PVG's
College of Engineering
Nashik

Department of Computer Engineering

LABORATORY MANUAL

2017-2018

COMPUTER NETWORK LABORATORY

TE-COMPUTER ENGINEERING

SEMESTER-I

Subject Code: 310248

TEACHING SCHEME SCHEME	**EXAMINATION**
Lectures: 4 Hrs/Week	Theory:70 Marks
Practical: 2 Hrs/Week	In sem:30 Marks
	Practical:50 Marks
	Term Work: 25 Marks

Name of Faculty

Prof. Akshay R. Jain

Asst. Professor

PUNE VIDYARTHI GRIHA'S

COLLEGE OF ENGINEERING, NASHIK.

INDEX

Sr. No	Title	Page No	Date of Conduction	Date of Submission	Sign
1	Part A: Setup a wired LAN using Layer 2 Switch and then IP switch of minimum four computers. It includes preparation of cable, testing of cable using line tester, configuration machine using IP addresses, testing using PING utility and demonstrate the PING packets captured traces using Wireshark Packet Analyzer Tool. Part B: Extend the same Assignment for Wireless using Access Point				
2	Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.(50% students will perform Hamming Code and others will perform CRC)				
3	*Write a program to simulate Go back N and Selective Repeat Modes of Sliding Window* Protocol in peer to peer mode and demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.				
4	Write a program to demonstrate subletting and find the subnet masks.				
5	Write a program using TCP socket for wired network for following a.Say Hello to Each other (For all students)				

PUNE VIDYARTHI GRIHA'S

COLLEGE OF ENGINEERING, NASHIK.

INDEX

	b. File transfer (For all students) c. Calculator (Arithmetic) (50% students) d. Calculator (Trigonometry) (50% students) Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.				
6	Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.				
7	Write a program to analyze following packet formats captured through Wireshark for wired network. 1. Ethernet 2. IP 3.TCP 4. UDP				
8	Write a program for DNS lookup. Given an IP address input, it should return URL and vice-versa				
9	Installing and configure DHCP server and write a program to install the software on remote machine.				
10	Write a program using TCP sockets for wired network to implement a.Peer to Peer Chat b.Multiuser Chat				
11	Write a program using UDP sockets for wired network to implement a.Peer to Peer Chat b.Multiuser Chat				
12	Use network simulator NS2 to implement: a. Monitoring traffic for the given				

PUNE VIDYARTHI GRIHA'S

COLLEGE OF ENGINEERING, NASHIK.

INDEX

	topology b. Analysis of CSMA and Ethernet protocols c. Network Routing: Shortest path routing, AODV. d. Analysis of congestion control (TC P and UDP).				
13	Configure RIP/OSPF/BGP using packet Tracer				

Group A Assignments

Assignment No.	1
Title	**Study of Existing LAN**
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 1

Title: Study of Existing LAN

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.

Problem Statement: Setup a wired LAN using Layer 2 Switch and then IP switch of minimum four computers. It includes preparation of cable, testing of cable using line tester, configuration machine using IP addresses, testing using PING utility and demonstrate the PING packets captured traces using Wireshark Packet Analyzer Tool.

Part B: Extend the same Assignment for Wireless using Access Point.

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 Iternnet 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).

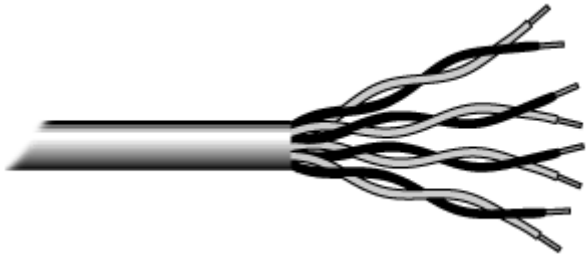


Fig.1. Unshielded twisted pair

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	LocalTalk & Telephone (Rarely used)
3	16 Mbps	10BaseT Ethernet
4	20 Mbps	Token Ring (Rarely used)

5	100 Mbps (2 pair) 1000 Mbps (4 pair)	100BaseT Ethernet Gigabit Ethernet
5e	1,000 Mbps	Gigabit 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.

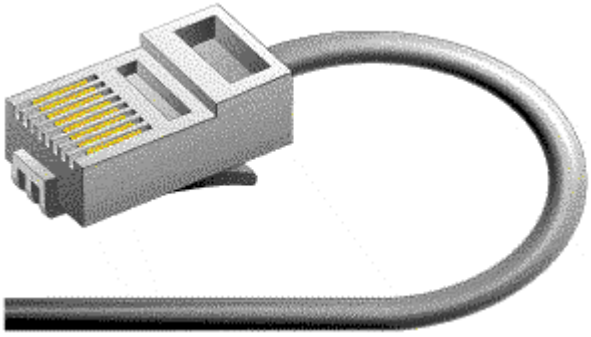


Fig. 2. RJ-45 connector

Shielded Twisted Pair (STP) Cable

Although UTP cable is the least expensive cable, it may be susceptible to radio and electrical frequency interference (it should not be too close to electric motors, fluorescent lights, etc.). If you must place cable in environments with lots of potential interference, or if you must place cable in extremely sensitive environments that may be susceptible to the electrical current in the UTP, shielded twisted pair may be the solution. Shielded cables can also help to extend the maximum distance of the cables.

Shielded twisted pair cable is available in three different configurations:

1. Each pair of wires is individually shielded with foil.
2. There is a foil or braid shield inside the jacket covering all wires (as a group).
3. There is a shield around each individual pair, as well as around the entire group of wires (referred to as double shield twisted pair).

Coaxial Cable

Coaxial cabling has a single copper conductor at its center. A plastic layer provides insulation between the center conductor and a braided metal shield (See fig. 3). The metal shield helps to block any outside interference from fluorescent lights, motors, and other computers.

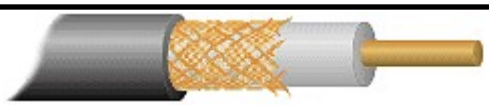


Fig. 3. Coaxial cable

Although coaxial cabling is difficult to install, it is highly resistant to signal interference. In addition, it can support greater cable lengths between network devices than twisted pair cable. The two types of coaxial cabling are thick coaxial and thin coaxial.

Thin coaxial cable is also referred to as thinnet. 10Base2 refers to the specifications for thin coaxial cable carrying Ethernet signals. The 2 refers to the approximate maximum segment length being 200 meters. In actual fact the maximum segment length is 185 meters. Thin coaxial cable has been popular in school networks, especially linear bus networks.

Thick coaxial cable is also referred to as thicknet. 10Base5 refers to the specifications for thick coaxial cable carrying Ethernet signals. The 5 refers to the maximum segment length being 500 meters. Thick coaxial cable has an extra protective plastic cover that helps keep moisture away from the center conductor. This makes thick coaxial a great choice when running longer lengths in a linear bus network. One disadvantage of thick coaxial is that it does not bend easily and is difficult to install.

Coaxial Cable Connectors

The most common type of connector used with coaxial cables is the Bayone-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 screw, onto the cable.

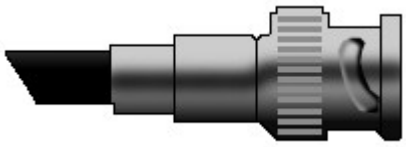


Fig. 4. BNC connector

Fiber Optic Cable

Fiber optic cabling consists of a center glass core surrounded by several layers of protective materials (See fig. 5). It transmits light rather than electronic signals eliminating the problem of electrical interference. This makes it ideal for certain environments that contain a large amount of electrical interference. It has also made it the standard for connecting networks between buildings, due to its immunity to the effects of

moisture and lighting.

Fiber optic cable has the ability to transmit signals over much longer distances than coaxial and twisted pair. It also has the capability to carry information at vastly greater speeds. This capacity broadens communication possibilities to include services such as video conferencing and interactive services. The cost of fiber optic cabling is comparable to copper cabling; however, it is more difficult to install and modify. 10BaseF refers to the specifications for fiber optic cable carrying Ethernet signals.

The center core of fiber cables is made from glass or plastic fibers (see fig 5). A plastic coating then cushions the fiber center, and kevlar fibers help to strengthen the cables and prevent breakage. The outer insulating jacket made of teflon or PVC.

Fig. 5. Fiber optic cable

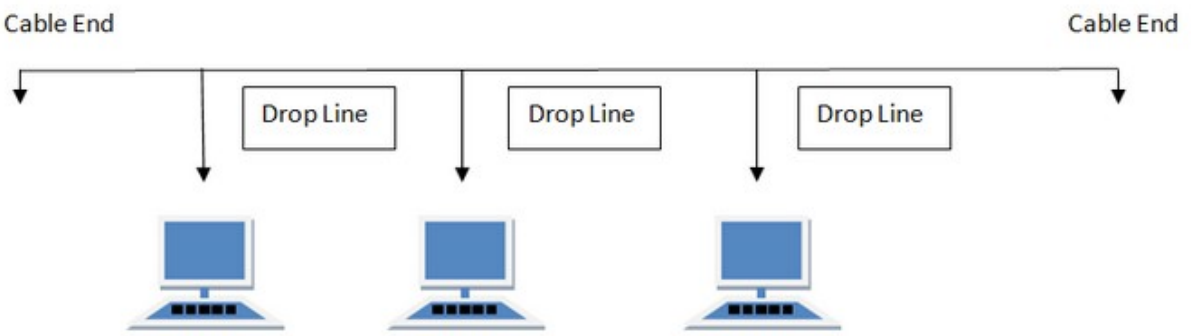
There are two common types of fiber cables -- single mode and multimode. Multimode cable has a larger diameter; however, both cables provide high bandwidth at high speeds. Single mode can provide more distance, but it is more expensive.

Types of Network Topology:

Network Topology is the schematic description of a network arrangement, connecting various nodes(sender and receiver) through lines of connection.

BUS Topology:

Bus topology is a network type in which every computer and network device is connected to single cable. When it has exactly two endpoints, then it is called Linear Bus topology.



Features of Bus Topology

- It transmits data only in one direction.
- Every device is connected to a single cable

Advantages of Bus Topology

1. It is cost effective.
2. Cable required is least compared to other network topology.
3. Used in small networks.
4. It is easy to understand.
5. Easy to expand joining two cables together.

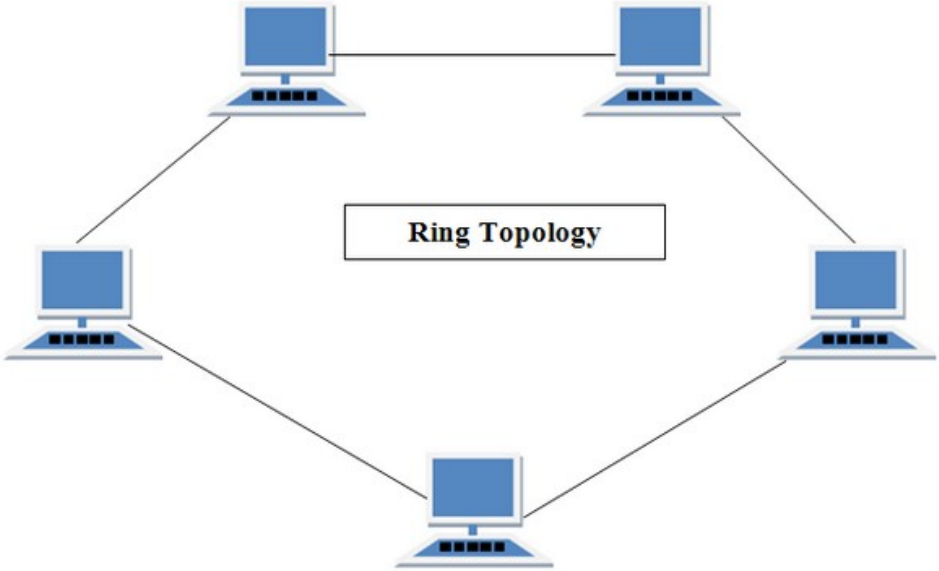
Disadvantages of Bus Topology

1. Cables fails then whole network fails.
2. If network traffic is heavy or nodes are more the performance of the network decreases.
3. Cable has a limited length.
4. It is slower than the ring topology.

RING Topology

It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbors for each device.

Features of Ring Topology



1. A number of repeaters are used for Ring topology with large number of nodes, because if someone wants to send some data to the last node in the ring topology with 100 nodes, then the data will have to pass through 99 nodes to reach the 100th node. Hence to prevent data loss repeaters are used in the network.
2. The transmission is unidirectional, but it can be made bidirectional by having 2 connections between each Network Node, it is called **Dual Ring Topology**.
3. In Dual Ring Topology, two ring networks are formed, and data flow is in opposite direction in them. Also, if one ring fails, the second ring can act as a backup, to keep the network up.
4. Data is transferred in a sequential manner that is bit by bit. Data transmitted, has to pass through each node of the network, till the destination node.

