DEPARTMENT OF CSE

****LAB MANUAL****

****ACADEMIC YEAR: 2022-23****

***(2019 Batch - Regulation 2020)***

****Programme(UG/PG) : UG****

****Semester : III B.Tech., I Semester****

****Course Code: CS3105****

****Course Title : COMPUTER NETWORKS LAB****

****Prepared By****

****Dr. Syed Sadat Ali,****

****Professor,****

****Dept. of CSE, NCET.****



**NIMRA COLLEGE OF ENGINEERING & TECHNOLOGY(23)**

**NIMRA NAGAR, JUPUDI, IBRAHIMPATNAM-521 456.**

**(Approved by AICTE, Recognized by Govt. of A.P. Affliated to JNTUK, Kakinada.)**

**INTERNAL ASSESSMENT MARK SPLIT UP**

**Observation (Day to Day work) : 10 Marks**

**Record: 5 Marks**

1. **GENERAL INSTRUCTIONS**

**1.1 General discipline in the lab**

* + Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
  + Students will not be allowed to enter late in the lab.
  + Students will not leave the class till the period is over.
  + Students should come prepared for their experiment.
  + Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
  + Students must get the connection of the hardware setup verified before switching on the power supply.
  + Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
  + Violating the above code of conduct may attract disciplinary action.
  + Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.

**1.2 Attendance**

* Attendance in the lab class is compulsory.
* Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
* On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**

**1.3 Preparation and Performance**

* Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self study reference is provided on LMS.
* Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
* Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
* Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
* Refer **Annexure 1** for Lab Report Format

1. **LIST OF EXPERIMENTS**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Experiment** | **Page No.** |
| 1. | Study of Network devices in detail and connect the computers in Local Area Networks. | 6 |
| 2. | Write a Program to implement the data link layer framing methods such as ( i ) Character Stuffing ( ii ) Bit Stuffing. | 11, 14 |
| 3. | Write a Program to implement the data link layer framing methods Checksum. | 16 |
| 4. | Write a Program for Hamming Code Generation for error deduction and correction. | 20 |
| 5. | Write a Program to implement on a data set of characters the three CRC polynomials - CRC 12, CRC 16 and CRC CCIP. | 22 |
| 6 | Write a Program to implement Slidinng window protocol for Goback N . | 25 |
| 7. | Write a Program to implement Slidinng window protocol for Selective repeat. | 28 |
| 8. | Write a Program to implement Stop and Wait Protocol. |  |
| 9. | Write a program for congestion control using leaky bucket algorithm. |  |
| 10. | Write a program to implement Dijkstra’s algorithm to compute Shortes path through a graph. | 32 |
| 11. | Write a Program to implement Distance vector routing algorithm buy obtaining routing table at each node. (Take an example subnet grapoh with weights indicating delay between nodes). | 35 |
| 12. | Write a Program to implement Broadcast tree by taking subnet of nosts. | 38 |
| 13. | Wireshark   1. Packet Captuire Using wire shark. 2. Starting Wire Shark. 3. Viewing Captured Traffic. 4. Analysis and Statistics & Filters. | 41  47  51  52 |
| 14. | How to run Nimap scan. | 55 |
| 15. | Operating System Detection using Nmap. | 67 |
| 16. | Do the following using NS2 Simulator.  i . NS2 Simulator-Introduction.  ii . Simulate to Find the Number of Packets Dropped.  iii . Simulate to Find the Numnber of Packets Dropped by TCP/UDP.  iv . Simulate to Find the Numnber of Packets Dropped to Congestion.  v . Simulate to Compare Data Rate & Throughput. |  |

|  |  |
| --- | --- |
| **Marks Distribution** | |
| **Continuous Evaluation (15 Marks)** | **End Semester Exam (35 Marks)** |
| **Each experiment shall be evaluated for 10 marks and at the end of the semester proportional marks shall be awarded out of 10.** | **End semester practical exam conduction and evaluation will be conducted by the External Examiner appointed by JNTUK, Kakinada..** |
| **Following is the breakup of 10 marks for each**  **4 Marks: Observation & conduct of experiment. Teacher may ask questions about experiment.**  **3 Marks: For report writing**  **3 Marks: For the 15 minutes quiz to be conducted in every lab.** |

