Cost
2        : 3.0 ( 1 )
From Node:Cost
3        : 2.0 ( 1 )
From Node:Cost
4        : 4.0 ( 3 )
From Node:Cost
5        : 5.0 ( 6 )
From Node:Cost
6        : 3.0 ( 3 )
```



CONCLUSION:

Illustrated the link state routing algorithms and protocols. All the simulation results were verified successfully.

REFERENCES:

1. Foundations of Python 3 Network Programming, John Goerzen, Brandon Rhodes, Second Edition, Apress Publisher; ISBN-10: 1430258543, ISBN-13: 978-1430258544
2. Foundations of Python Network Programming by Beaulne, Alexandre Goerzen, John Membrey, Peter Rhodes, Brandon 2014.
3. (2008) Algorithms and Data Structures - The Basic Toolbox, Kurt Mehlorn, Peter Sanders, Springer; ISBN: 978-3-540-77977-3, e-ISBN: 978-3-540-77978-0

CONTRIBUTION:

1. Narendran S [CB.EN.U4CCE20036] – Code Work on Heaps + Testing with Node Information + Graph Visualization
2. Narun T [CB.EN.U4CCE20037] – Code Work on Graphs + Debugging
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Code Work on Linking + Reviewing
4. Santosh [CB.EN.U4CCE20053] – Code Work on Nodes + Documentation

EXPERIMENT 5: SIMULATION OF WIRELESS PERSONAL, LOCAL AND WIDE AREA NETWORKS

TEAM 9

AIM

To simulate wireless personal, local and wide area networks using Cisco Packet Tracer.

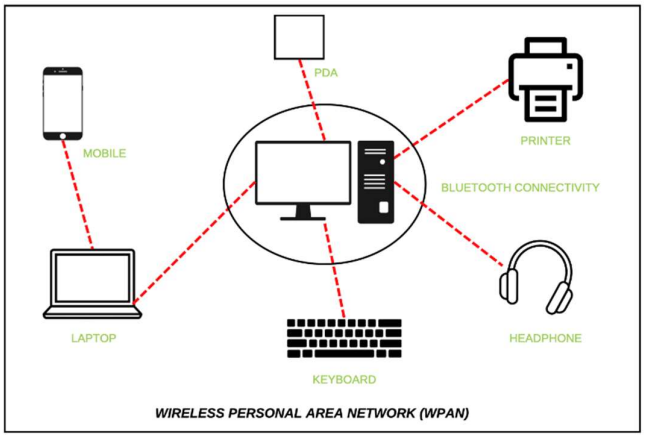
SOFTWARES REQUIRED

1. Oracle VM VirtualBox 6.1.38, Oracle Corporation
2. Ubuntu 22.04 (64-bit) Operating System
3. Cisco Packet Tracer 8.2.0 64-bit, Cisco Systems Inc.

THEORY

Personal Area Network (PAN): -

1. Range – Electronic devices in a user's local vicinity are connected by a personal area network (PAN). A personal area network (PAN) links technological items that are normally within a single user's range, which is roughly 10 meters (33 feet) apart.

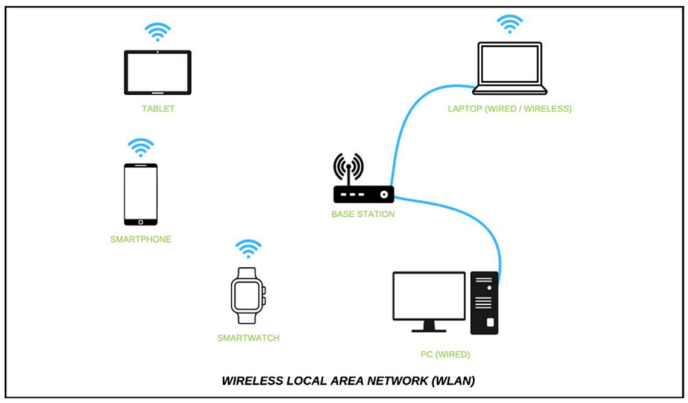


2. Components – Laptops, smartphones, tablets, wearable technology, printers, and entertainment equipment are the most common PAN components. A desktop computer, a wireless mouse, and wireless headphones, for instance, can all be linked together, but only the computer can establish a direct connection to the Internet.
3. Working – Although devices within a PAN can communicate with one another and exchange data, PANs typically lack a router and cannot connect directly to the Internet. But a PAN device can also be linked to a LAN, and the LAN can subsequently be linked to the Internet.
4. Data Transfer – This kind of network is made to allow wired or wireless communication and resource, data, and application sharing across devices in a small office or home office (SOHO) setting.
5. Use Cases – The link between a Bluetooth earpiece and a smartphone is one of the most typical real-world instances of a PAN. PANs can link various electronic devices including keyboards, printers, tablets, and laptops.

6. Examples – There are wired and wireless PAN network connections available. Wireless connection technologies include Bluetooth, Wi-Fi, IrDA, and Zigbee, while wired connection methods include USB and FireWire.

Local Area Network (LAN): -

1. Range – A group of devices connected in a single physical location, such as a building, office, or home, is known as a local area network (LAN).
2. Property – The fact that a LAN connects devices that are located in a single, constrained region is its sole distinguishing feature, regardless of size. A wide area network (WAN) or metropolitan area network (MAN), in contrast, spans a wider geographic area. A few WANs and MANs link numerous LANs collectively.

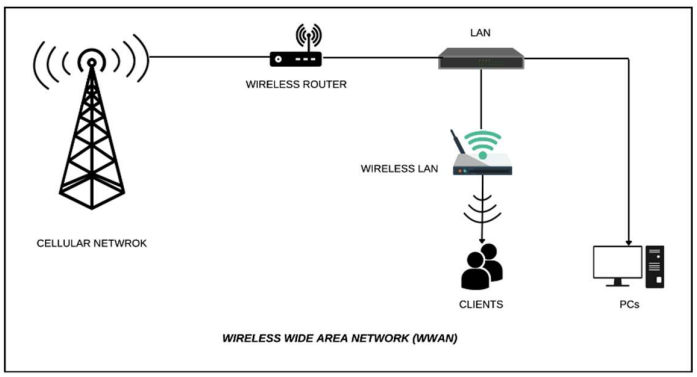


3. Components – A local area network (LAN) is made up of cables, access points, switches, routers, and additional parts that allow devices to connect to internal servers, web servers, and other LANs via wide area networks.
