Assignment No.	**A-1**
Title	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.
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

Assignment No. A1

Title: 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.

Objectives: To establish a wired LAN for four computers.

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.

Outcomes:

Develop and demonstrate a wired LAN for four computers.

Tools Required:

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

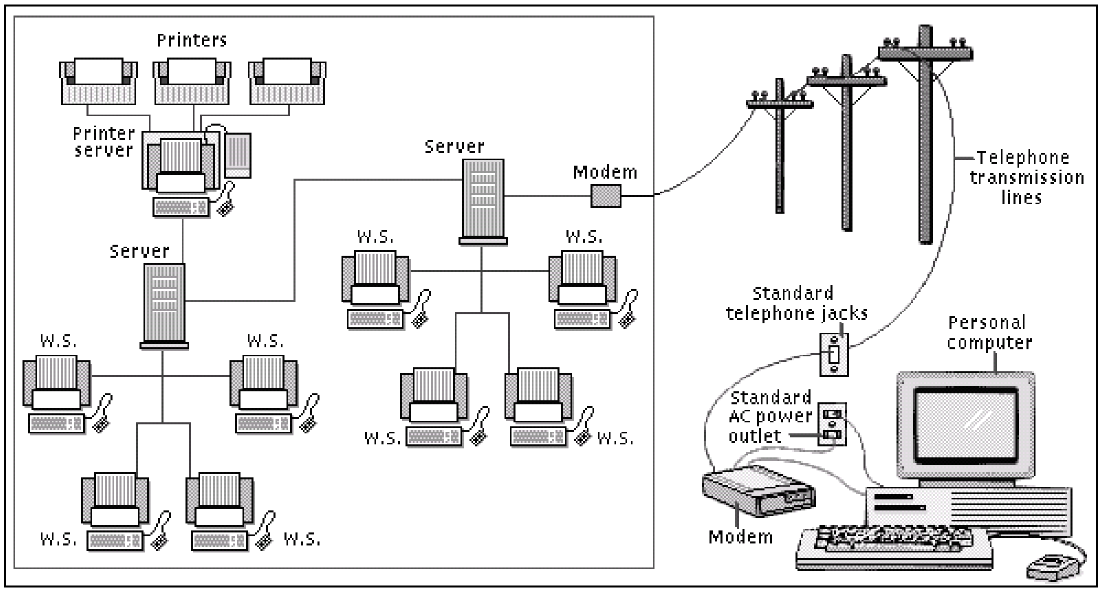
Software: Open source O.S.andwireshark

Theory:

Introduction:-

Computer Networks, the widespread sharing of information among groups of computers and their users, a central part of the information age. The popular adoption of the personal computer (PC) and the local area network (LAN) during the 1980s has led to the capacity to access information on a distant database; download an application from overseas; send a message to a friend in a different country; and share files with a colleague—all from a personal computer.

The networks that allow all this to be done so easily are sophisticated and complex entities. They rely for their effectiveness on many cooperating components. The design and deployment of the worldwide computer network can be viewed as one of the great technological wonders of recent decades.



Computer Network

Networks are connections between groups of computers and associated devices that allow users to transfer information electronically. The local area network shown on the left is representative of the setup used in many offices and companies. Individual computers, called work stations (WS), communicate to each other via cable or telephone line linking to servers. Servers are computers exactly like the WS, except that they have an administrative function and are devoted entirely to monitoring and controlling WS access to part or all of the network and to any shared resources (such as printers). The red line represents the larger network connection between servers, called the backbone; the blue line shows local connections. A modem (modulator/demodulator) allows computers to transfer information across standard telephone lines. Modems convert digital signals into analogue signals and back again, making it possible for computers to communicate, or network, across thousands of miles.

Study of Network Devices:-

NIC (**Network Interface Card)**:-

Each computer includes the File server or a Network will have a card plugged in the PCI Expunction slot or will have on-board NIC (Network Interface Card), which will provide connectivity among the workstation in the network through cables.

Type's of Card:-

1. Arc net card (2.5 mbits/sec)
2. Ethernet card (10/100 mbps)
3. Token Ring card (4-16 mbits/sec)

Hub/Switch:-

These devices are used for Re-directing traffic, i.e. in a **Star** Topology the central device is used to ECHO/Re-Direct the packets coming from one workstation/node to the Destination workstation/node.

This is done by using the devices like Hub/Switch, during the present situation **Hub's are absolute due to their disadvantages of Echoing a packet from one node to all, which leads to increasing N/W traffic and packet Collision.**

Type of Hub:-

1. **Passive Hub:-**
   It is a device which do not require any type of power supply and does not boost incoming signal, it just echo the incoming signal to all nodes.

2. **Active Hub :-**

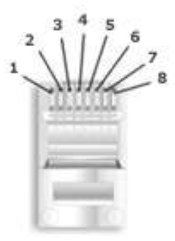
It is a device which requires power supply and boosts the incoming signal and echoes the signal to all nodes.

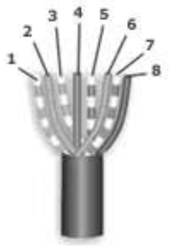
Hub where absolute due to use of an intelligent device called **Switch** which reads the destination adders and sends the incoming packet to it.

paring Rules and Color Code:-

The CAT 5 Cable consist of 8 wires which comes pares of White/Blue, Blue, White/Orange, Orange, White/Green, Green, White/Brown, Brown and they are coded for **Straight** and **Cross** combinations respectively.

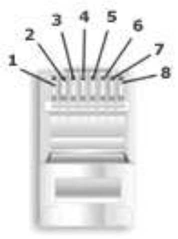
Straight:-

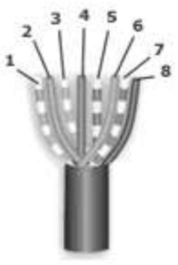




Pair #	Wire	Pin #
1-White/Blue	White/Blue	5
	Blue	4
2-Wht./Orange	White/Orange	1
	Orange	2
3-White/Green	White/Green	3
	Green	6
4-White/Brown	White/Brown	7
	Brown	8

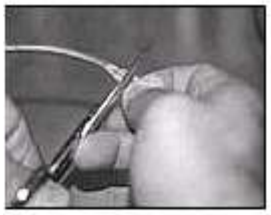
Cross:-





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

	Brown	8



Connections among devices:-

Node to Node	-	Straight – Cross,
Switch to Node	-	Straight – Straight,
Switch to Switch	-	Straight – Cross.

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

1. Skin off the cable jacket approximately 1" or slightly more.
2. Un-twist each pair, and straighten each wire between the fingers.
3. Place the wires in the order of one of the two diagrams shown above .Bring all of the wires together, until they touch.
4. At this point, recheck the wiring sequence with the diagram.

5. Optional: Make a mark on the wires at 1/2" from the end of the cable jacket.
6. Hold the grouped (and sorted) wires together tightly, between the thumb, and the forefinger.

7. Cut all of the wires at a perfect 90 degree angle from the cable at 1/2" from the end of the cable jacket. This is a very critical step. If the wires are not cut straight, they may not all make contact. We suggest using a pair of scissors for this purpose.
8. Conductors should be at a straight 90 degree angle, and be 1/2" long, prior to insertion into the connector.
9. Insert the wires into the connector (pins facing up).
10. Push moderately hard to assure that all of the wires have reached the end of the connector. Be sure that the cable jacket goes into the back of the connector by about 3/16".
11. Place the connector into a crimp tool, and squeeze hard so that the handle reaches its full swing.
12. Repeat the process on the other end. For a straight through cable, use the same wiring.
13. Use a cable tester to test for proper continuity.

Cable Testing Tool:-

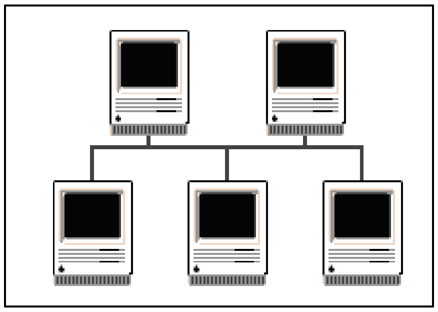
It is a tool used for testing weather there is no cut in between two terminals and to identify the type of pair crimp with.

Study of Topologies:-

What is a Topology?

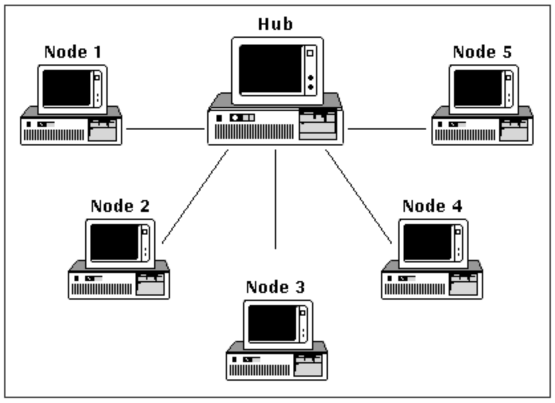
The physical topology of a network refers to the configuration of cables, computers, and other peripherals. Physical topology should not be confused with logical topology which is the method used to pass information between workstations.

1. Bus Topologies:-



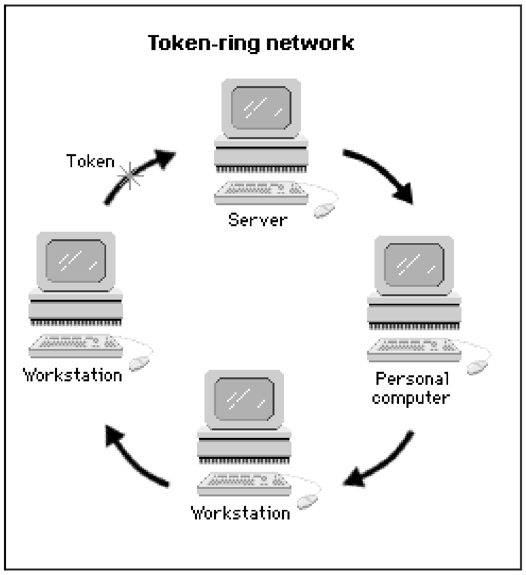
In a bus network configuration, each node is connected to one main communications line. With this arrangement, even if one of the nodes goes down, the rest of the network can continue to function normally.

2. Star Topologies:-



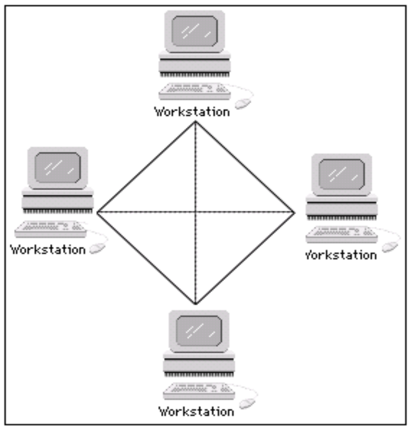
A star network consists of several nodes connected to a central hub/switch in a star-shaped configuration. Messages from individual nodes pass directly to the hub/switch, which determines any further routing.

3. Ring Topology:-



Token Ring Network, in computer science, a LAN formed in a ring (closed loop) topology that uses token passing as a means of regulating traffic. On a token ring network, a token governing the right to transmit is passed from one station to the next in a physical circle. If a station has information to transmit, it "seizes" the token, marks it as being in use, and inserts the information. The "busy" token, plus message, is then passed around the circle, copied when it arrives at its destination, and eventually returned to the sender. The sender removes the attached message and then passes the freed token to the next station in line. Token ring networks are defined in the IEEE 802.5 standards.

