

# **Yeshwantrao Chavan College of Engineering**

(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)

**Wanadongri Hingna Road, Nagpur-441 110**

**Session 2022-2023**

**Department of Computer Technology**



**LAB MANUAL**

**ON**

**Computer Networks Lab  
(AIDS 2204)**

**THIRD SEMESTER (AIDS)**

---

# YESHWANTRAO CHAVAN COLLEGE OF ENGINEERING, NAGPUR

## DEPARTMENT OF COMPUTER TECHNOLOGY

### CT2302: LAB: COMPUTER NETWORKS

#### List of Practical's

Sr. No.	Name of Practical	Date of Completion	Mapped CO
1	Use Network Utility Command like ping, ipconfig, netstat, tracert to observe the network details.	Before MSE I	CO2,CO4
2	To implement Hamming Code using C and C++.		CO3, CO4
3	To implement Dijkstra's Routing algorithm using backtracking approach.		
4	Use traffic monitoring tool Wireshark to observe network traffic with packet details.	Before MSEII	CO3,CO4
5	Configure network using Cisco Packet Tracer software and show packet transmission from source to destination.		CO4
6	Configure network using Distance vector routing protocol in Cisco Packet Tracer	Before Last Teaching Day	CO4
7	Use Openssl command to perform Asymmetric key encryption(RSA) and also implement RSA algorithm.		CO4
8	Client server communication using socket programming.		CO1, CO2, CO4
9	Advanced Practical: Study of NSG tool		CO1, CO2

**Course Teacher**

Prof. Shweta N. Jain

**HOD**

Dr. R.D. Wajgi

---

## Experiment No. 1

**Aim:-** Use Network Utility Command like ping, ipconfig, netstat, tracert to observe the network details.

### Theory:

**Ping:** The ping command helps to verify IP-level connectivity. When troubleshooting, you can use ping to send an ICMP echo request to a target host name or IP address. Use ping whenever you need to verify that a host computer can connect to the TCP/IP network and network resources. You can also use ping to isolate network hardware problems and incompatible configurations.

It is usually best to verify that a route exists between the local computer and a network host by first using the ping command and the IP address of the network host to which you want to connect. Try pinging the IP address of the target host to see if it responds, as follows:

### Ping IP\_address

You should perform the following steps when using ping:

1. Ping the loopback address to verify that TCP/IP is installed and configured correctly on the local computer.

ping 127.0.0.1

2. Ping the IP address of the local computer to verify that it was added to the network correctly.

ping IP\_address\_of\_local\_host

3. Ping the IP address of the default gateway to verify that the default gateway is functioning and that you can communicate with a local host on the local network.

ping IP\_address\_of\_default\_gateway

4. Ping the IP address of a remote host to verify that you can communicate through a router.

ping IP\_address\_of\_remote\_host

The ping command uses Windows Sockets-style name resolution to resolve a computer name to an IP address, so if pinging by address succeeds, but pinging by name fails, then the problem lies in address or name resolution, not network connectivity.

---

If you cannot use ping successfully at any point, confirm that:

- The computer was restarted after TCP/IP was installed and configured.
- The IP address of the local computer is valid and appears correctly on the General tab of the Internet Protocol (TCP/IP) Properties dialog box.
- IP routing is enabled and the link between routers is operational.

**Ipconfig:**(short for interface configuration) is a system administration utility in [Unix- like](#) operating systems to configure, control, and query [TCP/IP network interface](#) parameters from a [command line interface](#) (CLI) or in system configuration scripts. Ifconfig originally appeared in 4.2BSD as part of the [BSD TCP/IP](#) suite.

To run the Ipconfig.exe utility, at a command prompt, type ipconfig, and then add any appropriate Option

## SYNTAX

ipconfig [/? | /all | /renew adapter | /release adapter | /flushdns | /displaydns | /registerdns | /showclassid adapter | /setclassid adapter classid

The adapter connection name can use wildcard characters (\* and ?).

## OPTIONS

/?            Displays this help message

/all          Displays full configuration information

/release     Releases the IP address for the specified adapter

/renew       Renews the IP address for the specified adapter

/flushdns    Purges the DNS Resolver cache

/registerdns Refreshes all DHCP leases and reregisters DNS names

/displaydns Displays the contents of the DNS Resolver Cache

/showclassid Displays all the DHCP ClassIds allowed for the specified adapter

/setclassid Modifies the DHCP ClassId

The default (with no parameters specified) is to display only the IP address, subnet mask, and default gateway for each adapter that is bound to TCP/IP.

---

For /all, Ipconfig displays all of the current TCP/IP configuration values, including the IP address, subnet mask, default gateway, and Windows Internet Naming Service (WINS) and DNS configuration.

For /release and /renew, if no adapter name is specified, the IP address leases for all adapters that are bound to TCP/IP are released or renewed.

For /setclassid, if no ClassId is specified, the ClassId is removed.

## EXAMPLES

ipconfig            Show information

ipconfig /all        Show detailed information

ipconfig /renew      Renew all adapters

ipconfig /renew EL\*    Renew any connection whose name starts EL

ipconfig /release \*Con\*    Release all matching connections, for example, "Local Area Connection 1" or "Local Area Connection2"

**NETSTAT:** netstat (network statistics) is a [command-line tool](#) that displays [network connections](#) (both incoming and outgoing), routing tables, and a number of network interface statistics. It is available on [Unix](#), [Unix-like](#), and [Windows NT-based operating systems](#).

It is used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement

Parameters used with this command must be prefixed with a hyphen (-) rather than a slash (/).

-a	Displays all active connections and the TCP and <a href="#">UDP ports</a> on which the computer is listening.
-b	Displays the binary (executable) program's name involved in creating each connection or listening port. (Windows XP, 2003 Server and newer Windows operating systems (not Microsoft Windows 2000 or other non-Windows operating systems)) On Mac OS X when combined with
-i	the total number of bytes of traffic will be reported.

-e	Displays <a href="#">ethernet</a> statistics, such as the number of <a href="#">bytes</a> and packets sent and received. This parameter can be combined with -s.
-f Windows	Displays fully qualified domain names < <a href="#">FQDN</a> > for foreign addresses (only available on Windows Vista and newer operating systems).
-f FreeBSDAddress Family	Limits display to a particular socket address family, unix, inet, inet6
-g	Displays multicast group membership information for both IPv4 and IPv6 (may only be available on newer operating systems)
-i	Displays network interfaces and their statistics (not available under Windows)
-m	Displays the STREAMS statistics.
-n	Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names.
-o	Displays active TCP connections and includes the process ID (PID) for each connection. You can find the application based on the PID on the Processes tab in Windows Task Manager. This parameter can be combined with -a, -n, and -p. This parameter is available on Microsoft Windows XP, 2003 Server (and Windows 2000 if a hotfix is applied). <sup>[2]</sup>
-p Windows and <a href="#">BSD</a> :Protocol	Shows connections for the protocol specified by Protocol. In this case, the Protocol can be tcp, udp, tcpv6, or udpv6. If this parameter is used with -s to display statistics by protocol, Protocol can be tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6, or ipv6.
-p LinuxProcess	Show which processes are using which sockets (similar to -b under Windows) (you must be root to do this)

-P SolarisProtocol	Shows connections for the protocol specified by Protocol. In this case, the Protocol can be ip, ipv6, icmp, icmpv6, igmp, udp, tcp, or rawip.
-r	Displays the contents of the <a href="#">IP routing table</a> . (This is equivalent to the route print command under Windows.)
-s	Displays statistics by protocol. By default, statistics are shown for the <a href="#">TCP</a> , <a href="#">UDP</a> , <a href="#">ICMP</a> , and <a href="#">IP</a> protocols. If the IPv6 protocol for Windows XP is installed, statistics are shown for the TCP over <a href="#">IPv6</a> , UDP over IPv6, <a href="#">ICMPv6</a> , and IPv6 protocols. The -p parameter can be used to specify a set of protocols.
-t	Linux: Displays only TCP connections.
-w	FreeBSD: Displays wide output - doesn't truncate hostnames or IPv6 addresses
-v	When used in conjunction with -b it will display the sequence of components involved in creating the connection or listening port for all executables.
Interval	Redisplays the selected information every Interval seconds. Press CTRL+C to stop the redisplay. If this parameter is omitted, netstat prints the selected information only once.
-h (unix) /? (windows)	Displays help at the command prompt.

Netstat provides statistics for the following:

- Proto - The name of the protocol ([TCP](#) or [UDP](#)).
- Local Address - The [IP](#) address of the local computer and the port number being used. The name of the local computer that corresponds to the [IP](#) address and the name of the port is shown unless the -n parameter is specified. If the port is not yet established, the port number is shown as an asterisk (\*).

- Foreign Address - The [IP](#) address and port number of the remote computer to which the socket is connected. The names that corresponds to the [IP](#) address and the port are shown unless the -n parameter is specified. If the port is not yet established, the port number is shown as an asterisk (\*).
- State - Indicates the state of a [TCP](#) connection. The possible states are as follows: CLOSE\_WAIT, CLOSED, ESTABLISHED, FIN\_WAIT\_1, FIN\_WAIT\_2, LAST\_ACK, LISTEN, SYN\_RECEIVED, SYN\_SEND, and TIME\_WAIT. For more information about the states of a TCP connection, see [RFC 793](#).