Advantages of Ring Topology

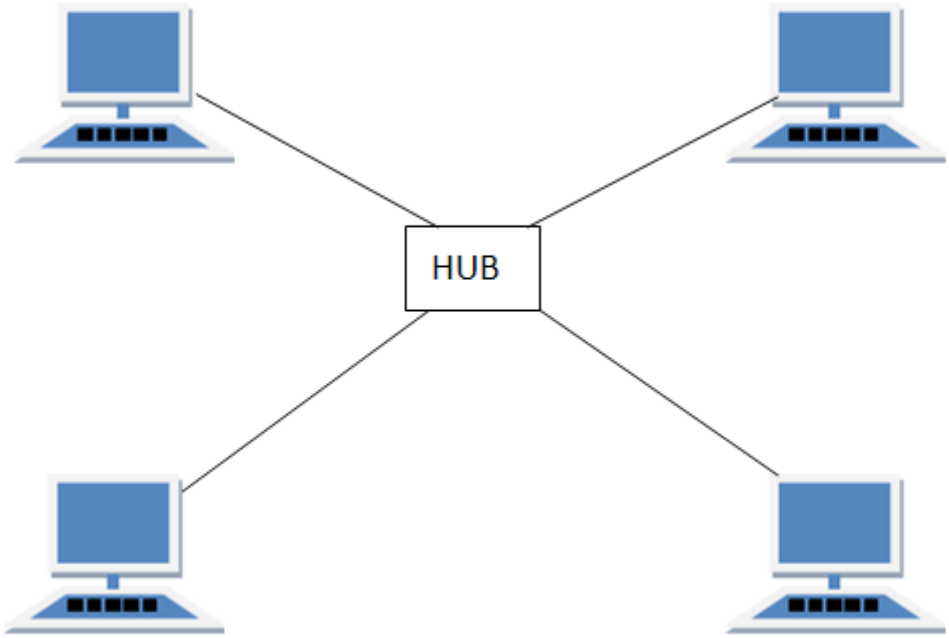
1. Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
2. Cheap to install and expand

Disadvantages of Ring Topology

1. Troubleshooting is difficult in ring topology.
2. Adding or deleting the computers disturbs the network activity.
3. Failure of one computer disturbs the whole network.

STAR Topology

In this type of topology all the computers are connected to a single hub through a cable. This hub is the central node and all others nodes are connected to the central node.



Features of Star Topology

1. Every node has its own dedicated connection to the hub.
2. Hub acts as a repeater for data flow.
3. Can be used with twisted pair, Optical Fibre or coaxial cable.

Advantages of Star Topology

1. Fast performance with few nodes and low network traffic.
2. Hub can be upgraded easily.
3. Easy to troubleshoot.
4. Easy to setup and modify.
5. Only that node is affected which has failed, rest of the nodes can work smoothly.

Disadvantages of Star Topology

1. Cost of installation is high.

2. Expensive to use.
3. If the hub fails then the whole network is stopped because all the nodes depend on the hub.
4. Performance is based on the hub that is it depends on its capacity

Network Devices:

Hubs

Hub is one of the basic icons of networking devices which works at physical layer and hence connect networking devices physically together. Hubs are fundamentally used in networks that use **twisted pair cabling** to connect devices.

They are designed to transmit the packets to the other appended devices without altering any of the transmitted packets received.

They act as pathways to direct electrical signals to travel along. They transmit the information regardless of the fact if data packet is destined for the device connected or not.

Hub falls in two categories:

Active Hub: They are smarter than the passive hubs. They not only provide the path for the data signals infact they regenerate, concentrate and strengthen the signals before sending them to their destinations. Active hubs are also termed as '**repeaters'**.

Passive Hub: They are more like point contact for the wires to built in the physical network. They have nothing to do with modifying the signals.

Switches

Switches are the linkage points of an Ethernet network. Just as in hub, devices in switches

are connected to them through twisted pair cabling. But the difference shows up in the manner both the devices; hub and a switch treat the data they receive.

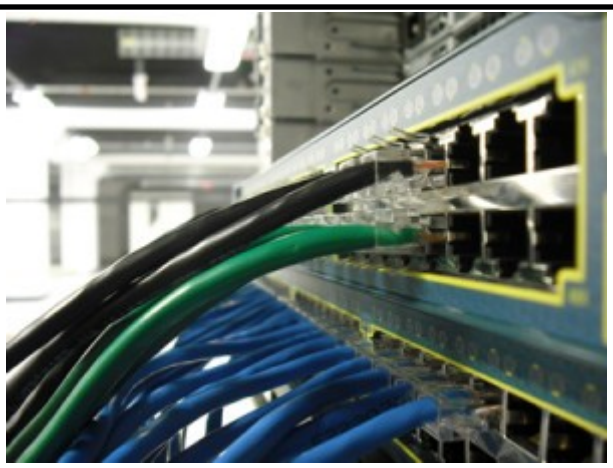
Hub works by sending the data to all the ports on the device whereas a **switch** transfers it only to that port which is connected to the destination device. A switch does so by having an in-built learning of the MAC address of the devices connected to it.

Since the transmission of data signals are well defined in a **switch** hence the network performance is consequently enhanced. Switches operate in **full-duplex** mode where devices can send and receive data from the switch at the simultaneously unlike in half-duplex mode.

The transmission speed in switches is double than in Ethernet hub transferring a 20Mbps connection into 30Mbps and a 200Mbps connection to become 300Mbps. Performance improvements are observed in networking with the extensive usage of switches in the modern days.

The following method will elucidate further how data transmission takes place via switches:

- **Cut-through transmission**: It allows the packets to be forwarded as soon as they are received. The method is prompt and quick but the possibility of error checking gets overlooked in such kind of packet data transmission.
- **Store and forward**: In this switching environment the entire packet are received and 'checked' before being forwarded ahead. The errors are thus eliminated before being propagated further. The downside of this process is that error checking takes relatively longer time consequently making it a bit slower in processing and delivering.
- **Fragment Free**: In a fragment free switching environment, a greater part of the packet is examined so that the switch can determine whether the packet has been caught up in a collision. After the collision status is determined, the packet is forwarded.



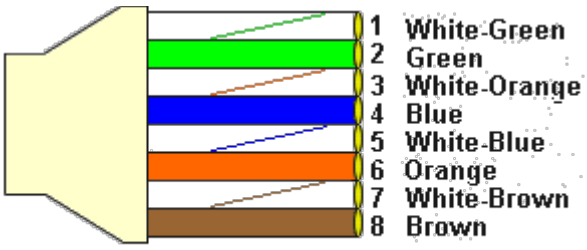
Bridges

A bridge is a computer networking device that builds the connection with the other bridge networks which use the same protocol. It works at the Data Link layer of the OSI Model and connects the different networks together and develops communication between them.

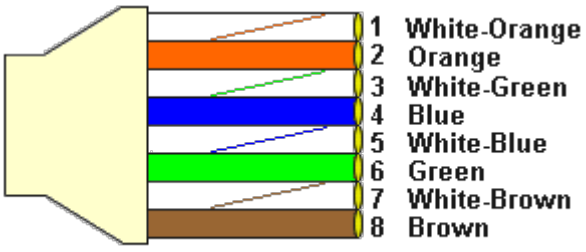
It connects two local-area networks; two physical LANs into larger logical LAN or two *segments* of the same LAN that use the same protocol.

Apart from building up larger networks, bridges are also used to segment larger networks into *smaller* portions.

Color Code:



568A CABLE END



568B CABLE END

Cable Preparation:



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

PING Command:

ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol (IP) network. It measures the round-trip time for messages sent from the originating host to a destination computer that are echoed back to the source.

The name comes from active sonar terminology that sends a pulse of sound and listens for the echo to detect objects under water, although it is sometimes interpreted as a backronym to *packet Internet groper*.Ping operates by sending Internet Control Message Protocol (ICMP/ICMP6) Echo Request packets to the target host and waiting for an ICMP Echo Reply.

The program reports errors, packet loss, and a statistical summary of the results, typically including the minimum, maximum, the mean round-trip times, and standard deviation of the mean.The command-line options of the ping utility and its output vary between the numerous implementations.

Conclusion: Thus we have studied wired lan setup and connection.

Assignment No.	3
Title	**Study of Hamming and CRC**
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 3

Title: Study of Hamming and CRC

OBJECTIVES:

1.To understand CRC and Hamming code

Problem Statement: Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.(50% students will perform Hamming Code and others will perform CRC)

Theory:

Hamming Distance:

One of the central concepts in coding for error control is the idea of the Hamming distance. The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words x and y as d(x, y).

The Hamming distance can easily be found if we apply the XOR operation (⊕) on the two words and count the number of 1s in the result.Note that the Hamming distance is a value greater than zero. The Hamming distance between two words is the number of differences between corresponding bits.

Example 10.4

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d(000, 011) is 2 because 000 ⊕ 011 is 011 (two 1s).

2. The Hamming distance d(10101, 11110) is 3 because 10101 ⊕ 11110 is 01011 (three 1s).

Minimum Hamming Distance

Although the concept of the Hamming distance is the central point in dealing with error detection and correction codes, the measurement that is used for designing a code is the minimum Hamming distance. In a set of words, the minimum Hamming distance is the smallest Hamming distance between all possible pairs. We use d min to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one.

The minimum Hamming distance is the smallest Hamming distance between all possible

pairs in a set of words.

Find the minimum Hamming distance of the coding scheme in Table 10.2. Solution We first find all the Hamming distances.

d(00000, 01011) = 3

d(01011, 10101) = 4

d(00000, 10101) = 3

d(01011, 11110) = 3

d(00000, 11110) = 4

d(10101, 11110) = 3

The d min in this case is 3.

Hamming Distance and Error

Before we explore the criteria for error detection or correction, let us discuss the relationship between the Hamming distance and errors occurring during transmission. When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error. In other words, the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is d(00000, 01101) = 3.

Minimum Distance for Error Detection

Now let us find the minimum Hamming distance in a code if we want to be able to detect up to s errors. If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. If our code is to detect up to s errors, the minimum distance between the valid codes must be s + 1, so that the received codeword does not match a valid codeword. In other words, if the minimum distance between all valid codewords is s + 1, the received codeword cannot be erroneously mistaken for another codeword. The distances are not enough (s + 1) for the receiver to accept it as valid. The error will be detected. We need to clarify a point here: Although a code with d min = s + 1 may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected.

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be d min = s + 1.

CRC:

A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents.

On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction, see bitfilters.

CRCs are so called because the *check* (data verification) value is a *redundancy* (it expands the message without adding information) and the algorithm is based on *cyclic* codes.

CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

The CRC was invented by W. Wesley Peterson in 1961; the 32-bit CRC function of Ethernet and many other standards is the work of several researchers and was published in 1975.

CRCs are based on the theory of cyclic error-correcting codes. The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks, was first proposed by W. Wesley Peterson in 1961. Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors: contiguous sequences of erroneous data symbols in messages.

This is important because burst errors are common transmission errors in many communication channels, including magnetic and optical storage devices. Typically an *n*-bit CRC applied to a data block of arbitrary length will detect any single error burst not longer than *n* bits and will detect a fraction $1/(1 - 2^{-n})$ of all longer error bursts.

Specification of a CRC code requires definition of a so-called generator polynomial. This polynomial becomes the divisor in a polynomial long division, which takes the message

as the dividend and in which the quotient is discarded and the remainder becomes the result.

The important caveat is that the polynomial coefficients are calculated according to the arithmetic of a finite field, so the addition operation can always be performed bitwise-parallel (there is no carry between digits). The length of the remainder is always less than the length of the generator polynomial, which therefore determines how long the result can be.

In practice, all commonly used CRCs employ the Galois field of two elements, GF(2). The two elements are usually called 0 and 1, comfortably matching computer architecture.

A CRC is called an *n*-bit CRC when its check value is *n* bits long. For a given *n*, multiple CRCs are possible, each with a different polynomial. Such a polynomial has highest degree *n*, which means it has *n* + 1 terms.

In other words, the polynomial has a length of *n* + 1; its encoding requires *n* + 1 bits. Note that most polynomial specifications either drop the MSB or LSB, since they are always 1. The CRC and associated polynomial typically have a name of the form CRC-*n*-XXX as in the table below.

The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC: it uses the generator polynomial *x* + 1 (two terms), and has the name CRC-1.

Application

A CRC-enabled device calculates a short, fixed-length binary sequence, known as the *check value* or *CRC*, for each block of data to be sent or stored and appends it to the data, forming a *codeword*.

When a codeword is received or read, the device either compares its check value with one freshly calculated from the data block, or equivalently, performs a CRC on the whole codeword and compares the resulting check value with an expected *residue* constant.

If the CRC check values do not match, then the block contains a data error.

The device may take corrective action, such as rereading the block or requesting that it be sent again. Otherwise, the data is assumed to be error-free (though, with some small

probability, it may contain undetected errors; this is the fundamental nature of error-checking).

Data integrity

CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of messages delivered. However, they are not suitable for protecting against intentional alteration of data.

Firstly, as there is no authentication, an attacker can edit a message and recompute the CRC without the substitution being detected. When stored alongside the data, CRCs and cryptographic hash functions by themselves do not protect against *intentional* modification of data.

