

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Department of Artificial Intelligence & Data Science

Vision:

Imparting quality education in the field of Artificial Intelligence and Data Science

Mission: • To include the culture of R and D to meet the future challenges in AI and DS.

- To develop technical skills among students for building intelligent systems to solve problems.
- To develop entrepreneurship skills in various areas among the students.
- To include moral, social and ethical values to make students best citizens of country.

Program Educational Outcomes:

1. To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
2. To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
3. To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
4. To prepare the graduates with strong managerial and communication skills to work effectively as individuals as well as in teams.

Program Specific Outcomes:

1. **Professional Skills-** The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, networking, artificial intelligence and data science for efficient design of computer-based systems of varying complexities.
2. **Problem-Solving Skills-** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

3. Successful Career and Entrepreneurship- The ability to employ modern computer languages, environments and platforms in creating innovative career paths to be an entrepreneur and to have a zest for higher studies.

Table of Contents

Contents

1. Guidelines to manual usage
2. Laboratory Objective
3. Laboratory Equipment/Software
4. Laboratory Experiment list
 - 4.1. Experiment No. 1
 - 4.2. Experiment No. 2
 - 4.3. Experiment No. 3
 - 4.4. Experiment No. 4
 - 4.5. Experiment No. 5
 - 4.6. Experiment No. 6
 - 4.7. Experiment No. 7
 - 4.8. Experiment No. 8
 - 4.9. Experiment No. 9
 - 4.10. Experiment No. 10
 - 4.11. Experiment No. 11
 - 4.12. Experiment No. 12
5. Appendix

1. Guidelines to manual usage

This manual assumes that the facilitators are aware of collaborative learning methodologies.

This manual will provide a tool to facilitate the session on Digital Communication modules in collaborative learning environment.

The facilitator is expected to refer this manual before the session.

Program Outcomes:

1. **Engineering knowledge:** An ability to apply knowledge of mathematics, including discrete mathematics, statistics, science, computer science and engineering fundamentals to model the software application.
2. **Problem analysis:** An ability to design and conduct an experiment as well as interpret data, analyze complex algorithms, to produce meaningful conclusions and recommendations.
3. **Design/development of solutions:** An ability to design and development of software system, component, or process to meet desired needs, within realistic constraints such as economic, environmental, social, political, health & safety, manufacturability, and sustainability.
4. **Conduct investigations of complex problems:** An ability to use research-based knowledge including analysis, design and development of algorithms for the solution of complex problems interpretation of data and synthesis of information to provide valid conclusion.
5. **Modern tool usage:** An ability to adapt current technologies and use modern IT tools, to design, formulate, implement and evaluate computer-based system, process, by considering the computing needs, limits and constraints.
6. **The engineer and society:** An ability of reasoning about the contextual knowledge of the societal, health, safety, legal and cultural issues, consequent responsibilities relevant to IT practices.
7. **Environment and sustainability:** An ability to understand the impact of engineering solutions in a societal context and demonstrate knowledge of and the need for sustainable development.
8. **Ethics:** An ability to understand and commit to professional ethics and responsibilities and norms of IT practice.
9. **Individual and team work:** An ability to apply managerial skills by working effectively as an individual, as a member of a team, or as a leader of a team in multidisciplinary projects.
10. **Communication:** An ability to communicate effectively technical information in speech, presentation, and in written form
11. **Project management and finance:** An ability to apply the knowledge of Information Technology and management principles and techniques to estimate time and resources needed to complete engineering project.
12. **Life-long learning:** An ability to recognize the need for, and have the ability to engage in independent and life-long learning.

Course Name: Computer Network

Course Code: (317524)

Course Outcomes

1. CO1: Analyse the requirements of network types, topology and transmission media
2. CO2: Demonstrate error control, flow control techniques and protocols and analyze them.
3. CO3: Demonstrate the subnet formation with IP allocation mechanism and apply various routing algorithms
4. CO4: Develop Client-Server architectures and prototypes
5. CO5: Implement web applications and services using application layer protocols.

CO to PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	-	2	-	2	1	1	1	-	1	-	1
CO2	-	3	-	1	1	-	-	1	-	-	-	-
CO3	3	2	1	1	-	-	-	1	-	-	1	1
CO4	-	1	2	1	1	1	-	-	-	-	-	1
CO5	2	3	-	-	1	-	-	-	1	-	-	-

CO to PSO Mapping:

	PSO1	PSO2	PSO3
CO1	2	1	1
CO2	2	2	1

2. Laboratory Objective

1. To learn computer network hardware and software components
2. To learn computer network topologies and types of networks
3. To develop an understanding of various protocols, modern technologies and applications
4. To learn modern tools for network traffic analysis
5. To learn network programming

3. Laboratory Equipment/Software

1. Computer/PC
2. Cisco Packet Tracer

4. Laboratory Experiment list

List of Assignments	
Group A	
1. Demonstrate the different types of topologies and types of transmission media by using a packet tracer tool.	
2. Setup a wired LAN using Layer 2 Switch. It includes preparation of cable, testing of cable using line tester, configuration machine using IP addresses, testing using PING utility and demonstrating the PING packets captured traces using Wireshark Packet Analyzer Tool.	
3. Use packet Tracer tool for configuration of 3 router network using one of the following protocol RIP/OSPF/BGP	
4. Write a program to implement link state /Distance vector routing protocol to find suitable path for transmission.	
Group B	
5. Write a program using TCP socket for wired network for following a. Say Hello to Each other b. File transfer	
6. Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines	
7. Study and Analyze the performance of HTTP, HTTPS and FTP protocol using Packet tracer tool.	
8. To study the SSL protocol by capturing the packets using Wireshark tool while visiting any SSL secured website (banking, e-commerce etc.).	
9. Illustrate the steps for implementation of S/MIME email security, POP3 through Microsoft® Office Outlook	
10. To study the IPsec (ESP and AH) protocol by capturing the packets using Wireshark tool	
Group C	

11. Installing and configuring DHCP server and assign IP addresses to client machines using DHCP server.

12. Write a program for DNS lookup. Given an IP address input, it should return URL and vice versa.

Assignment 1

Title: Types of topologies and types of transmission media.

Problem Statement:

Demonstrate the different types of topologies and types of transmission media using a packet tracer tool.

Objectives:

1. To understand the working of different types of topologies.
2. To understand the transmission media.
3. To understand the working of GNS3 tool.

Outcomes:

Demonstrate types of technologies and types of transmission media using GNS3 tool.

Tools Required:

Software: GNS3

Procedure:

- Open the GNS3 software
- Drag and drop 4 pcs using End Device Icons on the left.
- Select 8 port switch from switch icon list in the left.
- Make the connections using Straight through Ethernet cables
- Give IP address of the PCs as per table, ping between PCs and observe the transfer of data packets in real and simulation mode.

Theory:

Graphical Network Simulator-3 (GNS3)

GNS3 is a computer program used to learn and practice networking. It lets people simulate and experiment with networks, like the ones you might find in homes, offices, or the internet. With Packet Tracer, you can create, connect, and test devices like computers, routers, and switches to understand how they work together and how data travels through a network.

Types of Topologies:

Bus Topology

A BUS TOPOLOGY IN COMPUTER NETWORKING IS LIKE A SINGLE ROAD WHERE ALL THE HOUSES ARE CONNECTED. IN THIS SETUP, ALL DEVICES IN A NETWORK ARE CONNECTED TO A SINGLE CENTRAL CABLE, WHICH IS LIKE THE ROAD. THE DATA TRAVELS ALONG THIS CABLE, AND ANY DEVICE CAN PICK UP THE DATA IF IT'S MEANT FOR THEM, JUST LIKE A HOUSE CAN RECEIVE MAIL FROM THE MAIL TRUCK PASSING BY.

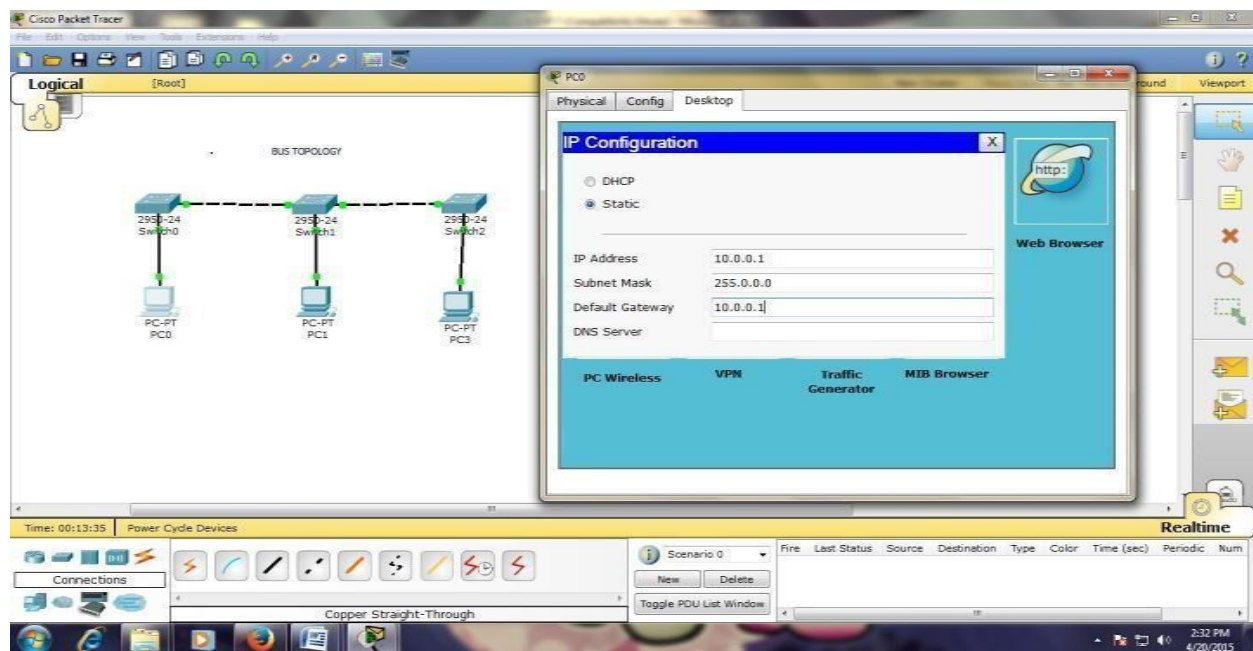


Fig -1: Design of bus topology

Star Topology:

A STAR TOPOLOGY IS LIKE A HUB IN THE CENTRE OF A WHEEL. IN THIS TYPE OF NETWORK SETUP, ALL DEVICES (LIKE COMPUTERS, PRINTERS, OR OTHER GADGETS) ARE CONNECTED DIRECTLY TO A CENTRAL HUB, OFTEN A SWITCH OR A ROUTER. THEY DON'T CONNECT TO EACH OTHER; INSTEAD, THEY ALL LINK TO THE CENTRAL HUB. THIS SETUP MAKES IT EASY TO ADD OR REMOVE DEVICES WITHOUT AFFECTING THE OTHERS. HOWEVER, IF THE CENTRAL HUB FAILS, THE WHOLE NETWORK MIGHT NOT WORK. IT'S LIKE HAVING ALL ROADS LEAD TO A CENTRAL TRAFFIC CIRCLE, WHICH IS THE HEART OF THE NETWORK

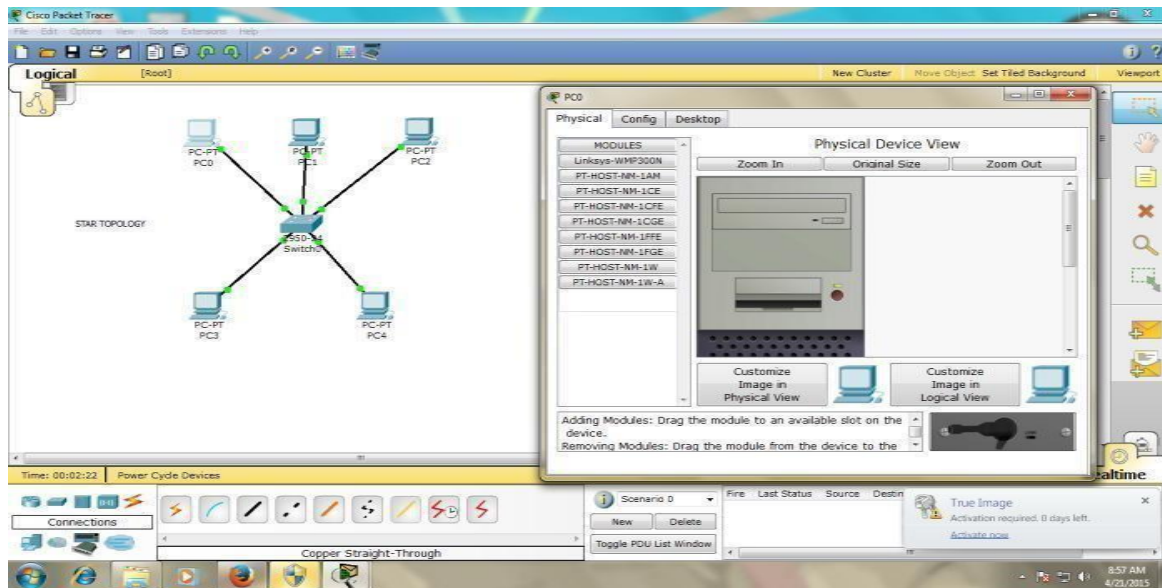


Fig -2: Design of star topology

Mesh Topology:

MESH TOPOLOGY IS A TYPE OF NETWORK SETUP WHERE EACH DEVICE IS CONNECTED TO EVERY OTHER DEVICE IN THE NETWORK. IT'S LIKE A WEB OF CONNECTIONS WHERE ALL DEVICES ARE DIRECTLY LINKED TO ONE ANOTHER. THIS MEANS IF YOU HAVE, LET'S SAY, FIVE DEVICES IN A MESH NETWORK, EACH OF THOSE FIVE DEVICES WILL HAVE CONNECTIONS TO THE OTHER FOUR.

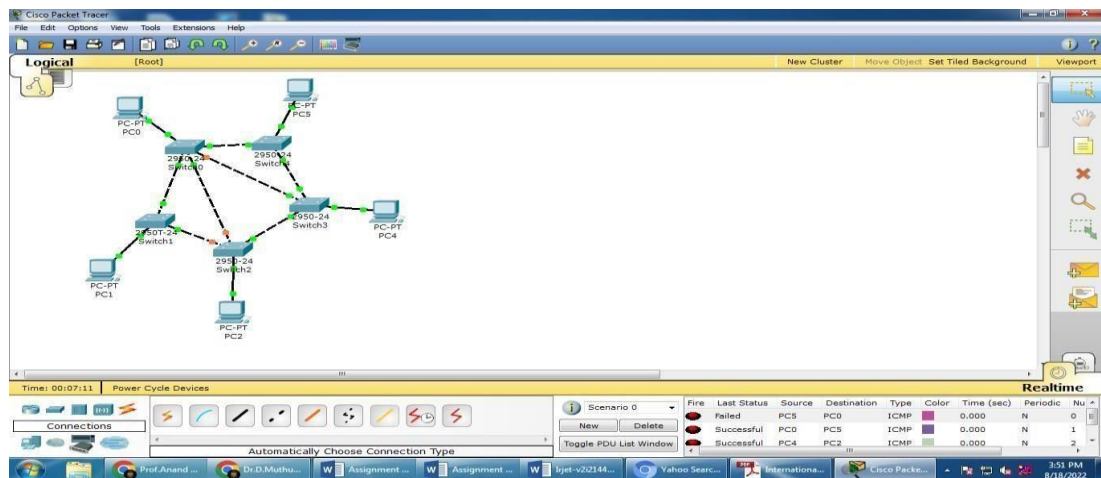


Fig -3: Design of mesh topology

Configuration of component:

Bus topology: To configure the IP address of an interface, we configure all PC one by one click on pc, open DESKTOP window, fill IP Address, Fill subnet mask and default gateway. After that, simulate the network using simulation

Star topology: To configure the IP address of an interface, we configure all PC one by one click on pc, open DESKTOP window, fill IP Address, Fill subnet mask and default gateway. After that, simulate the network using simulation.

Mesh topology: To configure the IP address of an interface, we configure all routers one by one. Click on router, open config window, and fill IP Address of serial port which are connected to router. Fill subnet mask, set clock rate and port status is ON. After that, simulate the network using simulation mode.