---

## Examples

To display the statistics for only the TCP or UDP protocols, type one of the following commands:

```
netstat -sp tcp
```

---



```
netstat -sp udp
```

To display active TCP connections and the process IDs every 5 seconds, type the following command (On Microsoft Windows, works on XP and 2003 only, or Windows 2000 with hotfix):

```
netstat -o 5
```

Mac     OS     X

```
version netstat -w
```

5

To display active TCP connections and the process IDs using numerical form, type the following command (On Microsoft Windows, works on XP and 2003 only, or Windows 2000 with hotfix):

```
netstat -no
```

To display all ports open by a process with id pid

```
netstat -aop | grep "pid"
```

### **Trace route:**

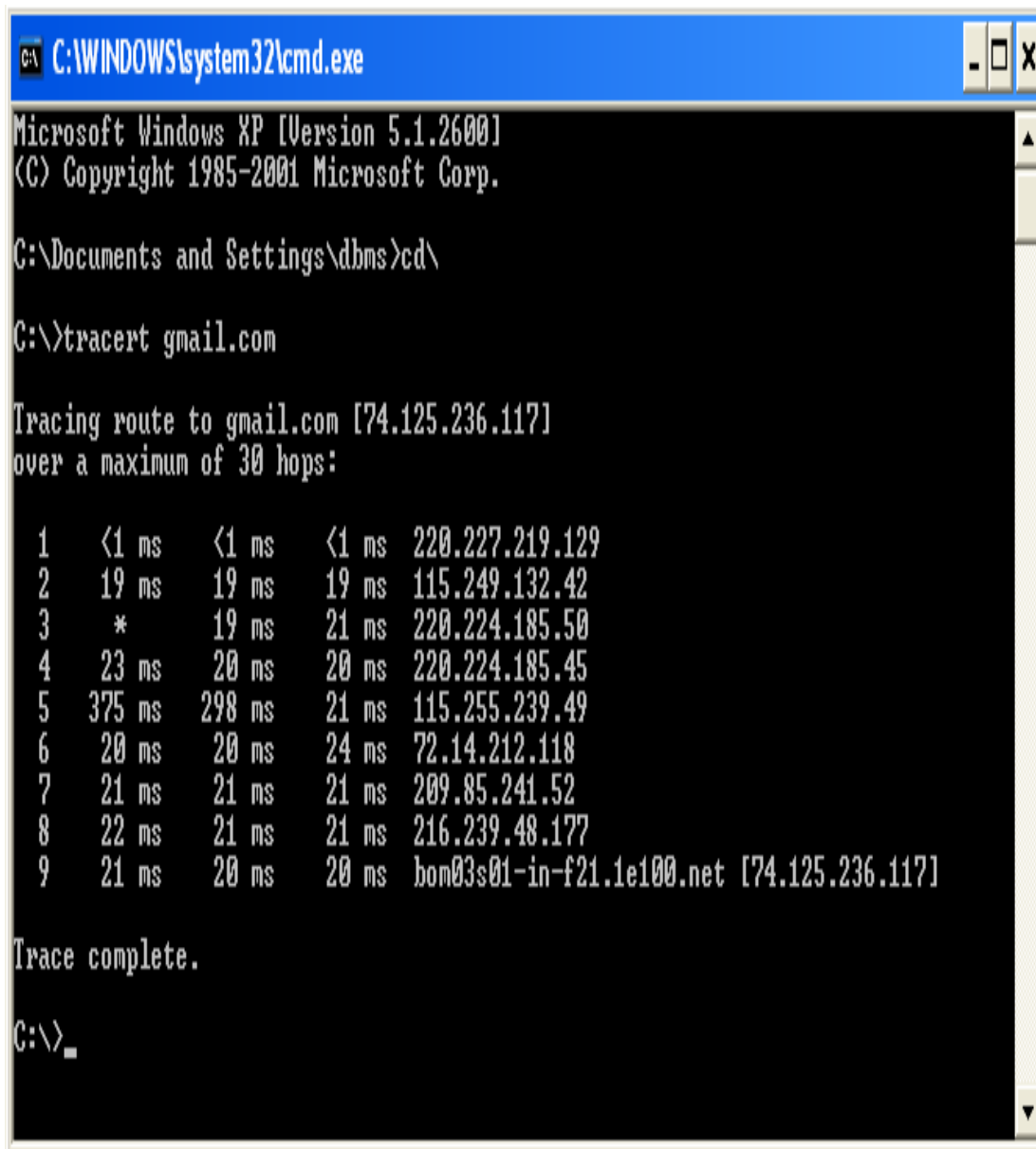
Trace route is a command which can show you the path a packet of information takes from your computer to one you specify. It will list all the routers it passes through until it reaches its destination, or fails to and is discarded. In addition to this, it will tell you how long each 'hop' from router to router takes.

In Windows, select Start > Programs > Accessories > Command Prompt. This will give you a window like the one below.

---

---

Enter the word `tracert`, followed by a space, then the domain name



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dbms>cd\

C:\>tracert gmail.com

Tracing route to gmail.com [74.125.236.117]
over a maximum of 30 hops:

  0  <1 ms    <1 ms    <1 ms    220.227.219.129
  1  19 ms     19 ms     19 ms     115.249.132.42
  2  *         19 ms     21 ms     220.224.185.50
  3  23 ms     20 ms     20 ms     220.224.185.45
  4  375 ms    298 ms    21 ms     115.255.239.49
  5  20 ms     20 ms     24 ms     72.14.212.118
  6  21 ms     21 ms     21 ms     209.85.241.52
  7  22 ms     21 ms     21 ms     216.239.48.177
  8  21 ms     20 ms     20 ms     bom03s01-in-f21.1e100.net [74.125.236.117]

Trace complete.

C:\>_
```

---

## **Experiment No.: 2**

**Aim:- To implement Hamming Code using C and C++.**

**Theory: -**

### **Error Detection and Correction**

Telephone system has three parts: the switches, the interoffice trunks, and the local loops. The first two are now almost entirely digital in most developed countries. The local loops are still analog twisted copper pairs and will continue to be so for years due to the enormous expense of replacing them. While errors are rare on the digital part, they are still common on the local loops. Furthermore, wireless communication is becoming more common, and the error rates here are orders of magnitude worse than on the interoffice fiber trunks. The conclusion is: transmission errors are going to be with us for many years to come. We have to learn how to deal with them.

As a result of the physical processes that generate them, errors on some media (e.g., radio) tend to come in bursts rather than singly. Having the errors come in bursts has both advantages and disadvantages over isolated single-bit errors. On the advantage side, computer data are always sent in blocks of bits. Suppose that the block size is 1000 bits and the error rate is 0.001 per bit. If errors were independent, most blocks would contain an error. If the errors came in bursts of 100 however, only one or two blocks in 100 would be affected, on average. The disadvantage of burst errors is that they are much harder to correct than are isolated errors.

### **3.2.1 Error-Correcting Codes**

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

---

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of  $m$  data (i.e., message) bits and  $r$  redundant, or check, bits. Let the total length be  $n$  (i.e.,  $n = m + r$ ). An  $n$ -bit unit containing data and check bits is often referred to as an  $n$ -bit codeword.

Given any two codewords, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just exclusive OR the two codewords and count the number of 1 bits in the result, for example:

```
10001001
10110001
00111000
```

The number of bit positions in which two code words differ is called the Hamming distance (Hamming, 1950). Its significance is that if two code words are a Hamming distance  $d$  apart, it will require  $d$  single-bit errors to convert one into the other.

In most data transmission applications, all  $2^m$  possible data messages are legal, but due to the way the check bits are computed, not all of the  $2^n$  possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code.

The error-detecting and error-correcting properties of a code depend on its Hamming distance. To detect  $d$  errors, you need a distance  $d + 1$  code because with such a code there is no way that  $d$  single-bit errors can change a valid codeword into another valid codeword. When the

---

receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct  $d$  errors, you need a distance  $2d + 1$  code because that way the legal codewords are so far apart that even with  $d$  changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.

As a simple example of an error-detecting code, consider a code in which a single parity bit is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords:

0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with  $m$  message bits and  $r$  check bits that will allow all single errors to be corrected. Each of the  $2^m$  legal messages has  $n$  illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the  $n$  bits in the  $n$ -bit codeword formed from it. Thus, each of the  $2^m$  legal messages requires  $n + 1$  bit patterns

dedicated to it. Since the total number of bit patterns is  $2^n$ , we must have  $(n + 1)2^m \leq 2^n$ .

Using  $n = m + r$ , this requirement becomes  $(m + r + 1) \leq 2^r$ . Given  $m$ , this puts a lower limit on the number of check bits needed to correct single errors.

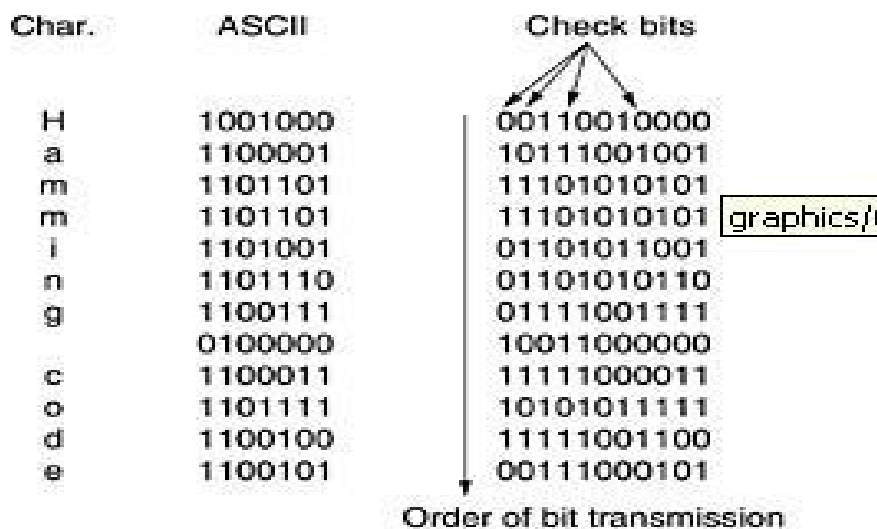
This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950). The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to

---

its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the  $m$  data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position  $k$  contributes to, rewrite  $k$  as a sum of powers of 2. For example,  $11 = 1 + 2 + 8$  and  $29 = 1 + 4 + 8 + 16$ . A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8).

When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit,  $k$  ( $k = 1, 2, 4, 8, \dots$ ), to see if it has the correct parity. If not, the receiver adds  $k$  to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Figure 3-7 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

**Figure:- Use of a Hamming code to correct burst errors.**



Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of  $k$  consecutive codeword are



arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all  $k$  bits have been sent, the second column is sent, and so on, as indicated in Fig. 3-7. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length  $k$  occurs, at most 1 bit in each of the  $k$  codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be blocks of  $km$  data bits immune to a single burst error of length  $k$  or less restored. This method uses  $kr$  check bits to make



## Experiment No.: 3

**Aim:** To implement Dijkstra's Routing algorithm using backtracking approach.

### **Theory:**

The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the previously-established route.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. **Nonadaptive algorithms** do not base their routing decisions on measurements or estimates of the current traffic and topology. Instead, the choice of the route to use to get from I to J (for all I and J) is computed in advance, off-line, and downloaded to the routers when the network is booted. This procedure is sometimes called static routing

**Adaptive algorithms**, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every  $\Delta$  sec, when the load changes or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time). In the following sections we will discuss a variety of routing algorithms, both static and dynamic.

Before we get into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is known as the **optimality principle**. It states that if router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route. To see this, call the part of the route from I to J  $r_1$  and the rest of the route  $r_2$ . If a route better than  $r_2$  existed from J to K, it could be concatenated with  $r_1$  to improve the route from I to K, contradicting our statement that  $r_1 r_2$  is optimal.

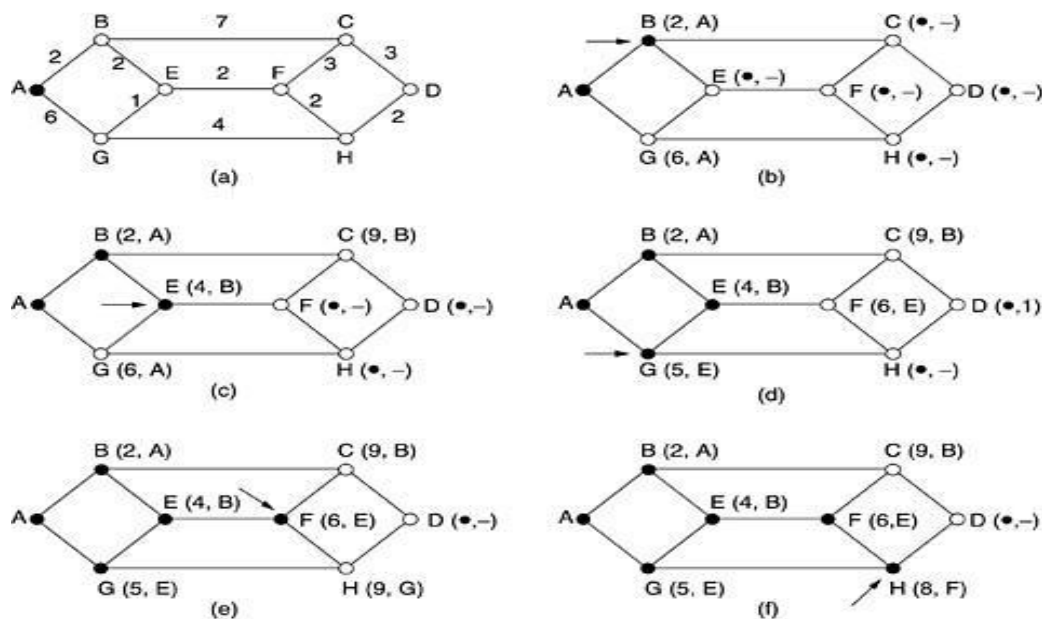
---

## Shortest Path Routing

Let us begin our study of feasible routing algorithms with a technique that is widely used in many forms because it is simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

The concept of a shortest path deserves some explanation. One way of measuring path length is the number of hops. Using this metric, the paths ABC and ABE in Fig. 5-7 are equally long. Another metric is the geographic distance in kilometers, in which case ABC is clearly much longer than ABE (assuming the figure is drawn to scale).

**Figure :** The first five steps used in computing the shortest path from A to D. The arrows indicate the working node.



However, many other metrics besides hops and physical distance are also possible. For example, each arc could be labeled with the mean queueing and transmission delay for some

standard test packet as determined by hourly test runs. With this graph labeling, the shortest path is the fastest path rather than the path with the fewest arcs or kilometers.

In the general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria or to a combination of criteria.

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959). Each node is labeled (in parentheses) with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Fig. 5-7(a), where the weights represent, for example, distance. We want to find the shortest path from A to D. We start out by marking node A as permanent, indicated by a filled-in circle. Then we examine, in turn, each of the nodes adjacent to A (the working node), relabeling each one with the distance to A. Whenever a node is relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later. Having examined each of the nodes adjacent to A, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-7(b). This one becomes the new working node.

We now start at B and examine all nodes adjacent to it. If the sum of the label on B and the distance from B to the node being considered is less than the label on that node, we have a shorter path, so the node is relabeled.

After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node with the

---

smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-7 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig. 5-7(c). At that point we have just made E permanent. Suppose that there were a shorter path than ABE, say AXYZ. There are two possibilities: either node Z has already been made permanent, or it has not been. If it has, then E has already been probed (on the round following the one when Z was made permanent), so the AXYZ path has not escaped our attention and thus cannot be a shorter path.

Now consider the case where Z is still tentatively labeled. Either the label at Z is greater than or equal to that at E, in which case AXYZ cannot be a shorter path than ABE, or it is less than that of E, in which case Z and not E will become permanent first, allowing E to be probed from Z.

This algorithm is given in Fig. 5-8. The global variables *n* and *dist* describe the graph and are initialized before shortest path is called. The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*. Since the shortest path from *t* to *s* in an undirected graph is the same as the shortest path from *s* to *t*, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, *path*, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

---

Dijkstra's algorithm to compute the shortest path through a graph.

```
#define MAX_NODES 1024          /* maximum number of nodes */
#define INFINITY 1000000000    /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                /* the path being worked on */
    int predecessor;            /* previous node */
    int length;                 /* length from source to this node */
    enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];

int i, k, min;
struct state *p;

for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;
do {
    for (i = 0; i < n; i++)
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }

    /* Find the tentatively labeled node with the smallest label. */
    k = 0; min = INFINITY;
    for (i = 0; i < n; i++)
        if (state[i].label == tentative && state[i].length < min) {
            min = state[i].length;
            k = i;
        }
    state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

---

## Experiment No. 4

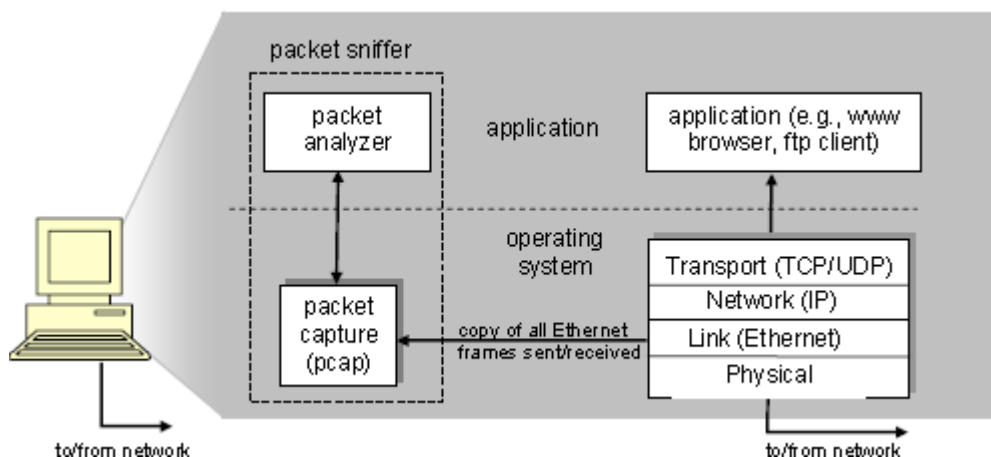
**Aim:** Use traffic monitoring tool Wireshark to observe network traffic with packet details.

### Theory:

#### **Packet Sniffer Structure**

The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures (“sniffs”) messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine.

Figure 10.1 shows the structure of a packet sniffer. At the right of Figure 10.1 are the protocols (In this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 10.1 is an addition to the usual software in the computer, and consists of two parts. The packet capture library receives a copy of every link-layer frame that is sent from or received by the computer. As the messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 10.1, the assumed physical media is an Ethernet, and so all upper layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives all messages sent/received from/by all protocols and applications executing in the computer.



**Figure 6.1: Packet sniffer structure**

The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must

“understand” the structure of all messages exchanged by protocols. For example, suppose user is interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 10.1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string “GET,” “POST,” or “HEAD,”

### **Wireshark packet sniffer**

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible. A network packet analyzer can be consider as a measuring device used to examine what's going on inside a network cable, just like a voltmeter is used by an electrician to examine what's going on inside an electric cable.

Wireshark allows user to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in the computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It is stable, has a large user base, rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, Token-Ring, FDDI, serial (PPP and SLIP), 802.11 wireless LANs, and ATM connections (if the OS on which it's running allows Wireshark to do so).