4. Mesh Topology:-

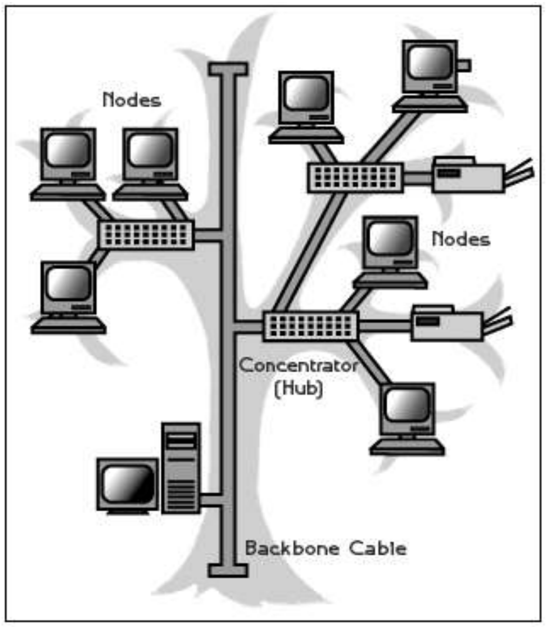


The type of network topology in which each of the nodes of the network is connected to each of the other nodes in the network with a point-to-point link – this makes it possible for data to be simultaneously transmitted from any single node to all of the other nodes.

Note: The physical fully connected mesh topology is generally too costly and complex for practical networks, although the topology is used when there are only a small number of nodes to be interconnected

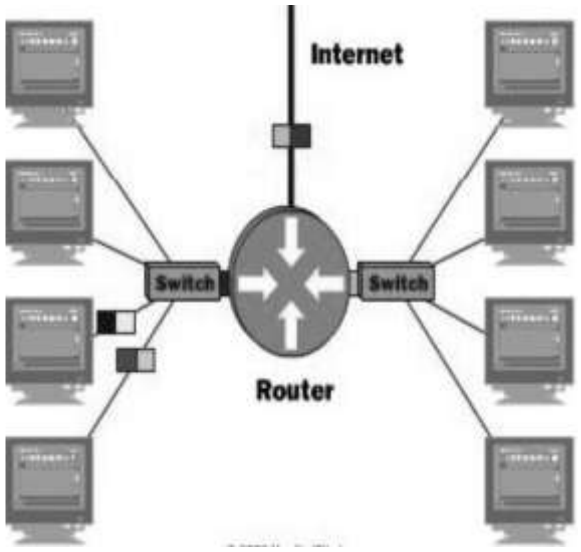
5. Hybrid/Tree Topology:-

A tree topology combines characteristics of linear bus and star topologies. It consists of groups of star-configured workstations connected to a linear bus backbone cable.



These topologies can also be mixed. For example, a bus-star network consists of a high-bandwidth bus, called the ***backbone,*** which connects a collection of slower-bandwidth star segments.

How Routers Work



Routers are the traffic cops of intranets. They make sure that all data gets sent to where it's supposed to go and that it gets sent via the most efficient route. Routers are also useful tools to make the most efficient use of the intranet. Routers are used to segment traffic and provide redundancy of routes. Routers use encapsulation to permit different protocols to be sent across otherwise incompatible networks.

Just as routers direct traffic on the Internet, sending information to its proper destination, routers on an intranet perform the same function. Routers-equipment that is a combination of hardware and software-can send the data to a computer on the same subnetwork inside the intranet, to another network on the intranet, or outside to the Internet. They do this by examining header information in IP packets, and then sending the data on its way. Typically, a router will send the packet to the next router closest to the final destination, which in turn sends it to an even closer router, and so on, until the data reaches its intended recipient.

A router has input ports for receiving IP packets, and output ports for sending those packets toward their destination. When a packet comes to the input port, the router examines the packet header, and checks the destination in it against a routing table-a database that tells the router how to send packets to various destinations.

Based on the information in the routing table, the packet is sent to a particular output port, which sends the packet to the next closest router to the packet's destination.

If packets come to the input port more quickly than the router can process them, they are sent to a holding area called an input queue. The router then processes packets from the queue in the order they were received. If the number of packets received exceeds the capacity of the queue (called the length of the queue), packets may be lost.

In a simple intranet that is a single, completely self-contained network, and in which there are no connections to any other network or the intranet, only minimal routing need be done, and so the routing table in the router is exceedingly simple with very few entries, and is constructed automatically by a program called *ifconfig*

Conclusion: Hence ,we have demonstrate a wired LAN for four computers.

Assignment No.	**A-02**
Title	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
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN Lab

ASSIGNMENT-A2

Title: 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

Objectives : To implement error detection and correction techniques
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.

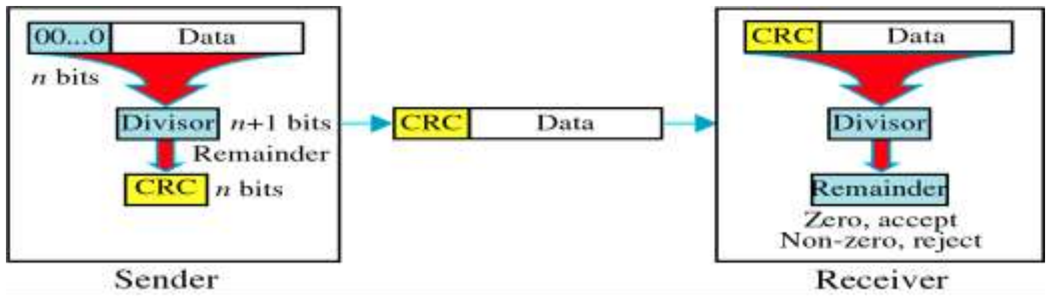
Outcome : Demonstrate Hamming Codes or CRC with example.

Software Requirements : Jdk and Wireshark

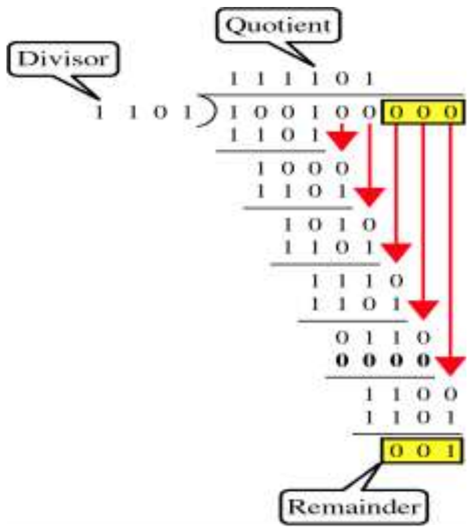
Hardware Requirements : Open source linux operating system.

THEORY:
Cyclic Redundancy Check: CRC

- Given a k-bit frame or message, the transmitter generates an n-bit sequence, known as a *frame check sequence (FCS)*, so that the resulting frame, consisting of (k+n) bits, is exactly divisible by some predetermined number.
- The receiver then divides the incoming frame by the same number and, if there is no remainder, assumes that there was no error.



Example:



Hamming code

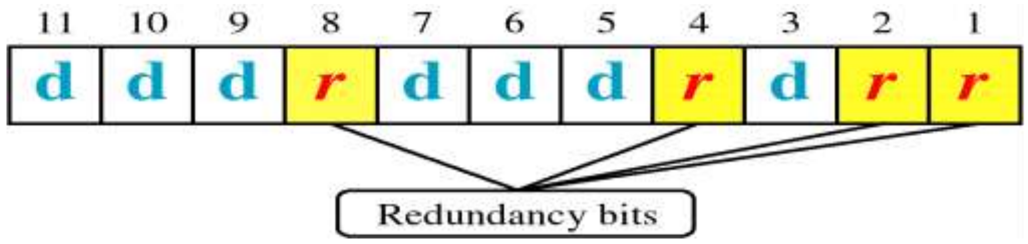
- Hamming codes are a family of linear error-correcting codes that generalize the Hamming(7,4)-code
- Invented by Richard Hamming in 1950

Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors.

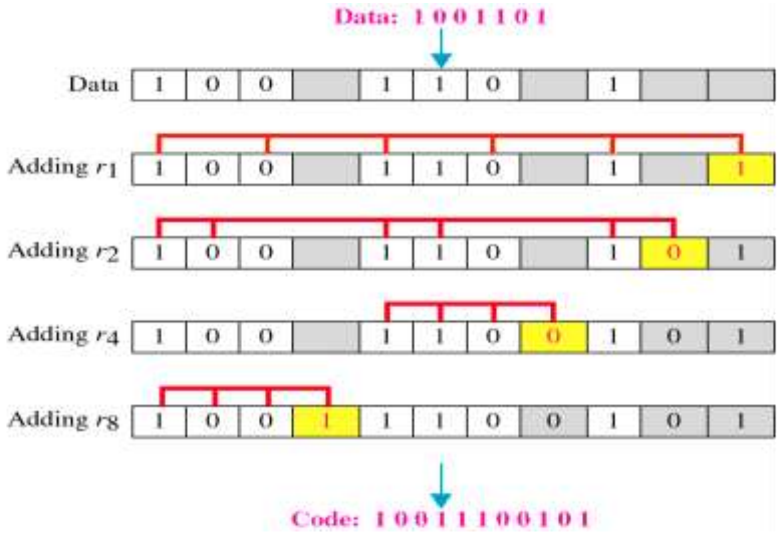
General algorithm

- The following general algorithm generates a single-error correcting (SEC) code for any number of bits.
- Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
- Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
- All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)
- All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
- Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.
- Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

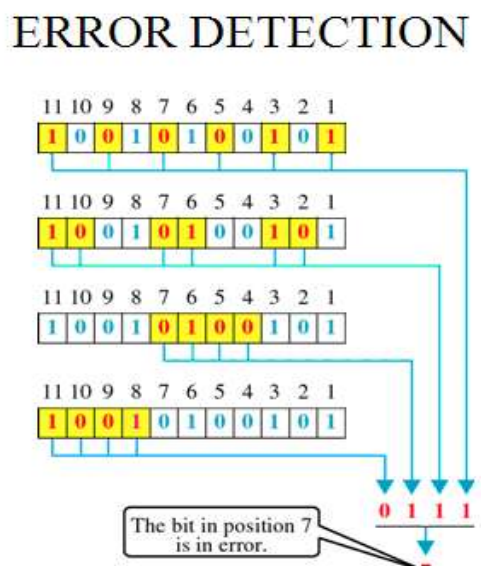
- Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
- Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.
- Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.
- In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.



Example : Error detection


Data: 1 0 0 1 1 0 1
Data 1 0 0 1 1 0 1
Adding r1 1 0 0 1 1 0 1 1
Adding r2 1 0 0 1 1 0 1 0 1
Adding r4 1 0 0 1 1 0 0 1 0 1
Adding r8 1 0 0 1 1 1 0 0 1 0 1
Code: 1 0 0 1 1 1 0 0 1 0 1

Error correction



Conclusion: Hence we have implemented CRC and Hamming code.

Assignment No.	**A-3**
Title	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.
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

Assignment No. A3

Title: 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.

Objectives: To demonstrate Go back N and Selective Repeat Modes of Sliding Window Protocol in peer to peer mode .

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.

Outcomes:

Demonstrate Go back N and Selective Repeat Modes and also captured packets usingWireshark Packet Analyzer Tool for peer to peer mode.