# 

# EXPERIMENT NO: 1

**(1 a)**

## NAME OF THE EXPERIMENT**:** Study of Network devices in details and connect the

## Computers in Local Area Network.

**OBJECTIVE: Study of Network devices used in LAN**

**RESOURCE : Computer Networks Text Book by Stallings.**

**Here is the common network device list:**

* Hub.
* Switch.
* Router.
* Bridge.
* Gateway.
* Modem.
* Repeater.
* Access Point.

## IMG_256

1. **Switch:**A **network switch** or **switching hub** is a computer networking device that connects network segments.The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.
2. **Router:** A **router** is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

## Hub: An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator

is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

1. **Bridge:** A **network bridge** connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term *bridge* formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. *Switch* or *Layer 2 switch* is often used interchangeably with *bridge*.Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.
2. **Gate Way:** In a communications network, a network node equipped for interfacing with

another network that uses different protocols.

* + A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.
  + A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

1. **Repeater:**Functioning at Physical Layer.A **repeater** is an electronic device that receives

a signal and retransmits it at a higher level and/or higher power, or onto the other side of

an obstruction, so that the signal can cover longer distances. Repeater have two ports, so

cannot be use to connect for more than two devices

## **Modem:** Modems (modulators-demodulators) are used to transmit digital signals over analog telephone lines. Thus, digital signals are converted by the modem into analog signals of different frequencies and transmitted to a modem at the receiving location. The receiving modem performs the reverse transformation and provides a digital output to a device connected to a modem, usually a computer. The digital data is usually transferred to or from the modem over a serial line through an industry standard interface, RS-232. Many telephone companies offer DSL services, and many cable operators  use modems as end terminals for identification and recognition of home and personal users. Modems work on both the Physical and Data Link layers.

## **Repeater:** A repeater is an electronic device that amplifies the signal it receives. You can think of repeater as a device which receives a signal and retransmits it at a higher level or higher power so that the signal can cover longer distances, more than 100 meters for standard LAN cables. Repeaters work on the Physical layer.

## **Access Point:** While an access point (AP) can technically involve either a wired or wireless connection, it commonly means a wireless device. An AP works at the second OSI layer, the Data Link layer, and it can operate either as a bridge connecting a standard wired network to wireless devices or as a router passing data transmissions from one access point to another.

Wireless access points (WAPs) consist of a transmitter and receiver (transceiver) device used to

create a wireless LAN (WLAN). Access points typically are separate network devices with a built-

in antenna, transmitter and adapter. APs use the wireless infrastructure network mode to provide a

connection point between WLANs and a wired Ethernet LAN. They also have several ports, giving

you a way to expand the network to support additional clients. Depending on the size of the network,

one or more APs might be required to provide full coverage. Additional APs are used to allow

access to more wireless clients and to expand the range of the wireless network.

Each AP is limited by its transmission range — the distance a client can be from an AP and still

obtain a usable signal and data process speed. The actual distance depends on the wireless standard,

the obstructions and environmental conditions between the client and the AP. Higher end APs have

high-powered antennas, enabling them to extend how far the wireless signal can travel.

APs might also provide many ports that can be used to increase the network’s size, firewall

capabilities and Dynamic Host Configuration Protocol (DHCP) service. Therefore, we get APs that

are a switch, DHCP server, router and firewall.

To connect to a wireless AP, you need a service set identifier (SSID) name. 802.11 wireless

networks use the SSID to identify all systems belonging to the same network, and client stations

must be configured with the SSID to be authenticated to the AP. The AP might broadcast the SSID,

allowing all wireless clients in the area to see the AP’s SSID. However, for security reasons, APs

can be configured not to broadcast the SSID, which means that an administrator needs to give client

systems the SSID instead of allowing it to be discovered automatically. Wireless devices ship with

default SSIDs, security settings, channels, passwords and usernames. For security reasons, it is

strongly recommended that you change these default settings as soon as possible because many

internet sites list the default settings used by manufacturers.