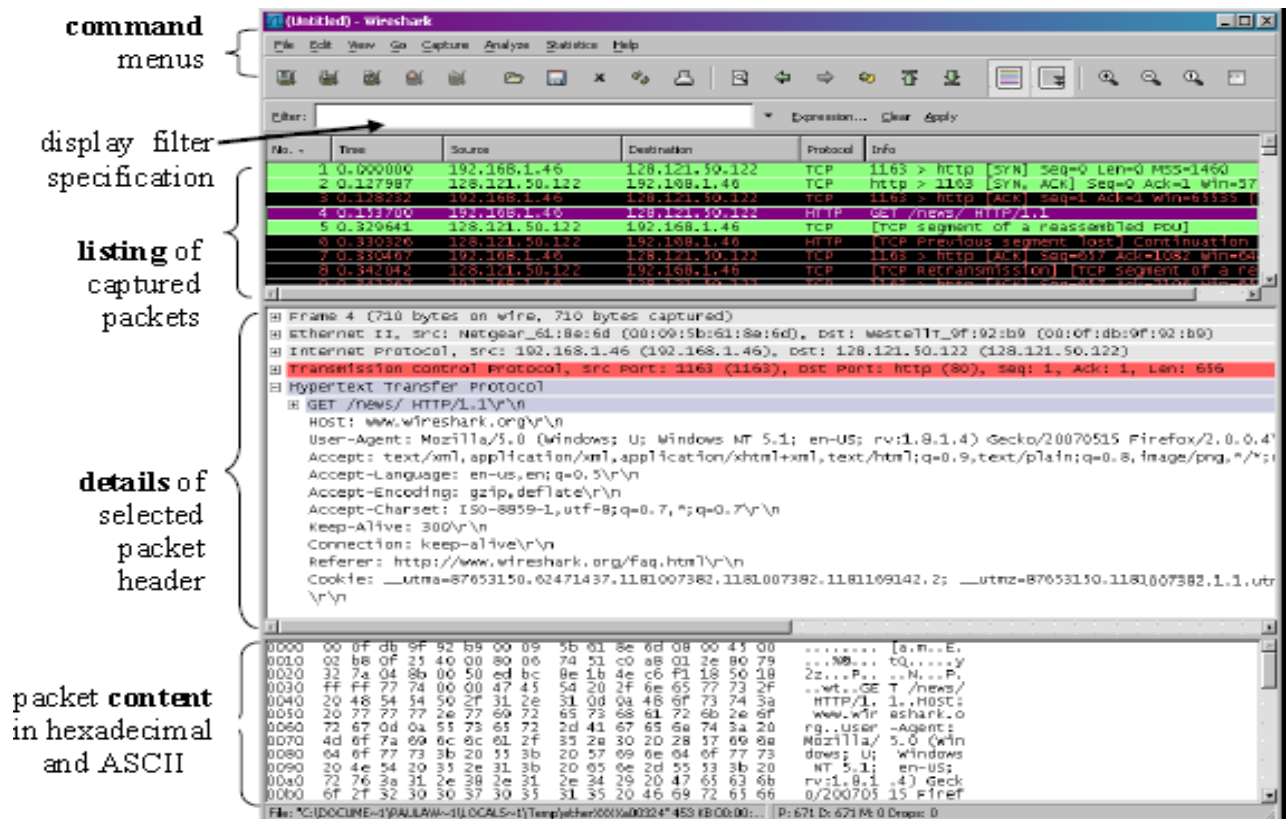
### **Getting Wireshark**

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the libpcap or WinPCap packet capture library.

### **Running Wireshark**



When the user runs the Wireshark program, the Wireshark graphical user interface shown in Figure 6.2 will be displayed. Initially, no data will be displayed in the various windows.



**Figure 6.2: Wireshark Graphical User Interface**

The Wireshark interface has five major components:

1. The command menus are standard pull down menus located at the top of the window. Out of which File and Capture menus are of interest. The File menu allows user to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows user to begin packet capture.
2. The packet-listing window displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
3. The packet-header details window provides details about the packet selected (highlighted) in the packet listing window. (To select a packet in the packet listing window, place the cursor over the packet's one-line summary in the packet listing window and click with the left mouse button.). These details include information about the

Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus-or-minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest level protocol that sent or received this packet are also provided.

4. The packet-contents window displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
5. Towards the top of the Wireshark graphical user interface, is the packet display filter field, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

### **Taking Wireshark for a Test Run.**

It is assumed that computer is connected to the Internet via a wired Ethernet interface. To run the Wireshark following steps are to be performed:

- Start up web browser, which will display the selected homepage.
- Start up the Wireshark software. User will initially see a window similar to that shown in Figure 10. 2, except that no packet data will be displayed in the packetlisting, packet-header, or packet-contents window, since Wireshark has not yet begun capturing packets.
- To begin packet capture, select the Capture pull down menu and select Options. This will cause the “Wireshark: Capture Options” window to be displayed, as shown in Figure 10.3.

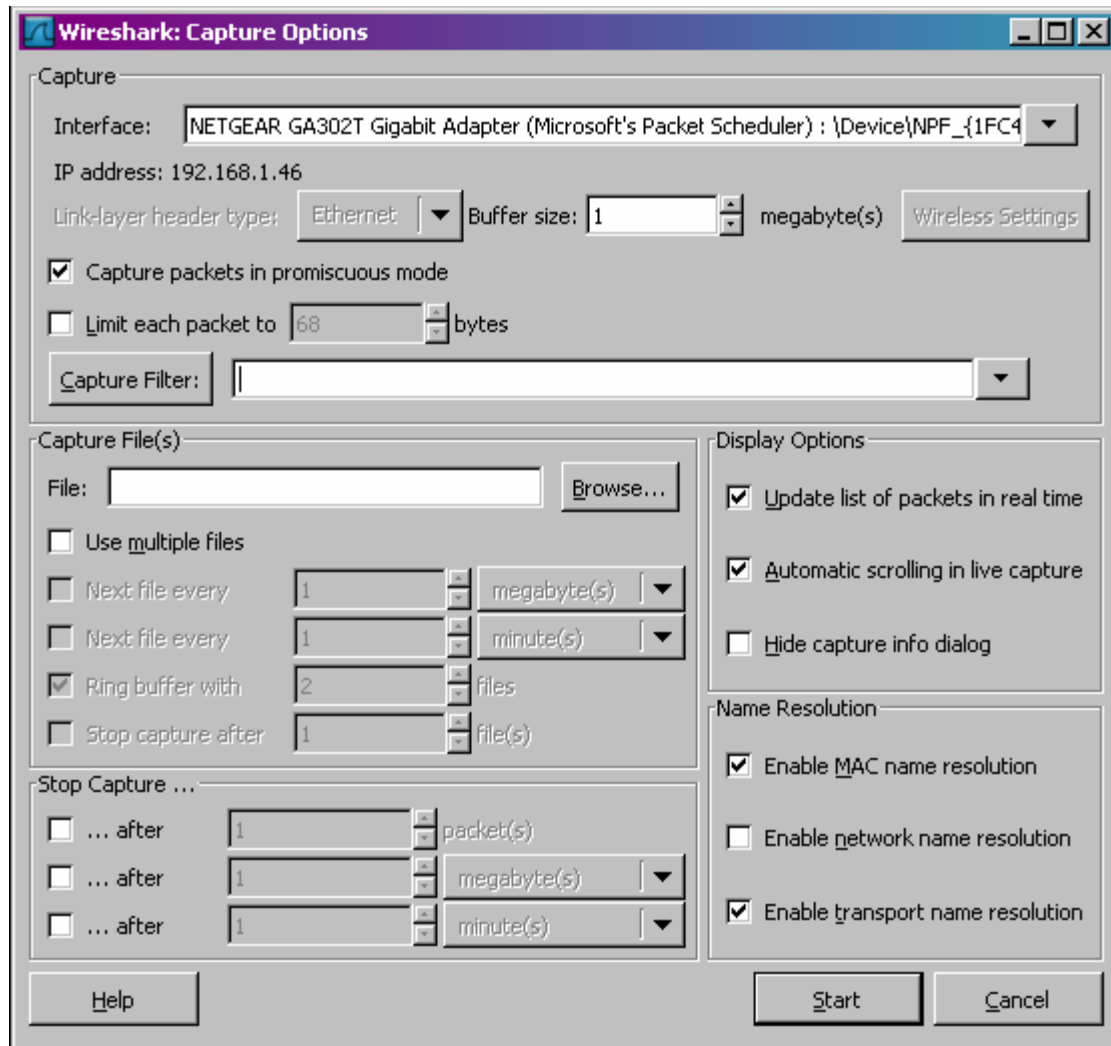
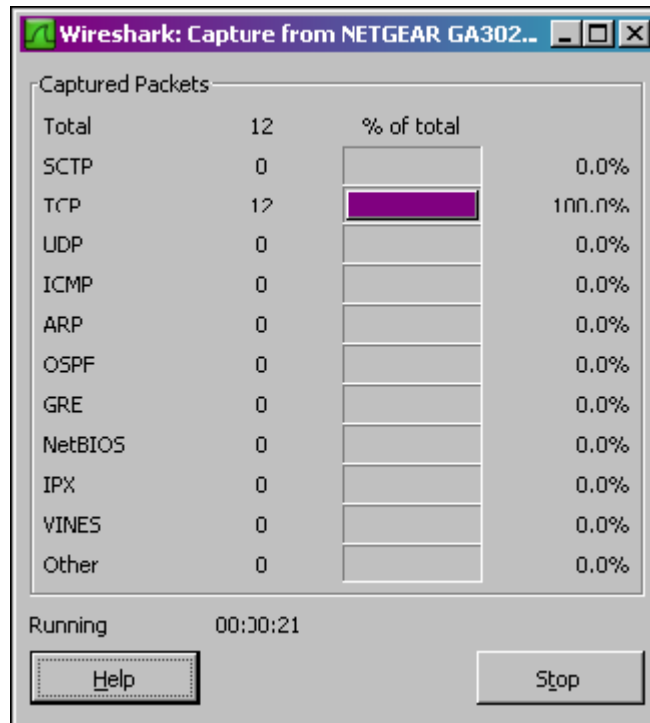


Figure 6.3: Wireshark Capture Options Window

- User can use most of the default values in this window, but uncheck “Hide capture info dialog” under Display Options. The network interfaces (i.e., the physical connections) that the computer has to the network will be shown in the Interface pull down menu at the top of the Capture Options window. In case if the computer has more than one active network interface (e.g., if user have both a wireless and a wired Ethernet connection), User will need to select an interface that is being used to send and receive packets (mostly likely the wired interface). After selecting the network interface (or using the default interface chosen by Wireshark), click Start. Packet capture will now begin - all packets being sent/received from/by the computer are now being captured by Wireshark!
- Once user begin packet capture, a packet capture summary window will appear, as shown in Figure 10.4. This window summarizes the number of packets of various types that are being captured, and (importantly!) contains the Stop button that will allow user to stop packet capture. Don't stop packet capture yet.