Simulation of network topology:

1. Simulation of bus topology:

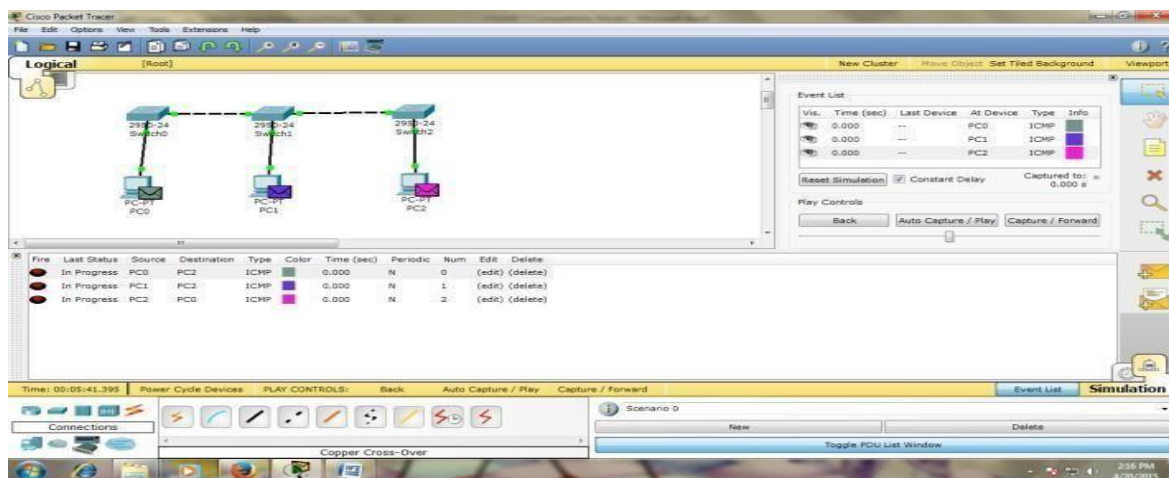


Fig.4 Simulation of Bus Topology

2. Simulation of star topology:

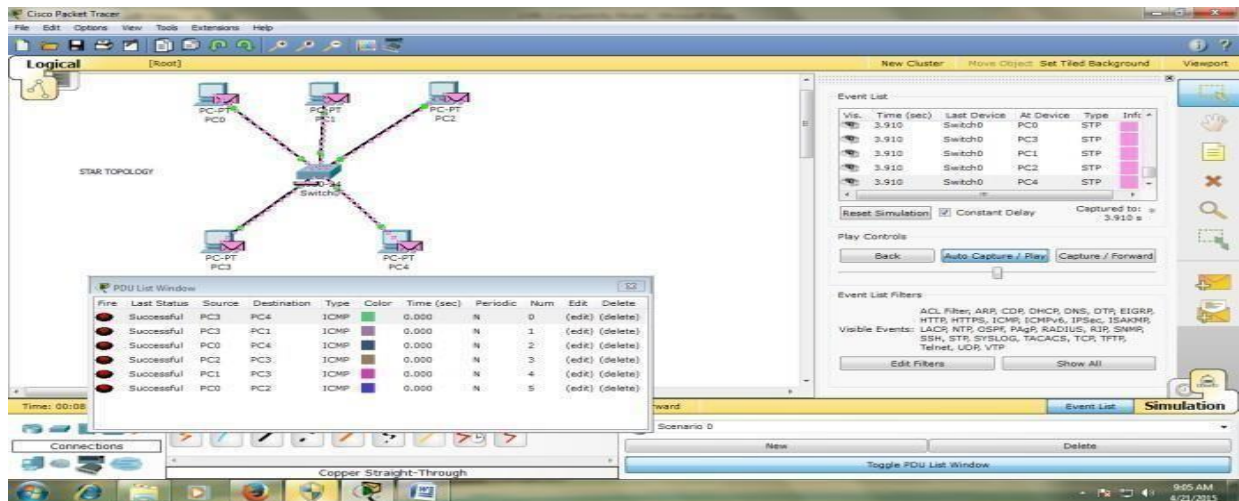


Fig.5 Simulation of Star Topology

3. Simulation of Mesh topology:

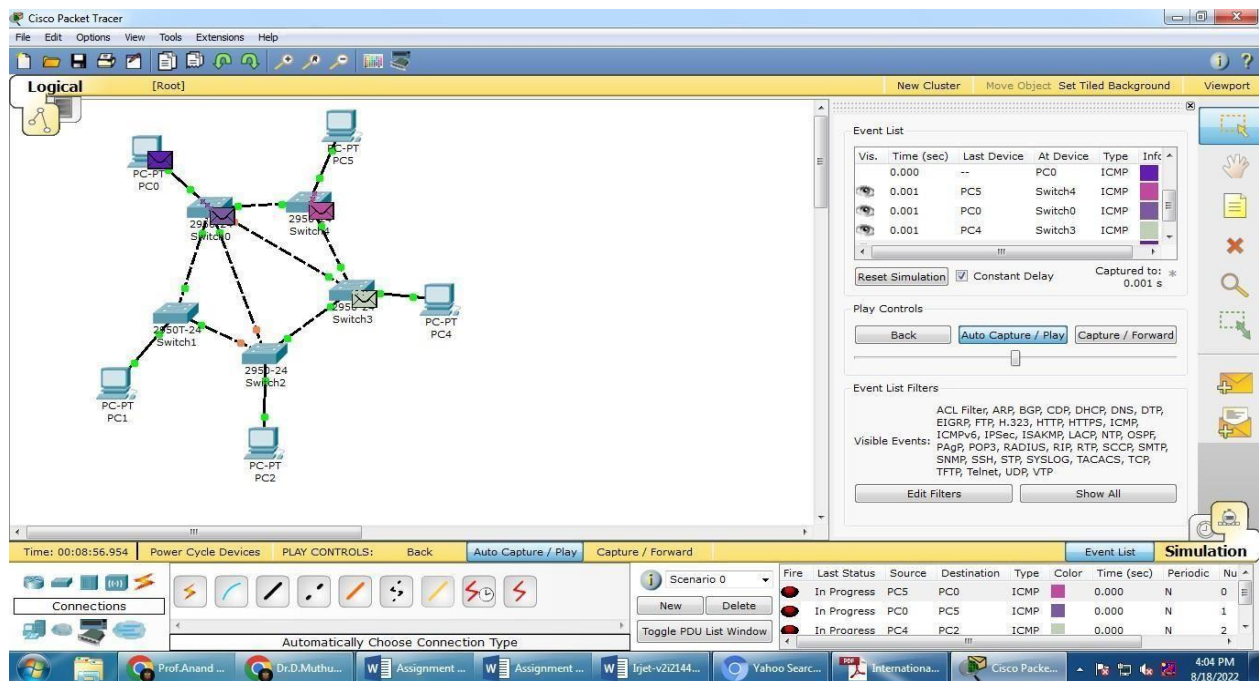


Fig.6 Simulation of Mesh Topology

Conclusion: Thus, we have implemented various topologies in a single network using GNS3. We have used switch configuration and send packet data from one device to another

Assignment 2

Title: Setup a wired LAN using switch.

Problem Statement:

Setup a wired LAN using Layer 2 Switch. It includes preparation of cable, testing of cable using line tester, configuration machine using IP addresses, testing using PING utility and demonstrating the PING packets captured traces using Wireshark Packet Analyzer Tool.

Objectives:

1. To understand the structure and working of various networks including the interconnecting devices used in them.
2. To get hands on experience of making and testing cables.

Outcomes:

Develop and demonstrate a wired LAN for four computers.

Tools Required:

Hardware: Computer, LAN Cards, RJ-45 Connectors, Switch, CAT-5 Cable, Cable tester, Crimping tool, etc.

Software: Open source O.S. and wireshark

Theory:

LAN - Local Area Network

A LAN connects network devices over a relatively short distance. A networked office building, school, or home usually contains a single LAN, though sometimes one building will contain a few small LANs (perhaps one per room), and occasionally a LAN will span a group of nearby buildings.

MAN-Metropolitan Area Network

A network spanning a physical area larger than a LAN but smaller than a WAN, such as a city. A MAN is typically owned and operated by a single entity such as a government body or large corporation.

WAN:

A **wide area network (WAN)** is a telecommunications network or computer network that extends over a large geographical distance. Wide area networks are often established with leased telecommunication circuits. Business, education and government entities use wide area

networks to relay data to staff, students, clients, buyers, and suppliers from various locations across the world. In essence, this mode of telecommunication allows a business to effectively carry out its daily function regardless of location. The Internet may be considered a WAN

What is Network Cabling?

Cable is the medium through which information usually moves from one network device to another. There are several types of cable which are commonly used with LANs. In some cases, a network will utilize only one type of cable, other networks will use a variety of cable types. The type of cable chosen for a network is related to the network's topology, protocol, and size. Understanding the characteristics of different types of cable and how they relate to other aspects of a network is necessary for the development of a successful network. The following sections discuss the types of cables used in networks and other related topics.

- Unshielded Twisted Pair (UTP) Cable
- Shielded Twisted Pair (STP) Cable
- Coaxial Cable
- Fiber Optic Cable
- Cable Installation Guides
- Wireless LANs
- Unshielded Twisted Pair (UTP) Cable

Twisted pair cabling comes in two varieties: shielded and unshielded. Unshielded twisted pair (UTP) is the most popular and is generally the best option for school networks (See fig. 1).

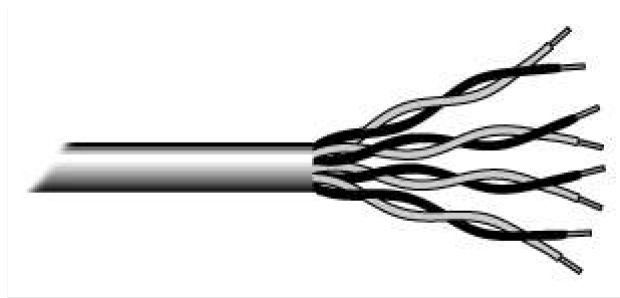


Fig.1. Unshielded twisted pair

The quality of UTP may vary from telephone-grade wire to extremely high-speed cable. The cable has four pairs of wires inside the jacket. Each pair is twisted with a different number of twists per inch to help eliminate interference from adjacent pairs and other electrical devices. The tighter the twisting, the higher the supported transmission rate and the greater the cost per foot. The EIA/TIA (Electronic Industry Association/Telecommunication Industry Association)

has established standards of UTP and rated six categories of wire (additional categories are emerging).

Categories of Unshielded Twisted Pair

Category	Speed	Use
1	1 Mbps	Voice Only (Telephone Wire)
2	4 Mbps	Local Talk & Telephone (Rarely used)
3	16 Mbps	10BaseT Ethernet
4	20 Mbps	Token Ring (Rarely used)
5	100Mbps (2 pair)	100BaseT Ethernet
6	10,000 Mbps	Gigabit Ethernet

Unshielded Twisted Pair Connector

The standard connector for unshielded twisted pair cabling is an RJ-45 connector. This is a plastic connector that looks like a large telephone-style connector (See fig. 2). A slot allows the RJ-45 to be inserted only one way. RJ stands for Registered Jack, implying that the connector follows a standard borrowed from the telephone industry. This standard designates which wire goes with each pin inside the connector.



Fig. 2. RJ-45 connector

Shielded Twisted Pair (STP) Cable

Shielded Twisted Pair (STP) cable is a type of wire that's commonly used to connect electronic devices and create networks. It consists of pairs of insulated wires twisted together, just like in regular Ethernet cables. What makes it different is the shielding, which is like a protective layer around the twisted pairs.

The shielding in STP cable is typically made of metal, such as aluminium or copper. It acts like a protective Armor, guarding the wires inside from interference or electrical "noise" that can disrupt data signals. This noise can come from various sources, like other electronic devices or power cables.

Coaxial Cable

Coaxial cable is a type of wire used to transmit signals, like TV or internet, from one place to another. It looks like a thick cable with a metal wire running through the center, and it's surrounded by insulating material and a metal shield. The central wire carries the signal, while the shield helps protect it from interference and signal loss. Coaxial cables are commonly used for connecting things like cable TV, satellite dishes, and internet modems to your devices because they are good at carrying high-frequency signals over long distances.



Coaxial Cable Connectors

The most common type of connector used with coaxial cables is the Bayonet-Neill-Concelman (BNC) connector (See fig. 4). Different types of adapters are available for BNC connectors, including a T-connector, barrel connector, and terminator. Connectors on the cable are the weakest points in any network. To help avoid problems with your network, always use the BNC connectors that crimp, rather than screw, onto the cable.



Fig:4 BNC Connector

Fiber Optic Cable

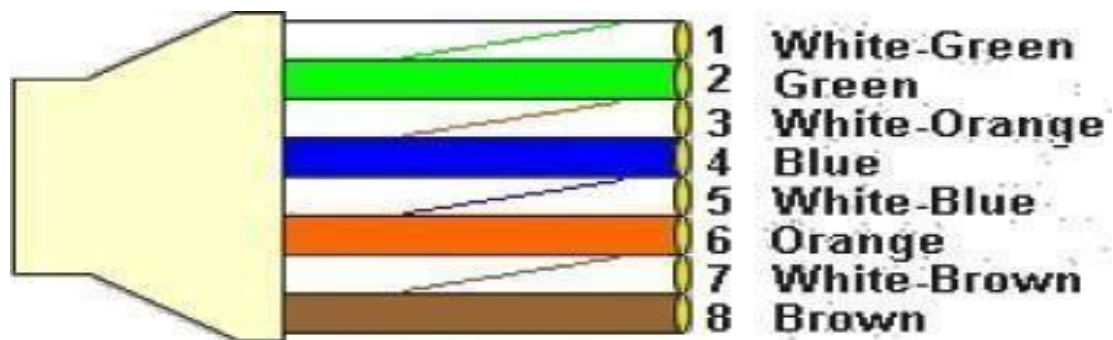
A Fiber optic cable is a special kind of cable used to send information using light instead of electricity. Inside the cable, there is a very thin strand of glass or plastic that can carry light signals. These signals represent data, like internet, phone calls, or videos. The light travels super-fast through the cable, allowing for very quick and efficient communication. Fiber optic cables are like high-speed highways for information, making them a popular choice for transmitting data over long distances and at high speeds.



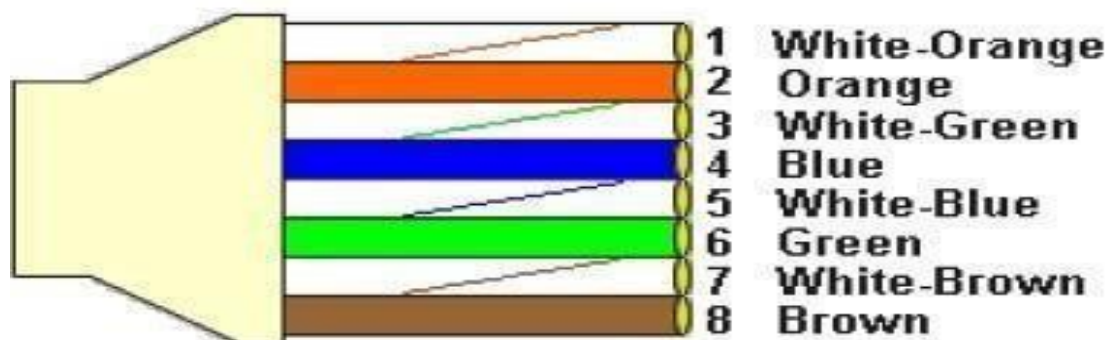
Fig. 5: Fiber optic cable

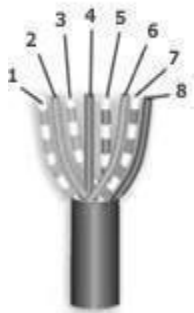
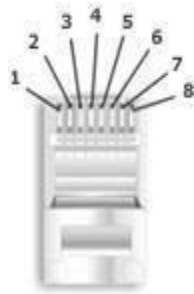
Pairing Rules and Color Code:-

Colour Code

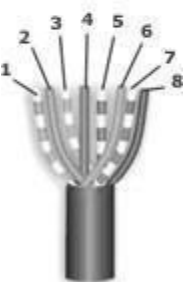


568A CABLE END



Straight:-

Pair #	Wire	Pin #
1-White/Blue	White/Blue	5
	Blue	4
2-Wht./Orange	White/Orange	1
	Orange	2
3-White/Green	White/Green	3
	Green	6
4-White/Brown	White/Brown	7
	Brown	8

Cross

Pair #	Wire	Pin #
1-White/Blue	White/Blue	5
	Blue	4
2-White/Green	White/Green	1
	Green	2
3-White/Orange	White/Orange	3
	Orange	6
4-White/Brown	White/Brown	7
	Brown	8

Connections among devices:-

Node to Node -Straight -Cross,
 Switch to Node -Straight - Straight,
 Switch to Switch - Straight

How to Crimp a Cat 5 cable with RJ 45 Connector:-

1. Skin off the cable jacket approximately 1" or slightly more.
2. Un-twist each pair, and straighten each wire between the fingers.
3. Place the wires in the order of one of the two diagrams shown above. Bring all of the wires together, until they touch.
4. At this point, recheck the wiring sequence with the diagram.
5. Optional: Make a mark on the wires at 1/2" from the end of the cable jacket.
6. Hold the grouped (and sorted) wires together tightly, between the thumb, and the forefinger.
7. Cut all of the wires at a perfect 90 degree angle from the cable at 1/2" from the end of the cable jacket. This is a very critical step. If the wires are not cut straight, they may not all make contact. We suggest using a pair of scissors for this purpose.
8. Conductors should be at a straight 90 degree angle, and be 1/2" long, prior to insertion into the connector.
9. Insert the wires into the connector (pins facing up).
10. Push moderately hard to assure that all of the wires have reached the end of the connector. Be sure that the cable jacket goes into the back of the connector by about 3/16".
11. Place the connector into a crimp tool, and squeeze hard so that the handle reaches its full swing.
12. Repeat the process on the other end. For a straight through cable, use the same wiring.
13. Use a cable tester to test for proper continuity.

Cable Preparation:



Orange – 1 & 2 Green – 3 & 6
Blue- 4 & 5 Brown- 7 & 8

PING Command:

The "ping" command is like saying "Hello, are you there?" to another computer. When you use the ping command on your computer, it sends a small message to another computer over a network. If the other computer is working and connected, it will reply with a message back. This helps you check if a computer or device is online and how fast it can communicate with your computer. It's a simple way to test the connection and measure the speed between your computer and another one on a network, like the internet.

```
PS C:\> ping google.com

Pinging google.com [216.58.195.142] with 32 bytes of data:
Reply from 216.58.195.142: bytes=32 time=61ms TTL=128
Reply from 216.58.195.142: bytes=32 time=61ms TTL=128
Reply from 216.58.195.142: bytes=32 time=56ms TTL=128
Reply from 216.58.195.142: bytes=32 time=63ms TTL=128

Ping statistics for 216.58.195.142:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 56ms, Maximum = 63ms, Average = 60ms
```

Conclusion: Thus, we have studied wired LAN setup and connection.

Assignment 3

Problem Definition:

- **Problem Definition: Use packet Tracer tool for configuration of 3 router network using one of the following protocol RIP/OSPF/BGP**

- **Prerequisite:**

1. Routing Protocols.
2. Basics of Packet Tracer.

- **Learning Objectives:**

1. To Understand Simulation Tool.
2. Should Able to Configure Routing Protocols

1.3 Theory:

1.3.1 Introduction

ROUTING INFORMATION PROTOCOL:

Routing Information Protocol, or RIP in short, is like a set of directions for computers in a network. Imagine you have a GPS that tells you the best way to reach a destination. RIP is similar, but for computers in a network.

RIP helps routers (devices that decide how to send data around a network) figure out the quickest path to send data from one place to another. It does this by sharing information about all the different routes and how far away they are. Routers use this information to make decisions about where to send data, so it reaches its destination efficiently.

RIP is a simple and old-fashioned way for computers in a network to talk to each other about the best routes, and it's like the network's own version of a GPS.