Tools Required:

Hardwar: PC-2

Software: jdk compiler and wireshark.

Theory:

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.

Flow Control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data should not be allowed to overwhelm the receiver. Receiver should also be able to inform the transmitter before its limits (this limit may be amount of memory used to store the incoming data or the processing power at the receiver end) are reached and the sender must send fewer frames. Hence, Flow control refers to the set of procedures used to restrict the amount of data the transmitter can send before waiting for acknowledgment.

There are two methods developed for flow control namely Stop-and-wait and Sliding-window Sliding window algorithms, used by TCP, permit multiple data packets to be in simultaneous transit, making more efficient use of network bandwidth.

Sliding Window Protocol:

With the use of multiple frames for a single message, the stop-and-wait protocol does not perform well. Only one frame at a time can be in transit. Efficiency can be greatly improved by allowing multiple frames to be in transit at the same time. Efficiency can also be improved by making use of the full-duplex line. To keep track of the frames, sender station sends sequentially numbered frames. Since the sequence number to be used occupies a field in the frame, it should be of limited size. If the header of the frame allows k bits, the sequence numbers range from 0 to 2k – 1. Sender maintains a list of sequence numbers that it is allowed to send (sender window).

The size of the sender’s window is at most 2k – 1. The sender is provided with a buffer equal to the window size. Receiver also maintains a window of size 2k – 1. The receiver acknowledges a frame by sending an ACK frame that includes the sequence number of the next frame expected. This also explicitly announces that it is prepared to receive the next N frames, beginning with the number specified. This scheme can be used to acknowledge multiple frames. It could receive frames 2, 3, 4 but withhold ACK until frame 4 has arrived. By returning an ACK with sequence number 5, it acknowledges frames 2, 3, 4 in one go. The receiver needs a buffer of size 1.

Sliding window algorithm is a method of flow control for network data transfers. TCP, the Internet's stream transfer protocol, uses a sliding window algorithm.

A sliding window algorithm places a buffer between the application program and the network data flow. For TCP, the buffer is typically in the operating system kernel, but this is more of an implementation detail than a hard-and-fast requirement.

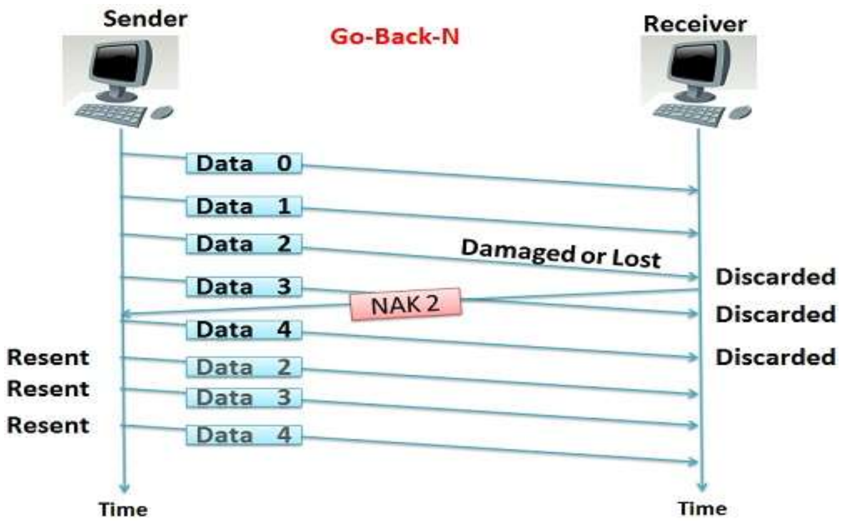
Data received from the network is stored in the buffer, from where the application can read at its own pace. As the application reads data, buffer space is freed up to accept more input from the network. The window is the amount of data that can be "read ahead" - the size of the buffer, less the amount of valid data stored in it. Window announcements are used to inform the remote host of the current window size.

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*.

Go-Back-N Protocol and “Selective Repeat Protocol” are the sliding window protocols. The sliding window protocol is primarily an error control protocol, i.e. it is a method of error detection and error correction. The basic difference between go-back-n protocol and selective repeat protocol is that the “go-back-n protocol” retransmits all the frames that lie after the frame which is damaged or lost. The “selective repeat protocol” retransmits only that frame which is damaged or lost.

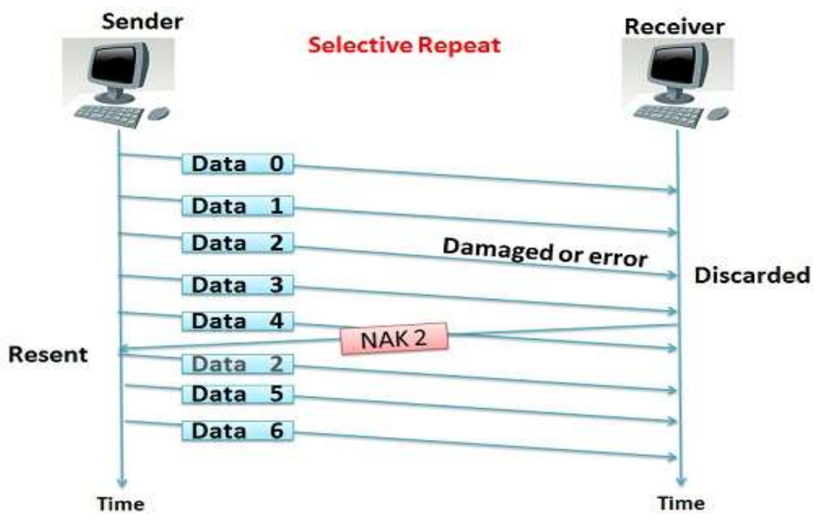
Go back N ARQ

In the Go-Back-N Protocol, the sequence numbers are modulo 1!",
where*m*is the size of the sequence number field in bits.



Selective Repeat ARQ

Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend *N* frames when just one frame is damaged;only the damaged frame is resent. This mechanism is called Selective RepeatARQ.



Key Differences Between Go-Back-N and Selective Repeat

1. Go-Back-N protocol is design to retransmit all the frames that are arrived after the damaged or a lost frame. On the other hand, Selective Repeat protocol retransmits only that frame that is damaged or lost.
2. If the error rate is high i.e. more frames are being damaged and then retransmitting all the frames that arrived after a damaged frame waste the lots of bandwidth. On the other hand, selective repeat protocol re-transmits only damaged frame hence, minimum bandwidth is wasted.
3. All the frames after the damaged frame are discarded and the retransmitted frames arrive in a sequence from a damaged frame onwards, so, there is less headache of sorting the frames hence it is less complex. On the other hand only damaged or suspected frame is retransmitted so, extra logic has to be applied for sorting hence, it is more complicated.
4. Go-Back-N has a window size of N-1 and selective repeat have a window size <=(N+1)/2.
5. Neither sender nor receiver need the sorting algorithm in Go-Back-N whereas, receiver must be able to sort the as it has to maintain the sequence.
6. In Go-Back-N receiver discards all the frames after the damaged frame hence, it don't need to store any frames. Selective repeat protocol does not discard the frames arrived after the damaged frame instead it stores those frames till the damaged frame arrives successfully and is sorted in a proper sequence.
7. In selective repeat NAK frame refers to the damaged frame number and in Go-Back-N, NAK frame refers to the next frame expected.
8. Generally the Go-Back-N is more is use due to its less complex nature instead of Selective Repeat protocol.

Conclusion: Hence we have implemented of sliding window protocol(Go back N and Selective Repeat).

Assignment No.	**A-4**
Title	Write a program to demonstrate subletting and find the subnet masks. .
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

ASSIGNMENT-4

Tittle: To demonstrate subnetting and find subnet mask.

Objectives : To understand subnetting concepts and also find subnet mask of network.

Problem Statement: Write a program to demonstrate subnetting and find subnet mask.

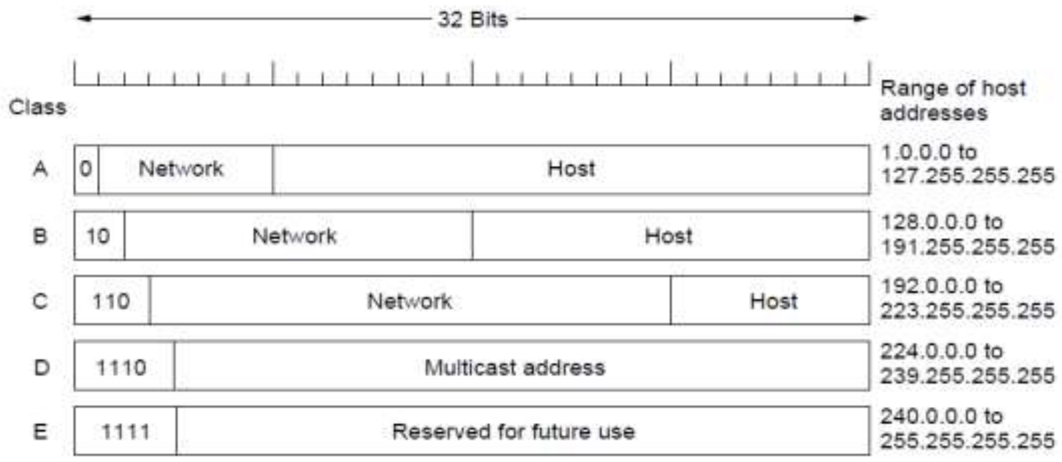
Outcome : Demonstrate subnetting concepts with examples.

Software Requirements : Jdk and python

Hardware Requirements : Open source linux operating system.
THEORY:

What is IP address?

An Internet Protocol address (IP address) is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the Internet Protocol for communication. An IP address serves two principal functions: host or network interface identification and location addressing. IP address is a 32 bit number. It is universally unique


32 Bits
Class
Range of host addresses
A
0
Network
Host
1.0.0.0 to 127.255.255.255
B
10
Network
Host
128.0.0.0 to 191.255.255.255
C
110
Network
Host
192.0.0.0 to 223.255.255.255
D
1110
Multicast address
224.0.0.0 to 239.255.255.255
E
1111
Reserved for future use
240.0.0.0 to 255.255.255.255

What is subnet?

A sub network, or subnet, is a logical, visible subdivision of an IP network. The practice of dividing a network into two or more networks is called sub netting. Computers that belong to a subenet are addressed with a common identical, most-significant bit-group in their IP . This results in the logical division of an IP address into two fields, a network or routing prefix and the rest field or host identifier. The rest field is an identifier for a specific host or network interface.

For the purpose of network management, an IP address is divided into two logical parts, the network prefix and the host identifier or rest field. All hosts on a sub network have the same network prefix. This routing prefix occupies the most-significant bits of the address. The number of bits allocated within a network to the internal routing prefix may vary between subnets, depending on the network architecture. While in IPv6 the prefix must consist of a set of contiguous 1-bits, in IPv4 this is not enforced, though there is no advantage to using non-contiguous 1-bits. The host part is a unique local identification and is either a host number on the local network or an interface identifier.

What is subnet masking?

An IP address has two components, the network address and the host address. A subnet mask separates the IP address into the network and host addresses (<network><host>). Subnetting further divides the host part of an IP address into a subnet and host address (<network><subnet><host>) if additional sub network is needed. It is called a subnet mask because it is used to identify network address of an IP address by performing a bitwise AND operation on the net mask.