4. Working – The computers in each department might be conceptually connected to the same switch in an office with many departments, such as accounting, IT support, and administration, but they might be segmented to operate separately.
5. Virtualization – The growth of virtualization has also sped up the creation of virtual LANs, which let network managers divide and logically organize network nodes without having to make significant changes to their infrastructure.
6. Data Transfer – The benefits of a LAN are the same as those of any networked group of devices. The devices can share files, print to shared printers, access and even control one another, use a single Internet connection, and more.
7. Use Cases – LANs are being used by households, restaurants, coffee shops, shops, and schools in addition to companies and educational institutions.
8. Growth – Even while the advantages of connecting devices to a network have long been acknowledged, it wasn't until the widespread adoption of Wi-Fi technology that LANs started to become typical in almost all types of environments.

Wide Area Network (WAN): -

1. Range – A vast network of information that is not connected to a single location is referred to as a wide area network (also known as WAN). Through a WAN provider, WANs may help devices from all over the world communicate, exchange information, and do much more.

2. Internet – The internet is regarded as the world's largest WAN, making WANs crucial for global commerce as well as for daily use.



3. Working – These networks are frequently built by service providers who then rent out their WAN to organizations including businesses, universities, governments, and the general public.
4. Data Transfer – No matter where they are, as long as they have access to the established WAN, these clients can utilize the network to relay and store data or connect with other users.
5. Use Cases – Different linkages, such as virtual private networks (VPNs) or lines, wireless networks, cellular networks, or internet access, can be used to enable access.

There are also more types of telecommunications networks, such as:

- Home Area Networks (HAN)
- Neighbourhood Area Networks (NAN)
- Field (or Factory) Area Networks (FAN)
- Metropolitan Area Networks (MAN)
- Internet (or Cloud) Area Networks (IAN)

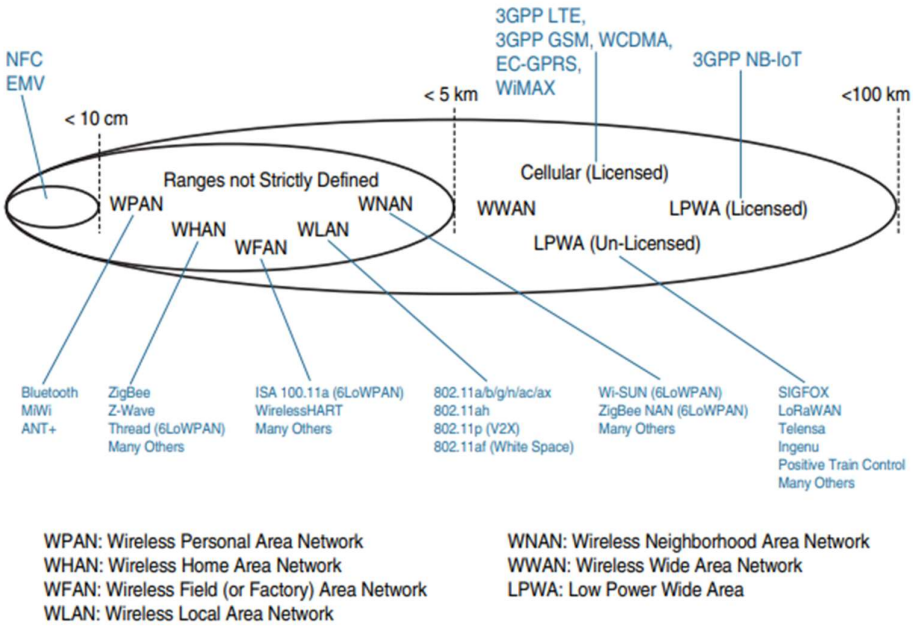


Figure 2-9 *Access Technologies and Distances*

Image Credits – IoT Fundamentals - Networking Technologies, Protocols, and Use Cases for the Internet of Things, Cisco Press, 2017

ALGORITHM

Personal Area Network (PAN): -

I. Choose the required components along with the IoT devices. Rename them conveniently and make connections using suitable connection types.
II. For the door and light configurations, set the network adapter as PT-IOT-NM-CFE.
III. For the router configuration,
   - Assign the IPv4 address and subnet mask in the suitable interface.
   - Turn the port status ON.
   - In the command line interface (CLI), enable the DHCP server to assign the IP address to the IoT devices.

IV. Assign IP configuration to the server by the static method. Turn ON IoT Registration Server Services. Create a registration server account using the desktop web browser.
V. Register each IoT device using remote server configuration.
VI. Add conditions for the devices in the IoT server as follows,

Enabled	Name	Condition	Actions
Yes	RFID-Valid	IoT4 Card ID = 1001	Set IoT4 Status to Valid
Yes	RFID-Invalid	IoT4 Card ID != 1001	Set IoT4 Status to Invalid
Yes	Door-Open	IoT4 Status is Valid	Set IoT2 Lock to Unlock
Yes	Door-Lock	IoT4 Status is Invalid	Set IoT2 Lock to Lock

VII. Discover Bluetooth (Portable Music Player & Speaker) devices and pair the same. Connect them to the home gateway by filling out the SSID and authentication details.

Local Area Network (LAN): -

I. Choose the required components along with the IoT devices. Rename them conveniently and make connections using suitable connection types.
II. Connect the PC and laptop to the router by removing the default physical connection and replacing it with the wireless interface.
III. Navigate to the PC desktop web browser and provide the authentication details. One can make changes such as IP address, DHCP server settings, router password, etc. over here. In the basic wireless settings,
   - Rename your network name to an appropriate one.
   - Set the security mode to WPA2 Personal.
   - Set the encryption type to AES.
   - Set a strong passphrase.

IV. For each end device, configure the SSID and WPA2-PSK authentication under the wireless interface. An alternative in the case of PC and laptop is to connect to the wireless network through desktop PC wireless.

V. If SSID broadcasting is disabled, manually set up a new network profile and choose the required options under the advanced settings. Repeat the same functionalities as in step III.

Wide Area Network (WAN): - Setup Pre-requisite – Integrate LAN and PAN steps to form WAN. Further steps include the following,

I. Create topologies for two separate wireless local area networks (WLANs).
   - Configure the wireless interface on each device with the IP address in the WLANs.