Any application that requires protection against such attacks must use cryptographic authentication mechanisms, such as message authentication codes or digital signatures (which are commonly based on cryptographic hash functions).

Secondly, unlike cryptographic hash functions, CRC is an easily reversible function, which makes it unsuitable for use in digital signatures.

Thirdly, CRC is a linear function with a property that ; as a result, even if the CRC is encrypted with a stream cipher that uses XOR as its combining operation (or mode of block cipher which effectively turns it into a stream cipher, such as OFB or CFB), both the message and the associated CRC can be manipulated without knowledge of the encryption key; this was one of the well-known design flaws of the Wired Equivalent Privacy (WEP) protocol.[5]

To compute an n-bit binary CRC, line the bits representing the input in a row, and position the $(n + 1)$-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row.

Example:

In this example, we shall encode 14 bits of message with a 3-bit CRC, with a polynomial $x^3 + x + 1$. The polynomial is written in binary as the coefficients; a 3rd-order polynomial has 4 coefficients ($1x^3 + 0x^2 + 1x + 1$). In this case, the coefficients are 1, 0, 1 and 1. The

result of the calculation is 3 bits long.

Start with the message to be encoded:

11010011101100

This is first padded with zeros corresponding to the bit length *n* of the CRC. Here is the first calculation for computing a 3-bit CRC:

11010011101100 000 <--- input right padded by 3 bits
1011 <--- divisor (4 bits) = $x^3 + x + 1$

01100011101100 000 <--- result

The algorithm acts on the bits directly above the divisor in each step. The result for that iteration is the bitwise XOR of the polynomial divisor with the bits above it.

The bits not above the divisor are simply copied directly below for that step. The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

11010011101100 000 <--- input right padded by 3 bits
1011 <--- divisor
01100011101100 000 <--- result (note the first four bits are the XOR with the divisor beneath, the rest of the bits are unchanged)
1011 <--- divisor ...
00111011101100 000
1011
00010111101100 000
1011
00000001101100 000 <--- note that the divisor moves over to align with the next 1 in the dividend (since quotient for that step was zero)
1011 (in other words, it doesn't necessarily move one bit per iteration)
00000000110100 000
1011
00000000011000 000
1011

00000000001110 000
1011
00000000000101 000
101 1

00000000000000 100 <--- remainder (3 bits). Division algorithm stops here as dividend is equal to zero.

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row.

These *n* bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some postprocessing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

11010011101100 100 <--- input with check value
1011 <--- divisor
01100011101100 100 <--- result
1011 <--- divisor ...
00111011101100 100

.....

00000000001110 100
1011
00000000000101 100
101 1
---
0 <--- remainder

Conclusion: Thus we have studied Hamming and CRC Code.

Assignment No.	4
Title	Study of Go back N and Selective Repeat Modes of Sliding Window Protocol.
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 4

Title: Study of Go back N and Selective Repeat Modes of Sliding Window Protocol.

OBJECTIVES:

1.To understand working of Go back N and Selective Repeat Modes of Sliding Window Protocol.

Problem Statement: Write a program to simulate Go back N and Selective Repeat Modes of Sliding Window Protocol in peer to peer mode and demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Theory:

SLIDING WINDOW PROTOCOL

Sliding Window Protocols assumes two-way communication (full duplex). It uses two types of frames:

- Data
- Ack (sequence number of last correctly received frame)

The basic idea of sliding window protocol is that both sender and receiver keep a ``window" of acknowledgment. The sender keeps the value of expected acknowledgment; while the receiver keeps the value of expected receiving frame. When it receives an acknowledgment from the receiver, the sender advances the window. When it receives the expected frame, the receiver advances the window.

In transmit flow control, sliding window is a variable-duration window that allows a sender to transmit a specified number of data units before an acknowledgement is received or before a specified event occurs.

An example of a sliding window in packet transmission is one in which, after the sender fails to receive an acknowledgement for the first transmitted packet, the sender "slides" the window, i.e. resets the window, and sends a second packet. This process is repeated for the specified number of times before the sender interrupts transmission. Sliding window is sometimes (loosely) called *acknowledgement delay period*.

For example, supposing a fixed window size of *m* packets, a sender may send out packets

$[n \dots (n+m-1)]$ before receiving any acknowledgement. If acknowledgement arrives from the receiver for packet *n*, then the range (window) of unacknowledged

packets slides to $[(n+1) \dots (n+m)]$, and the sender is able to send out packet (*n*

+ *m*). In some way, "sliding" signifies a FIFO operation, trimming the range at one end, extending it at the other end.

The purpose of the sliding window is to increase throughput. Let's denote the round trip time with RTT. The time necessary to transfer and acknowledge *K* (a big number of)

packets is roughly $\mathrm{RTT} \cdot K/(2m)$ (in one round trip, 2*m* packets and 2*m* ACKs are delivered). However, the size of the window (in bytes) should not grow above "capacity of the path" (the sum of affected network buffer sizes of all hops along the path): windows that are too big do not increase throughput; they only increase latency, the number of packets transmitted out-of-order, and memory usage.

In practice, protocols often adapt the window size to the link's speed and actual saturation or congestion.

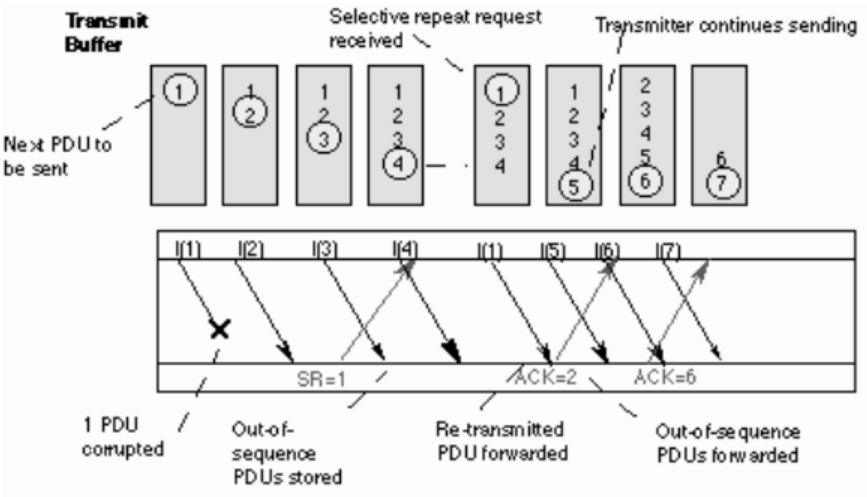
Selective Repeat ARQ

Selective Repeat ARQ is a specific instance of the Automatic Repeat-reQuest (ARQ) Protocol, in which the sending process continues to send a number of frames specified by a *window size* even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error.

The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every ACK it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its *window*.

The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its *window*, it re-sends the frame number given by the ACKs, and then continues where it left off.

The size of the sending and recieving windows must be equal, and half the maximum sequence number (assuming that sequence numbers are numbered from 0 to n-1) to avoid miscommunication in all cases of packets being dropped. The sender moves its window for every packet that is acknowledged.



APPLICATION

1. The sliding window implements reliability at both the data-link layer and the transport layer of the network protocol stack, like TCP/IP.

FAQS

Please refer the above theory for answers

1. Explain the Sliding Window protocol
2. What is difference between go-back-n and selective repeat?
3. What is the basic need for having sequence numbers?

Conclusion: Thus we have studied working of Go back N and Selective Repeat Modes of Sliding Window Protocol.

Assignment No.	5
Title	Write a program to demonstrate subletting and find the subnet masks.
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 5

Title: Write a program to demonstrate subletting and find the subnet masks.

OBJECTIVES:

1.To understand working of subletting and find the subnet masks.

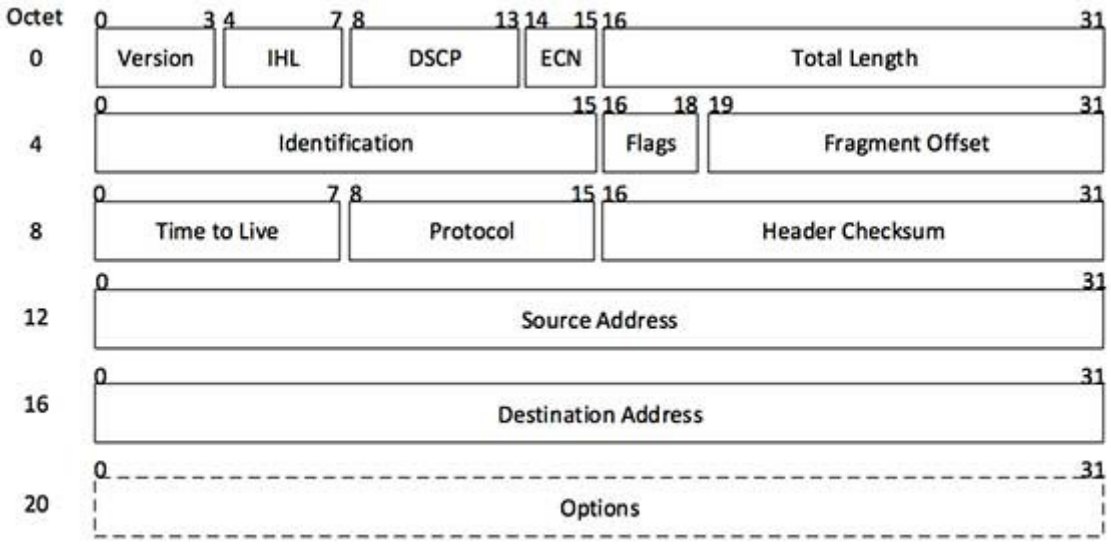
Problem Statement: Write a program to demonstrate subletting and find the subnet masks.

Theory:

Internet Protocol being a layer-3 protocol (OSI) takes data Segments from layer-4 (Transport) and divides it into packets. IP packet encapsulates data unit received from above layer and add to its own header information.

(IP Encapsulation)

The encapsulated data is referred to as IP Payload. IP header contains all the necessary information to deliver the packet at the other end.



[Image: IP Header]

IP header includes many relevant information including Version Number, which, in this context, is 4. Other details are as follows:

- **Version:** Version no. of Internet Protocol used (e.g. IPv4).
- **IHL:** Internet Header Length; Length of entire IP header.
- **DSCP:** Differentiated Services Code Point; this is Type of Service.
- **ECN:** Explicit Congestion Notification; It carries information about the congestion seen in the route.
- **Total Length:** Length of entire IP Packet (including IP header and IP Payload).
- **Identification:** If IP packet is fragmented during the transmission, all the fragments contain same identification number. to identify original IP packet they belong to.
- **Flags:** As required by the network resources, if IP Packet is too large to handle, these 'flags' tells if they can be fragmented or not. In this 3-bit flag, the MSB is always set to '0'.
- **Fragment Offset:** This offset tells the exact position of the fragment in the original IP Packet.
- **Time to Live:** To avoid looping in the network, every packet is sent with some TTL value set, which tells the network how many routers (hops) this packet can cross. At each hop, its value is decremented by one and when the value reaches zero, the packet is discarded.
- **Protocol:** Tells the Network layer at the destination host, to which Protocol this packet belongs to, i.e. the next level Protocol. For example protocol number of ICMP is 1, TCP is 6 and UDP is 17.
- **Header Checksum:** This field is used to keep checksum value of entire header which is then used to check if the packet is received error-free.
- **Source Address:** 32-bit address of the Sender (or source) of the packet.
- **Destination Address:** 32-bit address of the Receiver (or destination) of the packet.
- **Options:** This is optional field, which is used if the value of IHL is greater than 5. These options may contain values for options such as Security, Record Route, Time Stamp, etc.

IPv4 supports three different types of addressing modes.:

Unicast Addressing Mode:

In this mode, data is sent only to one destined host. The Destination Address field

contains 32- bit IP address of the destination host. Here the client sends data to the targeted server:

Broadcast Addressing Mode:

In this mode, the packet is addressed to all the hosts in a network segment. The Destination Address field contains a special broadcast address, i.e. **255.255.255.255**. When a host sees this packet on the network, it is bound to process it. Here the client sends a packet, which is entertained by all the Servers:

Multicast Addressing Mode:

This mode is a mix of the previous two modes, i.e. the packet sent is neither destined to a single host nor all the hosts on the segment. In this packet, the Destination Address contains a special address which starts with 224.x.x.x and can be entertained by more than one host.

Here a server sends packets which are entertained by more than one servers. Every network has one IP address reserved for the Network Number which represents the network and one IP address reserved for the Broadcast Address, which represents all the hosts in that network.

Hierarchical Addressing Scheme

IPv4 uses hierarchical addressing scheme. An IP address, which is 32-bits in length, is divided into two or three parts as depicted:

A single IP address can contain information about the network and its sub-network and ultimately the host. This scheme enables the IP Address to be hierarchical where a network can have many sub-networks which in turn can have many hosts.