- **OPEN SHORTEST PATH FIRST: (OSPF)**

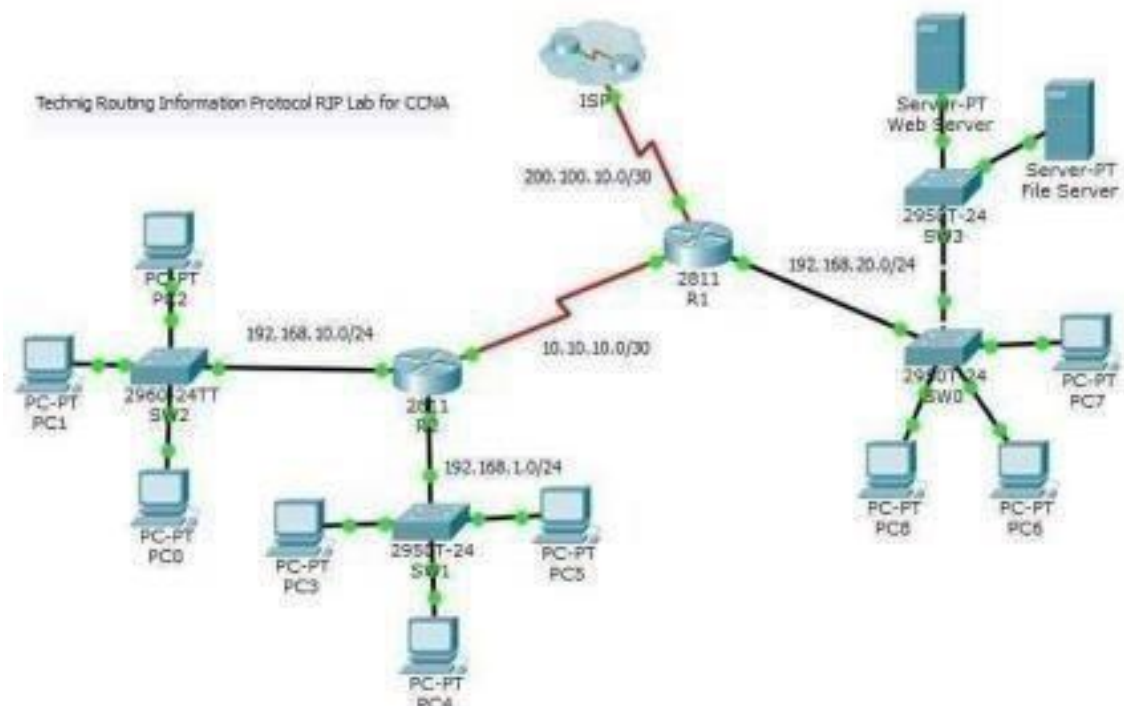
- OSPF is an interior gateway protocol (IGP) for routing Internet Protocol (IP) packets solely within a single routing domain, such as an autonomous system. It gathers link state information from available routers and constructs a topology map of the network. The topology is presented as a routing table to the Internet layer which

Department of Artificial Intelligence and Data Science Engineering, routes

packets based solely on their destination IP address.

- Open Shortest Path First (OSPF) is a routing protocol for Internet Protocol (IP) networks. It uses a link state routing (LSR) algorithm and falls into the group of interior gateway protocols (IGPs), operating within a single autonomous system (AS).

CONFIGURE ROUTING INFORMATION PROTOCOL (RIP)



Open the router 1 (**R1**) which is the main router connected to ISP router. Do the following command for RIP Routing.

```
R1>enable
```

```
R1#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
R1(config)#router rip
```

```
R1(config-router)#version 2
```

```
R1(config-router)#network 200.100.10.0 DHCP
```

message types:

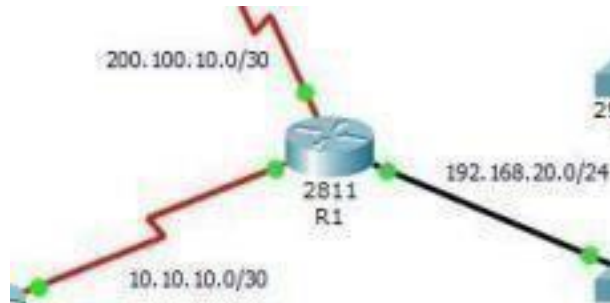
```
R1(config-router)#network 192.168.20.0
```

```
R1(config-router)#network 10.10.10.0
```

```
R1(config-router)#
```


After enabling router with enable command then go to privileged mode with configure terminal command. Now with router rip command, enable routing for all routers. The version 2 Command, configure routing information protocol with version two. And next set

all network id like the above network command. I have set all three network which connect directly to R1.



Now go to router R2 and configure routing protocol the same as router R1. On router 2 you must assign the network ids of all connected network the R2.

```
R2>enable
```

```
R2#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
R2(config)#router rip
```

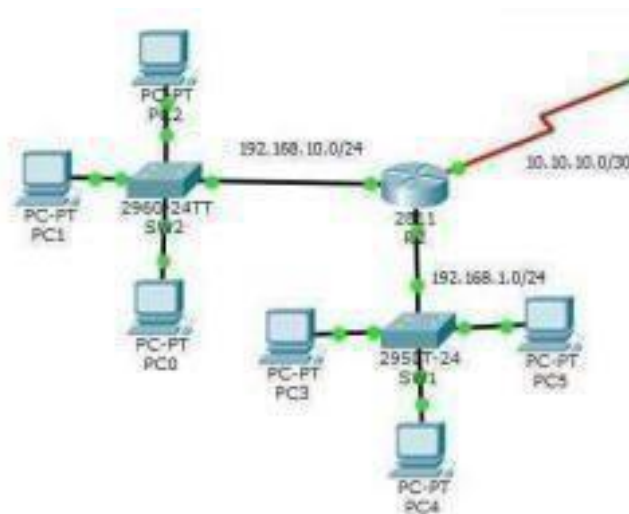
```
R2(config-router)#version 2
```

```
R2(config-router)#network 10.10.10.0
```

```
R2(config-router)#network 192.168.10.0
```

```
R2(config-router)#network 192.168.1.0
```

```
R2(config-router)#
```



For ISP router, just enter the network id 200.100.10.0, because only one network connected to ISP router.

```
ISP>enable
```

```
ISP#configure terminal
```

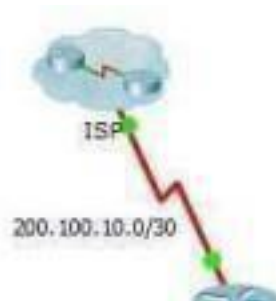
Enter configuration commands, one per line. End with CNTL/Z.

```
ISP(config)#router rip
```

```
ISP(config-router)#version 2
```

317524 Computer Network

```
ISP(config-router)#network 200.100.10.0 ISP(config-router)#
```



■ CONFIGURE OSPF ROUTING PROTOCOL

In the router R1 configure OSFP routing with Router ospf command.

```
R1>enable
```

```
R1#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
R1(config)#router ospf 1
```

```
R1(config-router)#network 20.10.10.0 0.0.0.3 area 0
```

```
R1(config-router)#network 10.10.10.0 0.0.0.3 area 0
```

```
R1(config-router)#network 10.10.10.4 0.0.0.3 area 0 R1(config-router)
```

The router OSPF command is enable OSPF routing on the router, and the 1 before OSFP is the process ID of the OSFP Protocol. You can set different process id from “1-65535” for each router. The network command with network ID “network 20.10.10.0” is the network identifier, and the “0.0.0.3” is the wildcard mask of 20.10.10.0 network. Wildcard

mask determine which interfaces to advertise, because OSPF advertise interfaces, not networks.

Now go to Router R3 and configure with the following commands.

```
R3>enable
```

```
R3#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
R3(config)#router ospf 1
```

```
R3(config-router)#network 192.168.1.0 0.0.0.255 area 0
```

```
R3(config-router)#network 10.10.10.0 0.0.0.3 area 0 Don?
```

So do the following for router R2.

```
R2>enable
```

```
R2#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
R2(config)#router ospf 1
```

```
R2(config-router)#network 192.168.10.0 0.0.0.255 area 0
```

```
R2(config-router)#network 10.10.10.4 0.0.0.3 area 0
```

OK, OSPF routing configuration has been finished successfully, now test your network whether they can ping with each other or not.

Conclusion: Hence, we have studied the packet tracer properly.

Assignment-4

Title: Write a program to implement link state /Distance vector routing protocol to find suitable path for transmission.

Objective: To understand working of Distance vector routing protocol.

Prerequisite:

1. Shortest path finding

2. Classification of routing Algorithm **Learning Objectives:**

1. Understand the concept Distance vector routing

2. Understand the Concept of Routing Algorithms

Theory:

Introduction:

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

Bellman Ford Basics – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors' distance vectors. Information kept by DV router -

- Each router has an ID
- Associated with each link connected to a router,
There is a link cost (static or dynamic).
- Intermediate hops Distance Vector Table Initialization -
- Distance to itself = 0

1. Distance to ALL other routers = infinity number. A router transmits its distance vector to each of its neighbour's in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbour's.
3. A router recalculates its distance vector when:
 - a. It receives a distance vector from a neighbour containing different information than before.
 - b. It discovers that a link to a neighbour has gone down.

The DV calculation is based on minimizing the cost to each destination

$D_x(y)$ = Estimate of least cost from x to y

$C(x,v)$ = Node x knows cost to each neighbour v

$D_x = [D_x(y): y \in N]$ = Node x maintains distance vector

Node x also maintains its neighbours' distance vectors –

For each neighbour v, x maintains $D_v = [D_v(y): y \in N]$

Distance Vector Routing:

- It is a dynamic routing algorithm in which each router computes distance between itself and each possible destination i.e. its immediate neighbours.
- The router share its knowledge about the whole network to its neighbours and accordingly updates table based on its neighbours.
- The sharing of information with the neighbours takes place at regular intervals.
- It makes use of Bellman Ford Algorithm for making routing tables.
- Problems – Count to infinity problem which can be solved by splitting horizon.
 - Good news spread fast and bad news spread slowly.
 - Persistent looping problem i.e. loop will be there forever.

Link State Routing:

- It is a dynamic routing algorithm in which each router shares knowledge of its neighbours with every other router in the network.
- A router sends its information about its neighbours only to all the routers through flooding.
- Information sharing takes place only whenever there is a change.
- It makes use of Dijkstra's Algorithm for making routing tables.
- Problems – Heavy traffic due to flooding of packets.
 - Flooding can result in infinite looping which can be solved by using Time to live (TTL) field.

Conclusion: Hence we have studied distance vector algorithm to find suitable path for transmission.

Program Code:

```
#include <iostream>

using namespace std;

struct node {
    int dist[20];
    int from[20];
} route[10];

int main()
{
    int dm[20][20], no;

    cout << "Enter no of nodes." << endl;
    cin >> no;
    cout << "Enter the distance matrix:" << endl;
    for (int i = 0; i < no; i++) {
        for (int j = 0; j < no; j++) {
            cin >> dm[i][j];

            /* Set distance from i to i as 0 */
            dm[i][i] = 0;
            route[i].dist[j] = dm[i][j];
            route[i].from[j] = j;
        }
    }

    int flag;
    do {
        flag = 0;
        for (int i = 0; i < no; i++) {
```

```

        for (int j = 0; j < no; j++) {
            for (int k = 0; k < no; k++) {
                if ((route[i].dist[j]) > (route[i].dist[k] + route[k].dist[j])) {
                    route[i].dist[j] = route[i].dist[k] + route[k].dist[j];
                    route[i].from[j] = k;
                    flag = 1;
                }
            }
        }
    }
} while (flag);

for (int i = 0; i < no; i++) {
    cout << "Router info for router: " << i + 1 << endl;
    cout << "Dest\tNext Hop\tDist" << endl;
    for (int j = 0; j < no; j++)
        printf("%d\t%d\t%d\n", j+1, route[i].from[j]+1, route[i].dist[j]);
    }
return 0;
}

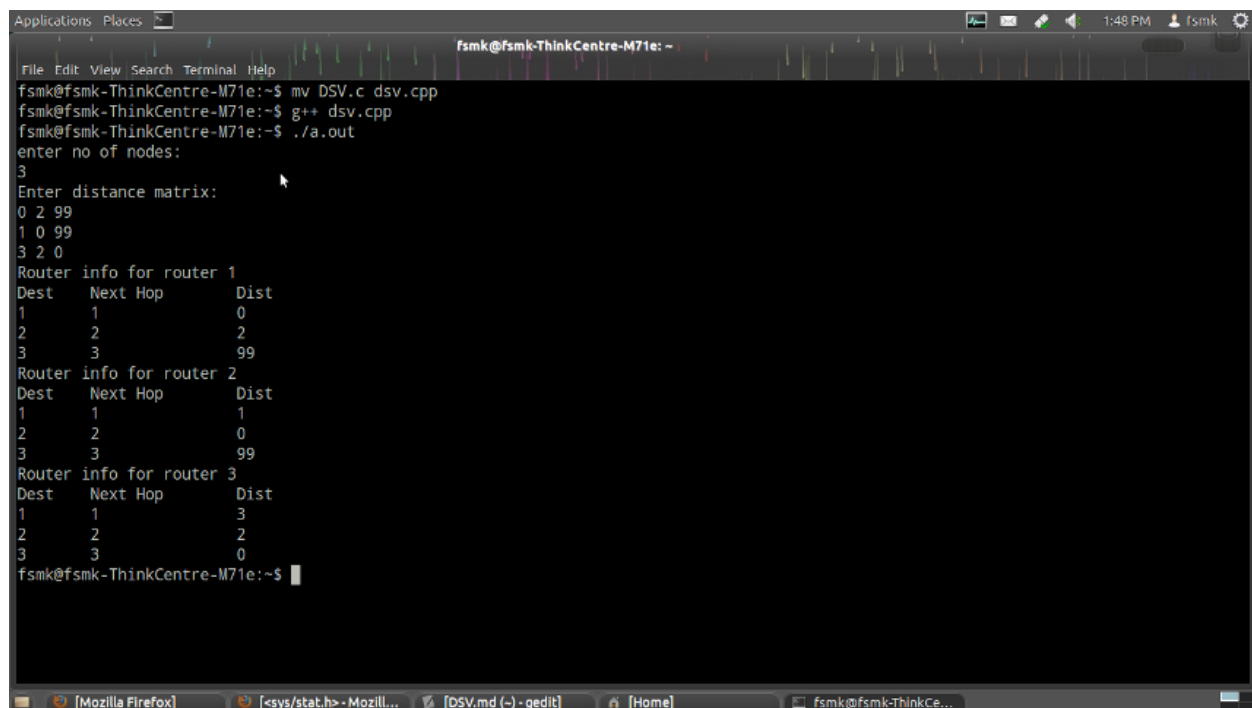
```

Output:

Commands for execution:-

Open a terminal.
 Change directory to the file location.
 Run g++ filename.cpp
 If there are no errors, run ./a.out

Screenshots:-



A terminal window on a Linux system (fsmk@fsmk-ThinkCentre-M71e) showing the execution of a C++ program for Dijkstra's algorithm. The user enters the number of nodes as 3 and provides a distance matrix. The program then outputs the shortest path information for three routers.

```
fsmk@fsmk-ThinkCentre-M71e:~$ mv DSV.c dsv.cpp
fsmk@fsmk-ThinkCentre-M71e:~$ g++ dsv.cpp
fsmk@fsmk-ThinkCentre-M71e:~$ ./a.out
enter no of nodes:
3
Enter distance matrix:
0 2 99
1 0 99
3 2 0
Router info for router 1
Dest  Next Hop  Dist
1      1         0
2      2         2
3      3        99
Router info for router 2
Dest  Next Hop  Dist
1      1         1
2      2         0
3      3        99
Router info for router 3
Dest  Next Hop  Dist
1      1         3
2      2         2
3      3         0
fsmk@fsmk-ThinkCentre-M71e:~$
```

The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The top status bar shows the time as 1:48 PM and the user as fsmk. The bottom taskbar includes icons for Mozilla Firefox, a terminal window, a file editor (gedit) for DSV.md, and a home button.

Assignment 5

Problem Definition:

Write a program using TCP socket for wired network for following

- a. Say Hello to Each other
- b. File transfer

Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Objective:

Set up connection TCP between two nodes.

- a. Say Hello to Each other
- b. File transfer

Theory:**Socket Programming:**

The Berkeley socket interface, an API, allows communications between hosts or between processes on one computer, using the concept of a socket. It can work with many different I/O devices and drivers, although support for these depends on the operating system implementation. This interface implementation is implicit for TCP/IP, and it is therefore one of the fundamental technologies underlying the Internet. It was first developed at the University of California, Berkeley for use on Unix systems. All modern operating systems now have some implementation of the Berkeley socket interface, as it has become the standard interface for connecting to the Internet. Programmers can make the socket interfaces accessible at three different levels, most powerfully and fundamentally at the RAW socket level. Very few applications need the degree of control over outgoing communications that this provides, so RAW sockets support was intended to be available only on computers used for developing Internet related technologies.

TCP

TCP provides the concept of a connection. A process creates a TCP socket by calling the `socket()` function with the parameters `PF_INET` or `PF_INET6` and `SOCK_STREAM`.

Server:

Setting up a simple TCP server involves the following steps:

1. Creating a TCP socket, with a call to `socket()`.
2. Binding the socket to the listen port, with a call to `bind()`. Before calling `bind()`, a programmer must declare a `sockaddr_in` structure, clear it (with `bzero()` or `memset()`), and the `sin_family` (`AF_INET` or `AF_INET6`), and fill its `sin_port` (the listening port, in network byte order) fields. Converting a short int to network byte order can be done by calling the function `htons()` (host to network short).
3. Preparing the socket to listen for connections (making it a listening socket), with a call to `listen()`.
4. Accepting incoming connections, via a call to `accept()`. This blocks until an incoming connection is received, and then returns a socket descriptor for the accepted connection. The initial descriptor remains a listening descriptor, and `accept()` can be called again at any time with this socket, until it is closed.
5. Communicating with the remote host, which can be done through `send()` and `recv()`.
6. Eventually closing each socket that was opened, once it is no longer needed, using `close()`.

Note that if there were any calls to `fork()`, each process must close the sockets it knew about (the kernel keeps track of how many processes have a descriptor open), and two processes should not use the same socket at once.

Client:

Setting up a TCP client involves the following steps:

1. Creating a TCP socket, with a call to `socket()`.
2. Connecting to the server with the use of `connect`, passing a `sockaddr_in` structure with the

sin_family set to AF_INET or AF_INET6, sin_port set to the port the endpoint is listening (in network byte order), and sin_addr set to the IPv4 or IPv6 address of the listening server (also in network byte order.)

1. Communicating with the server by send()ing and recv()ing. Terminating the connection and cleaning up with a call to close(). Again, if there were any calls to fork(), each process must close() the socket.

Functions:

1. socket():

socket() creates an endpoint for communication and returns a descriptor. socket() takes three arguments:

- ☐ domain, which specifies the protocol family of the created socket. For example:
- ☐ PF_INET for network protocol IPv4 or
- ☐ PF_INET6 for IPv6).

type, one of:

- ☐ SOCK_STREAM (reliable stream-oriented service)
- ☐ SOCK_DGRAM (datagram service)
- ☐ SOCK_SEQPACKET (reliable sequenced packet service), or
- ☐ SOCK_RAW (raw protocols atop the network layer).protocol

usually set to 0 to represent the default transport protocol for the specified domain and type values (TCP for PF_INET or PF_INET6 and SOCK_STREAM, UDP for those PF_ values and SOCK_DGRAM), but which can also explicitly specify a protocol.

The function returns -1 if an error occurred. Otherwise, it returns an integer representing the newly-assigned descriptor

Prototype:

int socket(int domain, int type, int protocol);

connect():

`connect()` returns an integer representing the error code: 0 represents success, while -1 represents an error. Certain types of sockets are connectionless, most commonly user datagram protocol sockets. For these sockets, `connect` takes on a special meaning: the default target for sending and receiving data gets set to the given address, allowing the use of functions such as `send()` and `recv()` on connectionless sockets.