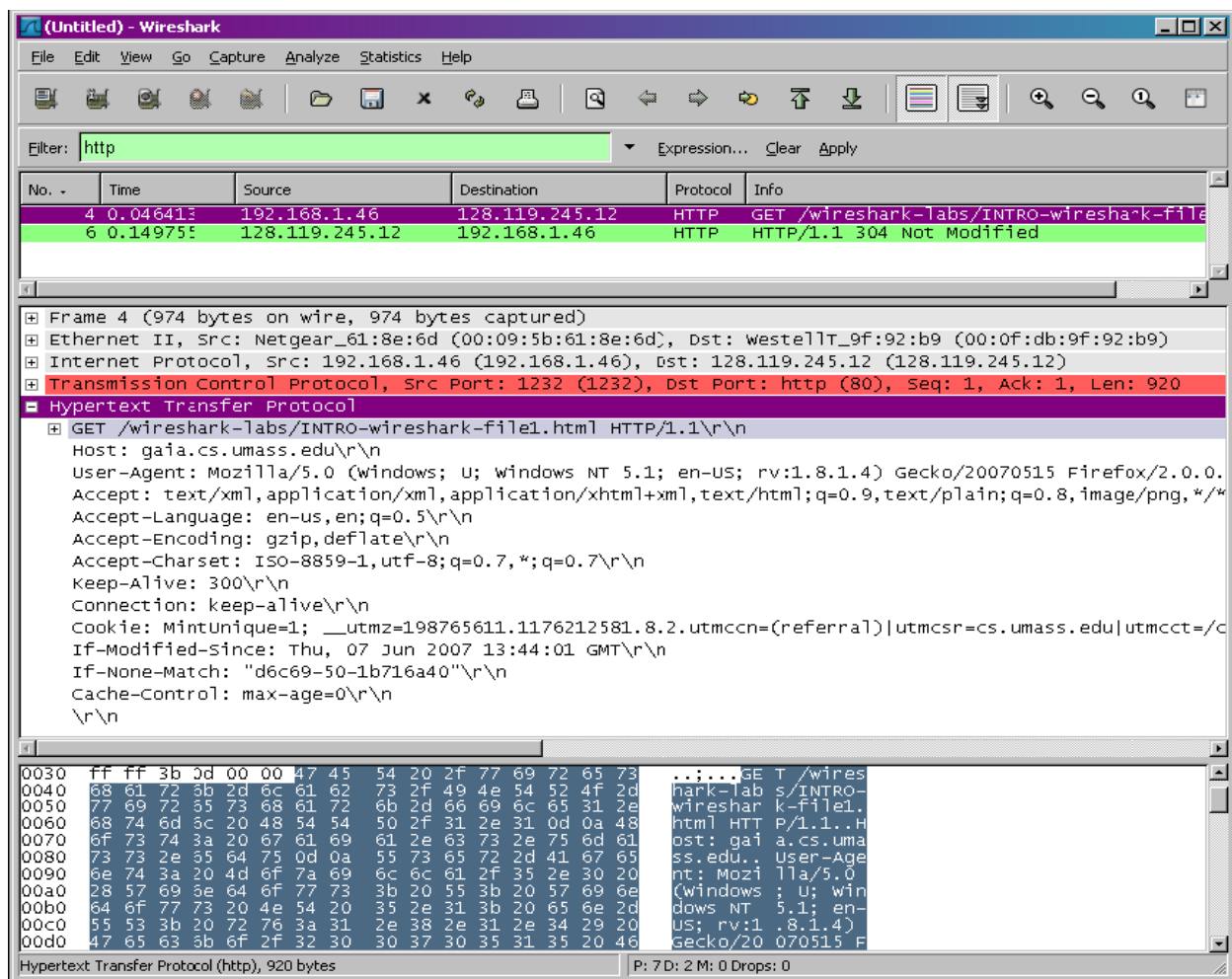
While Wireshark is running, enter the URL:  
<http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html>

and have that page displayed in user browser. In order to display this page, User browser will contact the HTTP server at [gaia.cs.umass.edu](http://gaia.cs.umass.edu) and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages will be captured by Wireshark.

- After the browser has displayed the [INTRO-wireshark-file1.html](http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html) page, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since user began packet capture. The main Wireshark window should now look similar to Figure 10.2. Now the live packet data is displayed that contains all protocol messages exchanged between user computer and other network entities! The HTTP message exchanges with the [gaia.cs.umass.edu](http://gaia.cs.umass.edu) web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the Protocol column in Figure 10.2). Even though the only action took was to download a web page, there were evidently many other protocols running on the computer that are unseen by the user.
- Type in “http” (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select Apply (to the right of where entered “http”). This will cause only HTTP message to be displayed in the packet-listing window.

---

- Select the first http message shown in the packet-listing window. This should be the HTTP GET message that was sent from user computer to the gaia.cs.umass.edu HTTP server. When HTTP GET message is select, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window<sup>3</sup>. By clicking plus and- minus boxes to the left side of the packet details window, minimize the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. Maximize the amount information displayed about the HTTP protocol. Now Wireshark display should look roughly as shown in Figure 10.5. (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).



## 10. Exit Wireshark

## Experiment No. 5

**Aim:** Configure network using Cisco Packet Tracer software and show packet transmission from source to destination.

**Theory:**

In this practical cisco packet tracer simulator is use to configure hub and switch. Analyse the operation of hubs and switches and identify the differnces between the transmission of packets in both the devices.

Cisco packet tracer:

Cisco Packet Tracer as the name suggests, is a tool built by Cisco. This tool provides a network simulation to practice simple and complex networks. The main purpose of Cisco Packet Tracer is to help users to learn the principles of networking with hands-on experience.

Key Features:

- Unlimited devices
- E-learning
- Customize single/multi user activities
- Interactive Environment
- Visualizing Networks
- Real-time mode and Simulation mode
- Self-paced
- Supports majority of networking protocols
- International language support
- Cross platform compatibility

Hub:

Hub is a very simple network device that is used in LANs. It is basically a multiport repeater. Hubs do not decide anything and forwards any traffic to all of the ports. So, they are not smart devices. They have multiple ports that connects different network equipments in the same network. But this devices the network bandwidth.

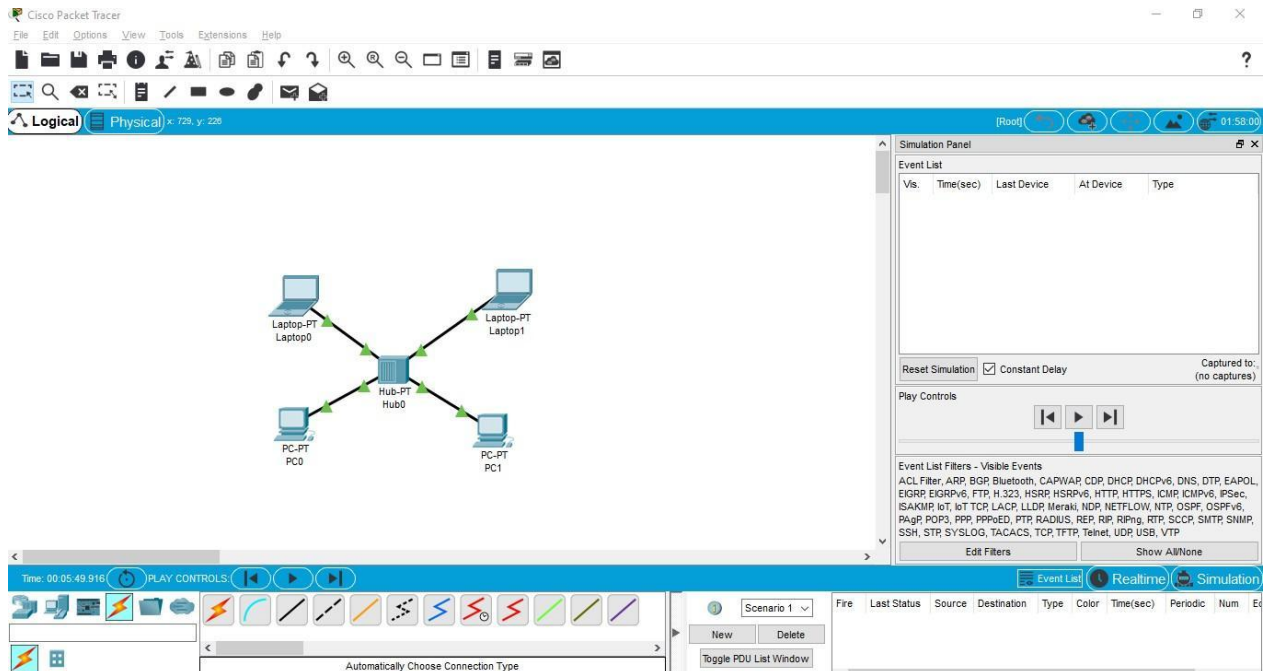
Hubs Operate at OSI Layer 1 Hubs are not MAC address aware. Whenever a frame is received on a port, it's flooded out to all the other ports apart from the one that it was received on.

Because of that, all hosts that are plugged in there are going to receive a frame, so they're going to have to process it at least as far as seeing that it's not for them.

Create the following network in Cisco Packet Tracer. And analyze the performance of hub in real time and simulation mode.

---





## Layer 2 Switch:

Layer 2 Switch is a network device that works in Layer 2 (Data Link Layer). It is a smart device that collects data and switches the traffic according to this data. Switches have many ports and with this characteristics, they expand the network through endpoints.

Switches, however, operate at Layer 2 of the OSI model. They've got physical ports on there.