Subnet Mask

The 32-bit IP address contains information about the host and its network. It is very necessary to distinguish both. For this, routers use Subnet Mask, which is as long as the size of the network address in the IP address. Subnet Mask is also 32 bits long. If the IP address in binary is ANDed with its Subnet Mask, the result yields the Network address. For example, say the IP Address is 192.168.1.152 and the Subnet Mask is 255.255.255.0 then:

IP	192.168.1.152	11000000	10101000	00000001	10011000	ANDed
Mask	255.255.255.0	11111111	11111111	11111111	00000000	
Network	192.168.1.0	11000000	10101000	00000001	00000000	Result

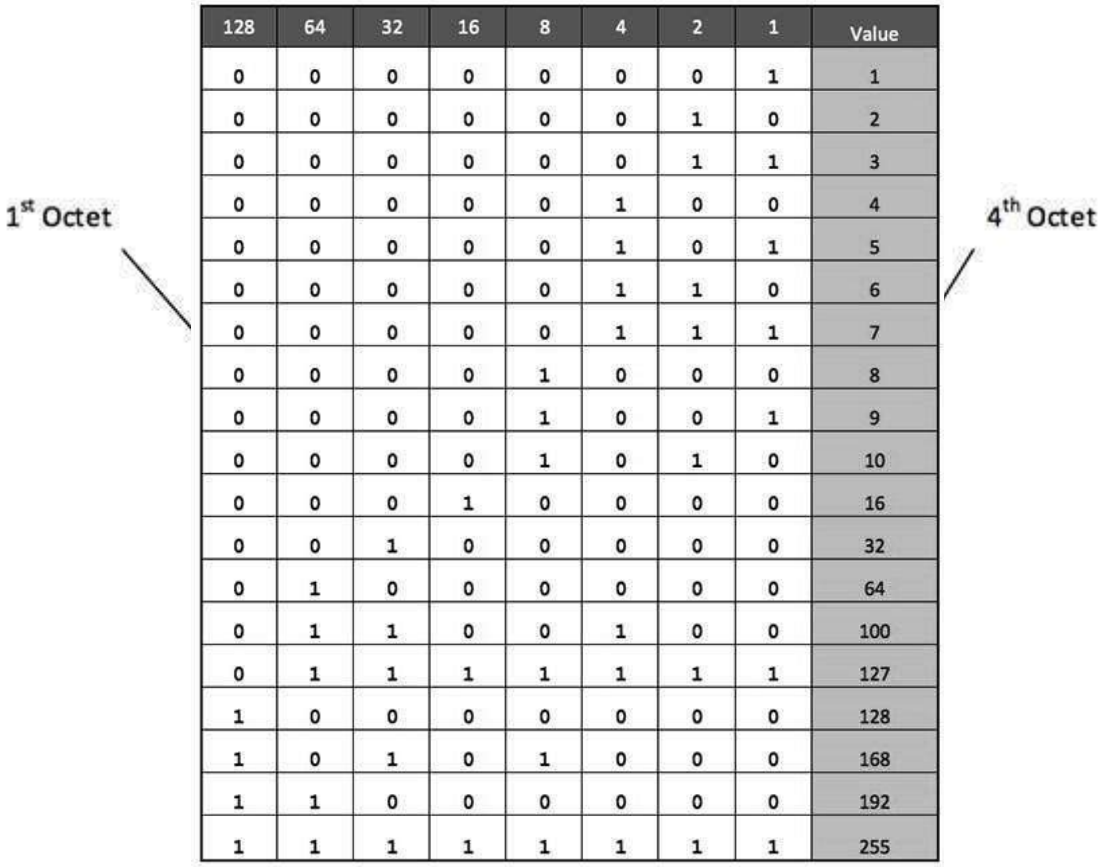
This way the Subnet Mask helps extract the Network ID and the Host from an IP Address. It can be identified now that 192.168.1.0 is the Network number and 192.168.1.152 is the host on that network.

Binary Representation

The positional value method is the simplest form of converting binary from decimal value. IP address is 32 bit value which is divided into 4 octets. A binary octet contains 8 bits and the value of each bit can be determined by the position of bit value '1' in the octet.

MSB	8th	7th	6th	5th	4th	3rd	2nd	1st	LSB
	1	1	1	1	1	1	1	1	
Positional Value	128	64	32	16	8	4	2	1	

Positional value of bits is determined by 2 raised to power (position – 1), that is the value of a bit 1 at position 6 is 2^(6-1) that is 2^5 that is 32. The total value of the octet is determined by adding up the positional value of bits. The value of 11000000 is 128+64 = 192. Some examples are shown in the table below:



128	64	32	16	8	4	2	1	Value
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	1	5
0	0	0	0	0	1	1	0	6
0	0	0	0	0	1	1	1	7
0	0	0	0	1	0	0	0	8
0	0	0	0	1	0	0	1	9
0	0	0	0	1	0	1	0	10
0	0	0	1	0	0	0	0	16
0	0	1	0	0	0	0	0	32
0	1	0	0	0	0	0	0	64
0	1	1	0	0	1	0	0	100
0	1	1	1	1	1	1	1	127
1	0	0	0	0	0	0	0	128
1	0	1	0	1	0	0	0	168
1	1	0	0	0	0	0	0	192
1	1	1	1	1	1	1	1	255

The number of networks and the number of hosts per class can be derived by this formula:

Number of networks = 2^network_bits

Number of Hosts/Network = 2^host_bits – 2

When calculating hosts' IP addresses, 2 IP addresses are decreased because they cannot be assigned to hosts, i.e. the first IP of a network is network number and the last IP is reserved for Broadcast IP.

Class A Address

The first bit of the first octet is always set to 0 (zero). Thus the first octet ranges from 1 – 127, i.e.

00000001 – 01111111
1 – 127

Class A addresses only include IP starting from 1.x.x.x to 126.x.x.x only. The IP range 127.x.x.x is reserved for loopback IP addresses.

The default subnet mask for Class A IP address is 255.0.0.0 which implies that Class A

addressing can have 126 networks (2^7-2) and 16777214 hosts (2^{24}-2).

Class A IP address format is thus: **0NNNNNNN**.HHHHHHHH.HHHHHHHH.HHHHHHHH

Class B Address

An IP address which belongs to class B has the first two bits in the first octet set to 10, i.e.

10000000 – **10**111111
128 – 191

Class B IP Addresses range from 128.0.x.x to 191.255.x.x. The default subnet mask for Class B is 255.255.x.x.

Class B has 16384 (2^{14}) Network addresses and 65534 (2^{16}-2) Host addresses.

Class B IP address format is: **10NNNNNN.NNNNNNNN**.HHHHHHHH.HHHHHHHH

Class C Address

The first octet of Class C IP address has its first 3 bits set to 110, that is:

11000000 – **110**11111
192 – 223

Class C IP addresses range from 192.0.0.x to 223.255.255.x. The default subnet mask for Class C is 255.255.255.x.

Class C gives 2097152 (2^{21}) Network addresses and 254 (2^8-2) Host addresses.

Class C IP address format is: **110NNNNN.NNNNNNNN.NNNNNNNN**.HHHHHHHH

Class D Address

Very first four bits of the first octet in Class D IP addresses are set to 1110, giving a range of:

11100000 – **1110**1111
224 – 239

Class D has IP address rage from 224.0.0.0 to 239.255.255.255. Class D is reserved for Multicasting. In multicasting data is not destined for a particular host, that is why there is

no need to extract host address from the IP address, and Class D does not have any subnet mask.

Class E Address

This IP Class is reserved for experimental purposes only for R&D or Study. IP addresses in this class ranges from 240.0.0.0 to 255.255.255.254. Like Class D, this class too is not equipped with any subnet mask.

Each IP class is equipped with its own default subnet mask which bounds that IP class to have prefixed number of Networks and prefixed number of Hosts per network. Classful IP addressing does not provide any flexibility of having less number of Hosts per Network or more Networks per IP Class.

CIDR or **Classless Inter Domain Routing** provides the flexibility of borrowing bits of Host part of the IP address and using them as Network in Network, called Subnet. By using subnetting, one single Class A IP address can be used to have smaller sub-networks which provides better network management capabilities.

Class A Subnets

In Class A, only the first octet is used as Network identifier and rest of three octets are used to be assigned to Hosts (i.e. 16777214 Hosts per Network). To make more subnet in Class A, bits from Host part are borrowed and the subnet mask is changed accordingly.

For example, if one MSB (Most Significant Bit) is borrowed from host bits of second octet and added to Network address, it creates two Subnets (2^1=2) with (2^{23}-2) 8388606 Hosts per Subnet.

The Subnet mask is changed accordingly to reflect subnetting. Given below is a list of all possible combination of Class A subnets:

Network Bits	Subnet Mask	Bits Borrowed	Subnets	Hosts/Subnet
8	255.0.0.0	0	1	16777214
9	255.128.0.0	1	2	8388606
10	255.192.0.0	2	4	4194302
11	255.224.0.0	3	8	2097150
12	255.240.0.0	4	16	1048574
13	255.248.0.0	5	32	524286
14	255.252.0.0	6	64	262142
15	255.254.0.0	7	128	131070
16	255.255.0.0	8	256	65534
17	255.255.128.0	9	512	32766
18	255.255.192.0	10	1024	16382
19	255.255.224.0	11	2048	8190
20	255.255.240.0	12	4096	4094
21	255.255.248.0	13	8192	2046
22	255.255.252.0	14	16384	1022
23	255.255.254.0	15	32768	510
24	255.255.255.0	16	65536	254
25	255.255.255.128	17	131072	126
26	255.255.255.192	18	262144	62
27	255.255.255.224	19	524288	30
28	255.255.255.240	20	1048576	14
29	255.255.255.248	21	2097152	6
30	255.255.255.252	22	4194304	2

In case of subnetting too, the very first and last IP address of every subnet is used for Subnet Number and Subnet Broadcast IP address respectively. Because these two IP addresses cannot be assigned to hosts, sub-netting cannot be implemented by using more than 30 bits as Network Bits, which provides less than two hosts per subnet.

Class B Subnets

By default, using Classful Networking, 14 bits are used as Network bits providing (2^{14}) 16384 Networks and (2^{16}-2) 65534 Hosts. Class B IP Addresses can be subnetted the same way as Class A addresses, by borrowing bits from Host bits. Below is given all possible combination of Class B subnetting:

Network Bits	Subnet Mask	Bits Borrowed	Subnets	Hosts/Subnet
16	255.255.0.0	0	0	65534
17	255.255.128.0	1	2	32766
18	255.255.192.0	2	4	16382
19	255.255.224.0	3	8	8190
20	255.255.240.0	4	16	4094
21	255.255.248.0	5	32	2046
22	255.255.252.0	6	64	1022
23	255.255.254.0	7	128	510
24	255.255.255.0	8	256	254
25	255.255.255.128	9	512	126
26	255.255.255.192	10	1024	62
27	255.255.255.224	11	2048	30
28	255.255.255.240	12	4096	14
29	255.255.255.248	13	8192	6
30	255.255.255.252	14	16384	2

Class C Subnets

Class C IP addresses are normally assigned to a very small size network because it can only have 254 hosts in a network. Given below is a list of all possible combination of subnetted Class B IP address:

Network Bits	Subnet Mask	Bits Borrowed	Subnets	Hosts/Subnet
24	255.255.255.0	0	1	254
25	255.255.255.128	1	2	126
26	255.255.255.192	2	4	62
27	255.255.255.224	3	8	30
28	255.255.255.240	4	16	14
29	255.255.255.248	5	32	6
30	255.255.255.252	6	64	2

Conclusion: Thus we have studied working of subletting and find the subnet masks.

Assignment No.	7
Title	working of TCP Socket
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 7

Title: Working of TCP Socket.

OBJECTIVES:

1.To understand working of TCP Socket.

Problem Statement: Write a program using TCP socket for wired network for following

a. Say Hello to Each other (For all students)

b. File transfer (For all students)

c. Calculator (Arithmetic) (50% students)

d. Calculator (Trigonometry) (50% students)

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

peer mode.

Theory:

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:

<sys/socket.h>

Definitions for the most basic of socket structures with the BSD socket API *<sys/types.h>*

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

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

<sys/un.h>

Definitions and data type declarations for SOCK_UNIX streams

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:

- Creating a TCP socket, with a call to socket().

- 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).

- Preparing the socket to listen for connections (making it a listening socket), with a call to listen().

- 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.

- Communicating with the remote host, which can be done through send() and recv().

- 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:
- Creating a TCP socket, with a call to socket().
- 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.)
- 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:

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: o PF_INET for network protocol IPv4 or
  - o 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() It 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.

Application:

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

Conclusion: Thus we have studied Working of TCP Socket.

Assignment No.	8
Title	working of UDP Socket
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 8

Title: Working of UDP Socket.

OBJECTIVES:

1.To understand working of UDP Socket.

Problem Statement: Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Theory:

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:

<sys/socket.h>
Definitions for the most basic of socket structures with the BSD socket API *<sys/types.h>*
Basic data types associated with structures within the BSD socket API *<netinet/in.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");
   recsize = recvfrom(sock, (void *)hz, 100, 0, (struct sockaddr
*)&sa, fromlen);
   printf ("recsize: %d\n ",recsize);
   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:

- socket
- pointer to buffer for data
- size of buffer
- flags (same as in read or other receive socket function)

- address struct of sending peer
- length of address struct of sending peer.