Prototype:

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

`bind()`:

`bind()` assigns a socket an address. When a socket is created using `socket()`, it is given an address family, but not assigned an address. Before a socket may accept incoming connections, it must be bound. `bind()` takes three arguments:

- `sockfd`, a descriptor representing the socket to perform the bind on `my_addr`, a pointer to a `sockaddr` structure representing the address to bind to.
- `addrlen`, a `socklen_t` field representing the length of the `sockaddr` structure.

It returns 0 on success and -1 if an error occurs.

Prototype:

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

`listen()`

`listen()` prepares a bound socket to accept incoming connections. This function is only applicable to the `SOCK_STREAM` and `SOCK_SEQPACKET` socket types. It takes two arguments:

- `sockfd`, a valid socket descriptor.

`backlog`, an integer representing the number of pending connections that can be queued up at any one time. The operating system usually places a cap on this value.

Once a connection is accepted, it is dequeued. On success, 0 is returned. If an error occurs, -1 is returned.

Prototype:

```
int listen(int sockfd, int backlog);
```

accept()

Programmers use `accept()` to accept a connection request from a remote host. It takes the following arguments:

- `sockfd`, the descriptor of the listening socket to accept the connection from.
- `cliaddr`, a pointer to the `sockaddr` structure that `accept()` should put the client's address information into.
- `addrlen`, a pointer to the `socklen_t` integer that will indicate to `accept()` how large the `sockaddr` structure pointed to by `cliaddr` is. When `accept()` returns, the `socklen_t` integer then indicates how many bytes of the `cliaddr` structure were actually used.
- The function returns a socket corresponding to the accepted connection, or -1 if an error occurs.

Prototype:

```
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

Blocking vs. nonblocking

Berkeley sockets can operate in one of two modes: blocking or non-blocking. A blocking socket will not "return" until it has sent (or received) all the data specified for the operation. This may cause problems if a socket continues to listen: a program may hang as the socket waits for data that may never arrive. A socket is typically set to blocking or nonblocking mode using the `fcntl()` or `ioctl()` functions.

Cleaning up

The system will not release the resources allocated by the `socket()` call until a `close()` call occurs. This is especially important if the `connect()` call fails and may be retried. Each call to `socket()` must have a matching call to `close()` in all possible execution paths.

Algorithm:**Server Program**

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
```

```
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();
```

```
Send to client: os.writeBytes("Hello\n")
```

5. Close socket:

```
client.close();
```

Client Program

1) Create a Socket Object:

```
Socket client = new Socket(server, port_id);
```

2) Create I/O streams for communicating with the server.

```
is = new DataInputStream(client.getInputStream());
```

```
os = new DataOutputStream(client.getOutputStream());
```

3) Perform I/O or communication with the server:

```
Receive data from the server: String line = is.readLine();
```

```
Send data to the server: os.writeBytes("Hello\n");
```

4) Close the socket when done:

5) client.close();

CONCLUSION

Thus we have successfully implemented the socket programming for TCP

Program Code:

1. Simple Hello

Client.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

int main()
{
    int clientSocket;
    char buffer[1024];
    struct sockaddr_in serverAddr;
    socklen_t addr_size;

    /*--- Create the socket. The three arguments are: ---*/
    /* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) */
    clientSocket = socket(PF_INET, SOCK_STREAM, 0);

    /*--- Configure settings of the server address struct ---*/
    /* Address family = Internet */
    serverAddr.sin_family = AF_INET;
    /* Set port number, using htons function to use proper byte order */
    serverAddr.sin_port = htons(7891);
    /* Set IP address to localhost */
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    /* Set all bits of the padding field to 0 */
    memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

    /*--- Connect the socket to the server using the address struct ---*/
    addr_size = sizeof serverAddr;
    connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);

    /*--- Read the message from the server into the buffer ---*/
    recv(clientSocket, buffer, 1024, 0);
```

```
/*—— Print the received message ——*/
```

```
printf("Data received: %s",buffer);
```

```
return 0;
```

```
}
```

Server.c

```
#include <stdio.h>
```

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <string.h>
```

```
int main()
```

```
{
```

```
int welcomeSocket, newSocket;
```

```
char buffer[1024];
```

```
struct sockaddr_in serverAddr;
```

```
struct sockaddr_storage serverStorage;
```

```
socklen_t addr_size;
```

```
/*—— Create the socket. The three arguments are: ——*/
```

```
/* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) */
```

```
welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
```

```
/*—— Configure settings of the server address struct ——*/
```

```
/* Address family = Internet */
```

```
serverAddr.sin_family = AF_INET;
```

```
/* Set port number, using htons function to use proper byte order */
```

```
serverAddr.sin_port = htons(7891);
```

```
/* Set IP address to localhost */
```

```
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
/* Set all bits of the padding field to 0 */
```

```
memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
```

```
/*—— Bind the address struct to the socket ——*/
```

```
bind(welcomeSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
```

```
/*—— Listen on the socket, with 5 max connection requests queued ——*/
```

```
if(listen(welcomeSocket,5)==0)
```

```
printf("Listening\n");
```

```
else
```

```
printf("Error\n");
```

```
/*—— Accept call creates a new socket for the incoming connection ——*/
```

```
addr_size = sizeof serverStorage;
```

```
newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage, &addr_size);
```

```
/*—— Send message to the socket of the incoming connection ——*/
```

```
strcpy(buffer,"Hello World\n");
```

```
send(newSocket,buffer,13,0);
```

```
return 0;
```

```
}
```

```
/*OUTPUT CLIENT
```

```
iotlab@iotlab-Veriton-M200-B360:~$ cd TCP\ Socket/
```

```
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket$ cd Simple\ Hello/
```

```
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/Simple Hello$ gcc client_simple_hello.c -o client
```

```
client_simple_hello.c: In function 'main':
```

```
client_simple_hello.c:23:30: warning: implicit declaration of function 'inet_addr'; did you mean 's6_addr'? [-Wimplicit-function-declaration]
```

```
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
~~~~~
```

```
s6_addr
```

```
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/Simple Hello$ ./client
```

```
Data received: Hello World
```

```
OUTPUT SERVER
```

```
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/Simple Hello$ gcc server_simple_hello.c -o server
```

server_simple_hello.c: In function 'main':

server_simple_hello.c:24:30: warning: implicit declaration of function 'inet_addr'; did you mean 's6_addr'? [-Wimplicit-function-declaration]

```
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

~~~~~

s6\_addr

iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/Simple Hello\$ ./server

Listening

\*/

## 2. File transfer

### Client.c

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
#include <netinet/in.h>
```

```
#include <netdb.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <errno.h>
```

```
#include <arpa/inet.h>
```

```
int main(void)
```

```
{
```

```
    int sockfd = 0;
```

```
    int bytesReceived = 0;
```

```
    char recvBuff[256];
```

```
    memset(recvBuff, '0', sizeof(recvBuff));
```

```
    struct sockaddr_in serv_addr;
```

```
    /* Create a socket first */
```

```
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
```

```
    {
```

```
        printf("\n Error : Could not create socket \n");
```

```
        return 1;
```

---

```
}
```

```
/* Initialize sockaddr_in data structure */
```

```
serv_addr.sin_family = AF_INET;  
serv_addr.sin_port = htons(5000); // port  
serv_addr.sin_addr.s_addr = inet_addr("172.16.6.168");
```

```
/* Attempt a connection */
```

```
if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)  
{  
    printf("\n Error : Connect Failed \n");  
    return 1;  
}
```

```
/* Create file where data will be stored */
```

```
FILE *fp;  
fp = fopen("sample_file.txt", "ab");  
if(NULL == fp)  
{  
    printf("Error opening file");  
    return 1;  
}
```

```
/* Receive data in chunks of 256 bytes */
```

```
while((bytesReceived = read(sockfd, recvBuff, 256)) > 0)  
{  
    printf("Bytes received %d\n", bytesReceived);  
    // recvBuff[n] = 0;  
    fwrite(recvBuff, 1, bytesReceived, fp);  
    // printf("%s \n", recvBuff);  
}
```

```
if(bytesReceived < 0)  
{  
    printf("\n Read Error \n");  
}
```

---

```
    return 0;
}
```

### **Server.c**

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>

int main(void)
{
    int listenfd = 0;
    int connfd = 0;
    struct sockaddr_in serv_addr;
    char sendBuff[1024];
    int numrv;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);

    printf("Socket retrieve success\n");

    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

---

```
if(listen(listenfd, 10) == -1)
{
    printf("Failed to listen\n");
    return -1;
}
```

```
while(1)
{
    connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
```

```
/* Open the file that we wish to transfer */
```

```
FILE *fp = fopen("sample_file.txt", "rb");
if(fp == NULL)
{
    printf("File open error");
    return 1;
}
```

```
/* Read data from file and send it */
```

```
while(1)
{
    /* First read file in chunks of 256 bytes */
    unsigned char buff[256] = {0};
    int nread = fread(buff, 1, 256, fp);
    printf("Bytes read %d \n", nread);
```

```
/* If read was success, send data. */
```

```
if(nread > 0)
{
    printf("Sending \n");
    write(connfd, buff, nread);
}
```

```
/*
```

```
* There is something tricky going on with read ..
```

```
* Either there was error, or we reached end of file.
```

```
*/
```

```
    if (nread < 256)
    {
        if (feof(fp))
            printf("End of file\n");
        if (ferror(fp))
            printf("Error reading\n");
        break;
    }

}

close(connfd);
sleep(1);
}

return 0;
}

/*OUTPUT SERVER
iotlab@iotlab-Veriton-M200-B360:~$ cd TCP\ Socket/
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket$ cd File\ Transfer/
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/File Transfer$ gcc Server_file.c -o server
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/File Transfer$ ./server
Socket retrieve success
Bytes read 0
End of file

OUTPUT CLIENT
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/File Transfer$ gcc Client_file.c -o client
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/File Transfer$ ./client
iotlab@iotlab-Veriton-M200-B360:~/TCP Socket/File Transfer$
```

---

## **Assignment 6**

### **Problem Definition:**

**Problem Definition: Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines**

### **1.1 Prerequisite:**

- a) Socket Header      b) Network Programming      c) Ports

### **Learning Objectives:**

1. To understand Work of Socket
2. Different methods associated with Client & Server Socket

### **New Concepts:**

1. Client Server Communication
2. Port Address

### **1.3 Theory:**

#### **1.3.1 Introduction**

What is UDP?

UDP is a connectionless and unreliable transport protocol. The two ports serve to identify the end points within the source and destination machines. User Datagram Protocol is used, in place of TCP, when a reliable delivery is not required. However, UDP is never used to send important data such as web-pages, database information, etc. Streaming media such as video, audio and others use

UDP because it offers speed. **Why UDP is faster than TCP?**

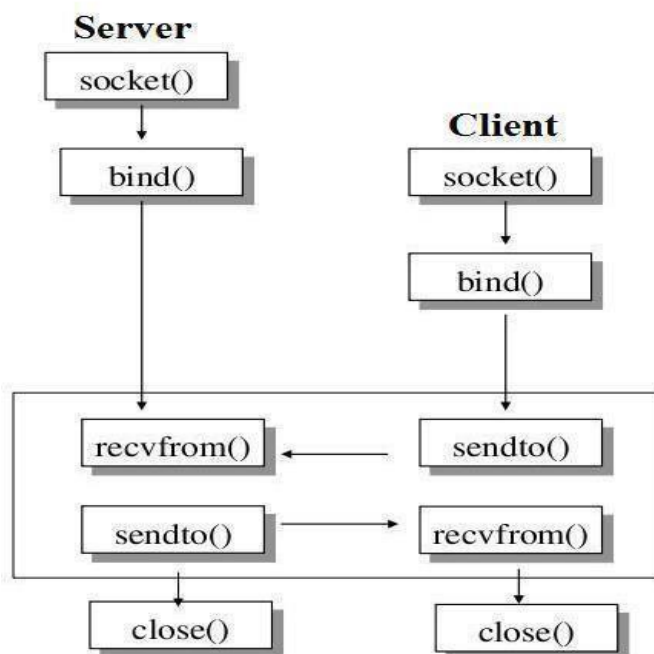
The reason UDP is faster than TCP is because there is no form of flow control. No error checking, error correction, or acknowledgment is done by UDP. UDP is only concerned with speed. So when, the data sent over the Internet is affected by collisions, and errors will be present. UDP packet's called as user datagrams with 8 bytes header. A format of

user datagrams is shown in figure 3. In the user datagrams first 8 bytes contains header information and the remaining bytes contain data.

### **LINUX SOCKET PROGRAMMING:**

The Berkeley socket interface, an API, allows communications between hosts or between processes on one computer, using the concept of a socket. It can work with many different I/O devices and drivers, although support for these depends on the operating-system implementation. This interface implementation is implicit for TCP/IP, and it is therefore one of the fundamental technologies underlying the Internet. It was first developed at the University of California, Berkeley for use on Unix systems. All modern operating systems now have some implementation of the Berkeley socket interface, as it has become the standard interface for connecting to the Internet. Programmers can make the socket interfaces accessible at three different levels, most powerfully and fundamentally at the RAW socket level. Very few applications need the degree of control over outgoing communications that this provides, so RAW sockets support was intended to be available only on computers used for developing Internet-related technologies. In recent years, most operating systems have implemented support for it anyway, including Windows XP. The header files: The Berkeley socket development library has many associated header files. They include: Definitions for the most basic of socket structures with the BSD socket API Basic data types associated with structures within the BSD socket API Definitions for the socketaddr\_in {} and other base data structures.

## **Connectionless Protocol**







## The header files:

The Berkeley socket development library has many associated header files. They include:

*<sys/socket.h>*

Definitions for the most basic of socket structures with the BSD **socket**

**API** *<sys/socket.h>*

Basic data types associated with structures within the BSD socket API *<sys/types.h>*

**Socket API** *<sys/types.h>*

Definitions for the `socketaddr_in{}` and other base data structures

*<sys/un.h>* Definitions and data type declarations for

SOCK\_UNIX streams

UDP: UDP consists of a connectionless protocol with no guarantee of delivery. UDP packets may arrive out of order, become duplicated and arrive more than once, or even not arrive at all. Due to the minimal guarantees involved, UDP has considerably less overhead than TCP. Being connectionless means that there is no concept of a stream or connection between two hosts, instead, data arrives in datagrams. UDP address space, the space of UDP port numbers (in ISO terminology, the TSAPs), is completely disjoint from that of TCP ports. Server: Code may set up a UDP server on port 7654 as follows:

```
sock =
socket(PF_INET,SOCK_DGRAM,0);    sa.sin_addr.s_addr =
INADDR_ANY;    sa.sin_port =    htons(7654); bound =
bind(sock,(struct sockaddr *)&sa, sizeof(struct sockaddr)); if (bound < 0)
fprintf(stderr, "bind(): %s\n",strerror(errno)); listen(sock,3); bind()
binds the socket to an address/port pair. listen() sets the length of the new connections queue.
while (1)
{ printf("recv test .... \n");
resize = recvfrom(sock, (void *)hz, 100, 0, (struct sockaddr *)&sa, fromlen); printf
("resize: %d\n ",resize); if
```

```

    (recsize < 0) fprintf(stderr, "%s\n",
    strerror(errno)); sleep(1); printf("datagram:
    %s\n",hz);
}

```

This infinite loop receives any UDP datagrams to port 7654 using `recvfrom()`. It uses the parameters: 1 socket 1 pointer to buffer for data 1 size of buffer 1 flags (same as in read or other receive socket function)

Client: A simple demo to send an UDP packet containing "Hello World!" to address 127.0.0.1, port 7654 might look like this:

```

#include #include #include #include #include #include int main(int argc,
char *argv[])
{
    int sock; struct sockaddr_in sa; int
    bytes_sent, buffer_length; char
    buffer[200]; sprintf(buffer,

    "Hello World!"); buffer_length = strlen(buffer) + 1;
    sock = socket(PF_INET, SOCK_DGRAM,
    0); sa.sin_family = AF_INET; sa.sin_addr.s_addr
    = htonl(0x7F000001); sa.sin_port = htons(7654); bytes_sent = sendto(sock, buffer,
    buffer_length, 0, &sa, sizeof(struct
    sockaddr_in) ); if(bytes_sent < 0) printf("Error sending packet: %s\n",
    strerror(errno) ); return 0;
}

```

In this code, `buffer` provides a pointer to the data to send, and `buffer_length` specifies the size of the buffer contents. Typical UDP client code

- Create UDP socket to contact server (with a given hostname and service port number) •
- Create UDP packet.

- Call send(packet), sending request to the server.
- Possibly call receive(packet) (if we need a reply).

Typical UDP Server code

- Create UDP socket listening to a well known port number.
- Create UDP packet buffer Call receive(packet) to get a request, noting the address of the client.
- Process request and send reply back with send(packet).

#### **APPLICATION :**

Socket programming is essential in developing any application over a network.

**CONCLUSION:** Thus we have studied Working of UDP Socket.

## **Assignment 7**

**Title:**

Study and Analyze the performance of HTTP, HTTPS and FTP protocol using Packet tracer tool.

**Theory:**

File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is an application layer protocol which moves files between local and remote file systems. It runs on the top of TCP, like HTTP. To transfer a file, 2 TCP connections are used by FTP in parallel: control connection and data connection.

**FTP Session:**

When a FTP session is started between a client and a server, the client initiates a control TCP connection with the server side. The client sends control information over this. When the server receives this, it initiates a data connection to the client side. Only one file can be sent over one data connection. But the control connection remains active throughout the user session. As we know HTTP is stateless i.e. it does not have to keep track of any user state. But FTP needs to maintain a state about its user throughout the session.

**Data Structures:**

FTP allows three types of data structures:

1. File Structure – In file-structure there is no internal structure and the file is considered to be a continuous sequence of data bytes.
2. Record Structure – In record-structure the file is made up of sequential records.
3. Page Structure – In page-structure the file is made up of independent indexed pages.

**Hypertext Transfer Protocol (HTTP)**

The Hypertext Transfer Protocol (HTTP) is an application-level protocol that uses TCP as an underlying transport and typically runs on port 80. HTTP is a stateless protocol i.e. server maintains no information about past client requests.

HTTP was invented alongside HTML to create the first interactive, text-based web browser: the original World Wide Web. Today, the protocol remains one of the primary means of using the Internet.

As a request-response protocol, HTTP gives users a way to interact with web resources such as HTML files by transmitting hypertext messages between clients and servers. HTTP clients generally use Transmission Control Protocol (TCP) connections to communicate with servers.