A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address. Subnet Mask is made by setting network bits to all "1"s and setting host bits to all "0"s. Within a given network, two host addresses are reserved for special purpose, and cannot be assigned to hosts. The "0" address is assigned a network address and "255" is assigned to a broadcast address, and they cannot be assigned to hosts.

A mask used to determine what subnet an IP address belongs to. An IP address has two components, the network address and the host address.

For example

consider the IP address 150.215.017.009. Assuming this is part of a Class B network, the first two numbers (150.215) represent the Class B network address, and the second two numbers (017.009) identify a particular host on this network.

Subnetting an IP network is to separate a big network into smaller multiple networks for reorganization and security purposes. All nodes (hosts) in a sub network see all packets transmitted by any node in a network. Performance of a network is adversely affected under heavy traffic load due to collisions and retransmissions.

Applying a subnet mask to an IP address separates network address from host address. The network bits are represented by the 1's in the mask, and the host bits are represented by 0's. Performing a bitwise logical AND operation on the IP address with the subnet mask produces the network address.

Conclusion:

Thus we have implemented subnetting program .

Assignment No.	**A-5**
Title	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)
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

ASSIGNMENT-A5

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:

TCP:

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model.

The client server model

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. A socket is one end of an interprocess communication channel. The two processes each establish their own socket.

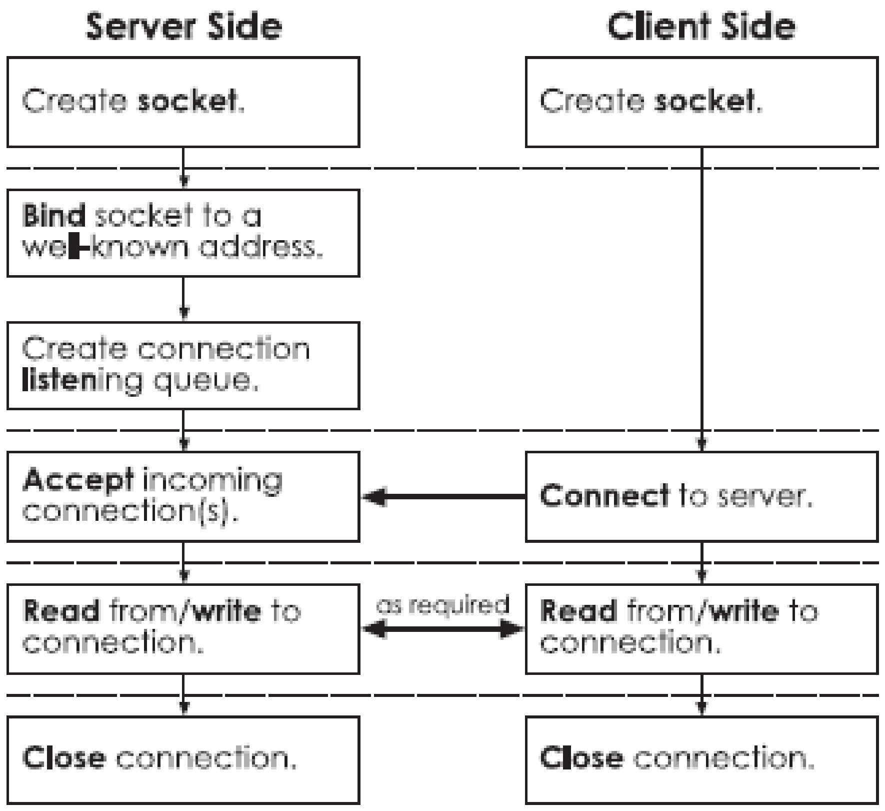
The steps involved in establishing a socket on the client side are as follows:

1. Create a socket with the socket( ) system call
2. Connect the socket to the address of the server using the connect( ) system call
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read ( ) and write ( ) system calls.

The steps involved in establishing a socket on the server side are as follows:

1. Create a socket with the socket ( ) system call
2. Bind the socket to an address using the bind ( ) system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the listen ( ) system call

4. Accept a connection with the accept ( ) system call. This call typically blocks until a client connects with the server.
5. Send and receive data


Server Side
Client Side
Create socket.
Create socket.
Bind socket to a well-known address.
Create connection listening queue.
Accept incoming connection(s).
Connect to server.
Read from/write to connection.
as required
Read from/write to connection.
Close connection.
Close connection.

Sockets are one of the most important IPC mechanisms on UNIX. Originally introduced in 4.2BSD asageneralisation of pipes, they later became the basis for the UNIX networking subsystem. Sockets arethe only IPC mechanism that allows communication between processes running on different machines.Essentially, it is an end-point of communication which may be bound to a name.But enough with the bland introduction. A socket is just a way to allow processes to talk to oneanother.

TCP and UDP
As you should remember from networking, on top of IP, there are two major transport protocols on topof which all other protocols are built: TCPand UDP These act as transportationmechanisms for other,higher-level, protocols.TCP is a reliable, connection-oriented protocol that transmits data as a stream of bytes. UDP, on theother hand, is an unreliable, connectionless protocol that sends data in chunks calleddatagrams
.

Creating a socket
Both the client and server need to create a socket before they can do anything. The client uses its socketto connect to a server whilst the server uses its socket for listening for new connections. Socket creation isdone using thesocket()call

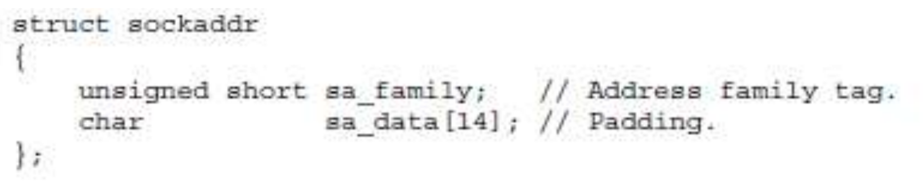
socket()
Creates a new socket. Returns a file descriptor representing the socket end-point, or-1
, if an error occurs.

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

af :is the address family to use with the socket. This could be either
AF_UNIXfor the UNIX address family(for local IPC), or
AF_INETfor the Internet address family (for network communication). TCP
requires that you useAF_INET,
type: is the socket type. This can beSOCK_STREAMfor connection-oriented, stream-based protocols likeTCP,SOCK_DGRAMfor datagram-based, connectionless protocols like UDP, orSOCK_RAWwhereyou want to use your own transportation protocol
protocol :is the transport protocol to use. It's best to pass0, which lets the system decide.

Binding the socket to a well-known address
For clients to connect to a server, they need to know its address, but sockets are created without an addressof their own and so must be assigned to one that the client will know about. This is done bybindingasocket to awell-known addresson the system. For instance, web-servers are usually bound to port 80 asthis is the well-known port for the HTTP protocol

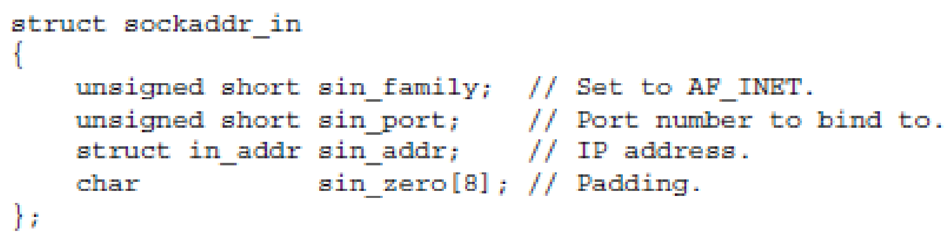
structsockaddr
This is a generic socket address structure provided to allow flexibility in passing socket addresses to functions.



```
struct sockaddr
{
    unsigned short sa_family;   // Address family tag.
    char           sa_data[14]; // Padding.
};
```

structsockaddr_in

structsockaddr_inis a specialised version ofstructsockaddrespecially for the
AF_INETaddress family.



```
struct sockaddr_in
{
    unsigned short sin_family;  // Set to AF_INET.
    unsigned short sin_port;    // Port number to bind to.
    struct in_addr sin_addr;    // IP address.
    char           sin_zero[8]; // Padding.
};
```

structin_addr

structin_addrrepresents an IP address. Why this structure exists and wasn't just incorporated directly intostructsockaddr_inis anybody's guess.Setting its one field,
s_addr, toINADDR_ANYwill leave it up to the server to choose an appropriate

host IP address, and this is usually the best thing to do.

```
struct in_addr
{
    unsigned long s_addr;   // IP address.
};
```

bind()

Binds a socket to a well-known address. This will return0if successful and-1
if not.

int bind(intfd, structsockaddr* addr, intlen);

fd: is the file descriptor of the socket to bind.
addr: points to the address to bind the socket to.
len: is the length of the address in bytes.

intBindSocket(intfd, unsigned short port)

```
{
    struct sockaddr_in addr;

    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;  // Let the host decide.
    addr.sin_port        = htons(port); // Port to bind to.

    return bind(fd, (struct sockaddr*) &addr, sizeof(addr));
}
```

listen()
Makes the socket listen for incoming connections, and sets up a connection queue for the socket.

int listen(intfd, intlen);

fd: is the file descriptor of the socket to put in passive mode.
Len: is the length of the connection queue **5**is the usual value used.

gethostbyname()

Allows you to discover a host's details (including its address) by specifying its name. This
returns astructure specifying the host's details, orNULLif it fails.

structhostent* gethostbyname(char* name);

connect()
Connects a socket to a given server, putting the socket inactive mode. Returns0
if successful, else-1

int connect(intfd, structsockaddr* addr, intlen);

fd: is the file descriptor of the socket to connect.
Addr: points to the address of the server to connect to.
len: is the length in bytes of the address

Accepting incoming connections
After the connection queue has been created, the server can accept incoming connections from clientswishing to talk to it. To accept one, it must call

accept()

Accepts a single incoming connection.Returns a file descriptor corresponding to the new connection, or-1if an error occurs.

int accept(intfd, structsockaddr* addr, int* len);

fd: is the file descriptor of the socket listening for incoming connections.
addr: is an address structure to hold the address of the client making the connection. Pass NULLhereifyou don't care about getting this information.
len: is pointer to a variable holding the length in bytes of the address structure passed in addr. On return-ing, this will hold the actual length of the client address. If you passed in NULLtoaddr, passNULLin here too.

Reading and Writing data
After the client has connected to the server and the server has accepted the connection, they can start
sending data back and forth. This is done with good old
read()
and
write()
. There are, however,
some caveats attached to reading data, however. .

write()
Writes data to a file descriptor.

int write(intfd, void* buf, unsigned int n);

fd: is the file descriptor to write to.
buf: is a buffer containing data to write.
n: is the number of bytes from the buffer to write.

read()
Reads data from a file descriptor. It returns the number of bytes actually read, or-1
if an error occurs.

int read(intfd, void* buf, unsigned int n);
fd:is the file descriptor to read from.
buf:is a buffer to write the data read to.
n:is the size in bytes of the buffer.