   - Set Wired Equivalent Privacy (WEP) key on each device in the WLANs.

II. Connect the two WLANs with a wired backbone network and configure the IP address on each device in the wired network.

III. Check communication between,
   - Wired Devices
   - Wireless Devices
   - Wired and Wireless Devices

IV. Outputs generated include,
   - Successful ping responses between different devices.
   - Sample screenshots of WEP key setting and IP address configuration.

Network Address Translation (NAT) Mini Demonstration in Wireless WAN: -

I. NAT allows a single device, such as a router, to act as an agent between the Internet (public network) and a local (private) network. This means only a single, unique IP address is required to represent an entire group of computers.

II. In static NAT, a private IP address is mapped to a public IP address, where the public address is always the same IP address (i.e., it has a static address). This allows an internal host such as a web server, to have an unregistered (private) IP address and still be reachable over the internet.

III. Configure the IP address on the interface in both routers, PCs and the server.

IV. Ping to the server from PC1 and then, configure static NAT.

STEP – I	STEP – II	STEP – III
configure terminal	int serial 0/0/0	ip nat inside source static 192.172.1.1 1.0.0.1
int fastEthernet 0/0	ip nat outside	ip route 0.0.0.0 0.0.0.0 serial 0/0/0
ip nat inside	ex	ip nat inside source static 192.172.1.2 1.0.0.1
ex		ip route 0.0.0.0 0.0.0.0 serial 0/0/0

V. Ping command on both PCs and check connectivity.

SCREENSHOTS OF INPUTS & OUTPUTS

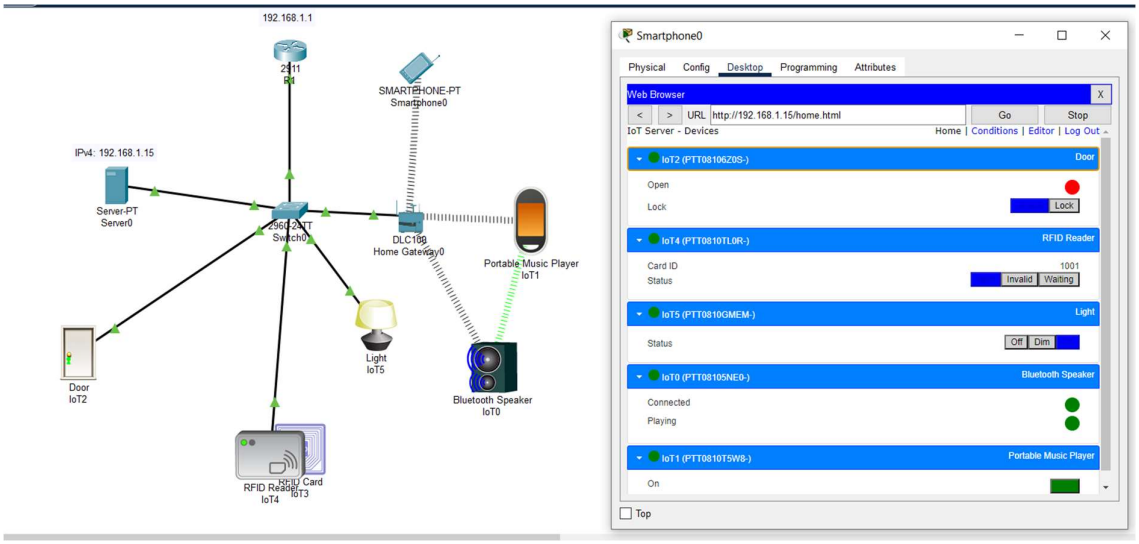


Figure 1 – Wireless Personal Area Network (WPAN)

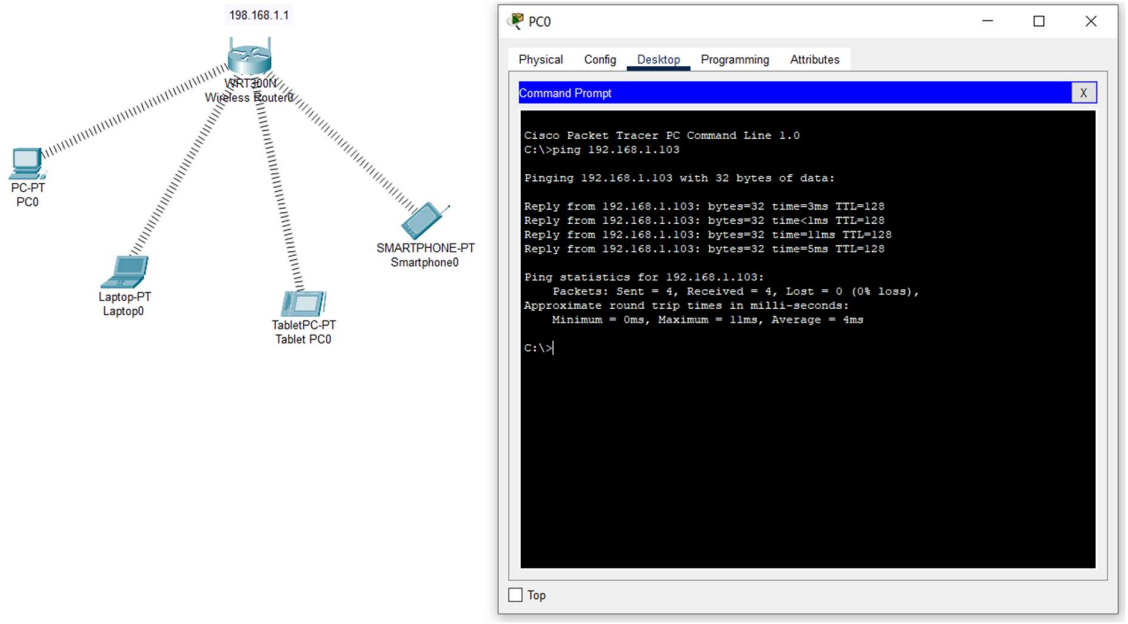


Figure 2 – Wireless Local Area Network (WLAN)

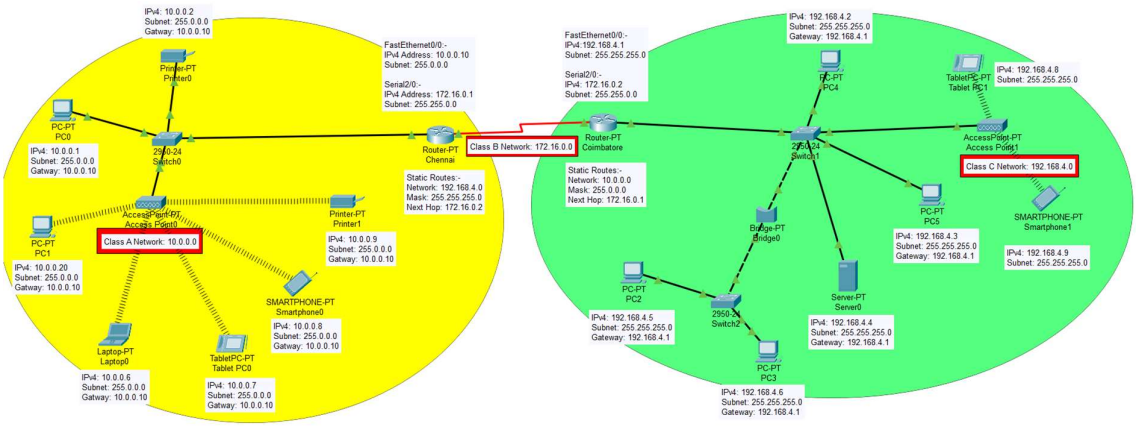


Figure 3 – Wireless Wide Area Network (WWAN)

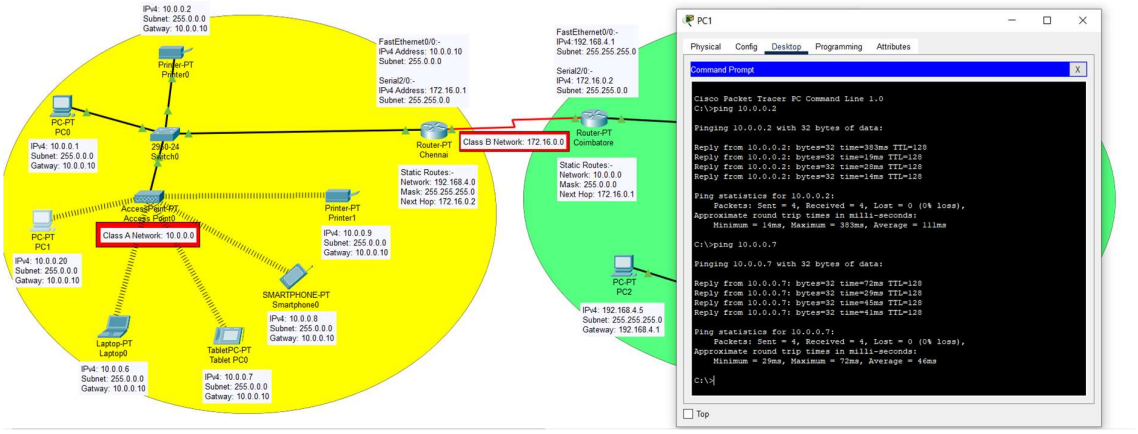


Figure 4 – Wireless Wide Area Network (WWAN)

Class A Network Command Prompt (Ping)

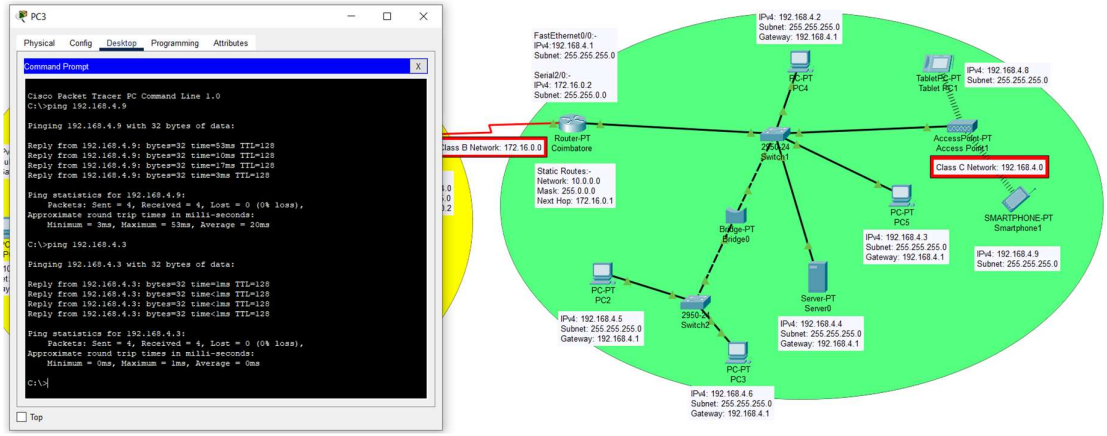


Figure 5 – Wireless Wide Area Network (WWAN)

Class B Network Command Prompt (Ping)

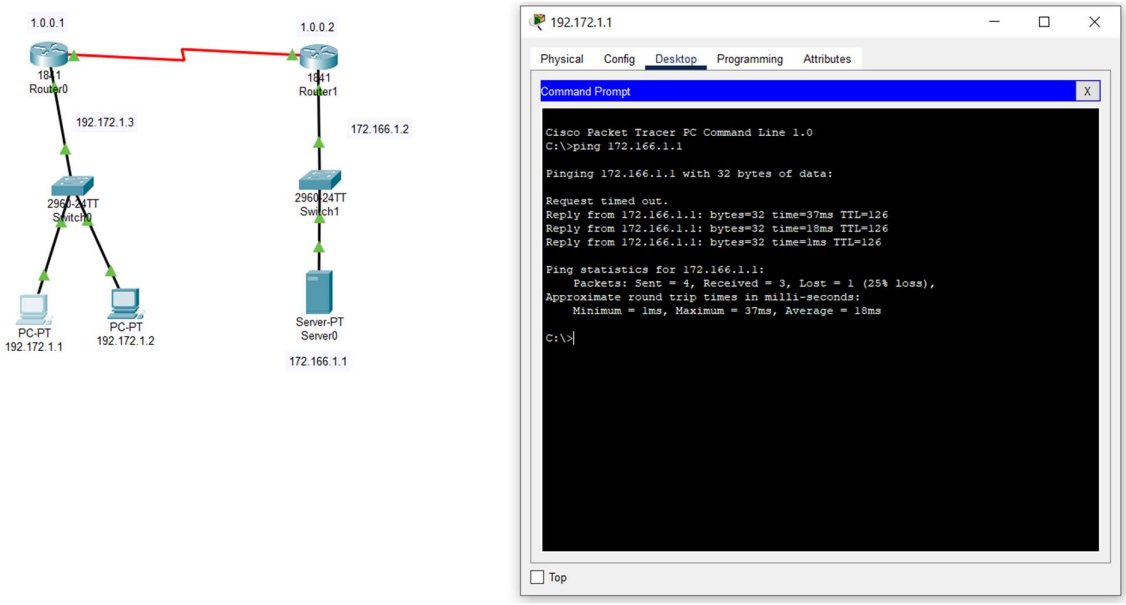


Figure 6 – Network Address Translation (NAT) Mini Demonstration in Wireless WAN

RESULT

Thus, simulated wireless personal, local and wide area networks using Cisco Packet Tracer. All the simulation results were verified successfully.