Access points can be fat or thin. Fat APs, sometimes still referred to as autonomous APs, need to be

manually configured with network and security settings; then they are essentially left alone to serve

clients until they can no longer function. Thin APs allow remote configuration using a controller.

Since thin clients do not need to be manually configured, they can be easily reconfigured and

monitored. Access points can also be controller-based or stand-alone.

## **Conclusion:** Having a solid understanding of the types of network devices available can help you design and built a network that is secure and serves your organization well. However, to ensure the ongoing security and availability of your network, you should carefully [monitor your network devices](https://www.netwrix.com/network_auditing_software_features.html?itm_source=blog&itm_medium=context&itm_campaign=network&itm_content=none&cID=70170000000kgEZ" \t "https://blog.netwrix.com/2019/01/08/network-devices-explained/_blank) and activity around them, so you can quickly spot hardware issues, configuration issues and attacks.

## NAME OF THE EXPERIMENT**:** Connect the Computers in Local Area Network.

**OBJECTIVE: Study of Network devices used in LAN**

**RESOURCE : Computer Networks Text Book by Stallings.**

**Aim:** Connect the computers in Local Area Network.

## Procedure: On the host computer

On the host computer, follow these steps to share the Internet connection:

1. Log on to the host computer as Administrator or as Owner.
2. Click **Start**, and then click **Control Panel**.

## Click Network and Internet Connections.

1. Click **Network Connections**.
2. Right-click the connection that you use to connect to the Internet. For example, if you

connect to the Internet by using a modem, right-click the connection that you want

Under Dial-up / other network available.

1. Click **Properties**.
2. Click the **Advanced** tab.
3. Under **Internet Connection Sharing**, select the **Allow other network users to**

**Connect through this computer's Internet connection** check box.

1. If you are sharing a dial-up Internet connection, select the **Establish a dial-up**

**connection whenever a computer on my network attempts to access the Internet**

check box if you want to permit your computer to automatically connect to the

Internet.

1. Click **OK**. You receive the following message:

When Internet Connection Sharing is enabled, your LAN adapter will be set to use IP address

192.168.0.1. Your computer may lose connectivity with other computers on your network. If

these other computers have static IP addresses, it is a good idea to set them to obtain their IP

addresses automatically. Are you sure you want to enable Internet Connection Sharing?

1. Click **Yes**.

The connection to the Internet is shared to other computers on the local area network (LAN).

The network adapter that is connected to the LAN is configured with a static IP

address of 192.168.0.1 and a subnet mask of 255.255.255.0

## On the client computer

To connect to the Internet by using the shared connection, you must confirm the LAN adapter IP configuration, and then configure the client computer. To confirm the LAN adapter IP configuration, follow these steps:

1. Log on to the client computer as Administrator or as Owner.
2. Click **Start**, and then click **Control Panel**.

## Click Network and Internet Connections.

1. Click **Network Connections**.
2. Right-click **Local Area Connection** and then click **Properties**.
3. Click the **General** tab, click **Internet Protocol (TCP/IP)** in the **connection uses**

**the following items** list, and then click **Properties**.

1. In the **Internet Protocol (TCP/IP) Properties** dialog box, click **Obtain an IP**

**address automatically** (if it is not already selected), and then click **OK**.

**Note:** You can also assign a unique static IP address in the range of 192.168.0.2

to 192.168.0.254. For example, you can assign the following static IP

address, subnet mask, and default gateway:

8. IP Address 192.168.31.202

9. Subnet mask 255.255.255.0

1. Default gateway 192.168.31.1
2. In the **Local Area Connection Properties** dialog box, click **OK**.
3. Quit Control Panel.