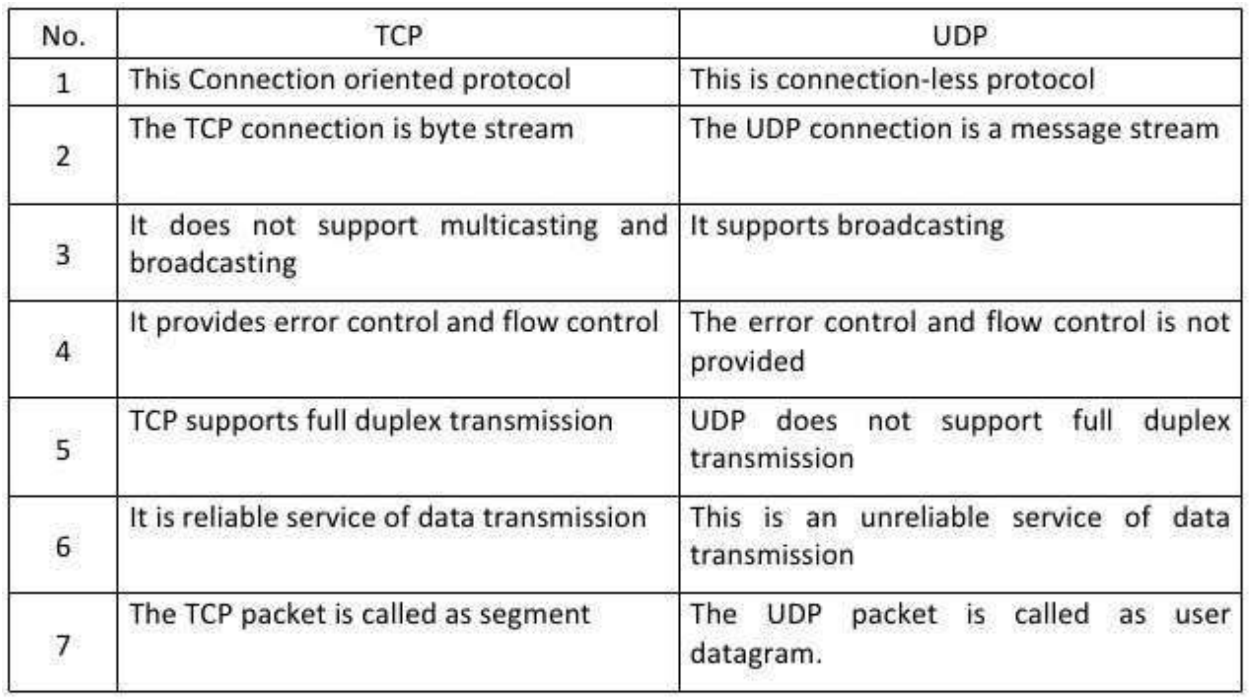
Closing the socket
Once the client or server is finished with a socket, it should call**close()**to deallocate it.

close()

Closes a file descriptor.

int close(intfd);

fd: is the file descriptor to close.



No.	TCP	UDP
1	This Connection oriented protocol	This is connection-less protocol
2	The TCP connection is byte stream	The UDP connection is a message stream
3	It does not support multicasting and broadcasting	It supports broadcasting
4	It provides error control and flow control	The error control and flow control is not provided
5	TCP supports full duplex transmission	UDP does not support full duplex transmission
6	It is reliable service of data transmission	This is an unreliable service of data transmission
7	The TCP packet is called as segment	The UDP packet is called as user datagram.

CONCLUSION:

Thus we have successfully implemented the socket programming for TCP using C.

Assignment No.	**A-6**
Title	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.
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

ASSIGNMENT-A6

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:

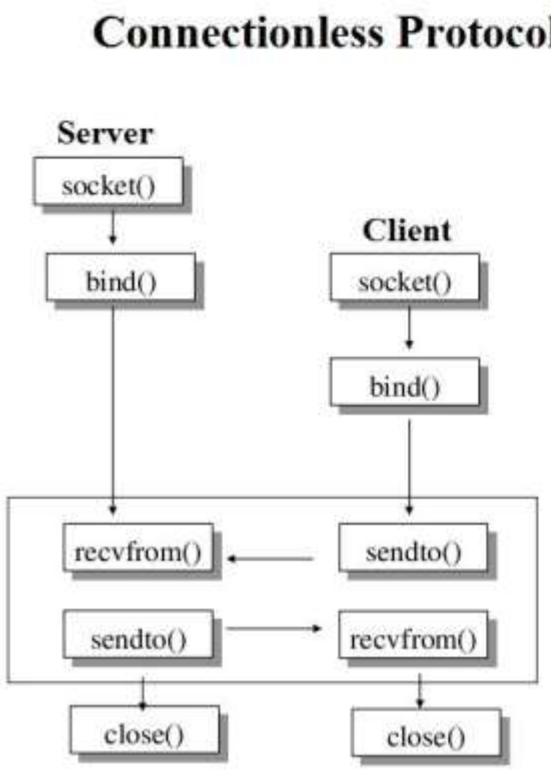
UDP:

UDP (User Datagram Protocol) is a communication protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end. Specifically, UDP doesn't provide sequencing of the packets that the data arrives in. This means that the application program that uses UDP must be able to make sure that the entire message has arrived and is in the right order. Network applications that want to save processing time because they have very small data units to exchange (and therefore very little message reassembling to do) may prefer UDP to TCP. The Trivial File Transfer Protocol (TFTP) uses UDP instead of TCP.

With a UDP socket a connection is NOT made, instead the sender just issues a message to its destination and hopes it gets there! The message uses a datagram of fixed length, often termed a record. Since there is no connection between client and server the client can send a datagram to one server and then immediately send a datagram to another server using the same socket UDP is a connectionless protocol.

Trivial File Transfer Protocol (TFTP) is a simple, lock-step, file transfer protocol which allows a client to get from or put a file onto a remote host.

TFTP is a simple protocol for transferring files, implemented on top of the UDP/IP protocols using IANA registered port number 69. TFTP was designed to be small and easy to implement, and therefore it lacks most of the advanced features offered by more robust file transfer protocols. TFTP only reads and writes files from or to a remote server. It cannot list, delete, or rename files or directories and it has no provisions for user authentication. Today TFTP is generally only used on local area networks (LAN).



CONCLUSION:

Thus we have successfully implemented the socket programming for UDP using C.

Assignment No.	**A-7**
Title	Write a program to analyze following packet formats captured through Wireshark for wired network. 1. Ethernet 2. IP 3.TCP 4. UDP .
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN La b

ASSIGNMENT-7

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

Objectives : To demonstrate data flow at various layer

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

Outcome: Student will able to demonstrate data flow from top-to-down and down to up for various protocol stacks at various layers and propose protocol model / framework for future network requirements
Tools Required: gcc complier and wireshark tool
THEORY:

Packet sniffer \ Packet analyzer:

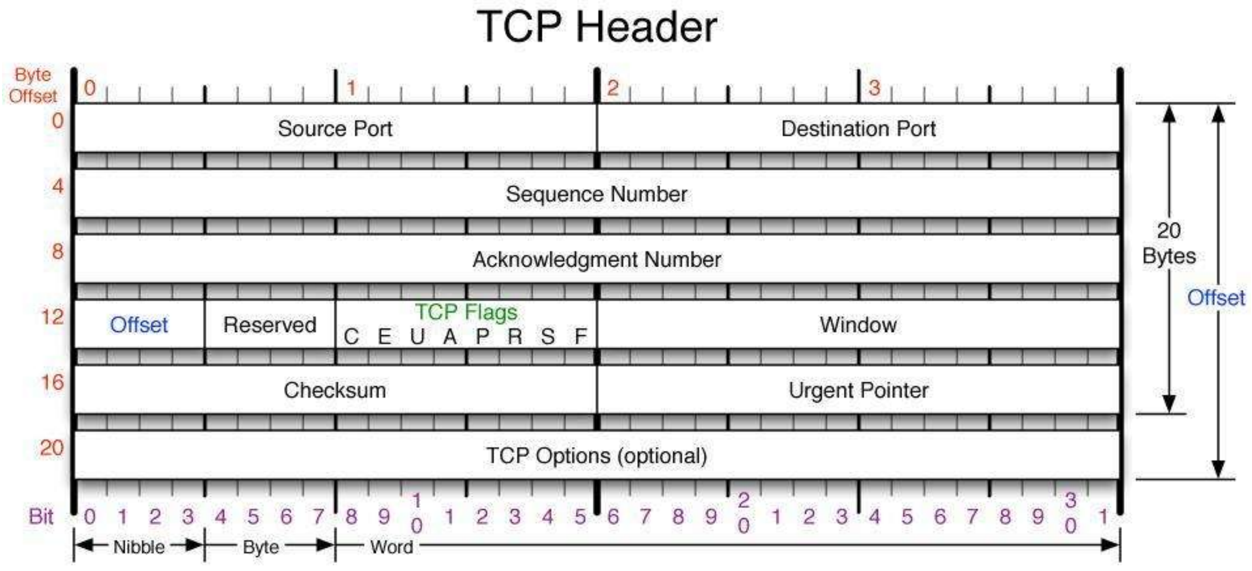
A packet analyzer (also known as a network analyzer, protocol analyzer or packet sniffer or for particular types of networks, an Ethernet sniffer or wireless sniffer) is a computer program or a piece of computer hardware that can intercept and log traffic passing over a digital network or part of a network. As data streams own across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate RFC or other specifications.

Different types of packet:

1. TCP:

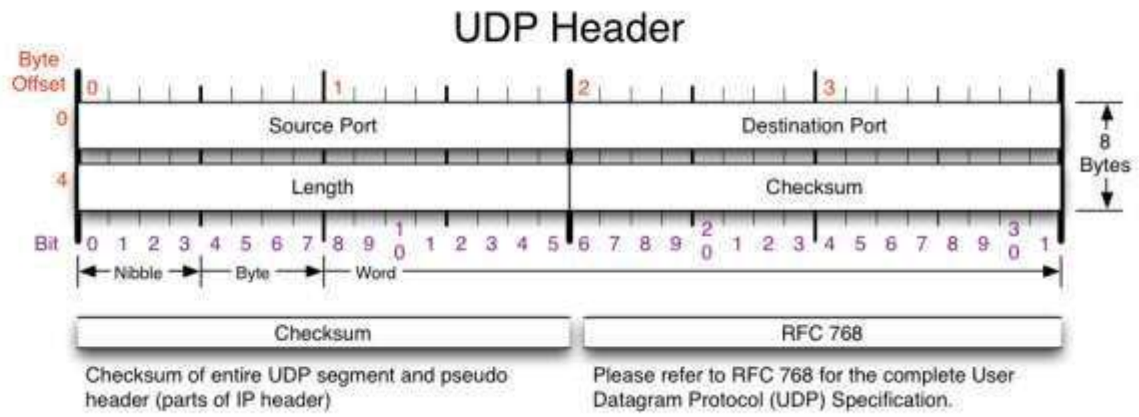
The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP. TCP provides reliable, ordered and error-checked delivery (or notification of failure to deliver) of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer. Web browsers use TCP when they connect to servers on the World Wide Web, and it is used to deliver email and transfer files from one location to another. The protocol corresponds to the transport layer of TCP/IP suite. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the

IP works by exchanging pieces of information called packets. A packet is a sequence of octets (bytes) and consists of a header followed by a body. The header describes the packet's source, destination and control information. The body contains the data IP is transmitting. Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, its source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details. TCP is a reliable stream delivery service that guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer over many networks is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends. The sender also maintains a timer from when the packet was sent, and retransmits a packet if the timer expires before the message has been acknowledged. The timer is needed in case a packet gets lost or corrupted.