This means that switches are MAC address aware. Whenever a frame is received, the switch will look at the source MAC address in the Layer 2 Ethernet header, and it will learn that MAC address. It will then add that MAC address to its MAC address table, which is a mapping between the MAC address and the port that is reachable on. If a unicast frame is later received where that MAC address is the destination, the switch will only send out the relevant port, unlike a hub that floods it out everywhere.

This is better for performance and security, as frames only go where they are required. Whenever a frame is received for the broadcast address or an unknown unicast address (it would be unknown because the switch hasn't learned about it yet), it will be flooded out all ports apart from the one it was received on.

Create the following network in Cisco Packet Tracer. Configure every device. And analyze the performance of switch in real time and simulation mode.

Cisco Packet Tracer

File Edit Options View Tools Extensions Help

Logical Physical x: 872, y: 257 [Root] 02:54:30

Simulation Panel

Event List

Vis.	Time(sec)	Last Device	At Device	Type
------	-----------	-------------	-----------	------

Reset Simulation ☒ Constant Delay Captured to: (no captures)

Play Controls

Event List Filters - Visible Events

ACL Filter, ARP, BGP, Bluetooth, CAPWAP, CDP, DHCP, DHCPv6, DNS, DTP, EAPOL, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPsec, ISAKMP, IoT, IoT TCR, LACP, LLDP, Meraki, NDR, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PPP, PPPoE, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDR, USB, VTP

Edit Filters Show All/None

Time: 00:05:47.294 PLAY CONTROLS

Scenario 0

New Delete

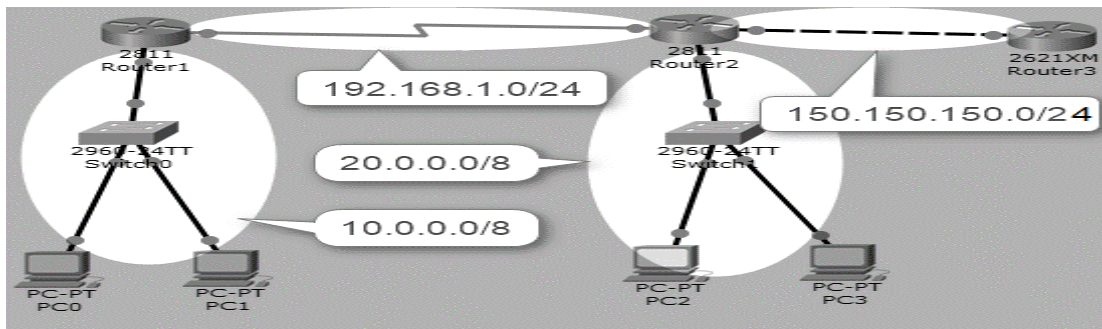
Toggle PDU List Window

Fire Last Status Source Destination Type Color Time(sec) Periodic Num Ex

## Experiment No. 6

**Aim:** Configure network using Distance vector routing protocol in Cisco Packet Tracer

**Theory:** Here, we will see communication enabled between PCs via Router in Packet Tracer. So, for this we need two PCs, a router, and two cross over cables to connect them. Important point is that we use cross over cable to connect PC to a router because they both use the same pins for transmission and receiving of data.



Sr. No.	Device	Interface	IP Address
1	Router1	Fa0/1	10.0.0.1/8
		S1/0	192.168.1.1/24
2	Router2	S1/0	192.168.1.2/24
		Fa0/0	20.0.0.1/8
		Fa0/1	150.150.150.1/24
3	Router3	Fa0/1	150.150.150.2/24
4	Switch1	N/A	N/A
5	Switch2	N/A	N/A
6	PC0	Fa0	10.0.0.2/8
7	PC1	Fa0	10.0.0.3/8
8	PC2	Fa0	20.0.0.2/8
9	PC3	Fa0	20.0.0.3/8

If you are using a simulator, such as Cisco Packet Tracer or GNS3, create the preceding topology and configure the devices as per the values mentioned in the following Table.

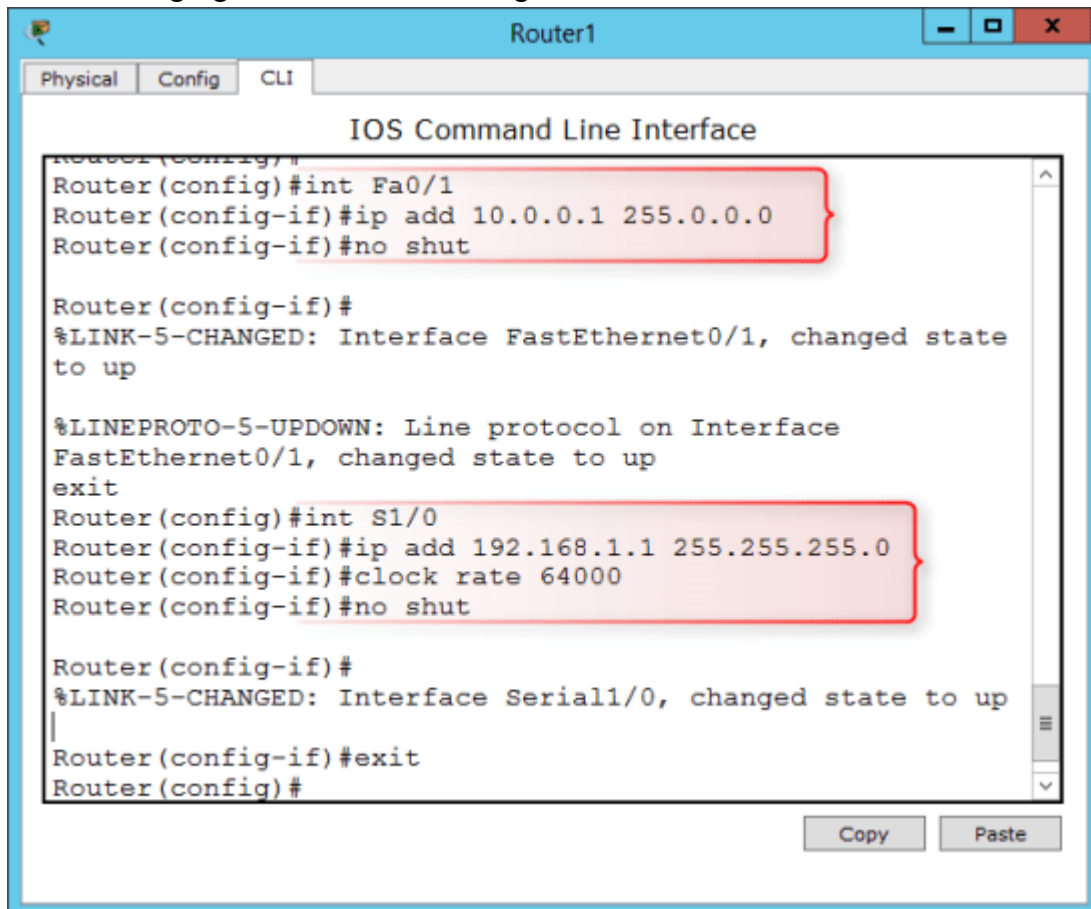
First of all, configure the IP addresses on each device. We assume that you know how to configure IP addresses. If you stuck in configuring IP addresses, click [here](#) to know how to configure IP address or you can refer the following example.

For example, to configure TCP/IP addresses on Router1, execute the following commands:

```
Router1(config)#interface fa0/1
Router1(config-if)#ip add 10.0.0.1 255.0.0.0
Router1(config-if)#no shut
Router1(config-if)#exit

Router1(config)#interface S1/0
Router1(config-if)#ip add 192.168.1.1 255.255.255.0
Router1(config-if)#clock rate 64000
Router1(config-if)#no shut
```

The following figure shows the IP configuration of Router1.



## ***Steps to Configure RIP Routing***

Once you have configured the appropriate IP addresses on each device, perform the following steps to configure RIP routing. The default version of RIP is RIPv1. In the later section, we will also configure RIPv2 routing.

1. On **Router1**, execute the following commands to configure **RIP** routing.

```
Router1(config)#router rip
Router1(config-router)#network 10.0.0.0
Router1(config-router)#network 192.168.1.0
Router1(config-router)#exit
```

2. On **Router2**, execute the following commands to configure **RIP** routing.

```
Router2(config)#router rip
Router2(config-router)#network 20.0.0.0
Router2(config-router)#network 192.168.1.0
Router2(config-router)#network 150.150.150.0
Router2(config-router)#exit Router2(config)#
```

3. On **Router3**, execute the following commands to configure **RIP** routing.

```
4. Router3(config)#router rip
5. Router3(config-router)#network 150.150.150.0
Router3(config-if)#exit
```

6. Once you have configured RIP routing protocol on each router, wait for a few seconds (let complete the convergence process), and then execute the **show ip route** command on any router to show the routing information.

```
Router(config)#do show ip route
```

7. In the following figure, you can see the routes learned by the RIP protocol on Router3.



```
Router3
Physical Config CLI
IOS Command Line Interface
Router#config t
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#router rip
Router(config-router)#network 150.150.150.0
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#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

R    20.0.0.0/8 [120/1] via 150.150.150.1, 00:00:22, FastEthernet0/1
     150.150.0.0/24 is subnetted, 1 subnets
C     150.150.150.0 is directly connected, FastEthernet0/1
R    192.168.1.0/24 [120/1] via 150.150.150.1, 00:00:22, FastEthernet0/1
Router#
```

## Verifying RIP Configuration

To verify and test the RIP configuration, perform the following steps:

1. To verify which routing protocol is configured, use the **show ip protocols** command.

```
Router#show ip protocols
```

2. To view the RIP messages being sent and received, use the **debug ip rip** command.

```
Router#debug ip rip
```

3. To stop the debugging process, use the **undebug all** command.

```
Router#undebug all
```

## Configure RIP Version 2 (RIPv2)

The configuration process of the RIPv2 protocol is similar to configuring RIPv1 protocol. To configure the RIPv2 routing protocol, you just need to type **version 2** command before

executing the **network** command. To configure the RIPv2 protocol, perform the following tasks on each router.