REFERENCES

1. Computer Networking - A Top-Down Approach, Jim Kurose, Keith Ross, Seventh Edition, Pearson, 2017.
2. Computer Networks - A Systems Approach, Larry L. Peterson, Bruce S. Davie, Fifth Edition, 2012
3. Performance Evaluation of Wide Area Network using Cisco Packet Tracer, Sanam. Nagendram, P. Sai Anil, E.V. S. Pavan, V. Amarendra, International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, No.6, November – December 2019
   (https://www.warse.org/IJATCSE/static/pdf/file/ijatcse38862019.pdf)
4. Wireless Local Area Network using Wireless Router in Packet Tracer – https://www.youtube.com/watch?v=kkZwRv6avfY&ab_channel=JunaidAnjum
5. RFID, Bluetooth and Smartphone using Cisco Packet Tracer – https://www.youtube.com/watch?v=cPewlj0kEtk&ab_channel=DrSKhan
6. Configuration of Wireless Heterogeneous Network using Cisco Packet Tracer – https://www.youtube.com/watch?v=UZwB7JTgYCc&ab_channel=Dr.ShubhangiKharche
7. Static NAT Configuration using Cisco Packet Tracer – https://www.youtube.com/watch?v=SJ9uXKk59GA&ab_channel=TechnoWorld

CONTRIBUTION – TEAM 9

<table>
<tr><th rowspan="2">Name</th><th rowspan="2">Roll Number</th><th colspan="3">Work Done</th></tr>
<tr><th>Cisco Packet Tracer</th><th>Documentation</th><th>Role</th></tr>
<tr><td>B Ambareesh</td><td>CB.EN.U4CCE20006</td><td>Network Address Translation (NAT)</td><td>Others</td><td>Review</td></tr>
<tr><td>Narendran S</td><td>CB.EN.U4CCE20036</td><td rowspan="2">Wireless Local Area Network (WLAN) & Wireless Personal Area Network (WPAN)</td><td rowspan="2">Theory (Text-Only)</td><td rowspan="2">Testing</td></tr>
<tr><td>Narun T</td><td>CB.EN.U4CCE20037</td></tr>
<tr><td>Pabbathi Greeshma</td><td>CB.EN.U4CCE20040</td><td rowspan="2">Wireless Wide Area Network (WWAN)</td><td rowspan="2">Algorithm</td><td rowspan="2">Debugging</td></tr>
<tr><td>Santosh</td><td>CB.EN.U4CCE20053</td></tr>
</table>

EXPERIMENT 6: NETWORK HEALTH MONITORING USING WIRESHARK PACKET SNIFFER

TEAM 9

AIM

To monitor network health using the Wireshark packet sniffer tool.

SOFTWARES REQUIRED

Wireshark 4.0.2 64-bit, the Wireshark Developer Community

Running on 64-bit Windows 10 (21H2), build 19044, with 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (with SSE4.2), with 7926 MB of physical memory, with GLib 2.72.3, with PCRE2 10.40 2022-04-14, with Qt 5.15.2, with Npcap version 1.71, based on libpcap version 1.10.2-PRE-GIT, with c-ares 1.18.1, with GnuTLS 3.6.3, with Gcrypt 1.10.1, with nghttp2 1.46.0, with brotli 1.0.9, with LZ4 1.9.3, with Zstandard 1.5.2, without AirPcap, with light display mode, without HiDPI, with LC_TYPE=English_India.utf8, binary plugins supported.