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

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 No.	09
Title	packet formats captured through Wireshark for wired network
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 09

Title: packet formats captured through Wireshark for wired network.

OBJECTIVES:

1.To understand packet formats captured through Wireshark for wired network.

Problem Statement: Write a program to analyze following packet formats captured through Wireshark for wired network. 1. Ethernet 2. IP 3.TCP 4. UDP.

Theory:

Ethernet:

Ethernet is a way of connecting computers together in a local area network. It has been the most widely used method of linking computers together in LANs since the 1990s. The basic idea of its design is that multiple computers have access to it and can send data at any time. This is comparatively easy to engineer.

If two computers send data at the same time, a collision will occur. When this happens, the data sent is not usable. In general, both computers will stop sending, and wait a random amount of time, before they try again. A special protocol was developed to deal with such problems. It is called Carrier sense multiple access with collision detection or CSMA/CD.

Different cable types

There are different Ethernet standards. Today, Ethernet cables look like thick telephone cables. They connect to boxes called hubs or switches. Each cable runs from a computer's network interface card (NIC) to such a box. This cable is called 10BaseT or 100BaseT, or 1000BaseT Cable.

All cable types:

- 10Base2 and 10Base5: These coaxial cables are like those used in television, but they are a bit thinner. They are also called "thinnet" or "coax". Each computer has a "T" plugged into it, and cables plug into each side of the "T". Sometimes, instead of a "T", a *vampire tap* is used which goes through the skin of a cable. It supports 10MBits per second transfer speed, and was the first to be adopted.
- 10BaseT: Cables look like thick phone cables, but with 8 copper wires instead of 2 or 4, and they go from each computer' to a Hub or a Switch. Supported speed is 10 MBit/second.
- 10BaseF: Same as 10BaseT, but cables transmit light pulses, instead of electrical signals.
- 100BaseT: Cables look the same as 10BaseT, but can run at up to 100 MBits per second

- 1000BaseT: Cables look the same as 10BaseT, but can run at up to 1GBit (1000MBit) per second.

Today, the cables for 10BaseT, 100BaseT, and 1000BaseT are the same. They use unshielded twisted pair of Category 5 (UTP-Cat5) or 5e for that. Shielded cable (STP-Cat5 or Cat5e) can be used when there is a lot of electrical noise, and Category 6 (UTP-Cat6 or STP-Cat6) works better with faster signals such as 1GBit or 10GBit.

Internet Protocol

The **Internet Protocol** (**IP**) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

Historically, IP was the connectionless datagram service in the original *Transmission Control Program* introduced by Vint Cerf and Bob Kahn in 1974; the other being the connection-oriented Transmission Control Protocol (TCP). The Internet protocol suite is therefore often referred to as TCP/IP.

The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol Version 6 (Ipv6).

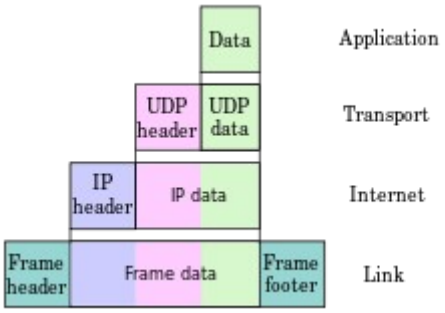
Function:

The Internet Protocol is responsible for addressing hosts, encapsulating data into datagrams (including fragmentation and reassembly) and routing datagrams from a source host to a destination host across one or more IP networks.For these purposes, the Internet Protocol defines the format of packets and provides an addressing system.

Each datagram has two components: a header and a payload. The IP header includes source IP address, destination IP address, and other metadata needed to route and deliver the datagram. The payload is the data that is transported. This method of nesting the data payload in a packet with a header is called encapsulation.

IP addressing entails the assignment of IP addresses and associated parameters to host interfaces. The address space is divided into subnetworks, involving the designation of network prefixes. IP routing is performed by all hosts, as well as routers, whose main function is to transport packets across network boundaries. Routers communicate with

one another via specially designed routing protocols, either interior gateway protocols or exterior gateway protocols, as needed for the topology of the network.



TCP:

After going through the various layers of the Model, it's time to have a look at the TCP protocol and to study its functionality. This section will help the reader to get to know about the concepts and characteristics of the TCP, and then gradually dive into the details of TCP like connection establishment/closing, communication in TCP and why the TCP protocol is called a reliable as well as an adaptive protocol. This section will end with a comparison between UDP and TCP followed by a nice exercise which would encourage readers to solve more and more problems.

Before writing this section, the information has been studied from varied sources like TCP guide, RFC's, tanenbaum book and the class notes.

What is TCP?

In theory, a transport layer protocol could be a very simple software routine, but the TCP protocol cannot be called simple. Why use a transport layer which is as complex as TCP? The most important reason depends on IP's unreliability. In fact all the layers below TCP are unreliable and deliver the datagram hop-by-hop. The IP layer delivers the datagram hop-by-hop and does not guarantee delivery of a datagram; it is a connectionless system. IP simply handles the routing of datagrams; and if problems occur, IP discards the packet without a second thought, generating an error message back to the sender in the process. The task of ascertaining the status of the datagrams sent over a network and handling the resending of information if parts have been discarded falls to TCP.

Most users think of TCP and IP as a tightly knit pair, but TCP can be, and frequently is,

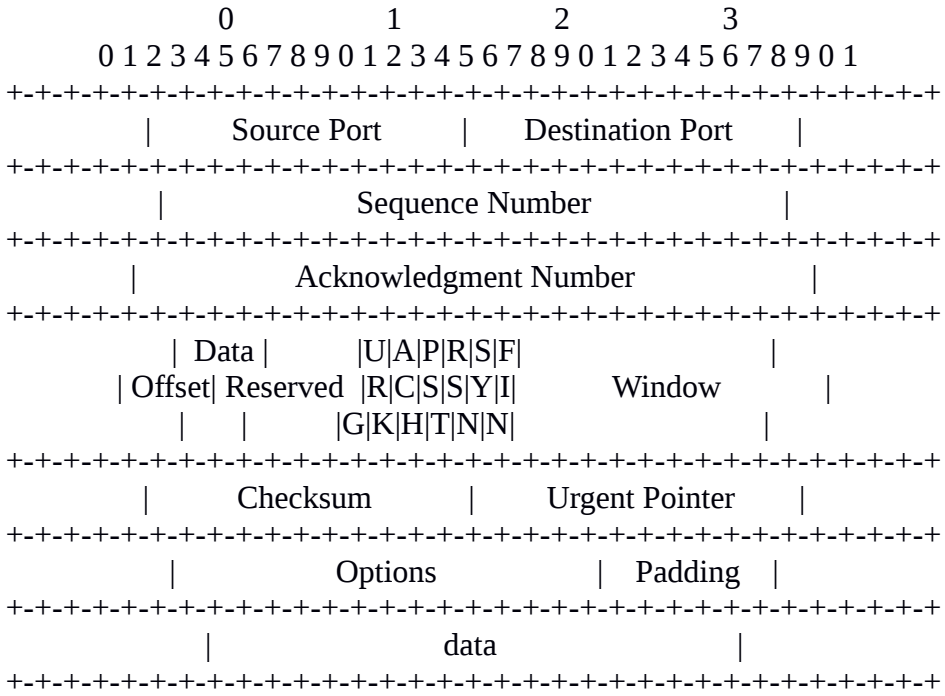
used with other transport protocols.

For example, TCP or parts of it are used in the File Transfer Protocol (FTP) and the Simple Mail Transfer Protocol (SMTP), both of which do not use IP.

The Transmission Control Protocol provides a considerable number of services to the IP layer and the upper layers. Most importantly, it provides a connection-oriented protocol to the upper layers that enable an application to be sure that a datagram sent out over the network was received in its entirety. In this role, TCP acts as a message-validation protocol providing reliable communications. If a datagram is corrupted or lost, it is usually TCP (not the applications in the higher layers) that handles the retransmission.

TCP Header structure

TCP segments are sent as Internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses. A TCP header follows the Internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.



TCP Header Format

Source Port: 16 bits The source port number.

Destination Port: 16 bits The destination port number.

Sequence Number: 32 bit The sequence number of the first data octet in this segment

(except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.
Reserved: 6 bits Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant

ACK: Acknowledgment field significant

PSH: Push Function

RST: Reset the connection

SYN: Synchronize sequence numbers

FIN: No more data from sender

Window: 16 bits The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the

TCP/Network interface in the arguments or results of calls by the TCP on the IP.

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

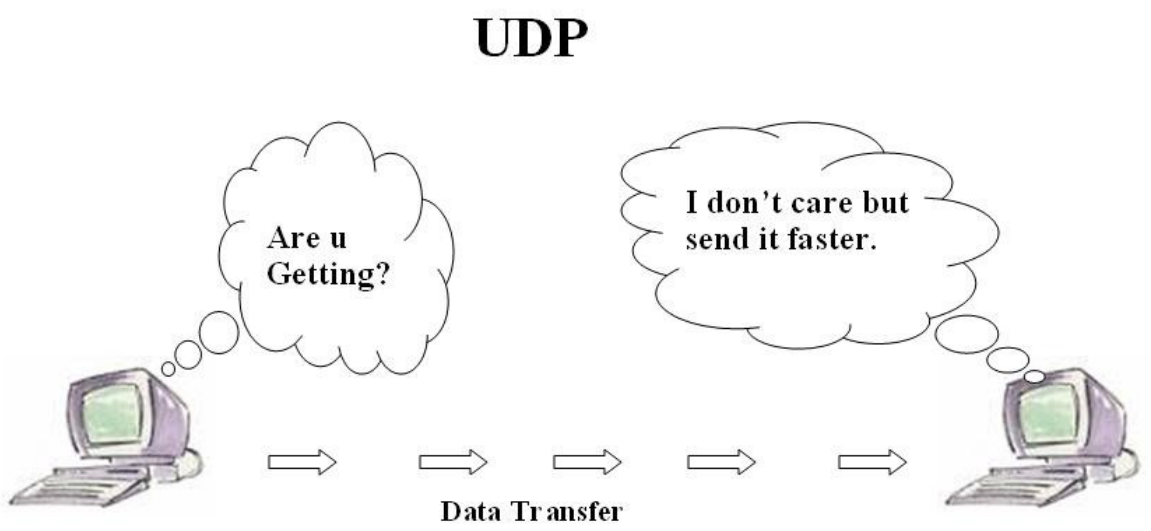
Urgent Pointer: 16 bits This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets. The option-length counts the two octets of option-kind and option-length as well as the option-data octets. Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

What is UDP?



'Figure 2: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 figur 3. In the user datagrams first 8 bytes contains header information and the remaining bytes contains data.

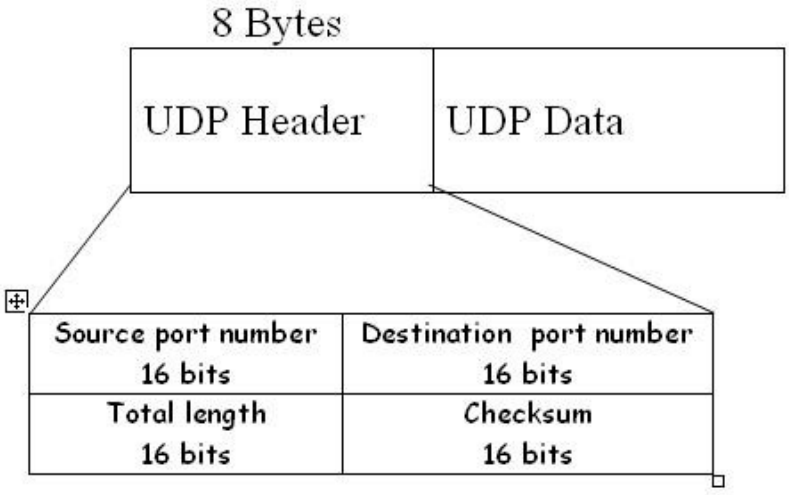


Figure 3:UDP datagrams

***Source port number*:** This is a port number used by source host,who is transferring data.

It is 16 bit longs. So port numbers range between 0 to 65,535.

***Destination port number*:** This is a port number used by Destination host, who is getting data. It is also 16 bits long and also same number of port range like source host.

length: Length field is a 16 bits field. It contains the total length of the user datagram, header and data.

***Checksum*:** The UDP checksum is optional. It is used to detect error fro the data. If the field is zero then checksum is not calculated. And true calculated then field contains 1.

Characteristics of UDP

The characteristics of UDP are given below.

• End-to-end. UDP can identify a specific process running on a computer.

• Unreliable, connectionless delivery (e.g. USPS)::

UDP uses a connectionless communication setup. In this UDP does not need to establish a connection before sending data. Communication consists only of the data segments themselves

• Same best effort semantics as IP

• No ack, no sequence, no flow control

• Subject to loss, duplication, delay, out-of-order, or loss of connection

• Fast, low overhead

1.Suit for reliable, local network

2.RTP(Real-Time Transport Protocol)

Conclusion: Thus we have studied packet formats captured through Wireshark for wired network.