1. On Router1, execute the following commands:
2. Router1(config)#router rip  
Router1(config-router)#version 2
3. On Router2, execute the following commands:
4. Router2(config)#router rip  
Router2(config-router)#version 2
5. On Router3, execute the following commands:
6. Router3(config)#router rip  
Router3(config-router)#version 2
7. Once you have executed the preceding commands, execute the following command on each router, and verify the configuration as shown in the following figure.

```
Router#show ip protocols
```

## Removing RIP Routing Configuration

If you have added a wrong network or route, you can remove that network from the routing table. In this section, we will learn how to remove the routes learned by the RIP protocol. To do this, perform the following tasks.

On **Router1**, execute the following commands.

```
Router1(config)#router rip
Router1(config-router)#no network 10.0.0.0
Router1(config-router)#no network 192.168.1.0
Router1(config-router)#exit
```

On **Router2**, execute the following commands.

```
Router2(config)#router rip
Router2(config-router)#no network 20.0.0.0
Router2(config-router)#no network 192.168.1.0
Router2(config-router)#no network 150.150.150.0
Router2(config-router)#exit
```

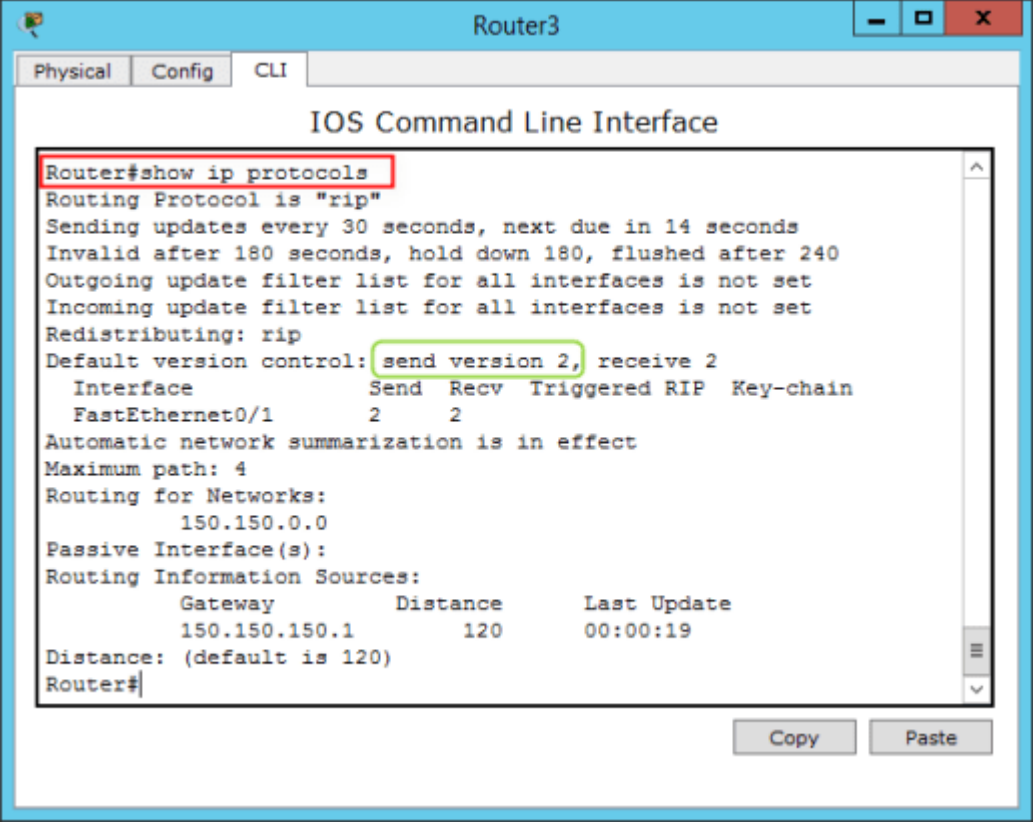
On **Router3**, execute the following commands.

```
Router3(config)#router rip
Router3(config-router)#no network 150.150.150.0
Router3(config-router)#exit
```





Now, execute the **show ip route** command and verify that the routes learned by the RIP routing protocol are deleted. If the routes are still available in the routing table, execute the **clear ip route \*** command.



The screenshot shows a window titled "Router3" with tabs for "Physical", "Config", and "CLI". The "CLI" tab is active, displaying the "IOS Command Line Interface". The command "Router#show ip protocols" has been entered and is highlighted with a red box. The output of the command is displayed below it, with "send version 2," highlighted in green. The output shows the following configuration details for the RIP protocol:

```
Router#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 14 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 2, receive 2
    Interface      Send Recv Triggered RIP Key-chain
    FastEthernet0/1    2      2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    150.150.0.0
  Passive Interface(s):
  Routing Information Sources:
    Gateway         Distance      Last Update
    150.150.150.1    120          00:00:19
  Distance: (default is 120)
Router#
```

At the bottom of the CLI window, there are "Copy" and "Paste" buttons.

## Experiment No. 7

**Aim:** Use Openssl command to perform Asymmetric key encryption(RSA) and also implement RSA algorithm.

### Theory:

#### PUBLIC KEY ENCRYPTION (RSA)

1. First we need to generate public-private key pair. Use the following command to do the same.

`openssl genrsa -out priv-key.pem`

This command generates the private key in file named priv-key.pem.

```
ubuntu@ubuntu-VB:~/Desktop/Crypto$ openssl genrsa -out priv-key.pem
Generating RSA private key, 512 bit long modulus
.....+++++
e is 65537 (0x10001)
```

We can see the contents of the above generated file.

```
ubuntu@ubuntu-VB:~/Desktop/Crypto$ cat priv-key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIBOgIBAAJBAJ5S+p8jM+lJyUfS64Ge+4m8DUMAuzIOypSP2y8o3YMD4nLEeWkq
QFHkLLEwvvh7AdxsCQsG4UPhtLt/Fh5m60CAwEAAQJBAJywStEMKednkWC/4cva
596ceh0BucIE2YDGDopRy0hi0Fn7QD81i7IvLq8YY2Qy4FD0UclHavI8uB6gYZKi
roECIQDRV6ldHLMbOTYjGbKZzBFOTMbonixQmFmZ4q0HAKnP+QIhAMGcZOjyZ4j4
rxJhzQev62UXRrB7jwToKkHE8400U35VAiBjxQWgrMbnmJKmk9680boexhSeVJQG
LTW2037gHn7P8QIgCENRdzOpQYpDhohQBMz4Qec9aBjN4Mq7yvTJ48p208UCIF01
WuLMZYmitYFav20TWPZNvQnIRdv00rtj/Z1U0A4m
-----END RSA PRIVATE KEY-----
```

2. Using this private key and the generated exponent, we can extract public key using the command:

`openssl rsa -pubout -in priv-key.pem -out pub-key.pem`

```
ubuntu@ubuntu-VB:~/Desktop/Crypto$ openssl rsa -pubout -in priv-key.pem -out pub-key.pem
writing RSA key
ubuntu@ubuntu-VB:~/Desktop/Crypto$ cat pub-key.pem
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAJ5S+p8jM+lJyUfS64Ge+4m8DUMAuzIO
ypSP2y8o3YMD4nLEeWkqQFHkLLEwvvh7AdxsCQsG4UPhtLt/Fh5m60CAwEAAQ==
-----END PUBLIC KEY-----
```

(This public key can be sent to anyone, who wants to send message to you.)

3. The above displayed files are in base64 encoding format and all the parameters are mixed. To view them separately, we need to use:

`openssl rsa -text -in priv-key.pem`

It displays modulus, public exponent, private exponent, prime1 and prime2 and some other parameters(out of scope) very clearly.

```

ubuntu@ubuntu-VB:~/Desktop/Crypto$ openssl rsa -text -in priv-key.pem
Private-Key: (512 bit)
modulus:
 00:9e:52:fa:9f:23:33:e8:a3:c9:47:d2:eb:81:9e:
 fb:89:bc:0d:43:00:bb:32:0e:ca:94:8f:db:2f:28:
 dd:83:1d:e2:72:c4:79:69:2a:40:51:e4:94:b1:30:
 7a:f6:e1:ec:07:71:b0:24:2c:1b:85:0f:86:d2:ed:
 fc:58:79:9b:ad
publicExponent: 65537 (0x10001)
privateExponent:
 00:9c:b0:4a:d1:0c:29:e7:67:91:60:bf:e1:cb:da:
 e7:de:9c:7a:13:81:b9:c2:04:d9:80:c6:0e:8a:51:
 cb:48:62:d0:59:fb:40:3f:35:8b:b2:2f:2e:af:18:
 63:64:32:e0:50:f4:51:c9:47:6a:f2:3c:b8:1e:a0:
 61:92:a2:ae:81
prime1:
 00:d1:57:a9:5d:1c:b3:1b:39:36:23:19:b2:99:cc:
 17:ce:4c:c6:e8:9e:2c:50:98:59:99:e2:a3:87:00:
 a9:cf:f9
prime2:
 00:c1:9c:64:e8:f2:67:88:f8:af:12:61:cd:07:af:
 eb:65:17:46:b0:7b:8f:04:e8:2a:41:c4:f3:83:8e:
 53:7e:55

```

4. To begin encryption, let's create a message file (Here, I have created file message.txt with the content "1 am in Cyber Security Workshop.")

```

ubuntu@ubuntu-VB:~/Desktop/Crypto$ cat message.txt
1 am in Cyber Security Workshop.