THEORY & SCREENSHOTS OF I/O

(i) Capturing & Analysing Ethernet Networks

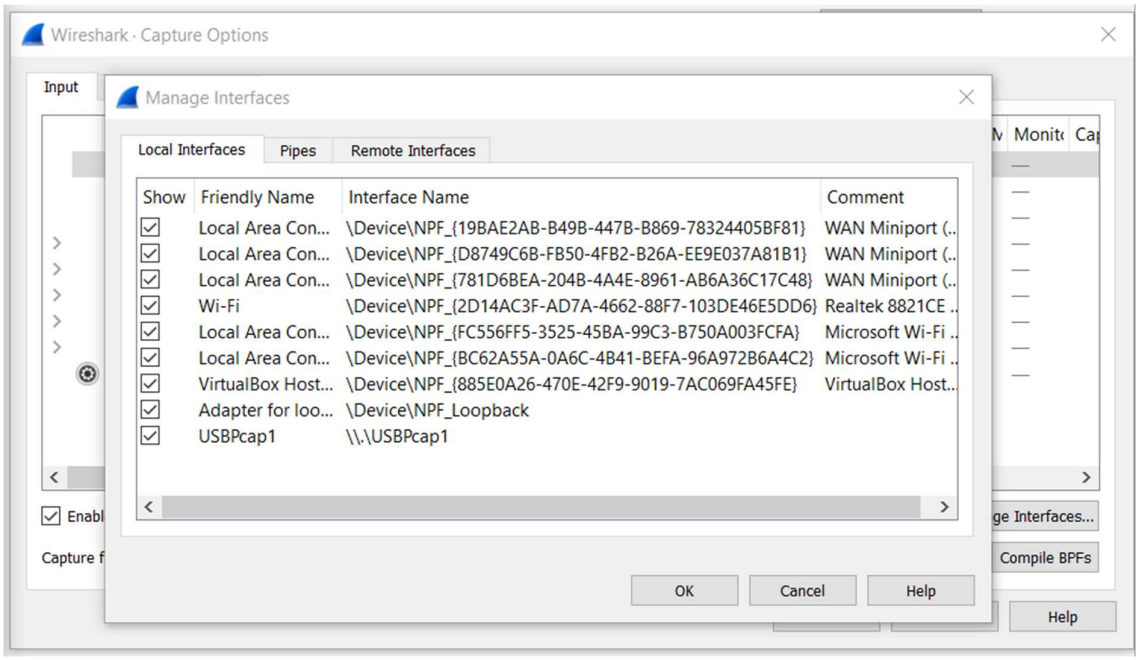


Figure 1 – Manage Interfaces

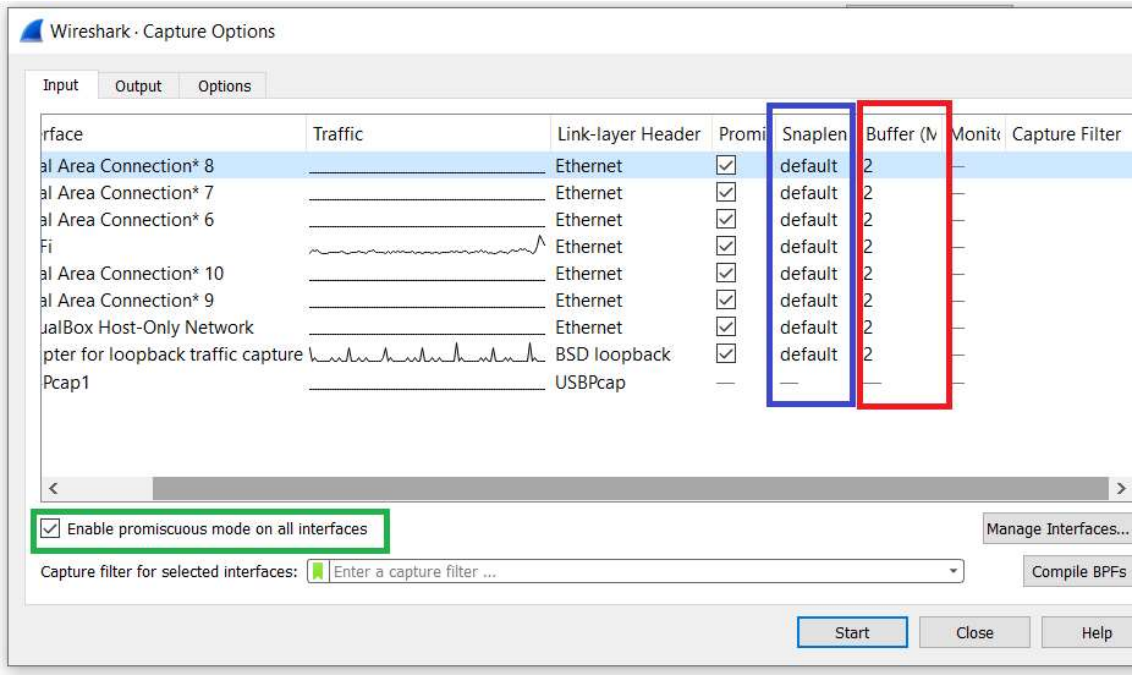


Figure 2 – Capture Options (Input)

As a practice, navigate to manage interfaces in the Wireshark capture options and check only the ones that you would use.

- Snaplength – It captures only a certain amount of data per frame instead of the entire payload, say the first 64 bytes of the frame, which includes most of the information that one needs such as the Ethernet part of the frame, the IP packet information, IP & TCP headers. One needs to be careful with that since there are possibilities of under-capturing certain data.
- Buffer – 2 megabytes of kernel buffer for a capture process. This is usually enough unless working in a high throughput environment.
- Enable promiscuous mode on all interfaces – It allows Wireshark to capture traffic not just to and from itself but also to other machines that are unicasting traffic among each other.

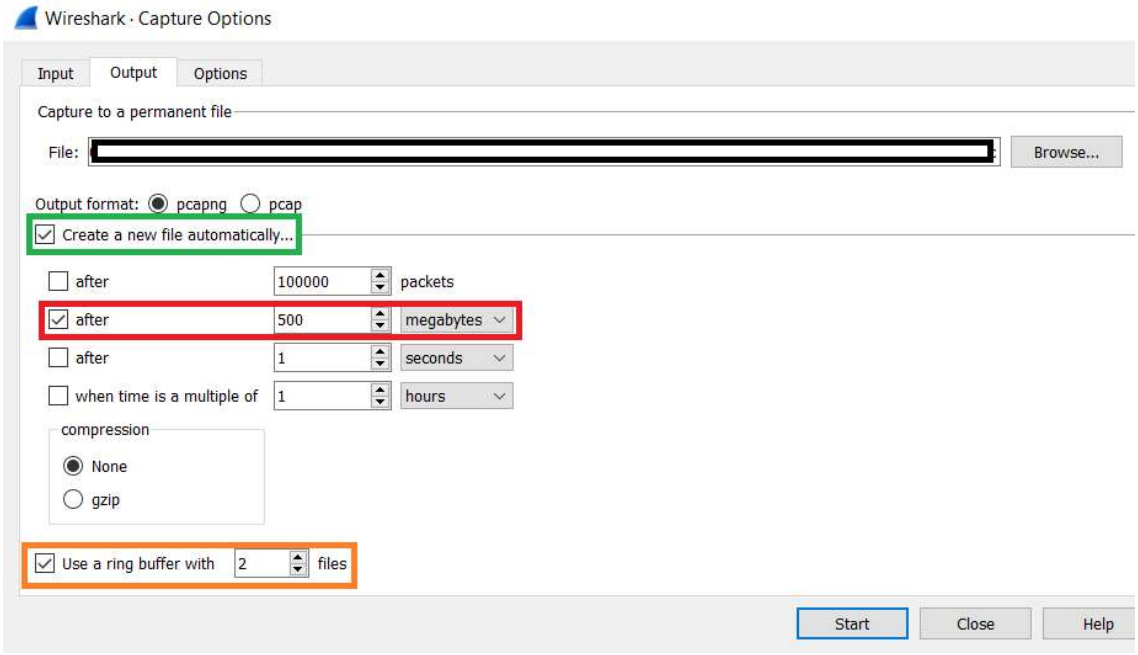


Figure 3 – Capture Options (Output)

Configure the place for Wireshark to save and configure some other settings that make Wireshark traffic easier to read. Two files will be overwritten continuously with 500 megabytes of data.