HTTP utilizes specific request methods in order to perform various tasks. All HTTP servers use the GET and HEAD methods, but not all support the rest of these request methods:

- GET requests a specific resource in its entirety

- ☐ HEAD requests a specific resource without the body content
- ☐ POST adds content, messages, or data to a new page under an existing web resource
- ☐ PUT directly modifies an existing web resource or creates a new URI if need be
- ☐ DELETE gets rid of a specified resource
- ☐ TRACE shows users any changes or additions made to a web resource
- ☐ OPTIONS shows users which HTTP methods are available for a specific URL
- ☐ CONNECT converts the request connection to a transparent TCP/IP tunnel
- ☐ PATCH partially modifies a web resource

#### HTTP Connections:

1. Non-Persistent
2. Persistent

#### **HTTPS (HTTP over SSL or HTTP Secure)**

HTTPS is an abbreviation of Hypertext Transfer Protocol Secure. It is a secure extension or version of HTTP. This protocol is mainly used for providing security to the data sent between a website and the web browser. It is widely used on the internet and used for secure communications. This protocol uses the 443 port number for communicating the data. This protocol is also called HTTP over SSL because the HTTPS communication protocols are encrypted using the SSL (Secure Socket Layer). By default, it is supported by various web browsers.

Those websites which need login credentials should use the HTTPS protocol for sending the data.

#### **Analyzing Site Traffic**

To show you the traffic level for a given site over a selected period of time: Step 1 Choose Analyze > Traffic > Site. Step 2 To change the data to see the top application traffic coming into a specific site, out of a specific site, or all traffic within, coming in and moving out of that site, use the traffic selector buttons. Step 3 To see site conversations about the conversation between sites to pinpoint specific applications or sites, select the Site Conversations button and choose filters from the Interactive Report to further pinpoint an application, data source, or time frame in question. Step 4 To view top applications transmitting and receiving traffic for the selected time period and drill down to collect more data utilizing capture data, real-time graphs, and application group detail), left click the Top N Application dashboard. Step 5 To see the criteria by which the Packet Analyzer classifies the amount of application traffic on this site over this period of time, use the view Application Distribution graph. Hover over graph parts to view detailed information on speed and percentages or left-click a graph element for other menu options.

#### **Analyzing Application Traffic**

To show you the traffic level for a given application over a selected period of time: Step 1 Choose Analyze > Traffic > Application. Step 2 To see data for a different time interval (when No data for select time interval displays), click Filter on the

Interactive Report, and expand the time range to allow more data to be viewed. Step 3 To focus in on a spike or area of interest, use the slider under the Application Traffic graph. Hover over the data points to see specific traffic details. Step 4 To see top application traffic details, click Top Application Traffic and choose filters from the Interactive Report to further pinpoint a data source, encapsulation method, or time frame in question. Step 5 To view top hosts transmitting and receiving traffic for the selected time period and drill down to collect more data utilizing capture data, real-time graphs, and application group detail), left-click a Top N Hosts graph element and select a specific task. Step 6 For example, select Hosts Detail to see the All Hosts window and the detailed information about all hosts. Table D-45 describes the fields in this window. Step 7 To show the criteria by which the Packet Analyzer classifies packets as that application, select one of the options under the Application Configuration. This is typically a list of TCP and/or UDP ports that identify the application. Some applications are identified by heuristic or other state-based algorithms. You can select Configure Application to configure specific applications in your network

### **Analyzing Host Traffic**

The Host Traffic Analysis window will show you at a quick glance the input and output of a particular host over a specified time range. It is available under the menu option Analyze > Traffic > Host. It will show you: • Input and output traffic for the host • Top N application activity of the host over the selected interval • Total application usage distribution for the host • Host Conversations—Shows detailed lists of all the conversations for a particular host.

Using the Packet Analyzer Application Programming Interface Packet Analyzer provides an Application Programming Interface (API) that allows you to configure and retrieve data from the Packet Analyzer. The API follows the commonly used Representational State Transfer (REST) style of providing services over HTTP or HTTPS. The Packet Analyzer REST API is also referred to as the Northbound Interface (NBI).

| No.                                                                                                                                             | Time                    | Source                  | Destination             | Protocol           | Length | Info                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|-------------------------|-------------------------|--------------------|--------|----------------------------------------------------------------------------------------------------------------|
| 14                                                                                                                                              | 3.646133                | 192.168.1.5             | 20.44.10.122            | TLSv1.2            | 283    | Client Hello                                                                                                   |
| 15                                                                                                                                              | 3.890712                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 1466   |                                                                                                                |
| 16                                                                                                                                              | 3.890712                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 1466   | Ignored Unknown Record                                                                                         |
| 19                                                                                                                                              | 3.891120                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 1466   | Ignored Unknown Record                                                                                         |
| 20                                                                                                                                              | 3.891120                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 698    | Ignored Unknown Record                                                                                         |
| 22                                                                                                                                              | 3.909597                | 192.168.1.5             | 20.44.10.122            | TLSv1.2            | 212    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message                                           |
| 24                                                                                                                                              | 4.154382                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 105    | Change Cipher Spec, Encrypted Handshake Message                                                                |
| 26                                                                                                                                              | 4.161599                | 192.168.1.5             | 20.44.10.122            | TLSv1.2            | 1050   | Application Data                                                                                               |
| 27                                                                                                                                              | 4.406289                | 20.44.10.122            | 192.168.1.5             | TLSv1.2            | 507    | Application Data                                                                                               |
| 44                                                                                                                                              | 7.490156                | 2402:e280:3e16:16ca::   | 2404:6800:4009:800::    | QUIC               | 1292   | Initial, DCID=f9137635effb49eb, PKN: 1, CRYPTO, PADDING, CRYPTO, CRYPTO, PADDING, PING, CRYPTO, PAD...         |
| 51                                                                                                                                              | 7.563659                | 2404:6800:4009:800::    | 2402:e280:3e16:16ca::   | QUIC               | 1292   | Protected Payload (KP0)                                                                                        |
| 113                                                                                                                                             | 7.792930                | 2402:e280:3e16:16ca::   | 2404:6800:4009:800::    | QUIC               | 1292   | Initial, DCID=cc51057bd672a4cf, PKN: 1, PADDING, CRYPTO, PING, CRYPTO, PING, PADDING, CRYPTO, CRYPT...         |
| 118                                                                                                                                             | 7.864655                | 2404:6800:4009:800::    | 2402:e280:3e16:16ca::   | QUIC               | 1292   | Protected Payload (KP0)                                                                                        |
| 127                                                                                                                                             | 7.872652                | 2404:6800:4009:800::    | 2402:e280:3e16:16ca::   | QUIC               | 1292   | Protected Payload (KP0)                                                                                        |
| 137                                                                                                                                             | 7.905852                | 2402:e280:3e16:16ca::   | 2404:6800:4009:800::    | QUIC               | 1292   | Initial, DCID=0ac6sef30370907f, PKN: 1, PING, PING, CRYPTO, PADDING, PING, PADDING, PING, PADDING, PING, PA... |
| 141                                                                                                                                             | 7.977491                | 2404:6800:4009:800::    | 2402:e280:3e16:16ca::   | QUIC               | 1292   | Protected Payload (KP0)                                                                                        |
| 229                                                                                                                                             | 11.434376               | 2402:e280:3e16:16ca::   | 2404:6800:4009:800::    | QUIC               | 1292   | Initial, DCID=21488d3620f14da9, PKN: 1, CRYPTO, PING, PADDING, CRYPTO, CRYPTO, PING, PADDING, PING, CRYPTO...  |
| > Frame 14: 283 bytes on wire (2204 bits), 283 bytes captured (2204 bits) on interface \Device\NPF_{080EAEDA-E451-41EE-890E-18A96640BF50}, id 0 |                         |                         |                         |                    |        |                                                                                                                |
| > Ethernet II, Src: IntelCor_6c:39:a5 (cc:2f:71:6c:39:a5), Dst: Talcang_T_65:92:8c (40:33:06:65:92:8c)                                          |                         |                         |                         |                    |        |                                                                                                                |
| > Internet Protocol Version 4, Src: 192.168.1.5, Dst: 20.44.10.122                                                                              |                         |                         |                         |                    |        |                                                                                                                |
| > Transmission Control Protocol, Src Port: 54953, Dst Port: 443, Seq: 1, Ack: 1, Len: 229                                                       |                         |                         |                         |                    |        |                                                                                                                |
| > Transport Layer Security                                                                                                                      |                         |                         |                         |                    |        |                                                                                                                |
| 0000                                                                                                                                            | 40                      | 13 06 65 92 8c cc 2f    | 71 6c 39 a5 00 00 45 00 | @3 e-.../ q19...E- |        |                                                                                                                |
| 0010                                                                                                                                            | 01 0d bb 4f 40 00 80 06 | 5e 48 c0 a0 01 05 14 2c | ...00...^H....          |                    |        |                                                                                                                |
| 0020                                                                                                                                            | 0e 7a d6 a9 01 bb 1c 33 | 81 91 f7 75 92 6f 50 18 | z-...3...uP.            |                    |        |                                                                                                                |
| 0030                                                                                                                                            | 05 00 0e 05 00 00 16 03 | 03 00 e0 01 00 00 dc 03 | .....                   |                    |        |                                                                                                                |
| 0040                                                                                                                                            | 03 63 4c 26 c1 13 3e ac | ae 55 16 26 b3 56 ef e7 | cL&...:U&V..            |                    |        |                                                                                                                |
| 0050                                                                                                                                            | 59 b3 41 da 48 a2 f8 d3 | ba f4 90 9e 30 2b e1 b4 | Y-A-M...-8+..           |                    |        |                                                                                                                |
| 0060                                                                                                                                            | ba 20 9a 2e 00 00 38 76 | 20 ba 4c a0 26 de 49 63 | ...-88...L&Ic           |                    |        |                                                                                                                |
| 0070                                                                                                                                            | c1 b6 15 39 82 52 9c 59 | ba 5f c5 18 de 50 96 c3 | ...9-R-Y...P-           |                    |        |                                                                                                                |
| 0080                                                                                                                                            | ff e5 00 26 c0 2c c0 20 | c0 30 c0 2f c0 24 c0 23 | ...8...4-0/\$-#         |                    |        |                                                                                                                |
| 0090                                                                                                                                            | c0 28 c0 27 c0 0a c0 09 | c0 14 c0 13 00 00 00 9c | {.....                  |                    |        |                                                                                                                |
| 00a0                                                                                                                                            | 00 3d 00 3c 00 35 00 2f | 00 0a 01 00 00 66 00 00 | ...<\$../.....          |                    |        |                                                                                                                |
| 00b0                                                                                                                                            | 00 23 00 21 00 00 1e 73 | 65 6c 66 2e 65 76 65 6e | -B-l...s elf.even       |                    |        |                                                                                                                |
| 00c0                                                                                                                                            | 74 73 2e 64 61 74 61 2e | 6d 69 63 72 6f 73 6f 66 | ts.data.microsoft       |                    |        |                                                                                                                |

http

| No.  | Time      | Source         | Destination    | Protocol | Length | Info                                       |
|------|-----------|----------------|----------------|----------|--------|--------------------------------------------|
| 1829 | 75.228436 | 192.168.1.5    | 164.100.150.66 | HTTP     | 556    | GET /rajnandgaon HTTP/1.1                  |
| 1834 | 75.361283 | 164.100.150.66 | 192.168.1.5    | HTTP     | 512    | HTTP/1.1 301 Moved Permanently (text/html) |
| 1920 | 81.560595 | 192.168.1.5    | 164.100.129.81 | HTTP     | 557    | GET /rajnandgaon HTTP/1.1                  |
| 1926 | 81.662494 | 164.100.129.81 | 192.168.1.5    | HTTP     | 510    | HTTP/1.1 302 Found (text/html)             |

> Frame 1829: 556 bytes on wire (4448 bits), 556 bytes captured (4448 bits) on interface \Device\NPF\_{080EAEDA-E451-41EE-890E-18A96640BF50}, id 0

> Ethernet II, Src: IntelCor\_6c:39:a5 (cc:2f:71:6c:39:a5), Dst: TalcengT\_65:92:8c (40:33:06:65:92:8c)

> Internet Protocol Version 4, Src: 192.168.1.5, Dst: 164.100.150.66

> Transmission Control Protocol, Src Port: 55093, Dst Port: 80, Seq: 1, Ack: 1, Len: 502

> Hypertext Transfer Protocol

0000 40 33 06 65 92 8c cc 2f 71 6c 39 a5 00 00 45 00 @3 e.../ q19...E-

0010 02 1e 7d 03 40 00 80 06 7f 82 c0 a8 01 05 a4 64 ...}... ..d

0020 95 4d 47 35 00 50 90 36 e6 ff 0e a4 4f a5 50 18 B.5.P.6...O-P-

0030 02 00 df 07 00 00 47 45 54 20 2f 72 61 6e 6e 61 .....GE T /rajna

0040 6e 64 67 61 6f 6e 20 48 54 54 50 2f 31 2e 31 0d ndgaon H TTP/1.1

0050 0a 48 6f 73 74 3a 20 77 77 72 2e 63 67 2e 6e 69 Host: www.cg.ni

0060 63 2e 89 6e 0d 0a 43 6f 6e 5e 65 63 74 89 6f 6e c.in-Co nnection

0070 3a 20 6b 65 65 70 2d 61 6c 69 76 65 6d 00 55 70 ! keep-a live-Up

0080 67 72 63 64 65 2d 49 6e 73 65 63 75 72 65 2d 52 grade-In secure-R

0090 65 71 75 65 73 74 73 3a 20 31 0d 0a 95 73 65 72 equests: 1-User

00a0 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6e 61 2f -Agent: Mozilla/

00b0 35 2e 30 20 28 57 69 6e 64 6f 77 73 20 4e 54 20 5.0 (Win dows NT

00c0 31 30 2e 30 3b 20 57 69 6e 36 34 3b 20 78 36 34 10.0: Win dows NT

## Conclusion:

We Studied and analyze the performance of HTTP, HTTPS and FTP protocol using Packet tracer tool.



## **Assignment No. 8**

**Title:**

To study the SSL protocol by capturing the packets using Wireshark tool while visiting any SSL secured website (banking, e-commerce etc.)

**Outcomes:**

Retrieve SSL protocol by capturing the packets using Wireshark.

**Theory:**

**SSL, or Secure Sockets Layer**, is an encryption-based Internet security protocol. It was first developed by Netscape in 1995 for the purpose of ensuring privacy, authentication, and data integrity in Internet communications. SSL is the predecessor to the modern TLS encryption used today.

**How does SSL/TLS work?**

In order to provide a high degree of privacy, SSL encrypts data that is transmitted across the web. This means that anyone who tries to intercept this data will only see a garbled mix of characters that is nearly impossible to decrypt.

SSL initiates an authentication process called a handshake between two communicating devices to ensure that both devices are really who they claim to be.

SSL also digitally signs data in order to provide data integrity, verifying that the data is not tampered with before reaching its intended recipient.

There have been several iterations of SSL, each more secure than the last. In 1999 SSL was updated to become TLS.

**Why is SSL/TLS important?**

Originally, data on the Web was transmitted in plaintext that anyone could read if they intercepted the message. For example, if a consumer visited a shopping website, placed an order, and entered their credit card number on the website, that credit card number would travel across the Internet unconcealed.

SSL was created to correct this problem and protect user privacy. By encrypting any data that goes between a user and a web server, SSL ensures that anyone who intercepts the data can only see a scrambled mess of characters. The consumer's credit card number is now safe, only visible to the shopping website where they entered it.

SSL also stops certain kinds of cyber attacks: It authenticates web servers, which is

## Are SSL and TLS the same thing?

The image shows a Wireshark packet capture of a TLS handshake. The top pane displays the packet list, and the bottom pane shows the packet details for the selected packet (No. 1).

**Packet List:**

| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                 |
|-----|----------|----------------|----------------|----------|--------|------------------------------------------------------|
| 1   | 0.000000 | 192.168.1.102  | 172.194.79.100 | TCP      | 78     | 60245 → 443 [V] Seq=0 Win=0 Len=0                    |
| 2   | 0.019664 | 172.194.79.100 | 192.168.1.102  | TCP      | 74     | 443 → 60245 [V] Seq=1 Win=0 Len=0                    |
| 3   | 0.019829 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=1 Ack=1 Win=0 Len=0            |
| 4   | 0.021326 | 192.168.1.102  | 172.194.79.100 | TLSv1    | 188    | Client Hello                                         |
| 5   | 0.040746 | 172.194.79.100 | 192.168.1.102  | TCP      | 66     | 443 → 60245 [ACK] Seq=1 Ack=121 Win=0 Len=0          |
| 6   | 0.041634 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1484   | Server Hello                                         |
| 7   | 0.041807 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done                       |
| 8   | 0.041700 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=121 Ack=1730 Win=522920        |
| 9   | 0.088343 | 192.168.1.102  | 172.194.79.100 | TLSv1    | 292    | Client Key Exchange, Change Cipher Spec, Encrypted   |
| 10  | 0.085145 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 113    | Change Cipher Spec, Encrypted Handshake Message      |
| 11  | 0.085201 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=387 Ack=1777 Win=524288        |
| 12  | 0.085436 | 192.168.1.102  | 172.194.79.100 | TLSv1    | 329    | Application Data                                     |
| 13  | 0.130368 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data                                     |
| 14  | 0.130525 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=3127 Win=522384        |
| 15  | 0.137903 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data                                     |
| 16  | 0.137932 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=4877 Win=522384        |
| 17  | 0.138409 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data, Application Data, Application Data |
| 18  | 0.138500 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=5827 Win=522384        |
| 19  | 0.138632 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 316    | Application Data, Application Data                   |
| 20  | 0.138688 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=6877 Win=524288        |
| 21  | 0.140273 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                   |
| 22  | 0.140369 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=7427 Win=522384        |
| 23  | 0.144820 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data                                     |
| 24  | 0.144880 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=8777 Win=522384        |
| 25  | 0.144405 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data                                     |
| 26  | 0.144498 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=18127 Win=522384       |
| 27  | 0.138000 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 376    | Application Data, Application Data                   |
| 28  | 0.150441 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=18331 Win=524288       |
| 29  | 0.138000 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                   |
| 30  | 0.151495 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=11681 Win=522384       |
| 31  | 0.138287 | 172.194.79.100 | 192.168.1.102  | TLSv1    | 1418   | Application Data                                     |
| 32  | 0.155175 | 192.168.1.102  | 172.194.79.100 | TCP      | 66     | 60245 → 443 [ACK] Seq=480 Ack=13031 Win=522384       |

**Packet Details (No. 1):**

- Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
- Ethernet II, Src: Apple-aa:09:1d:79:1b:35:aa:09:1d, Dst: Cisco-Li-aa:09:1d:00:1b:1d:aa:09:1d
- Internet Protocol Version 4, Src: 192.168.1.102, Dst: 172.194.79.100
- Transmission Control Protocol, Src Port: 60245, Dst Port: 443, Seq: 0, Len: 0

## SSL PROTOCOL Wireshark :

### 1. Open the Wireshark trace

## Step 2: Inspect the Trace

Now we are ready to look at the details of some “SSL” messages.