2. UDP:

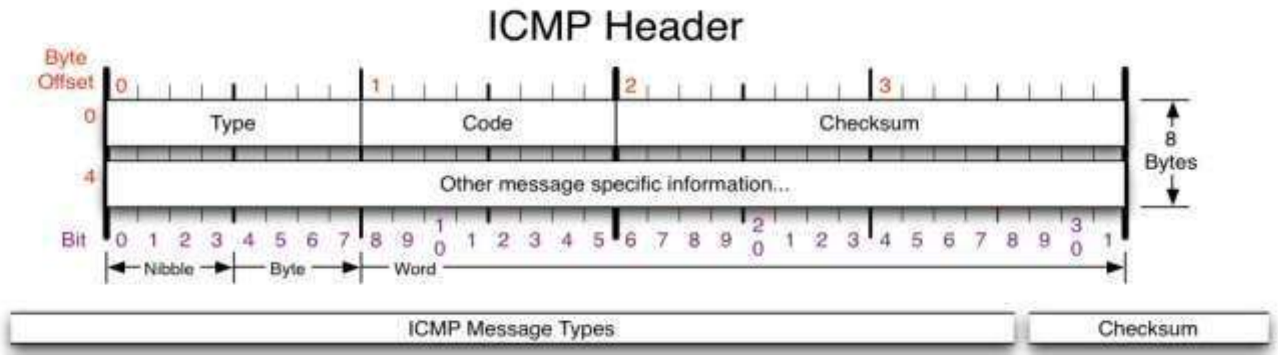
The User Datagram Protocol (UDP) is one of the core members of the Internet protocol Suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.



3.ICMP:

The Internet Control Message Protocol (ICMP) is one of the main protocols of the Internet Protocol Suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached.

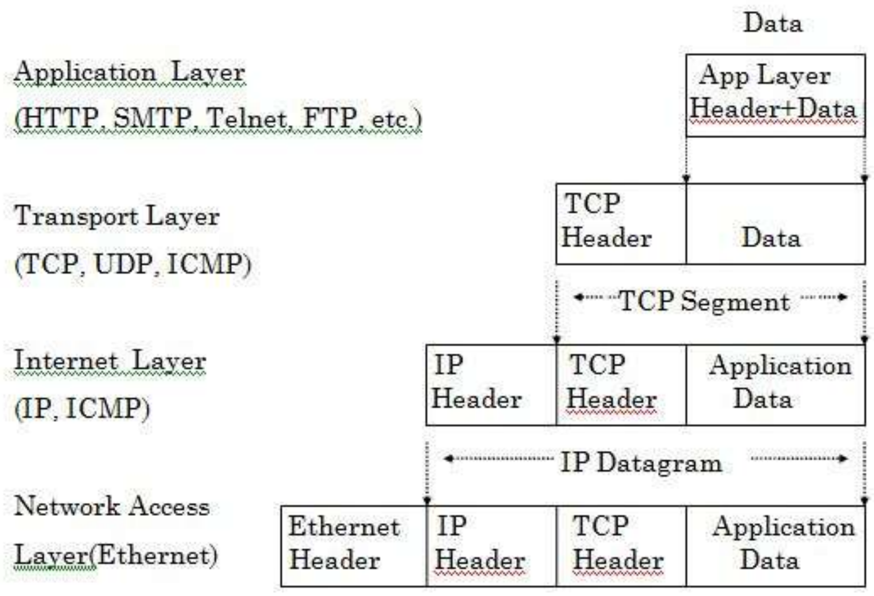
ICMP can also be used to relay query messages. It is assigned protocol number 1.ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the
exception of some diagnostic tools like ping and trace route). ICMP for Internet Protocol version 4 (IPv4) is also known as ICMPv4. IPv6 has a similar protocol, ICMPv6. The Internet Control Message Protocol is part of the Internet Protocol Suite, as defined in RFC 792. ICMP messages



are typically used for diagnostic or control purposes or generated in response to errors in IP operations. ICMP errors are directed to the source IP address of the originating packet.

4.IGMP:

The Internet Group Management Protocol (IGMP) is a communications protocol used by hosts and adjacent routers on IP networks to establish multicast group memberships. IGMP is an integral part of IP multicast.IGMP can be used for one-to-many networking applications such as online streaming video and gaming, and allows more efficient use of resources when supporting these types of applications. IGMP messages are carried in bare IP packets with IP protocol. There is no transport layer used with IGMP messaging, similar to the Internet Control Message Protocol. Membership Queries are sent by multicast routers to determine which multicast addresses are of interest to systems attached to its network. Routers periodically send General Queries to refresh the group membership state for all systems on its network. Group-Specific Queries are used for determining the reception state for a particular multicast address.



TCP/IP model

CONCLUSION:

Hence we have implemented packet formats captured through Wireshark for wired network. 1. Ethernet 2. IP 3.TCP 4.UDP .

Assignment No.	**A-8**
Title	Write a program for DNS lookup. Given an IP address input, it should return URL & vice versa..
Class	T.E. I(Comp.Engg.)
Date	
Subject	CN Lab

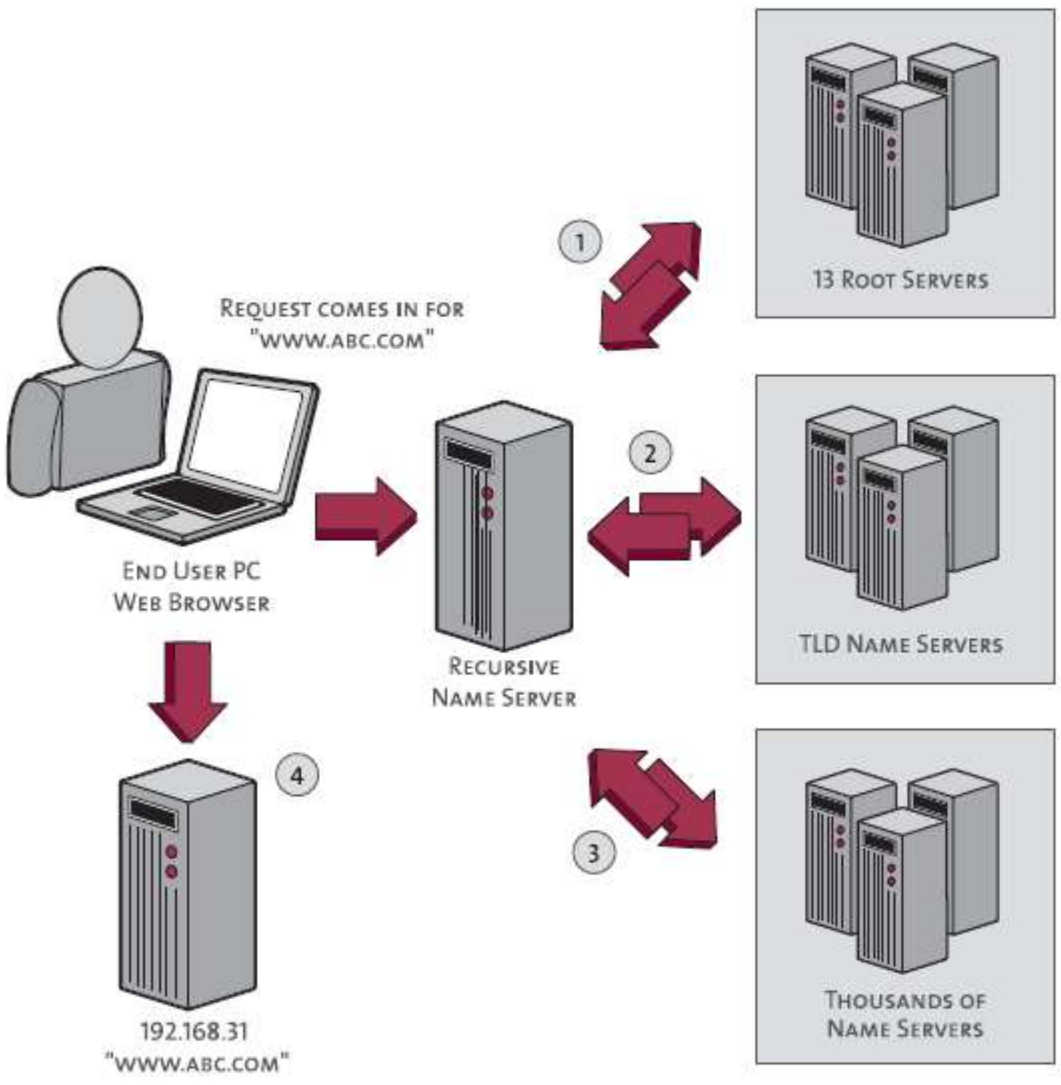
ASSIGNMENT-A8

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

Theory:-A DNS lookup, in a general sense, is the process by which a DNS record is returned from a DNS server. This is like looking up a phone number in a phone book - that is why it is referred to as a "lookup". Interconnected computers, servers and smart phones need to know how to translate the email addresses and domain names people use into meaningful numerical addresses. A DNS lookup performs this function.

The basic idea of DNS is that humans can't easily remember long strings of digits like machines can, but can much more easily remember words. So, when you type in a domain name like www.techopedia.com, the request is forwarded to a DNS server (whether locally or at an ISP), which returns the corresponding IP address. This address is then used by all the computers and routers to channel the request and responses of a user's session. The result is the user sees web pages as expected or has email show up in an in-box.

The two types of DNS lookups are forward DNS lookups and reverse DNS lookups.Forward DNS lookup is using an Internet domain name to find an IP address. Reverse DNS lookup is using an Internet IP address to find a domain name. When you enter the address for a Web site at your browser (the address is formally called the Uniform Resource Locator, or URL), the address is transmitted to a nearby router which does a forward DNS lookup in a routing table to locate the IP address. Forward DNS (which stands for domain name system) lookup is the more common lookup since most users think in terms of domain names rather than IP addresses. However, occasionally you may see a Web page with a URL in which the domain name part is expressed as an IP address (sometimes called a dot address) and want to be able to see its domain name. An Internet facility that lets you do either forward or reverse DNS lookup yourself is called nslookup. It comes with some operating systems or you can download the program and install it in your computer.



First your computer queries the name server (DNS server) it is set up to use. This is the recursive name server shown above.

The name server doesn't know the IP address for www.abc.com, so it will start the following chain of queries before it can report back the IP address to your computer (the numbers below correspond to the numbers in the image).

1. Query the **Internet root servers** to get the name servers for the .com TLD.
2. Query the .com **TLD name servers** to get the authoritative name servers for abc.com.
3. Query the **authoritative name servers for abc.com** to finally get the IP address for the host www.abc.com, then return that IP address to your computer.
4. Done! Now that your computer has the IP address for www.abc.com, it can access that host.

Conclusion:- Hence we have implemented DNSlookup.

Assignment No.	**B-01**
Title	**Lab Assignment on Unit V: (Use JAVA/PYTHON)** 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.
Class	T.E. I(Comp.Engg.)
Date	
Subject	CN Lab

ASSIGNMENT-B1

PROBLEM STATEMENT:

Lab Assignment on Unit V: (Use JAVA/PYTHON)
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:

Algorithm (Client Side)

1. Start.
2. Create a socket using socket() system call.
3. Connect the socket to the address of the server using connect() system call.
4. Send the requiredstring using send() system call.
5. Receive string from server by recv() system call.
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive string from client.
7. Reply to client
8. Stop.