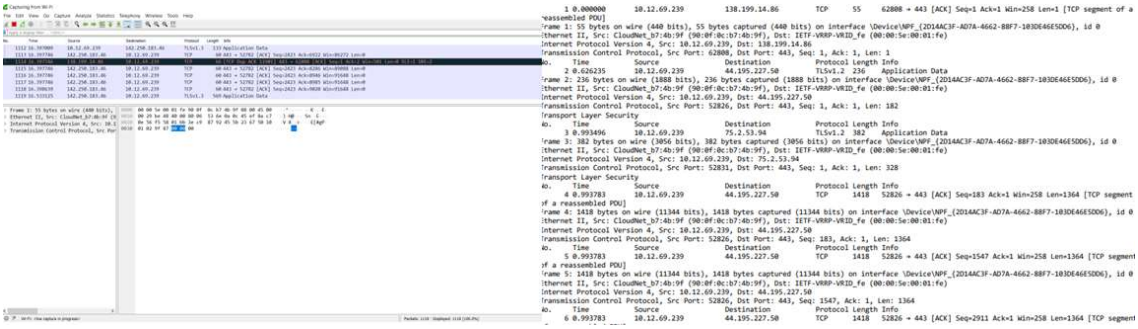


Figure 4 – Capturing from Wi-Fi

(ii) Capturing & Analysing 802.11 Wireless Networks

I. 802.11

- Institute of Electrical and Electronics Engineers' Standard for Wireless LAN communications.
- Standard is to define one medium access control (MAC) and several physical layer (PHY) specifications for wireless connectivity for fixed, portable, and moving stations (STAs) within a local area.

- Also offers regulatory bodies a means of standardizing access to one or more frequency bands for local area communication. Details the OSI Model's Layer 1 and Layer 2 protocols to be used for Wireless LAN.
- Uses radio waves as a physical layer and frames as a data link layer.

II. 802.11 Radio Wave Frequency Bands

- Several types of Wi-Fi standards exist - 802.11b, 802.11g, 802.11n, 802.11ah, etc. All standard occurs in the 2.4 GHz and 5 GHz frequency bands.
- The frequency band is divided into channels of equal bandwidth. To avoid interference, Wi-Fi devices can be configured to use different channels.
- Each standard uses different types of frequency modulation techniques.

III. 802.11 Frames

- Management Frames – Used for authentication, start, tearing-down and maintenance of Wi-Fi communications.
- Control Frames – Used to ease the flow of traffic. For example, signal if the medium is clear to send/receive data frames or acknowledge the receipt of error-free data frames.
- Data Frames – Encapsulate the actual data packets that need to be transmitted.

IV. 802.11 Regulations

- All Wi-Fi devices need to meet legal regulations about transmitter power, channels and interference. The is defined as a regdomain by IEEE. Each country or region (EU) can have its own set of regulations.

V. 802.11 Association Process

- The three 802.11 connection states are –
  - Not authenticated or associated.
  - Authenticated but not yet associated.
  - Authenticated and associated.
- A mobile station must be in the authenticated or associated state before data transmission can occur.