Assignment No.	11
Title	Understand IDNSLookup.
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	

Assignment No._ 11

Title: understand IDNSLookup.

OBJECTIVES:

1.To understand DNSLookup .

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

Theory:

What is DNS?

The "Domain Name System" What Internet users use to reference anything by name on the Internet The mechanism by which Internet software translates names to attributes such as addresses A globally distributed, scalable, reliable database Comprised of three components A "name space" Servers making that name space available Resolvers (clients) which query the servers about the name space.

DNS as a Lookup Mechanism:

Users generally prefer names to numbers Computers prefer numbers to names DNS provides the mapping between the two I have "x", give me "y"

DNS as a Database:

Keys to the database are "domain names" www.foo.com, 18.in-addr.arpa, 6.4.e164.arpa Over 200,000,000 domain names stored Each domain name contains one or more attributes Known as "resource records" Each attribute individually retrievable.

Global Distribution:

Data is maintained locally, but retrievable globally No single computer has all DNS data DNS lookups can be performed by any device Remote DNS data is locally cachable to improve performance.

How does DNS work?

DNS servers answer questions from both inside and outside their own domains. When a server receives a request from outside the domain for information about a name or address inside the domain, it provides the authoritative answer. When a server receives a request from inside its own domain for information about a name or address outside that domain, it passes the request out to another server -- usually one managed by its internet service provider. If that server does not know the answer or the authoritative source for the answer, it will reach out to the DNS servers for the top-level domain -- e.g., for all

of .com or .edu. Then, it will pass the request down to the authoritative server for the specific domain -- e.g., techtarget.com or stkate.edu; the answer flows back along the same path.

How does DNS increase web performance?

To promote efficiency, servers can cache the answers they receive for a set amount of time. This allows them to respond more quickly the next time a request for the same lookup comes in. For example, if everyone in an office needs to access the same training video on a particular website on the same day, the local DNS server will ordinarily only have to resolve the name once, and then it can serve all the other requests out of its cache. The length of time the record is held -- the time to live -- is configurable; longer values decrease the load on servers, shorter values ensure the most accurate responses.

DNS message format:

The DNS protocol uses two types of DNS messages, queries and replies, and they both have the same format. Each message consists of a header and four sections: question, answer, authority, and an additional space. A header field (*flags*) controls the content of these four sections.

The header section contains the following fields: *Identification*, *Flags*, *Number of questions*, *Number of answers*, *Number of authority resource records* (RRs), and *Number of additional RRs*. The identification field can be used to match responses with queries. The flag field consists of several sub-fields. The first is a single bit which indicates if the message is a query (0) or a reply (1). The second sub-field consists of four bits; if the value is 1, the present packet is a reply; if it is 2, the present packet is a status; if the value is 0, the present packet is a request.

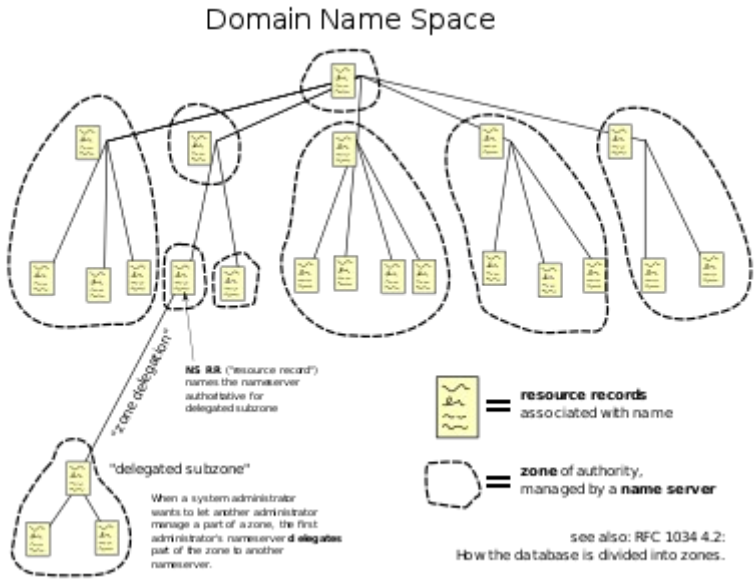
A single-bit sub-field indicates if the DNS server is authoritative for the queried hostname. Another single-bit sub-field indicates if the client wants to send a recursive query ("RD"). The next single-bit sub-field indicates if the replying DNS server supports recursion ("RA"), since not all DNS servers are configured to do this task. Another sub-field indicates if the request was truncated for some reason ("TC"), and a four-bit sub-field indicates status. The *question* section contains the domain name and type of record (A, AAAA, MX, TXT, etc.) being resolved.

The domain name is broken into discrete labels which are concatenated; each label is

prefixed by the length of that label. The *answer* section has the resource records of the queried name. A domain name may occur in multiple records if it has multiple IP addresses associated

Domain name space:

The domain name space consists of a tree data structure. Each node or leaf in the tree has a *label* and zero or more *resource records* (RR), which hold information associated with the domain name. The domain name itself consists of the label, possibly concatenated with the name of its parent node on the right, separated by a dot.[17] The tree sub-divides into *zones* beginning at the root zone. A DNS zone may consist of only one domain, or may consist of many domains and sub-domains, depending on the administrative choices of the zone manager. DNS can also be partitioned according to *class*; the separate classes can be thought of as an array of parallel namespace trees.



The hierarchical Domain Name System for class *Internet*, organized into zones, each served by a name server

Administrative responsibility over any zone may be divided by creating additional zones. Authority over the new zone is said to be *delegated* to a designated name server. The parent zone ceases to be authoritative for the new zone.

Domain name syntax:

The definitive descriptions of the rules for forming domain names appear in RFC 1035, RFC 1123, and RFC 2181. A domain name consists of one or more parts, technically called *labels*, that are conventionally concatenated, and delimited by dots, such as example.com.

The right-most label conveys the top-level domain; for example, the domain name www.example.com belongs to the top-level domain *com*.

The hierarchy of domains descends from right to left; each label to the left specifies a subdivision, or subdomain of the domain to the right. For example: the label *example* specifies a subdomain of the *com* domain, and *www* is a subdomain of example.com. This tree of subdivisions may have up to 127 levels.[*citation needed*]

A label may contain zero to 63 characters. The null label, of length zero, is reserved for the root zone. The full domain name may not exceed the length of 253 characters in its textual representation.In the internal binary representation of the DNS the maximum length requires 255 octets of storage, since it also stores the length of the name.

Although domain names may theoretically consist of any character representable in an octet, host names use a preferred format and character set. The characters allowed in their labels are a subset of the ASCII character set, consisting of characters *a* through *z*, *A* through *Z*, digits *0* through *9*, and hyphen. This rule is known as the *LDH rule* (letters, digits, hyphen). Domain names are interpreted in case-independent manner. Labels may not start or end with a hyphen.An additional rule requires that top-level domain names should not be all-numeric.

Internationalized domain names

The limited set of ASCII characters permitted in the DNS prevented the representation of names and words of many languages in their native alphabets or scripts. To make this possible, ICANN approved the Internationalizing Domain Names in Applications (IDNA) system, by which user applications, such as web browsers, map Unicode strings into the valid DNS character set using Punycode. In 2009 ICANN approved the installation of internationalized domain name country code top-level domains (*ccTLD*s). In addition, many registries of the existing top level domain names (*TLD*s) have adopted the IDNA

system.

Name servers

The Domain Name System is maintained by a distributed database system, which uses the client–server model. The nodes of this database are the name servers. Each domain has at least one authoritative DNS server that publishes information about that domain and the name servers of any domains subordinate to it. The top of the hierarchy is served by the root name servers, the servers to query when looking up (*resolving*) a TLD.

Conclusion: Thus we have studied DNS and DNSLookup.

Assignment No.	12
Title	**Study of DHCP**
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No._ 12

Title: understand Installing and configure DHCP server.

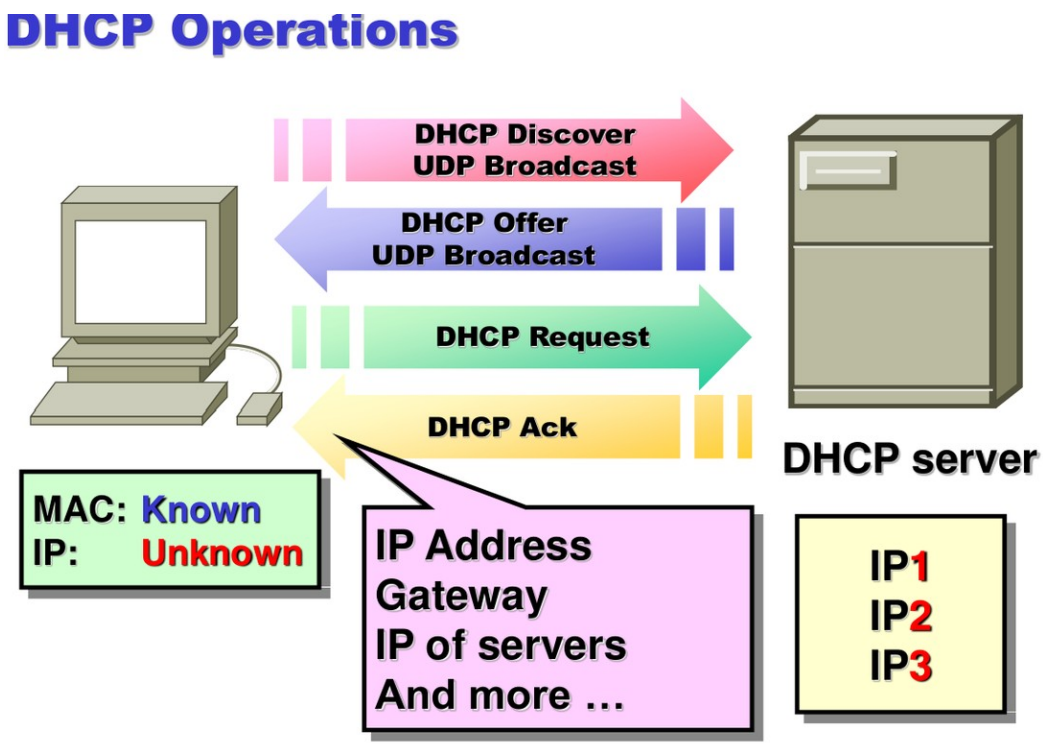
OBJECTIVES:

1.To understand Installing and configure DHCP server .

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

Theory: **Why Use DHCP?**

• Dynamic Host Configuration Protocol (DHCP) is network protocol for automatically assigning TCP/IP information to client machines. • DHCP is useful for fast delivery of client network configuration.When configuring the client system, the administrator can chooseDHCP and not have to enter an IP address, netmask, gateway, or DNS servers. The client retrieves this information from the DHCP server.• DHCP is also useful if an administrator wants to change the IP addresses of a large number of systems. Instead of reconfiguring all the systems, he can just edit one DHCP configuration file on the server for the new set of IP address. If the DNS servers for an organization changes, the changes are made on the DHCP server, not on the DHCP clients. Once the network is restarted on the clients (orthe clients are rebooted), the changes will take effect.



Configuring a DHCP Server

• You can configure a DHCP server using the configuration file/etc/dhcpd.conf.

• DHCP also uses the file /var/lib/dhcp/dhcpd.leases to store the client lease database.

• This file should not be modified by hand. DHCP lease information for each recently assigned IP address is automatically stored in the lease database. The information includes the length of the lease, to whom the IP address has been assigned, the start and end dates for the lease, and the MAC address of the network interface card that was used to retrieve the lease.

• Many RPM packages don't automatically install a /etc/dhcpd.conf file, but you can find a sample copy of dhcpd.conf in the following directory which you can always use as a guide:

• /usr/share/doc/dhcp-<version-number>/dhcpd.conf.sample

• # cp /usr/share/doc/dhcp-3.0pl1/dhcpd.conf.sample \ /etc/dhcpd.conf.

/etc/dhcpd.conf

• There are two types of statements in the configuration file:– Parameters — state how to perform a task, whether to perform a task, or what network configuration options to send to the

client.

– Declarations — describe the topology of the network, describe the clients, provide addresses for the clients, or apply a group of parameters to a group of declarations.

• Some parameters must start with the option keyword and are referred to as options.

/etc/dhcpd.conf

• The routers, subnet-mask, domain-name, domain-name- servers, and time-offset options are used for any host statements declared below it

• You must include a subnet declaration for every subnet in your network. If you do not, the DHCP server will fail to start

• Clients are assigned an IP address within the range

• To assign an IP address to a client based on the MAC address of the network interface card, use the hardware ethernet parameter within a host declaration.