2. To begin, enter and apply a display filter of “ssl”. (see below)

This filter will help to simplify the display by showing only SSL and TLS messages. It will exclude other TCP segments that are part of the trace, such as Acks and connection open/close.

| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                                 |
|-----|----------|----------------|----------------|----------|--------|----------------------------------------------------------------------|
| 4   | 0.021328 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 186    | Client Hello                                                         |
| 5   | 0.041634 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Server Hello                                                         |
| 7   | 0.041697 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done                                       |
| 9   | 0.060543 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 252    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| 10  | 0.100345 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 133    | Change Cipher Spec, Encrypted Handshake Message                      |
| 12  | 0.105436 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 239    | Application Data                                                     |
| 13  | 0.130468 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 15  | 0.137983 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 17  | 0.130469 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data, Application Data                 |
| 18  | 0.138832 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 318    | Application Data, Application Data                                   |
| 21  | 0.140271 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                                   |
| 23  | 0.144028 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 25  | 0.144465 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 27  | 0.150380 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 270    | Application Data, Application Data                                   |
| 28  | 0.188958 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                                   |
| 31  | 0.155107 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 33  | 0.155329 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Application Data                                                     |
| 34  | 0.163129 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Application Data, Application Data, Application Data                 |

Frame 4: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits) on interface 0  
 Ethernet II, Src: Apple\_A2105:1d (78:56:11:a2:05:1d), Dst: Cisco-Li\_a31a9:8d (08:00:0c:28:31:a9:8d)  
 Internet Protocol Version 4, Src: 192.168.1.102, Dst: 173.194.79.106  
 Transmission Control Protocol, Src Port: 68246, Dst Port: 443, Seq: 1, Ack: 1, Len: 128  
 Secure Sockets Layer

Figure 2: Trace of “SSL” traffic showing the details of the SSL header

3. Select a TLS message somewhere in the middle of your trace for which the Info reads “Application Data” & expand its Secure Sockets Layer block (by using the “+” expander or icon). For in stance, packet #12 (see below).

| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                                 |
|-----|----------|----------------|----------------|----------|--------|----------------------------------------------------------------------|
| 4   | 0.021328 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 186    | Client Hello                                                         |
| 5   | 0.041634 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Server Hello                                                         |
| 7   | 0.041697 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done                                       |
| 9   | 0.060543 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 252    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| 10  | 0.100345 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 133    | Change Cipher Spec, Encrypted Handshake Message                      |
| 12  | 0.105436 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 239    | Application Data                                                     |
| 13  | 0.130468 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 15  | 0.137983 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 17  | 0.130469 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data, Application Data                 |
| 18  | 0.138832 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 318    | Application Data, Application Data                                   |
| 21  | 0.140271 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                                   |
| 23  | 0.144028 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 25  | 0.144465 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 27  | 0.150380 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 270    | Application Data, Application Data                                   |
| 28  | 0.188958 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data, Application Data                                   |
| 31  | 0.155107 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1416   | Application Data                                                     |
| 33  | 0.155329 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Application Data                                                     |

Frame 12: 239 bytes on wire (1912 bits), 239 bytes captured (1912 bits) on interface 0  
 Ethernet II, Src: Apple\_A2105:1d (78:56:11:a2:05:1d), Dst: Cisco-Li\_a31a9:8d (08:00:0c:28:31:a9:8d)  
 Internet Protocol Version 4, Src: 192.168.1.102, Dst: 173.194.79.106  
 Transmission Control Protocol, Src Port: 68246, Dst Port: 443, Seq: 387, Ack: 1777, Len: 173  
 Secure Sockets Layer  
 TLSv1 Record Layer: Application Data Protocol: HTTP-over-TLS  
 Content Type: Application Data (23)  
 Version: TLS 1.0 (0x0301)  
 Length: 186  
 Encrypted Application Data: 32e72fc0f73e0c3a76cc498ad794fd08ee412b68a891114...

Application Data is a generic TLS message carrying contents for the application, such as the web page. It is a good place for us to start looking at TLS messages.

The lower layer protocol blocks are TCP and IP because SSL runs on top of TCP/IP.

The SSL layer contains a “TLS Record Layer”. This is the foundational sublayer for TLS. All messages contain records. Expand this block to see its details. Each record starts with a Content Type field. This tells us what is in the contents of the record. Then comes a Version identifier. It will be a constant value for the SSL connection. It is followed by a Length field giving the length of the record. Last comes the contents of the record. Application Data records are sent after SSL has secured the connection, so the contents will show up as encrypted data. To see within this block, we could configure Wireshark with the decryption key. This is possible, but outside of our scope. Note that, unlike other protocols we will see such as DNS, there may be multiple records in a single message. Each record will show up as its own block. Look at the Info column, and you will see messages with more than one block.

The Content-Type for a record containing “Application Data” is 23. The version constant used in this trace is 0x0301 which represents TLS 1.0. The Length covers only the payload of the Record Layer.

### Step 3: The SSL Handshake

An important part of SSL is the initial handshake that establishes a secure connection. The handshake proceeds in several phases. There are slight differences for different versions of TLS and depending on the encryption scheme that is in use. The usual outline for a brand-new connection is:

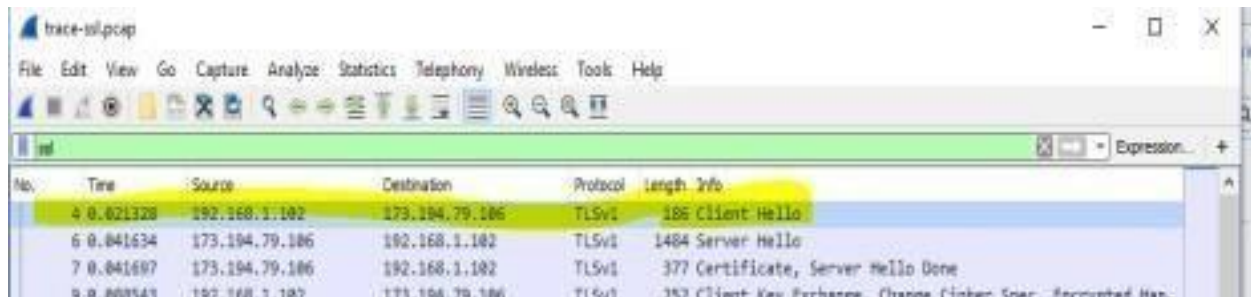
- a. Client (the browser) and Server (the web server) both send their Hellos
- b. Server sends its certificate to Client to authenticate (and optionally asks for Client Certificate) c. Client sends keying information and signals a switch to encrypted data. d. Server signals a switch to encrypted data.
- e. Both Client and Server send encrypted data.
- f. An Alert is used to tell the other party that the connection is closing.

Note that there is also a mechanism to resume sessions for repeat connections between the same client and server to skip most of steps b and c. However, we will not study session resumption.

### Hello Messages

*Next we will find and inspect the details of the Client Hello and Server Hello messages, including expanding the Handshake protocol block within the TLS Record.* For these initial messages, an encryption scheme is not yet established so the contents of the record are visible to us. They contain details of the secure connection setup in a Handshake protocol format.

#### 4. Select packet #4, which is a TLS Client Hello message



| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                         |
|-----|----------|----------------|----------------|----------|--------|--------------------------------------------------------------|
| 4   | 0.021328 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 186    | Client Hello                                                 |
| 6   | 0.041634 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Server Hello                                                 |
| 7   | 0.041697 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done                               |
| 9   | 0.000041 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 353    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake |

We can see several important fields here worth mentioning. First, the time (GMT seconds since midnight Jan 1, 1970) and random bytes (size 28) are included. This will be used later in the protocol to generate our symmetric encryption key. The client can send an optional session ID to quickly resume a previous TLS connection and skip portions of the TLS handshake. Arguably the most important part of the ClientHello message is the list of cipher suites, which dictate the key exchange algorithm, bulk encryption algorithm (with key length), MAC, and a pseudo-random function. The list should be ordered by client preference. The collection of these choices is a “cipher suite”, and the server is responsible for choosing a secure one it supports or return an error if it doesn’t support any. The final field specified in the specification is for compression methods. However, secure clients will advertise that they do not support compression (by passing “null” as the only algorithm) to avoid the CRIME attack. Finally, the ClientHello can have a number of different extensions. A common one is server\_name, which specifies the host name the connection is meant for, so web servers hosting multiple sites can present the correct certificate.

#### 5. Select packet #6, which is a TLS Server Hello message

The session ID sent by the server is 32 bytes long. This identifier allows later resumption of the session with an abbreviated handshake when both the client and server indicate the same value. In our case, the client likely sent no session ID as there was nothing to resume (see below)



| No. | Time     | Source         | Destination    | Protocol | Length | Info                           |
|-----|----------|----------------|----------------|----------|--------|--------------------------------|
| 4   | 0.021328 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 186    | Client Hello                   |
| 6   | 0.041634 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 1484   | Server Hello                   |
| 7   | 0.041697 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done |

Content Type: Handshake (22)  
Version: TLS 1.0 (0x0301)  
Length: 85

Handshake Protocol: Server Hello  
Handshake Type: Server Hello (2)  
Length: 81  
Version: TLS 1.0 (0x0301)

Random: 501778d3d52d556ed20e072f638f0a51e9724d66ef5f1376...  
GMT Unix Time: Jul 31, 2012 07:18:59.000000000 GMT Daylight Time  
Random Bytes: d52d556ed20e072f638f0a51e9724d66ef5f13769d3a52e0...  
Session ID Length: 32  
Session ID: 8530bdac95116ccb343798b36cb2fd79c1e278cba1af4145...

4

The Cipher method chosen by the Server is *TLS\_RSA\_WITH\_RC4\_128\_SHA* (0x0005). The Client will list the different cipher methods it supports, and the Server will pick one of these methods to use.

```
Session ID Length: 32
Session ID: 8530bdac95116ccb343798b36cb2fd79c1e278cbaf4145...
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Compression Method: null (0)
Extensions Length: 9
```

## Certificate Messages

6. Next, find and inspect the details of the Certificate message including expanding the Handshake protocol block within the TLS Record (see below for expansion of packet #7).

| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                       |
|-----|----------|----------------|----------------|----------|--------|------------------------------------------------------------|
| 4   | 0.021328 | 192.168.1.102  | 173.194.79.100 | TLSPv1   | 106    | Client Hello                                               |
| 6   | 0.041634 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1464   | Server Hello                                               |
| 7   | 0.041697 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 377    | Certificate, Server Hello Done                             |
| 9   | 0.080543 | 192.168.1.102  | 173.194.79.100 | TLSPv1   | 352    | Client Key Exchange, Change Cipher Spec, Encrypted Hand... |
| 10  | 0.105145 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 113    | Change Cipher Spec, Encrypted Handshake Message            |
| 12  | 0.105436 | 192.168.1.102  | 173.194.79.100 | TLSPv1   | 239    | Application Data                                           |
| 13  | 0.116460 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data                                           |
| 15  | 0.137903 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data                                           |
| 17  | 0.138409 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data, Application Data, Application Data       |
| 19  | 0.138532 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 316    | Application Data, Application Data                         |
| 21  | 0.140271 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data, Application Data                         |
| 23  | 0.144028 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data                                           |
| 25  | 0.144465 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data                                           |
| 27  | 0.150300 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 270    | Application Data, Application Data                         |
| 29  | 0.150950 | 173.194.79.100 | 192.168.1.102  | TLSPv1   | 1416   | Application Data, Application Data                         |

Secure Sockets Layer

- TLSPv1 Record Layer: Handshake Protocol: Certificate
  - Content Type: Handshake (22)
  - Version: TLS 1.0 (0x0301)
  - Length: 1625
  - Handshake Protocol: Certificate
    - Handshake Type: Certificate (11)
    - Length: 1625
    - Certificates Length: 1618
    - Certificates (1618 bytes)
      - Certificate Length: 806
        - Certificate: 30820321308202Bca0030201020104f9d06095b0902b54... (id-at-commonName=www.google.com,id-at-organizationName=Go...
        - Certificate Length: 807
          - Certificate: 30820321308202Bca0030201020104f9d06095b0902b54... (id-at-commonName=Thawte SOC CA,id-at-organizationName=Tha...

As with the Hellos, the contents of the Certificate message are visible because an encryption scheme is not yet established. It should come after the Hello messages.

Note it is the server that sends a certificate to the client, since it is the browser that wants to verify the identity of the server. It is also possible for the server to request certificates from the client, but this behavior is not normally used by web applications.

A Certificate message will contain one or more certificates, as needed for one party to verify the identity of the other party from its roots of trust certificates. You can inspect those certificates in your browser.

## 5 Client Key Exchange and Change Cipher Messages

7. Find and inspect the details of the Client Key Exchange and Change Cipher messages i.e. packet #9 (see below)

| No. | Time     | Source         | Destination    | Protocol | Length | Info                                                                 |
|-----|----------|----------------|----------------|----------|--------|----------------------------------------------------------------------|
| 7   | 0.041697 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 377    | Certificate, Server Hello Done                                       |
| 9   | 0.088543 | 192.168.1.102  | 173.194.79.106 | TLSv1    | 252    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| 10  | 0.105145 | 173.194.79.106 | 192.168.1.102  | TLSv1    | 113    | Change Cipher Spec, Encrypted Handshake Message                      |

The key exchange message is sent to pass keying information so that both sides will have the same secret session key. The change cipher message signals a switch to a new encryption scheme to the other party. This means that it is the last unencrypted message sent by the party.

Note how the Client Key Exchange has a Content-Type of 22, indicating the Handshake protocol. This is the same as for the Hello and Certificate messages, as they are part of the Handshake protocol.

✓ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange  
 Content Type: Handshake (22)  
 Version: TLS 1.0 (0x0301)  
 Length: 134  
 ✓ Handshake Protocol: Client Key Exchange

The Change Cipher Spec message has a Content-Type of 20, indicating the Change Cipher Spec protocol (see packet #10 – see below).

✓ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec  
 Content Type: Change Cipher Spec (20)  
 Version: TLS 1.0 (0x0301)  
 Length: 1  
 Change Cipher Spec Message

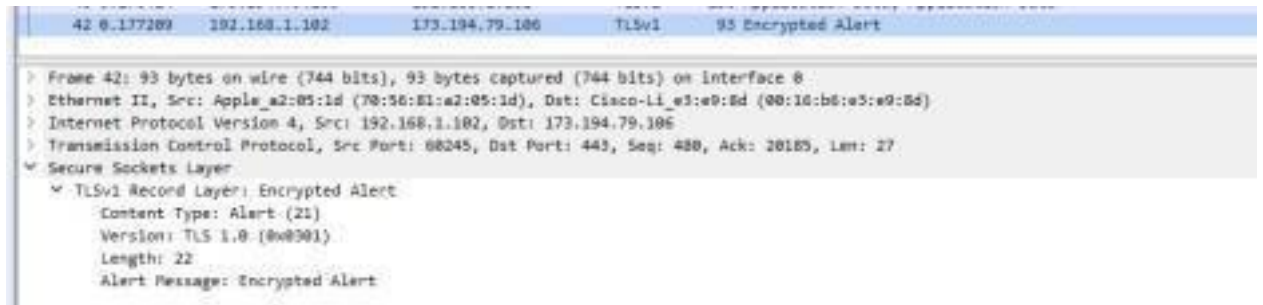
That is, this message is part of its own protocol and not the Handshake protocol. Both sides send the Change Cipher Spec message immediately before they switch to sending encrypted contents. The message is an indication to the other side. The contents of the Change Cipher Spec message are simply the value 1 as a single byte. Actually, it is the value “1” encrypted

under the current scheme, which uses no encryption for the handshake so that we can see it.

### Alert Message

8. Finally, find and inspect the details of an Alert message at the end of the trace (packet #42).

The Alert message is sent to signal a condition, such as notification that one party is closing the connection. You should find an Alert after the Application Data messages that make up the secure web fetch.



Note, the Content-Type value is 21 for Alert. This is a new protocol, different from the Handshake, Change Cipher Spec and Application Data values that we have already seen. The alert is encrypted; we cannot see its contents. Wireshark also describes the message as an “Encrypted Alert”. Presumably it is a “close\_notify” alert to signal that the connection is ending, but we can not be certain.

### Conclusion:

Hence we had studied the SSL protocol by capturing the packets using



## **Assignment 9**

**Problem Definition:** Illustrate the steps for implementation of S/MIME email security through Microsoft® Office Outlook. **Prerequisite:**

**Learning Objectives:**

1. Understand the concept and working of Encrypted mails

### **Theory**

#### **MIME (Secure/Multipurpose Internet Mail Extensions)**

S/MIME allows users to send encrypted and digitally signed emails. This protocol allows recipients of the email to be certain the email they receive is the exact message that began with the sender. It also helps ensure that a message going to an outbound recipient is from a specific sender and not someone assuming a false identity.

#### **How does S/MIME work?**

S/MIME provides cryptographic-based security services like authentication, message integrity, and digital signatures. All these elements work together to enhance privacy and security for both the sender and recipient of an email.

S/MIME also works with other technologies such as Transport Layer Security (TLS) which encrypts the path between two email servers. The protocol is also compatible with Secure Sockets Layer (SSL) which masks the connection between email messages and Office 365 (a common email service) servers.

In addition, BitLocker works in conjunction with S/MIME protocol, which encrypts data on a hard drive in a data center so if a hacker gets access, he or she won't be able to interpret the information.

**1. Safeguards sensitive data**

If you're sending information like your Social Security number over email, it's important that it's not easily stolen by hackers.

**2. Economical**

Instead of purchasing security equipment, you can simply rely on email encryption that's integrated directly on the server.

**3. Timesaving**

Instead of wasting time using several programs to make sure a connection is secure, you can rely on email encryption to do most of the work for you.

**4. Regulation compliance**

If you work in the healthcare industry, for example, and you haven't taken the right steps to secure medical data, you could be in violation of HIPAA laws [6]. Encryption helps you avoid those missteps.

**5. Protects against malware**

Malicious emails sometimes contain viruses masked as innocent email attachments. If you or someone else send an attachment using encrypted email, the email has a digital signature to prove its authenticity.

**How does email encryption work?**

If you don't want anyone but the receiver to see the contents of a message, encryption is vital. To the outsider, an encrypted email will have a bunch of random letters, digits, or symbols instead of readable text. The person with the private key to decrypt it, typically the receiver, will be able to read the email as usual.

**There are generally three encryption types available:**

- S/MIME encryption works as long as both the sender and recipient have mailboxes that support it. Windows Outlook is the most popular version that works with this method.

Gmail uses it as well.

- Office 365 Message Encryption is best for users with valid Microsoft Office licenses who can use this tool to encrypt the information and files sent via email. It's also a top choice for Outlook users
- PGP/MIME is a more affordable and popular option that other email clients may prefer to use. It's reliable and integrated into many of the apps we use today

Other email products may have their own brand of encryption, but the science behind it is the same. Only senders and recipients who have exchanged keys or digital signatures can communicate within the encrypted network.