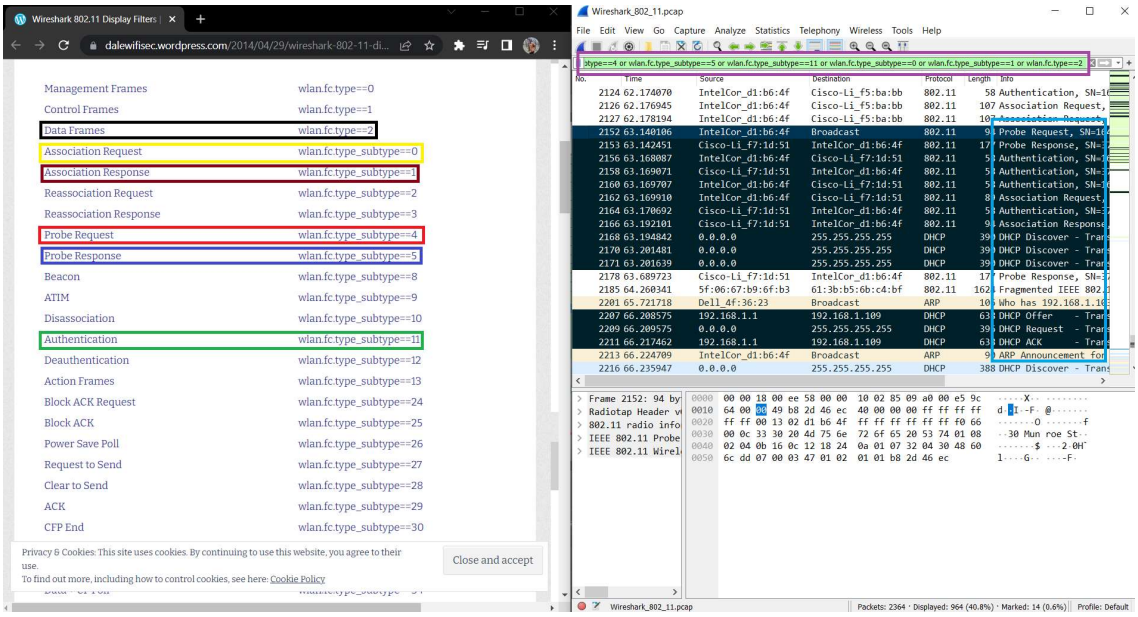


Figure 5 – Sort Packet Captures

Figure 6 – Flow Graph Statistics

(iii) Capturing & Analysing Bluetooth Traffic

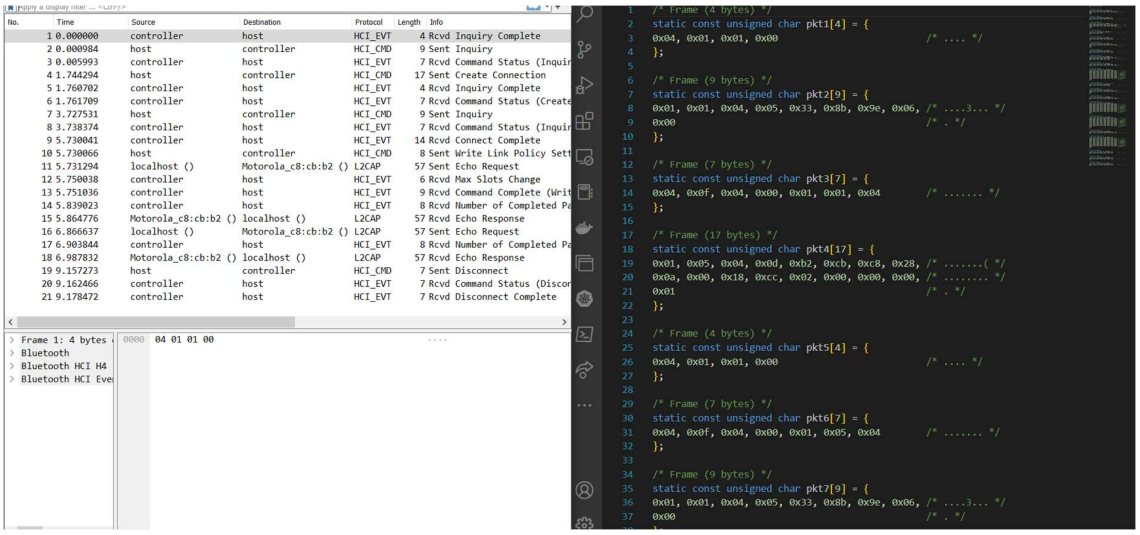


Figure 7 – Bluetooth Sample Capture File

(iv) Capturing & Analysing USB Traffic

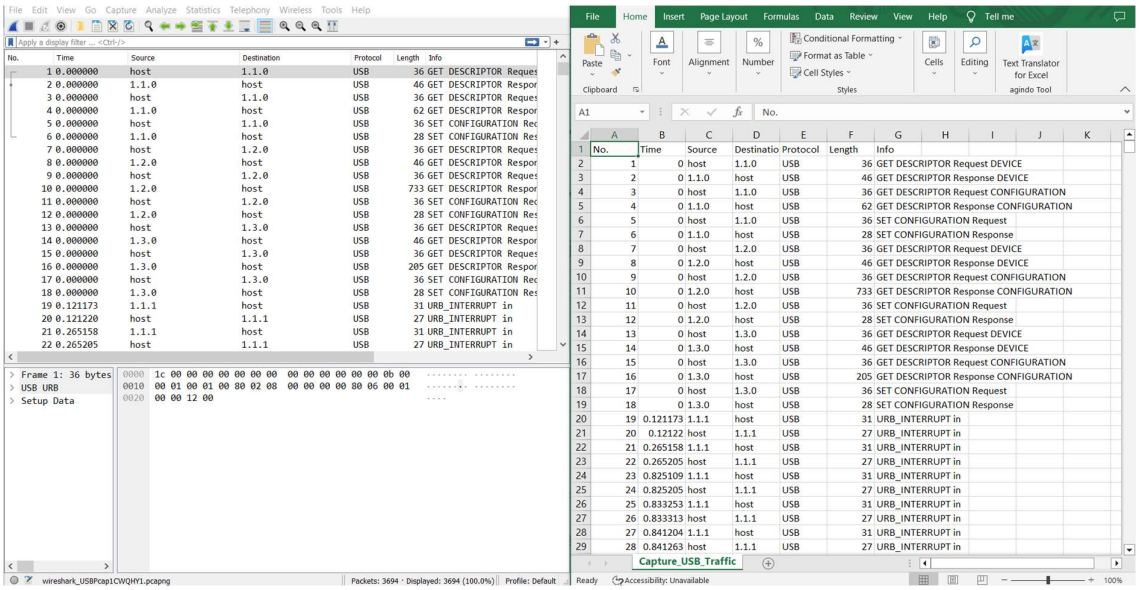


Figure 8 – Capture USB Traffic

RESULT

Thus, monitored network health using the Wireshark packet sniffer. All the simulation results were verified successfully.

REFERENCES

1. Wireshark Lab: 802.11 Wi-Fi v8.0, Supplement to Computer Networking: A Top-Down Approach, 8th ed., J.F. Kurose and K.W. Ross – http://www-net.cs.umass.edu/wireshark-labs/Wireshark_802.11_v8.0.pdf
2. "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012), vol., no., pp.1-3534, 14 Dec. 2016, DOI: 10.1109/IEEESTD.2016.7786995.
3. Understanding the IEEE 802.11 Standard for Wireless Networks – https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-80211.html
4. Wireshark 802.11 Display Filters – https://dalewifisec.wordpress.com/2014/04/29/wireshark-802-11-display-filters-2/
5. Bluetooth Wireshark Wiki – https://wiki.wireshark.org/Bluetooth
6. How to perform network traffic capture with Wireshark – https://www.youtube.com/watch?v=nWvscuxqais&t=306s&ab_channel=ChrisGreer
7. 802.11 Wireless Packet Capture – https://www.youtube.com/watch?v=3wKyUnFN7k8&ab_channel=AtishApajee
8. Capture USB Traffic with Wireshark – https://www.youtube.com/watch?v=Nix-QZ0gkOc&ab_channel=InformationSecurityNewspaper

CONTRIBUTION – TEAM 9

Name	Roll Number	Work Done – Wireshark & Corresponding Documentation
B Ambareesh	CB.EN.U4CCE20006	Capturing & Analysing 802.11 Bluetooth and USB Traffic
Narendran S	CB.EN.U4CCE20036	Capturing & Analysing Ethernet Networks
Narun T	CB.EN.U4CCE20037	
Pabbathi Greeshma	CB.EN.U4CCE20040	Capturing & Analysing 802.11 Wireless Networks
Santosh	CB.EN.U4CCE20053	

Postscript – B Ambareesh (CB.EN.U4CCE20006) gathered study materials and resources about Wireshark apart from the ones mentioned under references, i.e., the basic groundwork.