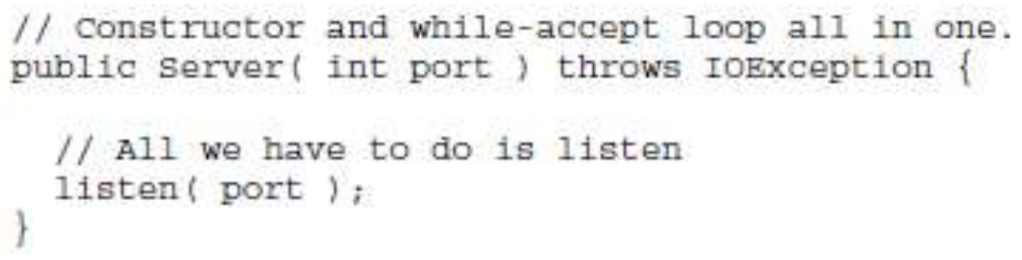
TCP is used as the transport layer protocol , since it provides reliable elivery which is critical for the given application. TCP does not provide timing guarantee, which is not very important in the given scenario

Server

The chat server does the following things
1. Accept multiple incoming connections for client.
2. Read incoming messages from each client and broadcast them to all other connected clients.
The server handles multiple chat clients with select based multiplexing. The select function monitors all the client sockets and the master socket for readable activity. If any of the client
socket is readable then it means that one of the chat client has send a message.

Listening on a port
The first thing we have to do is to get ready to receive incoming connections. To
Do this, we must listen on
a port.

The class
The Listener Class . We'll call it
Server.java
.
The constructor
The constructor for Server takes a single parameter -- a port number. This tells
Us what port to listen on when we are ready to start accepting connections. Here
is the constructor

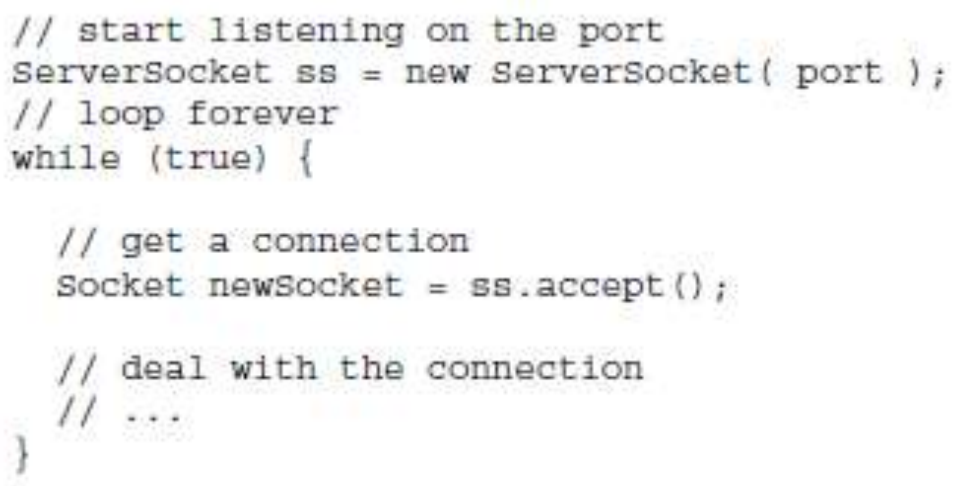


```
// Constructor and while-accept loop all in one.
public Server( int port ) throws IOException {

  // All we have to do is listen
  listen( port );
}
```

:
Accepting Sockets
Remember that your program will potentially be serving many clients from all over the Internet.
And these clients will be connecting to your server without regard to each other. That is, there's
no way to control the order, or the timing, with which the connections are arriving. As we'll see

later, multithreading is an excellent way to deal with these connections once they have come
in.
However, we're still trying to deal with the connections as they arrive.
The socket metaphor provides a straightforward solution: it *serializes* the incoming connections. That is, it makes them seem as if they are coming in one at a time, and arriving
just as you ask for them.
Here's what it looks like, in the abstract:



```
// start listening on the port
ServerSocket ss = new ServerSocket( port );
// loop forever
while (true) {

  // get a connection
  Socket newSocket = ss.accept();

  // deal with the connection
  // ...
}
```

The accept() routine is the key here. When this method of ServerSocket is called, it returns
a new Socket object that represents a new connection that has come in. After you've dealt with
this connection, you call accept() and get the next one.

Chat client
Now lets code the chat client that will connect to the above chat server. The client is based on the telnet program in python . It connects to a remote server, sends messages and receives messages.
The chat client does the following 2 things :

1. Listen for incoming messages from the server.
2. Check user input. If the user types in a message then send it to the server.

The Client is implemented using two threads, one each for incoming and outgoing messages. The main thread opens the socket and connects to the server. It then opens input buffered stream and print writer for in coming and outgoing messages respectively. It also creates a buffered stream to read from the console. The main thread takes care of the outgoing message, while another thread deals with incoming messages. In order to send a message, the user types the destination user name following by the message string on the console, which is then read into the buffered stream reading from the console. The main thread creates the formatted message by adding the destination user name to the beginning of the message string and writes it to the print writer stream. Control messages have the string “server” as the header.

Conclusion:- Hence we have implemented TCP socket for peer to peer and multiuser chat

Assignment No.	**B-02**
Title	**Lab Assignment on Unit V: (Use JAVA/PYTHON)** 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.
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN Lab

ASSIGNMENT-B2

PROBLEM STATEMENT:

Lab Assignment on Unit V: (Use JAVA/PYTHON)
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:

Algorithm (Client Side)

1. Start.
2. Create a socket using socket() system call.
3. Connect the socket to the address of the server using connect() system call.
4. Send the required string using send() system call.
5. Receive string from server by recv() system call.
6. Stop.

Algorithm (Server Side)

1. Start.
2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive string from client.
7. Reply to client
8. Stop.

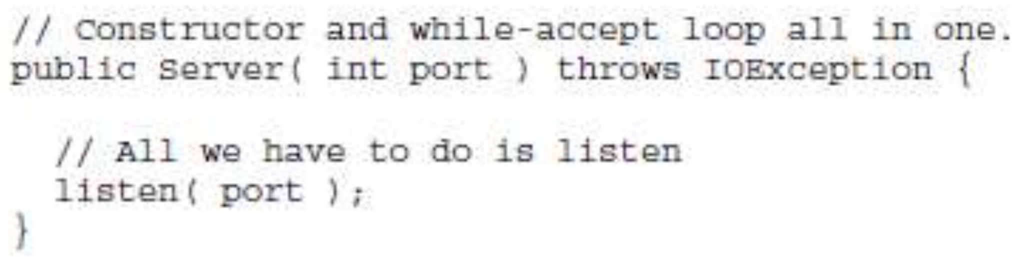
UDP is used as the transport layer protocol , it provides connectionless Services. UDP does not provide timing guarantee, which is not very important in the given scenario

Server

The chat server does the following things
1. Accept multiple incoming connections for client.
2. Read incoming messages from each client and broadcast them to all other connected clients.
The server handles multiple chat clients with select based multiplexing. The select function monitors all the client sockets and the master socket for readable activity. If any of the client
socket is readable then it means that one of the chat client has send a message.

Listening on a port
The first thing we have to do is to get ready to receive incoming connections. To Do this, we must listen on
a port.

The class
The Listener Class . We'll call it
Server.java
.
The constructor
The constructor for Server takes a single parameter -- a port number. This tells Us what port to listen on when we are ready to start accepting connections. Here is the constructor



```
// Constructor and while-accept loop all in one.
public Server( int port ) throws IOException {

  // All we have to do is listen
  listen( port );
}
```

:
Accepting Sockets
Remember that your program will potentially be serving many clients from all over the Internet.
And these clients will be connecting to your server without regard to each other. That is, there's
no way to control the order, or the timing, with which the connections are arriving. As we'll see

later, multithreading is an excellent way to deal with these connections once they have come
in.
However, we're still trying to deal with the connections as they arrive.
The socket metaphor provides a straightforward solution: it *serializes* the incoming connections. That is, it makes them seem as if they are coming in one at a time, and arriving
just as you ask for them.
Here's what it looks like, in the abstract:

```
// start listening on the port
ServerSocket ss = new ServerSocket( port );
// loop forever
while (true) {

  // get a connection
  Socket newSocket = ss.accept();

  // deal with the connection
  // ...
}
```

The accept() routine is the key here. When this method of ServerSocket is called, it returns
a new Socket object that represents a new connection that has come in. After you've dealt with
this connection, you call accept() and get the next one.

Chat client
Now lets code the chat client that will connect to the above chat server. The client is based on the telnet program in python . It connects to a remote server, sends messages and receives messages.
The chat client does the following 2 things :

1. Listen for incoming messages from the server.
2. Check user input. If the user types in a message then send it to the server.

The Client is implemented using two threads, one each for incoming and outgoing messages. The main thread opens the socket and connects to the server. It then opens input buffered stream and print writer for in coming and outgoing messages respectively. It also creates a buffered stream to read from the console. The main thread takes care of the outgoing message, while another thread deals with incoming messages. In order to send a message, the user types the destination user name following by the message string
on the console, which is then read into the buffered stream reading from the console. The main thread creates the formatted message by adding the destination user name to the beginning of the message string and writes it to the print writer stream. Control messages have the string “server” as the header.

Conclusion:- Hence we have implemented UDP socket for peer to peer and multiuser chat

/

Assignment No.	**B-03**
Title	Configure a Network using Distance Vector Routing Protocol. –RIP
Class	T.E.I (Comp.Engg.)
Date	
Subject	CN Lab

Assignment No. B3

Title: Configure a Network using Distance Vector Routing Protocol. -RIP

Objectives:

- Cable a network according to the Topology Diagram.
- Erase the startup configuration and reload a router to the default state.
- Perform basic configuration tasks on a router.
- Configure and activate interfaces.
- Configure RIP routing on all routers..
- Document the RIP configuration.

Problem Statement: To setup network, configure and verify RIP.

Outcomes: Perform basic configuration tasks on a router.

Tools Required:

Software: Packet Sniffer

Theory:

Introduction:-

ROUTING PROTOCOLS: RIP [ROUTING INFORMATION PROTOCOL]

RIP is an open standard routing protocol. It is a distance vectored routing protocols. It is a class-full routing protocol where updates are exchanged through broadcast. The routing table is exchanged every 30 seconds among the routers in the inter-network. The RIP protocol uses hop count as the metric to find the shortest path but the maximum allowable hop count is 15 by default. The RIP protocols is used only for a small network and is ineffective for a large network. The Administrative Distance of RIP is 120.

Some of the of key features of RIP protocol are:
It supports maximum 15 hops in a path.
It uses hops count metric to calculate the best path from a source to a destination network.

It sends routing updates (entire routing table) after every 30 seconds and when the network changes.
It uses UDP broadcast packets to exchange routing information.
The Administrative Distance (AD) value of the RIP protocol is 120.
It has two versions: RIPv1 and RIPv2.