How to send encrypted email in Outlook

Encrypting email may sound complicated, but it's not. Microsoft has a reputation for providing its users with simple ways to encrypt data, from files to folders to emails, too. It makes sense that they would include built-in tools for Outlook, their proprietary email system. You don't need a separate software tool or plug-in to start sending secure messages. Just follow these steps to begin.

### **1. Create a digital certificate**

For Outlook users, encrypting a single email is simple. First, you must have a digital signature. To create a digital signature:

1. Start in your Outlook window and click on the File tab
2. Select Options, then Trust Center, then Trust Center Settings
3. Select Email Security, Get a Digital ID

4. You'll be asked to choose a certification authority. This is entirely up to you as most are rated the same
5. You'll receive an email with your digital certificate/ID included
6. Go back into Outlook and select Options and the Security tab
7. In the Security Settings Name field, type in a name of your choosing
8. Ensure that S/MIME is selected from the Secure Message Format box and that Default Security Settings is checked as well
9. Go to Certificates and Algorithms, select Signing Certificate, and click Choose
10. Make sure the box is checked next to Secure Email Certificate, and check the box next to "Send These Certificates with Signed Messages"
11. Click OK to save your settings and start using Outlook

## **2. Use your digital signature**

**Now that you have a digital ID, you need to start using it:**

1. Open a new message to access the Tools tab
2. Click that, then Customize, and finally the Commands tab
3. From Categories, select Standard
4. From Categories, select Digitally Sign Message

## **3. Encrypt Outlook messages**

You can now send encrypted messages to a recipient with the next steps.

1. Open the window to compose a new message and select the Options tab, then More Options

2. Click the dialog box (triangle with arrow pointing down) in the lower-right corner
3. Choose Security Settings and check the box next to Encrypt message contents and attachments
4. Write your message as normal and send

After you've sent and received a message that you've both signed and encrypted, you don't have to sign it again. Outlook will remember your signature.

#### **4. Encrypt all Outlook messages**

You can encrypt each one, or you can use the steps below to encrypt all outgoing messages in Outlook:

Open the File tab in Outlook

Select Options, then Trust Center, and Trust Center Settings

From the Email Security tab, select Encrypted email

Check the box next to Encrypt content and attachments for outgoing messages

Use Settings to customize additional options, including certificates

#### **How to Send Encrypted Email**

Have you ever wondered about the security of your private email conversations? Whether at work, school, or home, sending emails comes with a bit of a risk. There's one thing you can do to discourage data breaches and attacks on your sensitive data, however. Use encrypted email. Learn how to practice this common-sense method for communicating in our step-by-step guide. But first, let's look at why you should embrace encryption for your email correspondence.

### How to Encrypt Email and Send Secure Messages

Emails sent over an open network can be intercepted and malicious actors can see email contents, attachments, or even take over your account.

To drive home the importance of email security, take a look at some alarming statistics that show the widespread cybersecurity issues that may have affected you in the past and still pose a threat today.

**Conclusion:** Thus we have studied the steps for implementation of S/MIME email security through Microsoft® Office Outlook.

## **Assignment No. 10**

**Title:**

To study the IPsec (ESP and AH) protocol by capturing the packets using Wireshark tool.

**Outcomes:**

Retrieve IPsec (ESP and AH) protocol by capturing the packets using Wireshark tool.

**Theory:**

**The IP security (IPSec)** is an Internet Engineering Task Force (IETF) standard suite of protocols between 2 communication points across the IP network that provide data authentication, integrity, and confidentiality. It also defines the encrypted, decrypted and authenticated packets. The protocols needed for secure key exchange and key management are defined in it.

**What Ports Does IPSEC Operate On?**

UDP port 500 should be opened as should IP protocols 50 and 51. UDP port 500 should be opened to allow for ISAKMP to be forwarded through the firewall while protocols 50 and 51 allow ESP and AH traffic to be forwarded respectively.<sup>2</sup>

**What is ISAKMP?**

ISAKMP stands for Internet Security Association and Key Management Protocol. These are two key components of an IPSEC VPN that must be in place in order for it to function normally and protect the public traffic that is being forwarded between the client and VPN server or VPN server to VPN server.

**What are ESP and AH?**

No, ESP is not Extra-Sensory Perception! ESP stands for Encapsulating Security Protocol and AH stands for Authentication Header.

**Encapsulating Security Protocol**

ESP gives protection to upper layer new protocols, with a Signed area indicating where a protected data packet has been signed for integrity, and an Encrypted area which indicates the information that's protected with confidentiality. Unless a data packet is being tunneled, ESP protects only the IP data payload (hence the name), and not the IP header.

some degree of traffic-level confidentiality, and an anti-replay service (a form of partial sequence integrity which guards against the use of commands or credentials which have been captured through password sniffing or similar attacks).<sup>3</sup>

### **Authentication Header**

Authentication Header (AH) is a new protocol and part of the Internet Protocol Security (IPsec) protocol suite, which authenticates the origin of IP packets (datagrams) and guarantees the integrity of the data. The AH confirms the originating source of a packet and ensures that its contents (both the header and payload) have not been changed since transmission.

If security associations have been established, AH can be optionally configured to defend against replay attacks using the sliding window technique.<sup>4</sup>

### **How Do They All Work Together?**

When properly configured, an IPSEC VPN provides multiple layers of security that ensure the security mode and integrity of the data that is being transmitted through the encrypted tunnel. This way an organization can feel confident that the data has not been intercepted and altered in transit and that they can rely on what they are seeing.

### **IPsec Protocols**

AH and/or ESP are the two protocols that we use to actually protect user data. Both of them can be used in transport or tunnel mode, let's walk through all the possible options.

#### **Authentication Header Protocol**

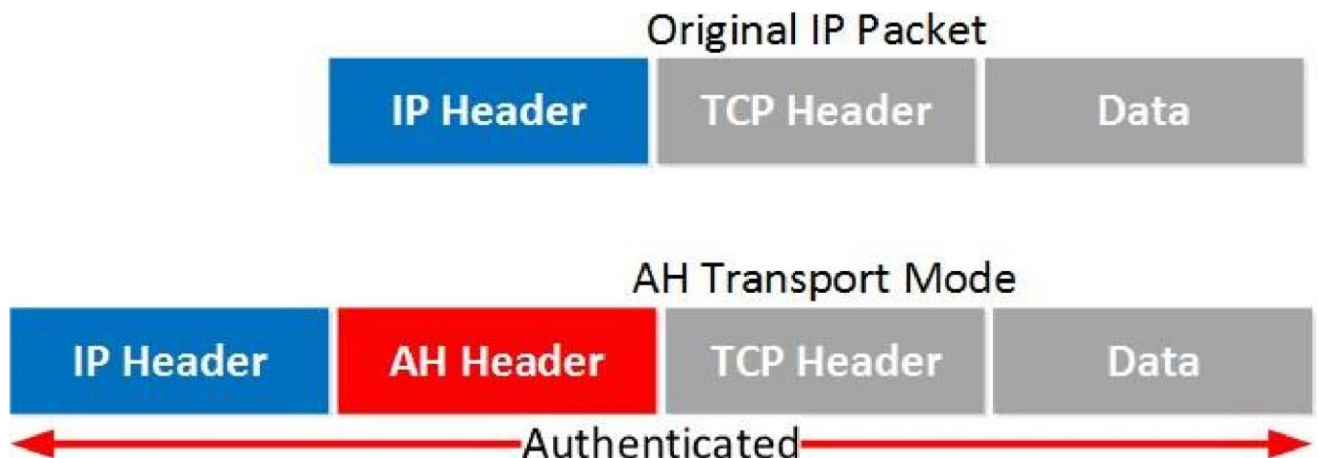
AH offers authentication and integrity but it doesn't offer any encryption. It protects the IP packet by calculating a hash value over almost all fields in the IP header. The fields it excludes are the ones that can be changed in transit (TTL and header checksum). Let's



start with transport mode...

### Transport Mode

Transport mode is simple, it just adds an AH header after the IP header. Here's an example of an IP packet that carries some TCP traffic:



And here's what that looks like in Wireshark:

```

* Frame 1: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on interface 0
  Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
  Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 124
    Identification: 0x0028 (40)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 255
    Protocol: Authentication Header (51)
    Header checksum: 0x21d3 [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  Authentication Header
    Next Header: ICMP (0x01)
    Length: 24
    AH SPI: 0xcf54ccdf
    AH Sequence: 30
    AH ICV: aa9cafe5ed06d6c74cb3c671
  Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x7994 [correct]
    Identifier (BE): 8 (0x0008)
    Identifier (LE): 2048 (0x0800)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Response frame: 2]
  Data (72 bytes)
  
```

Above you can see the AH header in between the IP header and ICMP header. This is a capture I took of a ping between two routers. You can see that AH uses 5 fields:

**Next Header:** this identifies the next protocol, ICMP in our example.

**Length:** this is the length of the AH header.

**SPI (Security Parameters Index):** this is a 32-bit identifier so the receiver knows to which flow this packet belongs.

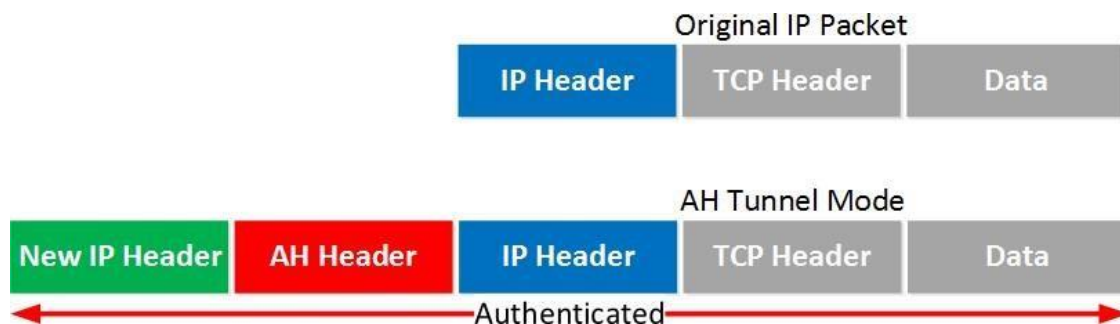
**Sequence:** this is the sequence number that helps against replay attacks.

**ICV (Integrity Check Value):** this is the calculated hash for the entire packet. The receiver also calculates a hash, when it's not the same you know something is wrong.

Let's continue with tunnel mode.

### Tunnel Mode

With tunnel mode we add a new IP header on top of the original IP packet. This could be useful when you are using private IP addresses and you need to tunnel your traffic over the Internet. It's possible with AH but it doesn't offer encryption:



The entire IP packet will be authenticated. Here's what it looks like in wireshark:

```

Frame 1: 158 bytes on wire (1264 bits), 158 bytes captured (1264 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
  Version: 4
  Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 144
  Identification: 0x0215 (533)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 255
  Protocol: Authentication Header (51)
  Header checksum: 0x1fd2 [validation disabled]
  Source: 192.168.12.1 (192.168.12.1)
  Destination: 192.168.12.2 (192.168.12.2)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Authentication Header
  Next Header: IPIP (0x04)
  Length: 24
  AH SPI: 0x646adc80
  AH Sequence: 5
  AH ICV: 606d214066853c0390cfe577
Internet Protocol version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
  Version: 4
  Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 100
  Identification: 0x003c (60)
  Flags: 0x00
    0... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 255
  Protocol: ICMP (1)
  Header checksum: 0x2209 [validation disabled]
  Source: 192.168.12.1 (192.168.12.1)
  Destination: 192.168.12.2 (192.168.12.2)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Internet Control Message Protocol

```

Above you can see the new IP header, then the AH header and finally the original IP packet that carries some ICMP traffic.

One problem with AH is that it doesn't play well with NAT / PAT. Fields in the IP header like TTL and the checksum are excluded by AH because it knows these will change. The IP addresses and port numbers however are included. If you change these with NAT, the ICV of AH fails.

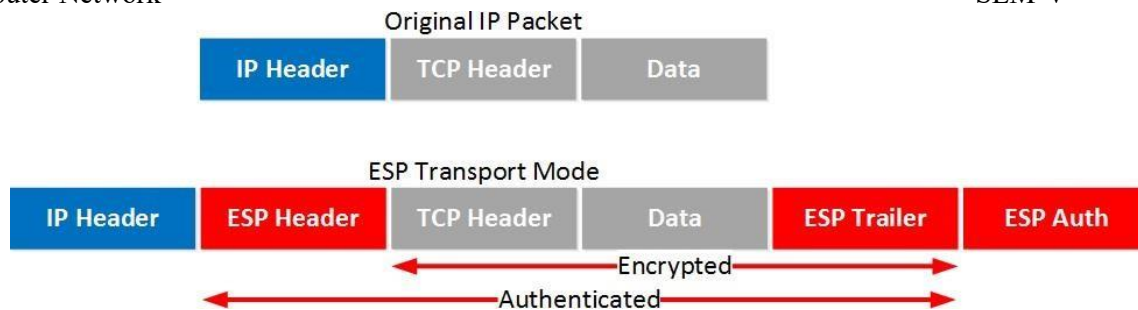
Let's continue with ESP...

### ESP (Encapsulating Security Payload) Protocol

ESP is the more popular choice of the two since it allows you to encrypt IP traffic. We can use it in transport or tunnel mode, let's look at both.

#### Transport Mode

When we use transport mode, we use the original IP header and insert an ESP header. Here's what it looks like:



Above you can see that we add an ESP header and trailer. Our transport layer (TCP for example) and payload will be encrypted. It also offers authentication but unlike AH, it's not for the entire IP packet. Here's what it looks like in wireshark:

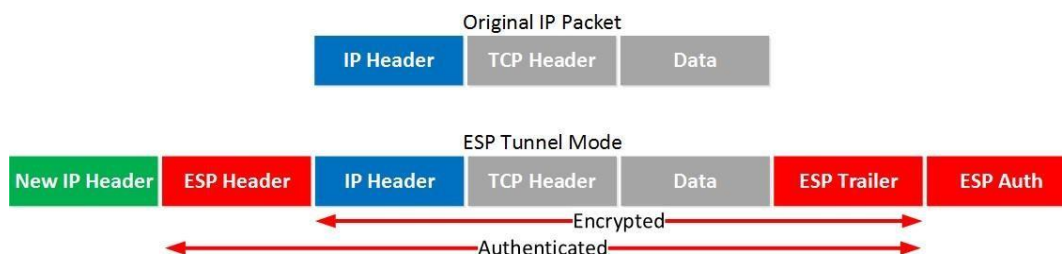
```

Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
  Version: 4
  Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 152
  Identification: 0x0042 (66)
  Flags: 0x00
    0... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 255
  Protocol: Encap Security Payload (50)
  Header checksum: 0x219e [validation disabled]
  Source: 192.168.12.1 (192.168.12.1)
  Destination: 192.168.12.2 (192.168.12.2)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Encapsulating Security Payload
  ESP SPI: 0x36cb42df (919290591)
  ESP Sequence: 1
  
```

Above you can see the original IP packet and that we are using ESP. The IP header is in cleartext but everything else is encrypted.

### Tunnel Mode

How about ESP in tunnel mode? This is where we use a new IP header which is useful for site-to-site VPNs:



It's similar to transport mode but we add a new header. The original IP header is now also encrypted. Here's what it looks like in wireshark:

```

* Frame 2: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
  Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
  Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 168
    Identification: 0x023e (574)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 255
    Protocol: Encap Security Payload (50)
    Header checksum: 0x1f92 [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  Encapsulating Security Payload
    ESP SPI: 0x8bb181a7 (2343666087)
    ESP Sequence: 5

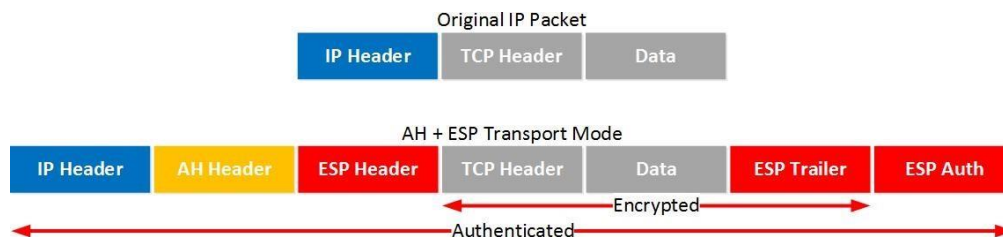
```

The output of the capture is above is similar to what you have seen in transport mode. The only difference is that this is a new IP header, you don't get to see the original IP header.

### AH and ESP

This one confuses a lot of people, it's possible to use AH and ESP at the same time. Let's check it out!

Transport Mode Let's start with transport mode, here's what the IP packet will look like:



With transport mode we will use the original IP header, followed by an AH and ESP header. The transport layer, payload and ESP trailer will be encrypted.

Because we also use AH, the entire IP packet is authenticated. Here's what it looks like in wireshark:



```
Frame 5: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
  Version: 4
  Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 164
  Identification: 0x0056 (86)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 255
  Protocol: Authentication Header (51)
  Header checksum: 0x217d [validation disabled]
  Source: 192.168.12.1 (192.168.12.1)
  Destination: 192.168.12.2 (192.168.12.2)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Authentication Header
  Next Header: Encap Security Payload (0x32)
  Length: 24
  AH SPI: 0xa90dc9aa
  AH Sequence: 1
  AH ICV: 157ba6cc340b1a30049ea551
Encapsulating Security Payload
  ESP SPI: 0xd2264f7a (3525726074)
  ESP Sequence: 1
```

Above you can see the original IP packet, the AH header and the ESP header.

### Conclusion:

Hence we had studied the IPsec (ESP and AH) protocol by capturing the packets using Wireshark tool.

## **Assignment 11**

**Problem Definition:** Installing and configure DHCP server and write a program to install the software on remote machine.

**Prerequisite:**

1. Knowledge about IP and Subnets.
2. Linux basic commands.

**Objectives:** 1. Understand the concept of DHCP.  
2. Configuring DHCP and installation of software.

**New Concepts:**

1. Crimping
2. Access Point Configuration

**Theory**

**Introduction**

- DHCP (Dynamic Host Configuration Protocol) is a protocol that lets network administrators manage centrally and automate the assignment of IP (Internet Protocol) configurations on a computer network.
- When using the Internet's set of protocols (TCP/IP), in order for a computer system to communicate to another computer system it needs a unique IP address.
- Without DHCP, the IP address must be entered manually at each computer system. DHCP lets a network administrator supervise and distribute IP addresses from a central point.
- The purpose of DHCP is to provide the automatic (dynamic) allocation of IP client configurations for a specific time period (called a lease period) and to eliminate the work necessary to administer a large IP network.
- When connected to a network, every computer must be assigned a unique address.
- However, when adding a machine to a network, the assignment and configuration of network (IP) addresses has required human action.
- The computer user had to request an address, and then the administrator would manually configure the machine. Mistakes in the configuration process are easy for novices to make, and can cause difficulties for both the administrator making the error as well as neighbors on the network.