```

5. To encrypt the message, using public key, use the command:

```
openssl rsautl -encrypt -in message.txt -pubin -inkey pub-key.pem -out cipher.bin
```

```

ubuntu@ubuntu-VB:~/Desktop/Crypto$ openssl rsautl -encrypt -in message.txt -pubin -inkey
pub-key.pem -out cipher.bin

```

6. To decrypt the message, private key and cipher text is required.

```
openssl rsautl -decrypt -in cipher.bin -inkey priv-key.pem -out decrypted.txt
```

```

ubuntu@ubuntu-VB:~/Desktop/Crypto$ openssl rsautl -decrypt -in cipher.bin -inkey priv-key
.pem -out decrypted.txt

```

7. The decrypted text is same as encrypted one.

```
ubuntu@ubuntu-VB:~/Desktop/Crypto$ cat decrypted.txt  
I am in Cyber Security Workshop.
```

Thus we have successfully demonstrated the use of public key encryption.

---

## Experiment No. 8

**Aim:** Client server communication using socket programming

The term network programming refers to writing programs that execute across multiple devices computers, in which the devices are all connected to each other using a network. The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide

the low-level communication details, allows to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols:

TCP: TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

UDP: UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

### **Socket Programming:**

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to

a server.

When the connection is made, the server creates a socket object on its end of the communication.

The client and server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

- The server invokes the accept method of the ServerSocket class. This method waits until a client connects to the server on the given port.

- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.

- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.

- On the server side, the accept method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a twoway communication protocol, so data can be sent across both streams at the same time. There are following usefull classes providing complete set of methods to implement sockets.

---



The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests

### Program for server

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
    try{
ServerSocket ss=new ServerSocket(6665);
        Socket s=ss.accept();//establishes connection
        DataInputStream dis=new DataInputStream(s.getInputStream());
        String str=(String)dis.readUTF();
        System.out.println("message= "+str);
        ss.close();
    }catch(Exception e){System.out.println(e);}
}
```

### Program for client

```
import java.io.*; import
    java.net.*; public class
    MyClient {
public static void main(String[] args) {
    try{
Socket s=new Socket("localhost",6665);
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
        dout.writeUTF("Hello Student");
        dout.flush();
        dout.close();
        s.close();
    }catch(Exception e){System.out.println(e);}
}
```

### Sample Output:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

E:\Smita Kapse\21-22\CN\Practical\Socket>javac MyClient.java
E:\Smita Kapse\21-22\CN\Practical\Socket>java MyClient
E:\Smita Kapse\21-22\CN\Practical\Socket>
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

E:\Smita Kapse\21-22\CN\Practical\Socket>javac MyServer.java
E:\Smita Kapse\21-22\CN\Practical\Socket>java MyServer
message= Hello Student
E:\Smita Kapse\21-22\CN\Practical\Socket>
```



## Experiment No. 9

**Aim:** Study of NSG tool..

**Theory:**

### NSG 2.1 - THE TCL SCRIPT GENERATOR

NS2 Scenarios Generator (NSG) is a tcl script generator tool used to generate TCL Scripts automatically . . . !!!

NSG is a Java based tool that runs on any platform and can generate TCL Scripts for Wired as well as Wireless Scenarios for Network Simulator - 2. The procedure to execute these TCL Scripts on NS-2 is same as those of manually written TCL Scripts.

Some of the main features of NS2 Scenarios Generator (NSG) are as mentioned below:

- (1) Creating Wired and Wireless nodes just by drag and drop.
- (2) Creating Simplex and Duplex links for Wired network.
- (3) Creating Grid, Random and Chain topologies.
- (4) Creating TCP and UDP agents. Also supports TCP Tahoe, TCP Reno, TCP New-Reno and TCP Vegas.
- (5) Supports Ad Hoc routing protocols such as DSDV, AODV, DSR and TORA.
- (6) Supports FTP and CBR applications.
- (7) Supports node mobility.
- (8) Setting the packet size, start time of simulation, end time of simulation, transmission range and interference range in case of wireless networks, etc.
- (9) Setting other network parameters such as bandwidth, etc for wireless scenarios.

### Launch NSG2 :

To execute NSG2, you have to install JAVA6.0 first. You can download JAVA6.0 NSG2 doesn't need to be installed in your computer. You just download it and launch it with following instruction under TERMINAL command environment.

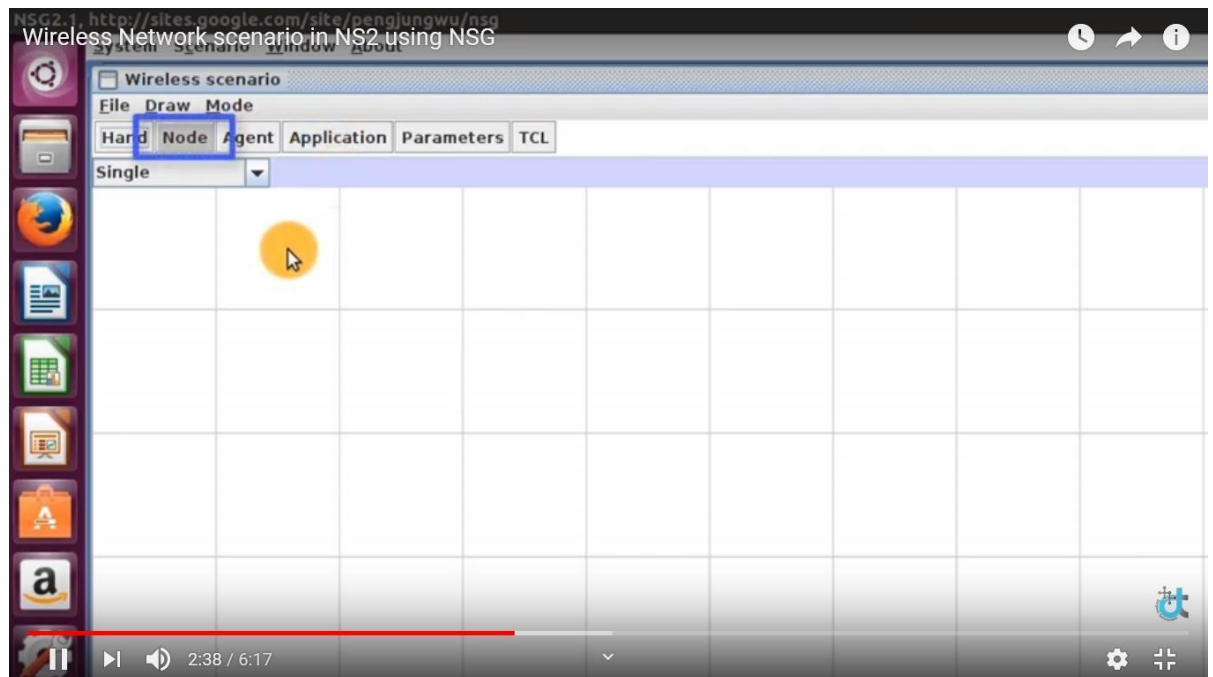
To Launch NSG2 execute following instructions:

- open terminal
- change directory into the folder where NSG2.1.jar is copied.
- type this command  
***java -jar NSG2.1.jar***
- it will directly open NSG tool

Steps for creation of scenario

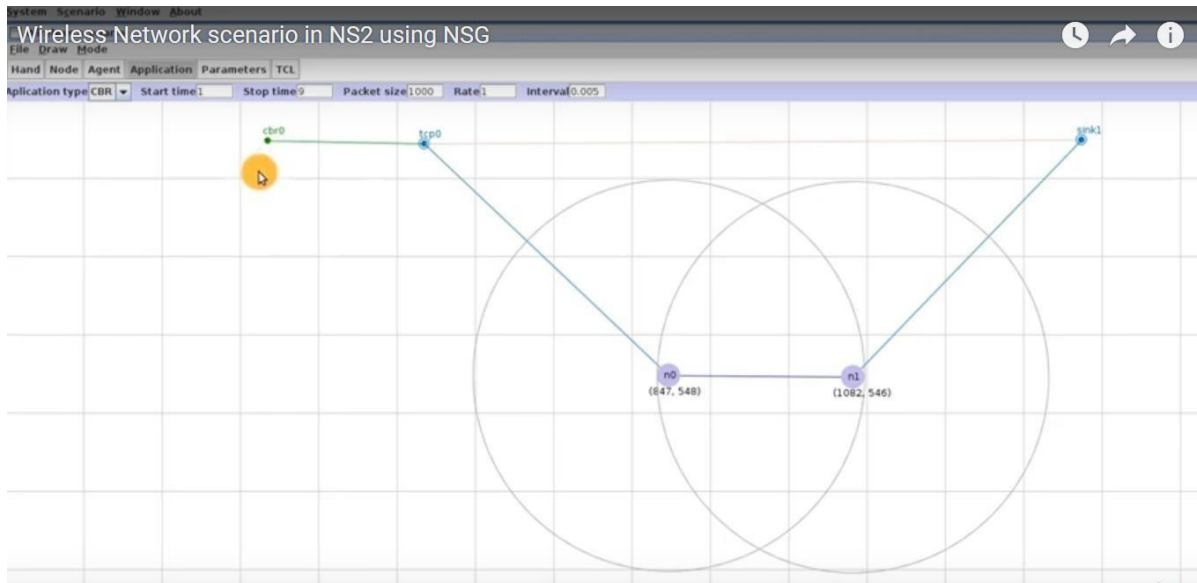
---

1. go to node tab □ set no and in the range of n0 click n1



2. Go to agent tab
  - Select TCP agent
  - Select node n0 □ click outside of it □ it will attach TCP agent to node n0
  - Selectn TCP sink
  - Select node n1 □ click outside of it □ it will attach TCP sink agent to node n1
  - Connect TCP and TCP sink
3. Go to application tab
  - Select CBR traffic
  - Set stop time ex:9
  - Select TCP agent□ attach cbr traffic to TCP agent

---



4. Go to parameter tab
  - Set simulation time
  - Trace file name
  - Nam file name
5. Click wireless tab
  - Change routing protocol Ex: AODV
6. Click channel tab set other parameters according to requirement
7. Click TCL tab
  - It will generate TCL script which contain TCL code for our scenario
  - Save this file with suitable name ex: aodv.tcl
  - Go to terminal □ type ns aodv.tcl
  - It will display output in nam animator

---