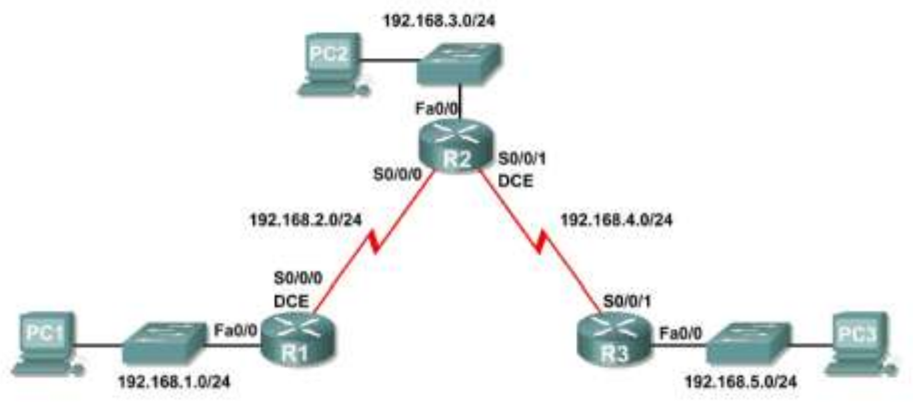
RIP TIMERS To manage the routing performance, RIP uses four different kinds of timers:

- Update timer: It is the time interval after which a router sends it's a copy of the routing table as update to the neighbor routers. The update timer is 30 sec by default.
- Invalid time: It is the time interval after which a router understands that the path to a network is invalid or becomes invalid. The invalid timer is 180 sec by default.
- Hold-down timer: It specifies the amount of time for which the information about the poorer routes are ignored. The hold-down timer is 180sec by default.
- Flush timer: It is the time before the invalid route is purged from the routing table. The flush timer is 240 sec by default.

Dis-advantages of RIP

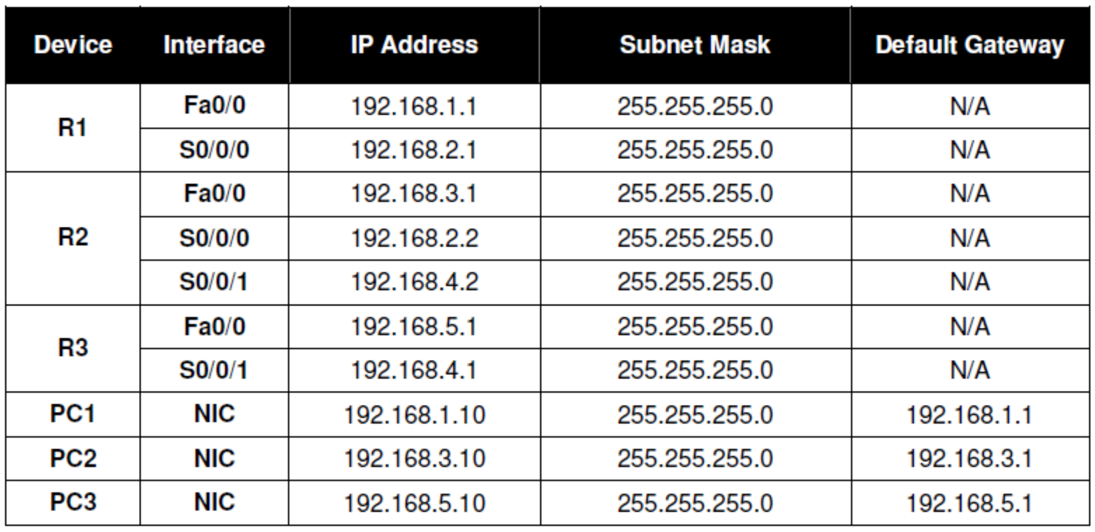
- It uses more bandwidth as updates are exchanged every 30 seconds where each update contains the complete routing table of the router.
- It does not uses bandwidth as the metric for calculation of the shortest path.
- RIP has a very slow convergence.
- RIP implementation can lead to routing loops in the network.
- RIP is only applicable to small network and is inefficient for larger networks.

Topology Diagram



Scenario A: Running RIPv1 on Classful Networks

Addressing Table



Device	Interface	IP Address	Subnet Mask	Default Gateway
R1	Fa0/0	192.168.1.1	255.255.255.0	N/A
	S0/0/0	192.168.2.1	255.255.255.0	N/A
R2	Fa0/0	192.168.3.1	255.255.255.0	N/A
	S0/0/0	192.168.2.2	255.255.255.0	N/A
	S0/0/1	192.168.4.2	255.255.255.0	N/A
R3	Fa0/0	192.168.5.1	255.255.255.0	N/A
	S0/0/1	192.168.4.1	255.255.255.0	N/A
PC1	NIC	192.168.1.10	255.255.255.0	192.168.1.1
PC2	NIC	192.168.3.10	255.255.255.0	192.168.3.1
PC3	NIC	192.168.5.10	255.255.255.0	192.168.5.1

Task 1: Prepare the Network.
Step 1: Cable a network that is similar to the one in the Topology Diagram.
You can use any current router in your lab as long as it has the required interfaces shown in the topology.
Step 2: Clear any existing configurations on the routers.

Task 2: Perform Basic Router Configurations.

Perform basic configuration of the R1, R2, and R3 routers according to the following guidelines:

1. Configure the router hostname.
2. Disable DNS lookup.
3. Configure an EXEC mode password.
4. Configure a message-of-the-day banner.
5. Configure a password for console connections.
6. Configure a password for VTY connections.

Task 3: Configure and Activate Serial and Ethernet Addresses.

Step 1: Configure interfaces on R1, R2, and R3.

Configure the interfaces on the R1, R2, and R3 routers with the IP addresses from the table under the
Topology Diagram.

Step 2: Verify IP addressing and interfaces.

Use the **show ip interface brief** command to verify that the IP addressing is correct and that the
interfaces are active.

When you have finished, be sure to save the running configuration to the NVRAM of the router.

Step 3: Configure Ethernet interfaces of PC1, PC2, and PC3.

Configure the Ethernet interfaces of PC1, PC2, and PC3 with the IP addresses and default gateways
from the table under the Topology Diagram.

Step 4: Test the PC configuration by pinging the default gateway from the PC.

Task 4: Configure RIP.

Step 1: Enable dynamic routing.

To enable a dynamic routing protocol, enter global configuration mode and use the **router** command.

Enter **router ?**at the global configuration prompt to a see a list of available routing protocols on your
router.

To enable RIP, enter the command **router rip** in global configuration mode.

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

Step 2: Enter classful network addresses.

Once you are in routing configuration mode, enter the classful network address for each directly
connected network, using the **network** command.

```
R1(config-router)#network 192.168.1.0
R1(config-router)#network 192.168.2.0
R1(config-router)#
```

The **network** command:
□□Enables RIP on all interfaces that belong to this network. These interfaces will now both send and
receive RIP updates.
□□Advertises this network in RIP routing updates sent to other routers every 30 seconds.
When you are finished with the RIP configuration, return to privileged EXEC mode and save the current
configuration to NVRAM.

```
R1(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R1#copy run start
```

Step 3: Configure RIP on the R2 router using the router rip and network commands.

```
R2(config)#router rip
R2(config-router)#network 192.168.2.0
R2(config-router)#network 192.168.3.0
R2(config-router)#network 192.168.4.0
R2(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R2#copy run start
```

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current
configuration to NVRAM.

Step 4: Configure RIP on the R3 router using the router rip and network commands.

```
R3(config)#router rip
R3(config-router)#network 192.168.4.0
R3(config-router)#network 192.168.5.0
R3(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R3# copy run start
```

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current
configuration to NVRAM.

Task 5: Verify RIP Routing.
Step 1: Use the show ip route command to verify that each router has all of the networks in the topology entered in the routing table.

Routes learned through RIP are coded with an **R** in the routing table. If the tables are not converged as shown here, troubleshoot your configuration. Did you verify that the configured interfaces are active? Did you configure RIP correctly? Return to Task 3 and Task 4 to review the steps necessary to achieve convergence.

```
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
* - candidate default, U - per-user static route, o - ODR
P - periodic downloaded static route
Gateway of last resort is not set
C 192.168.1.0/24 is directly connected, FastEthernet0/0
C 192.168.2.0/24 is directly connected, Serial0/0/0
R 192.168.3.0/24 [120/1] via 192.168.2.2, 00:00:04, Serial0/0/0
R 192.168.4.0/24 [120/1] via 192.168.2.2, 00:00:04, Serial0/0/0
R 192.168.5.0/24 [120/2] via 192.168.2.2, 00:00:04, Serial0/0/0
R1#
R2#show ip route
<Output omitted>
R 192.168.1.0/24 [120/1] via 192.168.2.1, 00:00:22, Serial0/0/0
C 192.168.2.0/24 is directly connected, Serial0/0/0
C 192.168.3.0/24 is directly connected, FastEthernet0/0
C 192.168.4.0/24 is directly connected, Serial0/0/1
R 192.168.5.0/24 [120/1] via 192.168.4.1, 00:00:23, Serial0/0/1
R2#
R3#show ip route
<Output omitted>
R 192.168.1.0/24 [120/2] via 192.168.4.2, 00:00:18, Serial0/0/1
R 192.168.2.0/24 [120/1] via 192.168.4.2, 00:00:18, Serial0/0/1
R 192.168.3.0/24 [120/1] via 192.168.4.2, 00:00:18, Serial0/0/1
```

```
C 192.168.4.0/24 is directly connected, Serial0/0/1
C 192.168.5.0/24 is directly connected, FastEthernet0/0
R3#
```

Step 2: Use the show ip protocols command to view information about the routing processes.

The **show ip protocols** command can be used to view information about the routing processes that
are occurring on the router. This output can be used to verify most RIP parameters to confirm that:

- RIP routing is configured
- The correct interfaces send and receive RIP updates
- The router advertises the correct networks
- RIP neighbors are sending updates

```
R1#show ip protocols
Routing Protocol is "rip"
Sending updates every 30 seconds, next due in 16 seconds
Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
Redistributing: rip
Default version control: send version 1, receive any version
Interface Send Recv Triggered RIP Key-chain
FastEthernet0/0 1 2 1
Serial0/0/0 1 2 1
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
192.168.1.0
192.168.2.0
Passive Interface(s):
Routing Information Sources:
Gateway Distance Last Update
192.168.2.2 120
Distance: (default is 120)
R1#
```

R1 is indeed configured with RIP. R1 is sending and receiving RIP updates on FastEthernet0/0 and
Serial0/0/0. R1 is advertising networks 192.168.1.0 and 192.168.2.0. R1 has one routing information

source. R2 is sending R1 updates.

Step 3: Use the debug ip rip command to view the RIP messages being sent and received.

Rip updates are sent every 30 seconds so you may have to wait for debug information to be displayed.

```
R1#debug ip rip
R1#RIP: received v1 update from 192.168.2.2 on Serial0/0/0
192.168.3.0 in 1 hops
192.168.4.0 in 1 hops
192.168.5.0 in 2 hops
RIP: sending v1 update to 255.255.255.255 via FastEthernet0/0 (192.168.1.1)
RIP: build update entries
network 192.168.2.0 metric 1
network 192.168.3.0 metric 2
network 192.168.4.0 metric 2
network 192.168.5.0 metric 3
RIP: sending v1 update to 255.255.255.255 via Serial0/0/0 (192.168.2.1)
RIP: build update entries
network 192.168.1.0 metric 1
```

The debug output shows that R1 receives an update from R2. Notice how this update includes all the
networks that R1 does not already have in its routing table. Because the FastEthernet0/0 interface
belongs to the 192.168.1.0 network configured under RIP, R1 builds an update to send out that interface.
The update includes all networks known to R1 except the network of the interface. Finally, R1 builds an
update to send to R2. Because of split horizon, R1 only includes the 192.168.1.0 network in the update.

Step 4: Discontinue the debug output with the undebug all command.

```
R1#undebug all
All possible debugging has been turned off
```

Conclusion: Thus we were able to setup network, configure and verify RIP