# EXPERIMENT NO: 2

**(2a)**

## NAME OF THE EXPERIMENT: Bit Stuffing.

**OBJECTIVE**: Implement the data link layer framing method.

**RESOURCE:** Turbo C

## PROGRAM LOGIC: The new technique allows data frames to contain an arbitrary number if bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with special bit pattern, 01111110, called a flag byte. Whenever the sender’s data link layer encounters five consecutive ones in the data , i t a u tomatical ly stuffs a 0 bit into the out going bit stream. This bit stuffing is analogous to c haracter stuf fin g, in w hich a D LE i s stuf fed into the out going character stream before DLE in the data.

## SOURCE CODE:

// BIT Stuffing program #include<stdio.h> #include<conio.h> #include<string.h>

void main()

{

int a[20],b[30],i,j,k,count,n; clrscr();

printf("Enter frame length:"); scanf("%d",&n);

printf("Enter input frame (0's & 1's only):"); for(i=0;i<n;i++)

scanf("%d",&a[i]); i=0;

count=1; j=0;

while(i<n)

{

if(a[i]==1)

{

b[j]=a[i];

for(k=i+1;a[k]==1 && k<n &&count<5;k++)

{

j++;

b[j]=a[k]; count++; if(count==5)

{

j++; b[j]=0;

}

i=k;

}

}

else

{

b[j]=a[i];

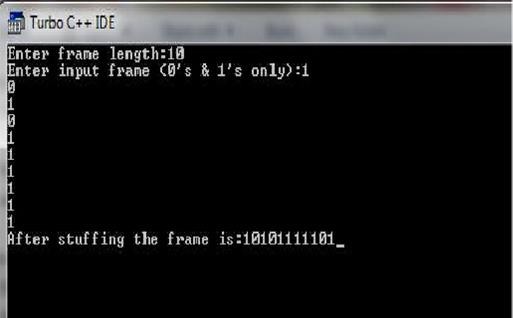
} i++; j++;

}

printf("After stuffing the frame is:"); for(i=0;i<j;i++)

printf("%d",b[i]); getch();

} OUTPUT:



## Viva questions:

1. What is Stuffing?
2. What is use of Stuffing?
3. With bit stuffing the boundary between two frames can be unambiquously recognize by?
4. is a analogous to character stuffing?
5. The senders data link layer encounters no of 1’s consecutively

**EXPERIMENT NO: 2**

## 2(b)

**NAME OF THE EXPERIMENT:** Character Stuffing. **OBJECTIVE:** Implement the data link layer framing methods. **RESOURCE:** Turbo C

## PROGRAM LOGIC:

The framing method gets around the problem of resynchronization after an error by ha vi ng ea ch fra me sta r t with th e ASCII c ha ra cter s equ en ce D L E ST X a nd the sequence DLE ETX. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE STX or DLE ETX characters to figure out. The data link la yer on t he recei vin g en d re mo ve s th e D LE b efore th e da ta a re giv en t o th e network layer. This technique is called character stuffing.

**PROCEDURE:** Go to debug -> run or press CTRL + F9 to run the program.

## SOURCE CODE:

//PROGRAM FOR CHARACTER STUFFING

#include<stdio.h> #include<conio.h> #include<string.h> #include<process.h> void main()

{

int i=0,j=0,n,pos;char a[20],b[50],ch; clrscr();

printf("enter string\n"); scanf("%s",&a); n=strlen(a);

printf("enter position\n"); scanf("%d",&pos); if(pos>n)

{

printf("invalid position, Enter again :"); scanf("%d",&pos);

}

printf("enter the character\n"); ch=getche();

b[0]='d';

b[1]='l';

b[2]='e';

b[3]='s';

b[4]='t';

b[5]='x'; j=6;

while(i<n)

{

if(i==pos-1)

{

b[j]='d';

b[j+1]='l';

b[j+2]='e';

b[j+3]=ch; b[j+4]='d';

b[j+5]='l';

b[j+6]='e';

j=j+7;

}

if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')