/etc/dhcpd.conf

subnet 192.168.1.0 netmask 255.255.255.0 {

The range of IP addresses the server will issue to

#DHCP enabled PC clients booting up on the network

range 192.168.1.10 192.168.1.100;

range 192.168.1.201 192.168.1.220;

Set the amount of time in seconds that

a client may keep the IP address

default-lease-time 86400;

max-lease-time 86400;

Set the default gateway to be used by

the PC clients

option routers 192.168.1.1;

Don't forward DHCP requests from this

NIC interface to any other NIC interfaces

option ip-forwarding off;

/etc/dhcpd.conf

Set the broadcast address and subnet mask

to be used by the DHCP clients

option broadcast-address 192.168.1.255;

option subnet-mask 255.255.255.0;

Set the DNS server to be used by the DHCP clients

option domain-name-servers 192.168.1.100;

If you specify a WINS server for your Windows clients,

you need to include the following option in the dhcpd.conf file:

option netbios-name-servers 192.168.1.100;

}

Starting and Stopping the Server

• Before you start the DHCP server for the first time, it will fail unless there is an existing dhcpd.leases file. To create the file if it does not exist, use the command

• #touch /var/lib/dhcp/dhcpd.leases

• If you have more than one network interface attached to the system, but you only want the DHCP server to start on one of the interface, you can configure the DHCP server to start only on that device. In /etc/sysconfig/dhcpd, add the name of the interface to the list of DHCPDARGS:

• DHCPDARGS=eth0

• Use the chkconfig command to get DHCP configured to start at boot:

• # chkconfig dhcpd on

• Use the /etc/init.d/dhcpd script to start/stop/restart DHCP after booting

• # /etc/init.d/dhcpd start

/etc/init.d/dhcpd stop

/etc/init.d/dhcpd restart

Configuration Steps

• 1. To install DHCP server on ubuntu, Type following command on terminal.

– sudo apt-get install isc-dhcp-server

• 2. Now we should configure DHCP server. Configuration file is stored at location

/etc/dhcp/dhcpd.conf .

• Use gedit to edit dhcpd.conf

A slightly different configuration for an internal subnet.

subnet 172.16.5.0 netmask 255.255.255.0 {

range 172.16.5.2 172.16.5.5;

option domain-name-servers 8.8.8.8;

option routers 172.16.1.1;

option broadcast-address 172.16.5.255;

default-lease-time 600;

max-lease-time 7200; }

3 . Now restart service by using following command.

– sudo service isc-dhcp-server restart

4. Now on client computer, in network configuration setting just choose automatic configuration. Thats it client get IP address automatically.

Steps for installation of Software on Remote Machine

1. Type following command for installation of ssh in command prompt-

– >sudo apt-get install ssh

2. Proceed with installation steps on Remote machine.

3. After installation, for obtaining remote access, type following command-

– >sudo ssh hostname@ipaddress

– For Example. >sudo ssh student@172.25.28.60

4. Enter the password for host machine then enter the password for remote machine.

5. After login for installation of any package such as SBCL package type following command:-

– >sudo apt-get install package_name.

– Example: >sudo apt-get install sbcl.

Conclusion: Thus we have studied Installing and configure DHCP server.

Group B Assignments

Assignment No.	B_3
Title	To understand TCP Scoket
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No. B_3

Title: To understand TCP Scoket .

OBJECTIVES:

1.To understand TCP Scoket .

Problem Statement: Write a program using TCP sockets for wired network to implement

a. Peer to Peer Chat

b. Multiuser Chat.

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

peer mode.

Theory:

Internet and WWW have emerged as global ubiquitous media for communication and changed the way we conduct science, engineering, and commerce. They are also changing the way we learn, live, enjoy, communicate, interact, engage, etc. The modern life activities are getting completely centered around or driven by the Internet. To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet. This created a huge demand for software designers and engineers with skills in creating new Internet-enabled applications or porting existing/legacy applications to the Internet platform. The key elements for developing Internet-enabled applications are a good understanding of the issues involved in implementing distributed applications and sound knowledge of the fundamental network programming models.

Client/Server Communication

At a basic level, network-based systems consist of a server , client , and a media for communication as shown in Fig. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called TCP/IP (Transport Control Protocol/Internet Protocol) stack, shown in Fig. 13.2. The transport layer comprises two

types of protocols, TCP (Transport Control Protocol) and UDP (User Datagram Protocol). The most widely used programming

interfaces for these protocols are sockets. TCP is a connection-oriented protocol that provides a reliable fl ow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet. UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival and sequencing. Example applications that use such services include Clock server and Ping.

The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. Port is represented by a positive (16-bit) integer value. Some ports have been reserved to support common/well known services:

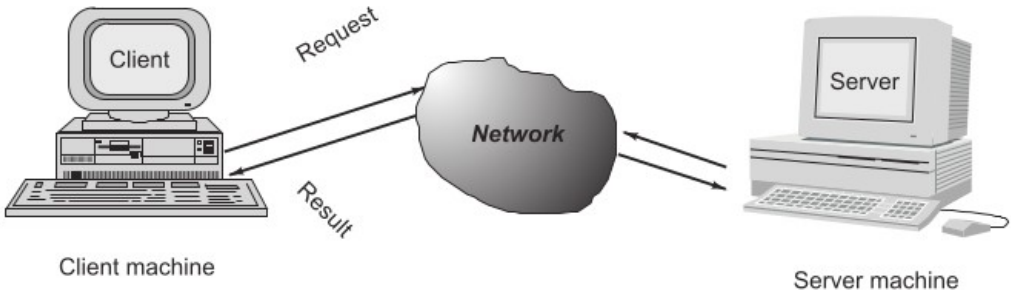


Fig. 13.1 Client – Server communication

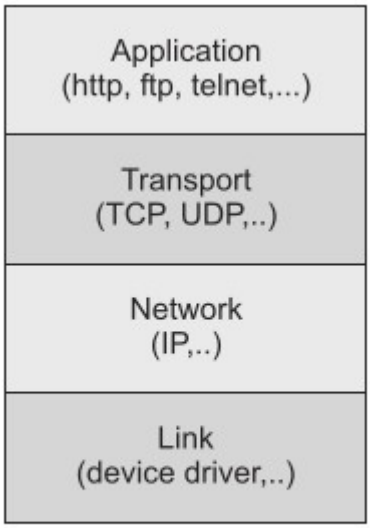


Fig. 13.2 TCP/IP software stack

Sockets and Socket-based Communication

Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing fi le I/O. In fact, socket handle is treated like fi le handle. The streams used in fi le I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. That means, a socket program written in Java language can communicate to a program written in non-Java (say C or C++) socket program. A server (program) runs on a specifi c computer and has a socket that is bound to a specifi c port. The

server listens to the socket for a client to make a connection request (see Fig. 13.4a). If everything goes well, the server accepts the connection (see Fig. 13.4b). Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

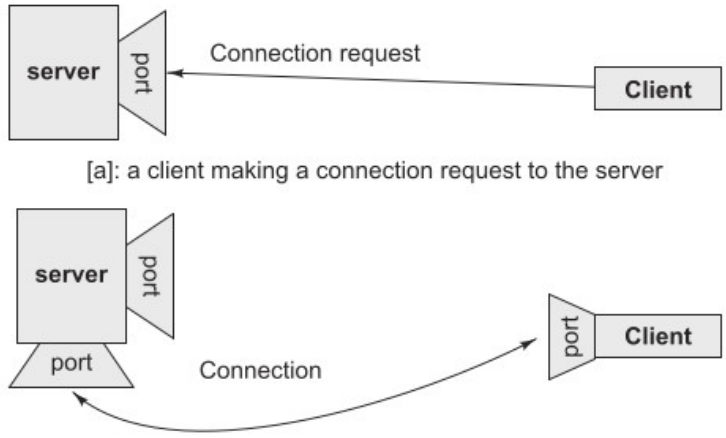


Fig. 13.4 Establishment of path for two-way communication between a client and server

TCP/IP SOCKET PROGRAMMING

The two key classes from the java.net package used in creation of server and client programs are:

∑ ServerSocket

∑ Socket

A server program creates a specifi c type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the fi le concept: developers have to open a socket, perform I/O, and close it. Figure 13.5 illustrates key steps involved in creating socket-based server and client programs.

A simple Server Program in Java The steps for creating a simple server program are:

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();

Conclusion: Thus we have studied Socket and TCP Socket.

Assignment No.	B_4
Title	To understand UDP Scoket
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No. B_4

Title: To understand UDP Scoket .

OBJECTIVES:

1.To understand UDP Scoket .

Problem Statement: Write a program using UDP sockets for wired network to implement

a. Peer to Peer Chat

b. Multiuser Chat.

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

peer mode.

Theory:

Internet and WWW have emerged as global ubiquitous media for communication and changed the way we conduct science, engineering, and commerce. They are also changing the way we learn, live, enjoy, communicate, interact, engage, etc. The modern life activities are getting completely centered around or driven by the Internet. To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet. This created a huge demand for software designers and engineers with skills in creating new Internet-enabled applications or porting existing/legacy applications to the Internet platform. The key elements for developing Internet-enabled applications are a good understanding of the issues involved in implementing distributed applications and sound knowledge of the fundamental network programming models.

Client/Server Communication

At a basic level, network-based systems consist of a server , client , and a media for communication as shown in Fig. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called TCP/IP (Transport Control

Protocol/Internet Protocol) stack, shown in Fig. 13.2. The transport layer comprises two types of protocols, TCP (Transport Control Protocol) and UDP (User Datagram Protocol). The most widely used programming

interfaces for these protocols are sockets. TCP is a connection-oriented protocol that provides a reliable fl ow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet. UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival and sequencing. Example applications that use such services include Clock server and Ping.

The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. Port is represented by a positive (16-bit) integer value. Some ports have been reserved to support common/well known services:

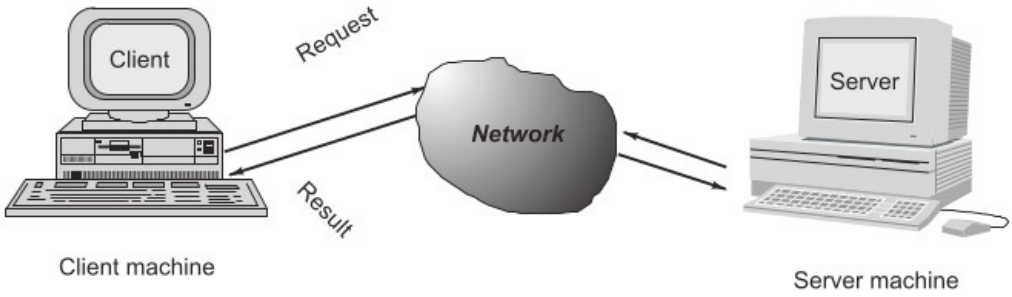


Fig. 13.1 Client – Server communication

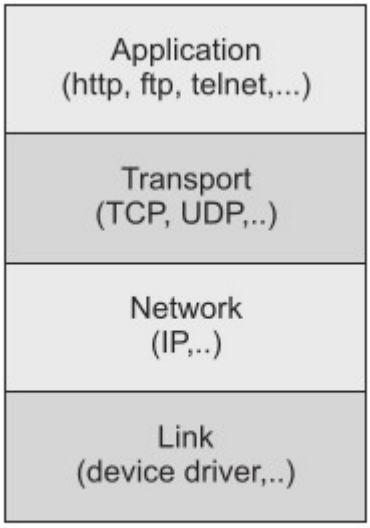


Fig. 13.2 TCP/IP software stack

Sockets and Socket-based Communication

Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing fi le I/O. In fact, socket handle is treated like fi le handle. The streams used in fi le I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. That means, a socket program written in Java language can communicate to a program written in non-Java (say C or C++) socket program. A server (program) runs on a specifi c computer and has a socket that is bound to a specifi c port. The

server listens to the socket for a client to make a connection request (see Fig. 13.4a). If everything goes well, the server accepts the connection (see Fig. 13.4b). Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

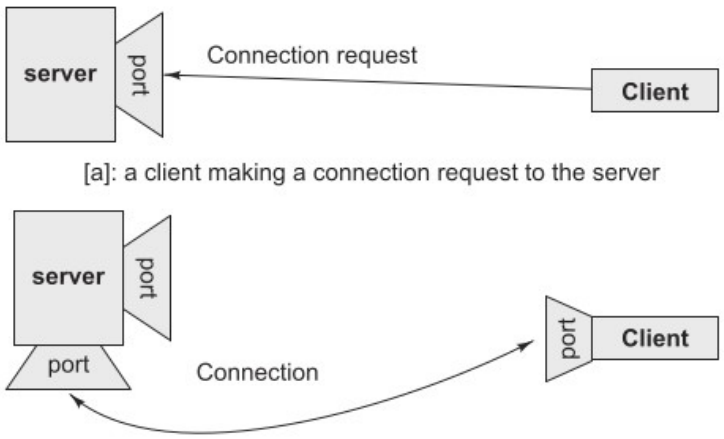


Fig. 13.4 Establishment of path for two-way communication between a client and server

UDP SOCKET PROGRAMMING

The previous two example programs used the TCP sockets. As already said, TCP guarantees the delivery of packets and preserves their order on destination. Sometimes these features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished

by the UDP protocol which conveys datagram packets. Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

The format of datagram packet is:

Java supports datagram communication through the following classes:

∑ DatagramPacket

∑ DatagramSocket

The class DatagramPacket contains several constructors that can be used for creating packet object.