Also, when mobile computer users travel between sites, they have had to relive this process for each different site from which they connected to a network.

- In order to simplify the process of adding machines to a network and assigning unique IP addresses manually, there is a need to automate the task.
- The introduction of DHCP alleviated the problems associated with manually assigning TCP/IP client addresses. Network administrators have quickly appreciated the importance, flexibility and ease-of-use offered in DHCP.

**Advantages of DHCP:-**

DHCP has several major advantages over manual configurations.

- Each computer gets its configuration from a "pool" of available numbers automatically for a specific time period (called a leasing period), meaning no wasted numbers.
- When a computer has finished with the address, it is released for another computer to use. Configuration information can be administered from a single point.
- Major network resource changes (e.g. a router changing address), requires only the DHCP server be updated with the new information, rather than every system.

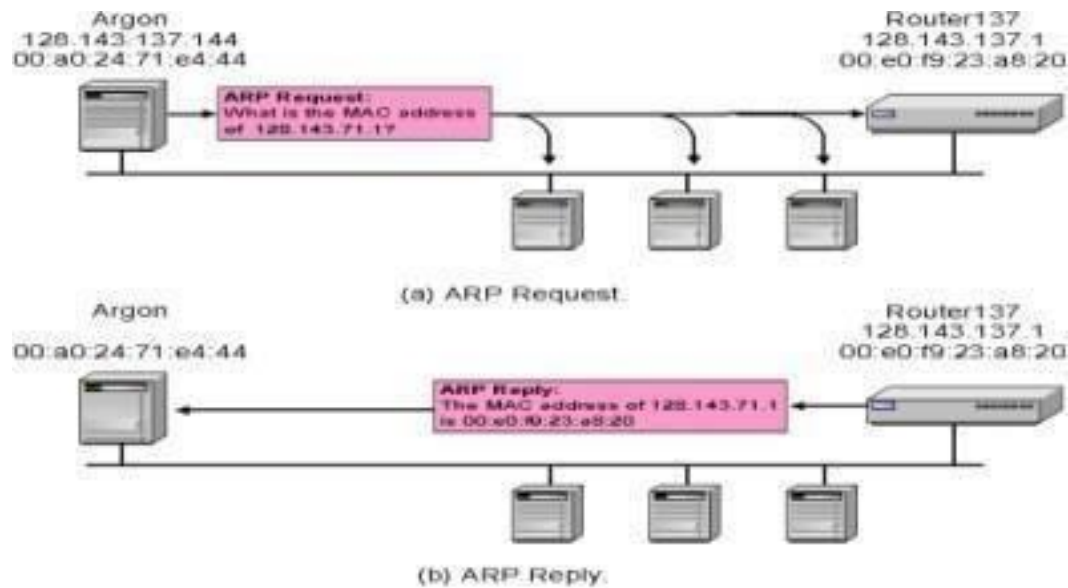
**DHCP message types:**

| Value | Message Type |
|-------|--------------|
| 1     | DHCPDISCOVER |
| 2     | DHCPOFFER    |
| 3     | DHCPREQUEST  |
| 4     | DHCPDECLINE  |
| 5     | DHCPACK      |
| 6     | DHCPNAK      |
| 7     | DHCPRELEASE  |
| 8     | DHCPINFORM   |

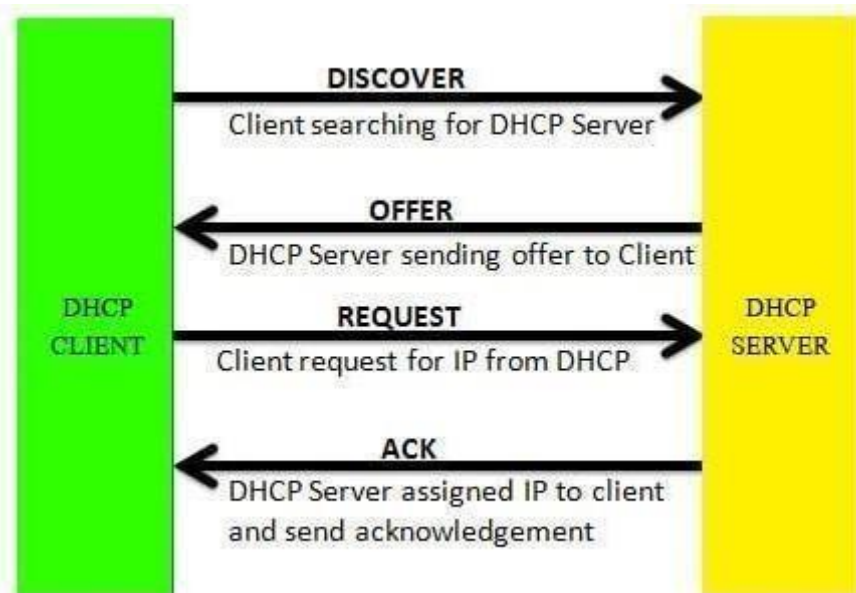
**DHCP Operations:-**

1. DHCP Discover





## 2. DHCP Offer



3. DHCP Discover: At this time, the DHCP client can start to use the IP address

4. DHCP Release: At this time, the DHCP client has released the IP address

### 5.4.4 Installing DHCP in Ubuntu:

Open terminal and type following commands:-

1. `sudo apt-get install isc-dhcp-server`
2. `sudo gedit /etc/dhcp/dhcpd.conf` then make changes in file....

```
default-lease-time 600; max-lease-time  
7200; option subnet-mask 255.255.255.0;  
option broadcast-address 10.1.32.255;  
subnet 192.168.1.0 netmask 255.255.255.0  
Range 10.1.32.10 10.1.32.20;}
```

3. default-lease-time 600; max-lease-time  
7200; option subnet-mask 255.255.255.0;  
option broadcast-address 10.1.32.255;  
subnet 192.168.1.0 netmask 255.255.255.0  
range 10.1.32.10 10.1.32.20; }
4. save file and close
5. again on terminal give following commands....  
sudo service isc-dhcp-server restart  
sudo service  
isc-dhcp-server start
6. On another PC in Internet properties change to Obtain IP address automatically and then  
check the IP address.

**Conclusion:**

Hence we Installed and Configured DHCP and studied Installation of Software on remote Machine.

## **Assignment – 12**

**Problem Definition:** Write a program for DNS lookup. Given an IP address input, it should return URL and vice versa.

### **Objective:**

- To get the host name and IP address.
- Map the host name with IP address and Vice-versa

### **Learning Objectives:**

- Understand what is Domain Name System and DNS lookup working.
- Understand what is DNS Structure and Hierarchy.

### **New Concepts:**

- Name Server and Domain Name System.
- DNS lookup, Zone

### **Theory:**

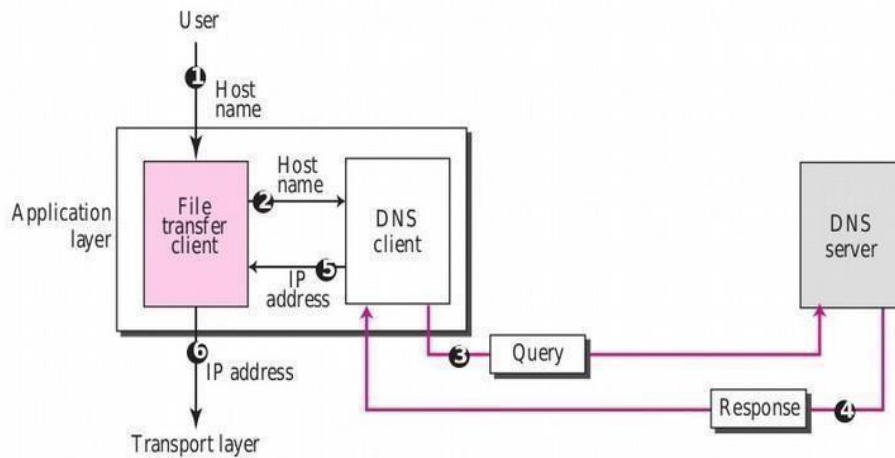
#### **Need for DNS:**

To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet. However, people prefer to use names instead of numeric addresses. Therefore, we need a system that can map a name to an address or an address to a name. When the Internet was small, mapping was done using a host file. The host file had only two columns: name and address. Every host could store the host file on its disk and update it periodically from a master host file. When a program or a user wanted to map a name to an

Address, the host consulted the host file and found the mapping.

Today, however, it is impossible to have one single host file to relate every address with a name and vice versa. The host file would be too large to store in every host. In addition, it would be impossible to update all the host files every time there is a change. One solution would be to store the entire host file in a single computer and allow access to this centralized information to every computer that needs mapping. But we know that this would create a huge amount of traffic on the Internet. Another solution, the one used today, is to divide this huge amount of information into smaller parts and store each part on a different computer. In this method, the host that needs

mapping can contact the closest computer holding the needed information. This method is used by the Domain Name System (DNS).



**Figure 1: How TCP/IP uses a DNS client and a DNS server to map a name to an address; the reverse mapping is similar.**

In Figure 1, a user wants to use a file transfer client to access the corresponding file transfer server running on a remote host. The user knows only the file transfer server name, such as forouzan.com.

However, the TCP/IP suite needs the IP address of the file transfer server to make the connection.

The following six steps map the host name to an IP address.

1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. We know that each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
4. The DNS server responds with the IP address of the desired file transfer server.
5. The DNS client passes the IP address to the file transfer server.
6. The file transfer client now uses the received IP address to access the file transfer server.

### NAME SPACE:

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses. In other words, the names must be unique because the addresses are unique.

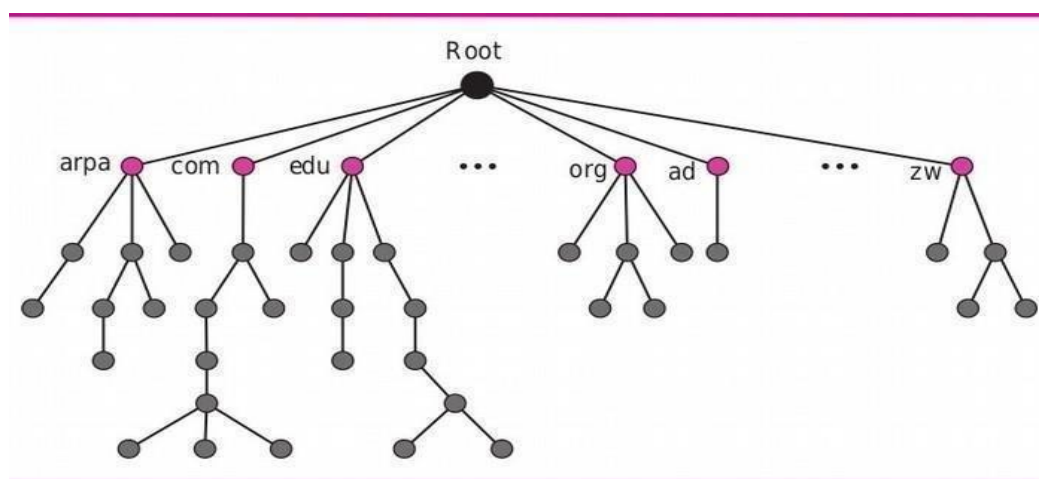
A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical. **Flat Name Space:**

In a flat name space, a name is assigned to an address. A name in this space is a sequence of characters without structure. The names may or may not have a common section; if they do, it has no meaning. The main disadvantage of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication.

### **Hierarchical Name Space;**

In a hierarchical name space, each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control the name spaces can be decentralized. A central authority can assign the part of the name that defines the nature of the organization and the name of the organization. **Domain Name Space:**

To have a hierarchical name space, a domain name space was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 (see Figure 2).



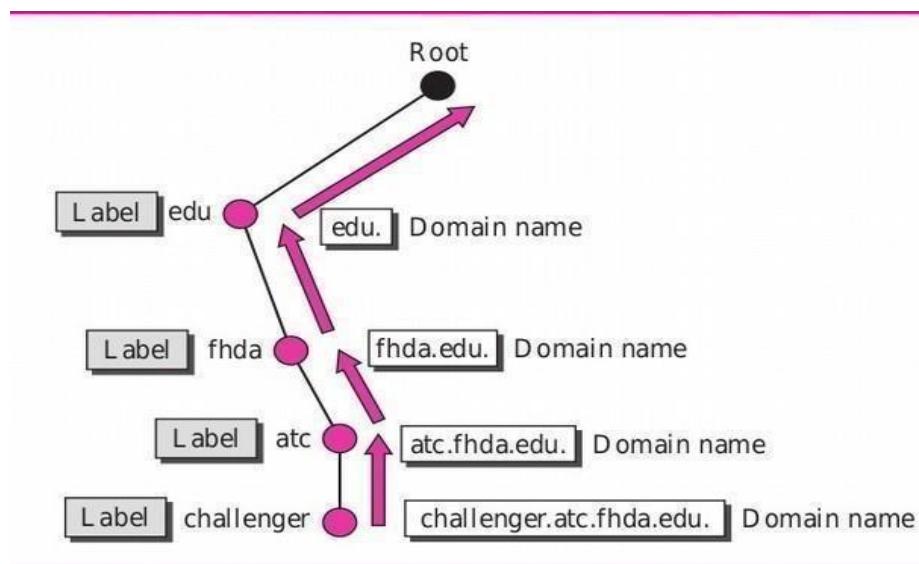
**Figure 2: Domain Name Space**

**Label:**

Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

**Domain Name:**

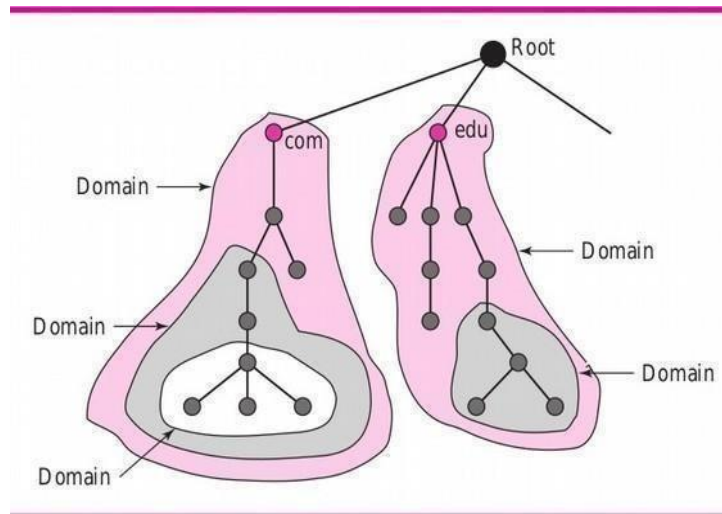
Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing.



**Figure 3 shows some domain names.**

**Domain:**

A domain is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. Figure 4 shows some domains. Note that a domain may itself be divided into domains (or subdomains as they are sometimes called).



**Figure 4 Domain**

### **RESOLUTION:**

Mapping a name to an address or an address to a name is called name-address resolution.

### **Resolver:**

DNS is designed as a client-server application. A host that needs to map an address to a name or a name to an address calls a DNS client called a resolver. The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the Information. After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it.

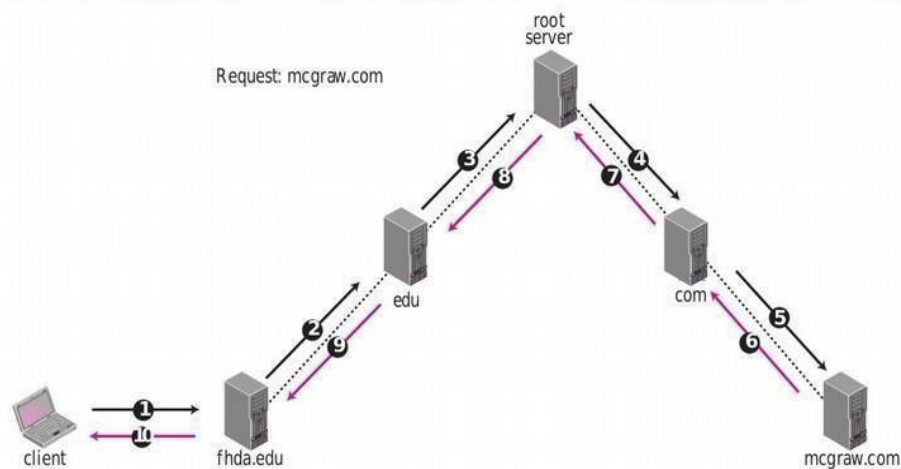
### **Mapping Names to Addresses:**

Most of the time, the resolver gives a domain name to the server and asks for the corresponding address. In this case, the server checks the generic domains or the country domains to find the mapping. If the domain name is from the generic domains section, the resolver receives a domain name such as “chal.atc.fhda.edu.” The query is sent by the resolver to the local **DNS server for resolution:**

If the local server cannot resolve the query, it either refers the resolver to other servers or asks other servers directly. If the domain name is from the country domains section, the resolver receives a domain name such as “ch.fhda.cu.ca.us.” The procedure is the same. Mapping

Addresses to Names a client can send an IP address to a server to be mapped to a domain name. As mentioned before, this is called a PTR query. To answer queries of this kind, DNS uses the inverse domain. However, in the request, the IP address is reversed and two labels, in-addr and arpa, are appended to create a domain acceptable by the inverse domain section. For example, if the resolver receives the IP address 132.34.45.121, the resolver first inverts the address and then adds the two labels before sending. The domain name sent is “121.45.34.132.in-addr.arpa.”, which is received by the local DNS and resolved. **Recursive Resolution:**

The client (resolver) can ask for a recursive answer from a name server. This means that the resolver expects the server to supply the final answer. If the server is the authority for the domain name, it checks its database and responds. If the server is not the authority, it sends the request to another server (the parent usually) and waits for the response. If the parent is the authority, it responds; otherwise, it sends the query to yet another server. When the query is finally resolved, the response travels back until it finally reaches the requesting client.



**CONCLUSION:** Thus we have successfully studied and implemented program for DNS lookup.



**Program code:**

```
import socket

def get_domain_name(ip_address):
    result = socket.gethostbyaddr(ip_address)
    return list(result)[0]

def get_Host_name_IP():
    try:
        host_name = 'www.google.com'
        host_ip = socket.gethostbyname(host_name)
        print("Hostname : ",host_name)
        print("IP : ",host_ip)
    except:
        print("Unable to get Hostname and IP")

print("Domain name using PTR DNS:")
print(get_domain_name("8.8.8.8"))
print(get_domain_name("13.251.106.90"))
print(get_domain_name("8.8.4.4"))
print(get_domain_name("23.23.212.126"))

# Driver code
get_Host_name_IP() #Function call
```

---