{

b[j]='d';

b[j+1]='l';

b[j+2]='e';

j=j+3;

}

b[j]=a[i]; i++;

j++;

}

b[j]='d';

b[j+1]='l';

b[j+2]='e';

b[j+3]='e';

b[j+4]='t';

b[j+5]='x';

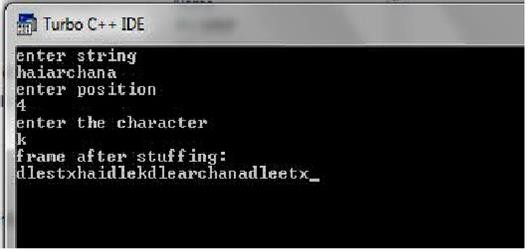
b[j+6]='\0';

printf("\nframe after stuffing:\n"); printf("%s",b);

getch();

}

**OUTPUT:**



## Viva Questions:

* 1. What is Character stuffing?
  2. What is the use of character stuffing?
  3. \_ \_ \_ \_ \_ \_are the delimiters for the character stuffing?
  4. Expand DLE STX?
  5. Expand DLE ETX?

# EXPERIMENT NO: 3

**NAME OF THE EXPERIMENT:** Program to implement data link layer framing method checksum.

**OBJECTIVE:**The [Checksum](https://www.geeksforgeeks.org/error-detection-code-checksum/) is an error detection method that detected errors in data/message while it is transmitted from sender to receiver. This method is used by the higher layer protocols and makes use of the Checksum Generator on the Sender side and Checksum Checker on the Receiver side.

**RESOURCE:** Turbo C

# PROGRAM LOGIC or APPROACH : The given problem can be divided into two

# following parts:

* Generating the [Checksum](https://www.geeksforgeeks.org/error-detection-in-computer-networks/) value of the **sender’s message** can be done using the following

steps:

* + - * Divide the message into the [binary strings](https://www.geeksforgeeks.org/tag/binary-string/) of the given **block size**.
      * All the [binary strings are added together](https://www.geeksforgeeks.org/program-to-add-two-binary-strings/) to get the **sum**.
      * The [One’s Complement of the binary string](https://www.geeksforgeeks.org/1s-2s-complement-binary-number/) representing the **sum** is the required

checksum value.

* Check if the value of the **received message** (i.e, **rec\_message + senders\_checksum**) is equal to**0**.
  + - * The checksum of the received message can be calculated similarly to the checksum

calculated in the above process.

* + - * If the checksum value is **0**, the message is transmitted properly with no errors

otherwise, some error has occurred during the transmission.

Below is the implementation of the above approach:

|  |
| --- |
| // C++ implementation of the above approach  #include <bits/stdc++.h>  **using** **namespace** std;    // Function to find the One's complement  // of the given binary string  string Ones\_complement(string data)  {  **for** (**int** i = 0; i < data.length(); i++) {  **if** (data[i] == '0')              data[i] = '1';  **else**              data[i] = '0';      }    **return** data;  }    // Function to return the checksum value of  // the give string when divided in K size blocks  string checkSum(string data, **int** block\_size)  {      // Check data size is divisible by block\_size      // Otherwise add '0' front of the data  **int** n = data.length();  **if** (n % block\_size != 0) {  **int** pad\_size = block\_size - (n % block\_size);  **for** (**int** i = 0; i < pad\_size; i++) {              data = '0' + data;          }      }        // Binary addition of all blocks with carry      string result = "";        // First block of data stored in result variable  **for** (**int** i = 0; i < block\_size; i++) {          result += data[i];      }        // Loop to calculate the block      // wise addition of data  **for** (**int** i = block\_size; i < n; i += block\_size) {            // Stores the data of the next bloack          string next\_block = "";    **for** (**int** j = i; j < i + block\_size; j++) {              next\_block += data[j];          }            // Stores the binary addition of two blocks          string additions = "";  **int** sum = 0, carry = 0;            // Loop to calculate the binary addition of          // the current two blocls of k size  **for** (**int** k = block\_size - 1; k >= 0; k--) {              sum += (next\_block[k] - '0')                     + (result[k] - '0');              carry = sum / 2;  **if** (sum == 0) {                  additions = '0' + additions;                  sum = carry;              }  **else** **if** (sum == 1) {                  additions = '1' + additions;                  sum = carry;              }  **else** **if** (sum == 2) {                  additions = '0' + additions;                  sum = carry;              }  **else** {                  additions = '1' + additions;                  sum = carry;              }          }            // After binary add of two blocks with carry,          // if carry is 1 then apply binary addition          string final = "";    **if** (carry == 1) {  **for** (**int** l = additions.length() - 1; l >= 0;                   l--) {  **if** (carry == 0) {                      final = additions[l] + final;                  }  **else** **if** (((additions[l] - '0') + carry) % 2                           == 0) {                      final = "0" + final;                      carry = 1;                  }  **else** {                      final = "1" + final;                      carry = 0;                  }              }                result = final;          }  **else** {              result = additions;          }      }        // Return One's complements of result value      // which represents the required checksum value  **return** Ones\_complement(result);  }    // Function to check if the received message  // is same as the senders message  **bool** checker(string sent\_message,               string rec\_message,  **int** block\_size)  {        // Checksum Value of the senders message      string sender\_checksum          = checkSum(sent\_message, block\_size);        // Checksum value for the receivers message      string receiver\_checksum = checkSum(          rec\_message + sender\_checksum, block\_size);        // If receivers checksum value is 0  **if** (count(receiver\_checksum.begin(),                receiver\_checksum.end(), '0')          == block\_size) {  **return** **true**;      }  **else** {  **return** **false**;      }  }    // Driver Code  **int** main()  {      string sent\_message          = "10000101011000111001010011101101";      string recv\_message          = "10000101011000111001010011101101";  **int** block\_size = 8;    **if** (checker(sent\_message,                  recv\_message,                  block\_size)) {          cout << "No Error";      }  **else** {          cout << "Error";      }    **return** 0;  } |

**Output**

No Error

***Time Complexity:****O(N)*  
 ***Auxiliary Space:****O(block\_size)*

# EXPERIMENT NO: 4

**NAME OF THE EXPERIMENT:** Write a program for Hamming Code generaion for Error

detection and correction

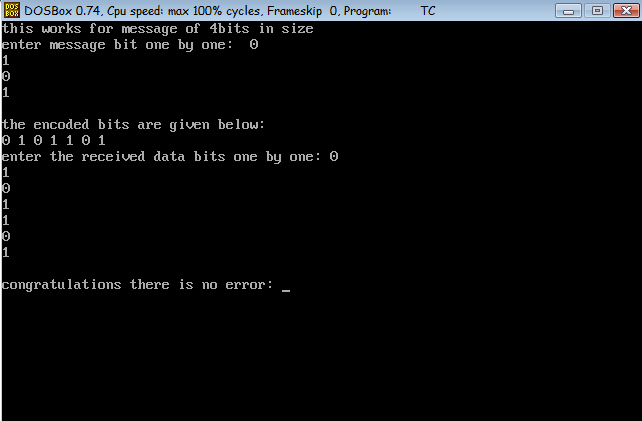
**OBJECTIVE:** Hamming Code generaion for Error detection and correction.