One of them is:

DatagramPacket(byte[] buf, int length, InetAddress address, int port);

This constructor is used for creating a datagram packet for sending packets of length length to the specifi ed port number on the specifi ed host. The message to be transmitted is indicated in the fi rst argument.

The key methods of DatagramPacket class are:

byte[] getData()

Returns the data buffer.

int getLength()

Returns the length of the data to be sent or the length of the data received.

void setData(byte[] buf)

Sets the data buffer for this packet.

void setLength(int length)

Sets the length for this packet.

The class DatagramSocket supports various methods that can be used for transmitting or receiving data a datagram over the network. The two key methods are:

void send(DatagramPacket p)

Sends a datagram packet from this socket.

void receive(DatagramPacket p)

Receives a datagram packet from this socket.

Conclusion: Thus we have studied Socket and UDP Socket.

Assignment No.	B_6
Title	To understand NS2
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No. B_6

Title: To understand network simulator NS2 .

OBJECTIVES:

1.To understand network simulator NS2 .

Problem Statement: Use network simulator NS2 to implement:

a. Monitoring traffic for the given topology

b. Analysis of CSMA and Ethernet protocols

c. Network Routing: Shortest path routing, AODV.

d. Analysis of congestion control (TCP and UDP).

Theory:

Network Simulator:

"ns-2" redirects here. For the fictional robot from the works of Isaac Asimov, see NS-2 (literary character). For the hepatitis C virus protein, see NS2 (HCV). For the video game, see Natural Selection 2.

"ns-3" redirects here. For the hepatitis C virus protein, see NS3 (HCV).For other uses of "NS", see NS (disambiguation).

ns (from **network simulator**) is a name for a series of discrete event network simulators, specifically **ns-1**, **ns-2** and **ns-3**. All of them are discrete-event computer network simulators, primarily used in research[4] and teaching. ns-3 is free software, publicly available under the GNU GPLv2 license for research, development, and use.

The goal of the ns-3 project is to create an open simulation environment for computer networking research that will be preferred inside the research community:[*citation needed*]

- It should be aligned with the simulation needs of modern networking research.
- It should encourage community contribution, peer review, and validation of the software.

Since the process of creation of a network simulator that contains a sufficient number of high-quality validated, tested and maintained models requires a lot of work, ns-3 project spreads this workload over a large community of users and developers.

History

ns-1

The first version of ns, known as ns-1, was developed at Lawrence Berkeley National Laboratory (LBNL) in the 1995-97 timeframe by Steve McCanne, Sally Floyd, Kevin Fall, and other contributors. This was known as the LBNL Network Simulator, and derived in 1989 from an earlier simulator known as REAL by S. Keshav.

ns-2

Ns-2 began as a revision of ns-1. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. In 2000, ns-2 development was support through DARPA with SAMAN and through NSF with CONSER, both at USC/ISI, in collaboration with other researchers including ACIRI.

Ns-2 incorporates substantial contributions from third parties, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems. For documentation on recent changes, see the version 2 change log.

ns-3

In 2006, a team led by Tom Henderson, George Riley, Sally Floyd, and Sumit Roy, applied for and received funding from the U.S. National Science Foundation (NSF) to build a replacement for ns-2, called ns-3. This team collaborated with the Planete project of INRIA at Sophia Antipolis, with Mathieu Lacage as the software lead, and formed a new open source project.

In the process of developing ns-3, it was decided to completely abandon backward-compatibility with ns-2. The new simulator would be written from scratch, using the C++ programming language. Development of ns-3 began in July 2006.

The first release, ns-3.1 was made in June 2008, and afterwards the project continued making quarterly software releases, and more recently has moved to three releases per year. ns-3 made its twenty first release (ns-3.21) in September 2014.

Current status of the three versions is:

- ns-1 development stopped around 2001. It is no longer developed nor maintained.
- ns-2 development stopped around 2010. It is no longer developed, and the last maintenance release was in 2013.
- ns-3 is still developed (but not compatible for work done on ns-2)

Simulation workflow:

The general process of creating a simulation can be divided into several steps:

1. ***Topology definition*:** To ease the creation of basic facilities and define their interrelationships, ns-3 has a system of containers and helpers that facilitates this

process.

2. ***Model development*:** Models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.
3. ***Node and link configuration*:** models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.
4. ***Execution*:** Simulation facilities generate events, data requested by the user is logged.
5. ***Performance analysis*:** After the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.
6. ***Graphical Visualization*:** Raw or processed data collected in a simulation can be graphed using tools like Gnuplot, matplotlib or XGRAPH.

The Ad Hoc On-Demand Distance Vector (AODV):

The Ad Hoc On-Demand Distance Vector (AODV) routing protocol enables multi-hop routing between participating mobile nodes wishing to establish and maintain an ad-hoc network. AODV is based upon the distance vector algorithm. The difference is that AODV is reactive, as opposed to proactive protocols like DV, i.e. AODV only requests a route when needed and does not require nodes to maintain routes to destinations that are not actively used in communications. As long as the endpoints of a communication connection have valid routes to each other, AODV does not play any role.

Features of this protocol include loop freedom and that link breakages cause immediate notifications to be sent to the affected set of nodes, but only that set. Additionally, AODV has support for multicast routing and avoids the Bellman Ford "counting to infinity" problem. The use of destination sequence numbers guarantees that a route is "fresh".

The algorithm uses different messages to discover and maintain links. Whenever a node wants to try and find a route to another node, it broadcasts a Route Request (RREQ) to all its neighbors. The RREQ propagates through the network until it reaches the destination or a node with a fresh enough route to the destination.

Then the route is made available by unicasting a RREP back to the source. The algorithm uses hello messages (a special RREP) that are broadcasted periodically to the immediate neighbors. These hello messages are local advertisements for the continued presence of the node and neighbors using routes through the broadcasting node will continue to mark the routes as valid.

If hello messages stop coming from a particular node, the neighbor can assume that the node has moved away and mark that link to the node as broken and notify the affected set of nodes by sending a link failure notification(a special RREP) to that set of nodes.

AODV also has a multicast route invalidation message, but because we do not cover multicast in this report we will not discuss this any further ODV needs to keep track of the following information for each route table entry:

Route table management:

Destination IP Address: IP address for the destination node.

Destination Sequence Number: Sequence number for this destination.

Hop Count: Number of hops to the destination.

Next Hop: The neighbor, which has been designated to forward packets to the destination for this route entry.

Lifetime: The time for which the route is considered valid.

Active neighbor list: Neighbor nodes that are actively using this route entry.

Request buffer: Makes sure that a request is only processed once.

Route discovery

A node broadcasts a RREQ when it needs a route to a destination and does not have one available. This can happen if the route to the destination is unknown, or if a previously valid route expires. After broadcasting a RREQ, the node waits for a RREP. If the reply is not received within a certain time, the node may rebroadcast the RREQ or assume that there is no route to the destination.

Forwarding of RREQs is done when the node receiving a RREQ does not have a route to the destination. It then rebroadcast the RREQ. The node also creates a temporary reverse route to the Source IP Address in its routing table with next hop equal to the IP address field of the neighboring node that sent the broadcast RREQ. This is done to keep track of a route back to the original node making the request, and might be used for an eventual RREP to find its way back to the requesting node.

The route is temporary in the sense that it is valid for a much shorter time, than an actual route entry. When the RREQ reaches a node that either is the destination node or a node with a valid route to the destination, a RREP is generated and unicasted back to the requesting node. While this RREP is forwarded, a route is created to the destination and when the RREP reaches the source node, there exists a route from the source to the destination.

Route maintenance:

When a node detects that a route to a neighbor no longer is valid, it will remove the routing entry and send a link failure message, a triggered route reply message to the neighbors that are actively using the route, informing them that this route no longer is valid. For this purpose AODV uses a active neighbor list to keep track of the neighbors that are using a particular route. The nodes that receive this message will repeat this procedure. The message will eventually be received by the affected sources that can chose to either stop

sending data or requesting a new route by sending out a new RREQ.

Conclusion: Thus we have studied NS2 and AODV.

Assignment No.	B_7
Title	Configure RIP/OSPF/BGP using packet Tracer.
Subject	Computer Network Lab
Class	T.E. (Comp)
Roll No.	
Date	
Signature	

Assignment No. B_7

Title: To understand Configure RIP/OSPF/BGP using packet Tracer.

OBJECTIVES:

1.To understand Configure RIP/OSPF/BGP using packet Tracer.

Problem Statement: Configure RIP/OSPF/BGP using packet Tracer.

Theory:

Static Routing:

Typically used in hosts Enter subnet mask, router (gateway), IP address Perfect for cases with few connections, doesn't change much E.g. host with a single router connecting to the rest of the Internet IP: 128.1.1.100

Dynamic Routing:

Most routers use dynamic routing Automatically build the routing tables As we saw previously, there are two major approaches

Link State Algorithms

Distance Vector Algorithms

First some terminology

AS = Autonomous System

Contiguous set of networks under one administrative authority

Common routing protocol

E.g. University of Alaska Statewide, Washington State University

E.g. Intel Corporation

A connected network

There is at least one route between any pair of nodes

RIP (Routing Information Protocol)

Distance vector algorithm

Open Standard Protocol

Classful routing protocol

Administrative Distance is 120

Distance metric: # of hops (max = 15 hops)

Distance vectors: exchanged every 30 sec via Response Message (also called advertisement)

Each advertisement: route to up to 25 destination nets

RIP: Link Failure and Recovery

If no advertisement heard after 180 sec neighbor/link declared dead

routes via neighbor invalidated

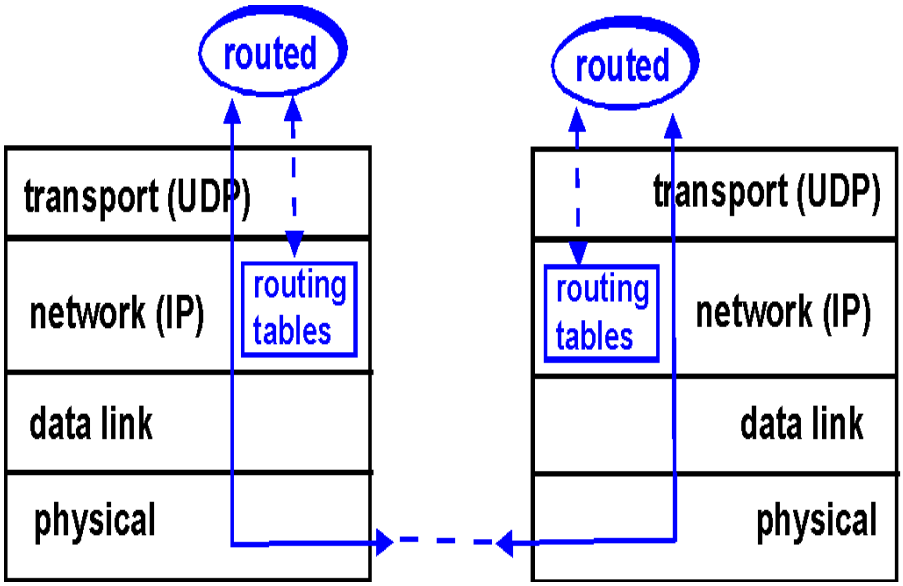
new advertisements sent to neighbors

neighbors in turn send out new advertisements (if tables changed)

link failure info quickly propagates to entire net

RIP Table processing

RIP routing tables managed by application-level process called route-d (daemon) advertisements sent in UDP packets, periodically repeated Why UDP?



RIP:

Advantages

Simplicity ; little to no configuration, just start routed upPassive version for hosts
If a host wants to just listen and update its routing table.

OSPF (Open Shortest Path First)

"Open": publicly available

RFC 2328

Uses Link State algorithm

LS packet dissemination

Topology map at each node

Route computation using Dijkstra's algorithm

OSPF advertisement carries one entry per neighbor router Metric is cost

Administrative Distance 110

Conceived as a successor to RIP

OSPF "advanced" features (not in RIP):

Security: all OSPF messages authenticated (to prevent malicious intrusion); TCP connections used Multiple same-cost paths allowed (only one path in RIP) For each link, multiple cost metrics for different Type Of Service (e.g., satellite link cost set "low" for best effort; high for real time) Integrated uni- and multicast support: Multicast OSPF (MOSPF) uses same topology data base as OSPF Hierarchical OSPF in large domains.

BGP Terminology:

Autonomous System :A collection of networks under a single administrative domain

Inter-domain Routing : Routing between the customer and the service provider

Internal Routing: Uses IGP protocol (RIP OSPF) to exchange routing information inside the AS

External Routing: Uses EGP protocol(BGP) to exchange routes between AS

IBGP: When BGP is used inside an AS

EBGP: When BGP is used between AS

Conclusion: Thus we have studied Configure RIP/OSPF/BGP using packet Tracer.