**RESOURCE:** Turbo C

## PROGRAM LOGIC:

1. #include<stdio.h>
2. #include<conio.h>
3. void main() {
4. int data[7],rec[7],i,c1,c2,c3,c;
5. printf("this works for message of 4bits in size \nenter message bit
6. one by one: ");
7. scanf("%d%d%d%d",&data[0],&data[1],&data[2],&data[4]);
8. data[6]=data[0]^data[2]^data[4];
9. data[5]=data[0]^data[1]^data[4];
10. data[3]=data[0]^data[1]^data[2];
11. printf("\nthe encoded bits are given below: \n");
12. for (i=0;i<7;i++) {
13. printf("%d ",data[i]);
14. }
15. printf("\nenter the received data bits one by one: ");
16. for (i=0;i<7;i++) {
17. scanf("%d",&rec[i]);
18. }
19. c1=rec[6]^rec[4]^rec[2]^rec[0];
20. c2=rec[5]^rec[4]^rec[1]^rec[0];
21. c3=rec[3]^rec[2]^rec[1]^rec[0];
22. c=c3\*4+c2\*2+c1 ;
23. if(c==0) {
24. printf("\ncongratulations there is no error: ");
25. } else {
26. printf("\nerron on the postion: %d\nthe correct message is \n",c);
27. if(rec[7-c]==0)
28. rec[7-c]=1; else
29. rec[7-c]=0;
30. for (i=0;i<7;i++) {
31. printf("%d ",rec[i]);
32. }
33. }
34. getch();
35. }

### Output



# EXPERIMENT NO: 5

**NAME OF THE EXPERIMENT:** Cyclic Redundancy Check.

**OBJECTIVE:** Implement on a data set of characters the three CRC polynomials – CRC12,

CRC 16 and CRC CCIP.

**RESOURCE:** Turbo C

## PROGRAM LOGIC:

CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be 2 power(-n). This scheme sometimes known as Cyclic Redundancy Code

**PROCEDURE:** Go to debug -> run or press CTRL + F9 to run the program.

## SOURCE CODE:

//PROGRAM FOR CYCLIC REDUNDENCY CHECK

#include<stdio.h> #include<conio.h>

int gen[4],genl,frl,rem[4]; void main()

{

int i,j,fr[8],dupfr[11],recfr[11],tlen,flag; clrscr();

frl=8; genl=4;

printf("enter frame:"); for(i=0;i<frl;i++)

{

scanf("%d",&fr[i]); dupfr[i]=fr[i];

}

printf("enter generator:"); for(i=0;i<genl;i++) scanf("%d",&gen[i]); tlen=frl+genl-1; for(i=frl;i<tlen;i++)

{

dupfr[i]=0;

}

remainder(dupfr); for(i=0;i<frl;i++)

{

recfr[i]=fr[i];

}

for(i=frl,j=1;j<genl;i++,j++)

{

recfr[i]=rem[j];

}

remainder(recfr); flag=0; for(i=0;i<4;i++)

{

if(rem[i]!=0) flag++;

}

if(flag==0)

{

printf("frame received correctly");

}

Else

{

printf("the received frame is wrong");

}

getch();

}

remainder(int fr[])

{

int k,k1,i,j; for(k=0;k<frl;k++)

{

if(fr[k]==1)

{

k1=k; for(i=0,j=k;i<genl;i++,j++)

{

rem[i]=fr[j]^gen[i];

}

for(i=0;i<genl;i++)

{

fr[k1]=rem[i]; k1++;

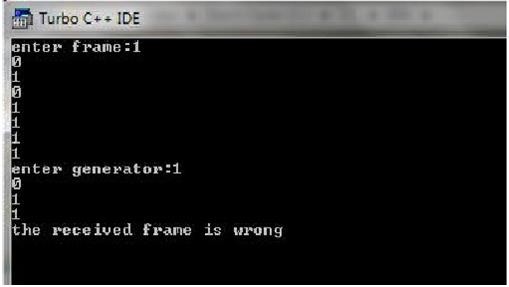
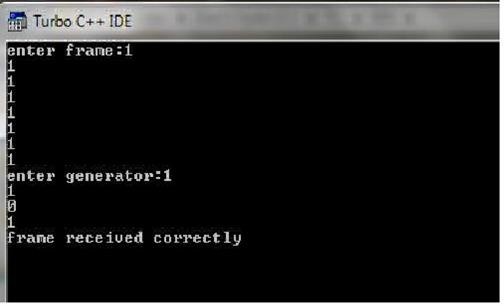
}

}

}

}

**OUTPUT:**



## Viva Questions:

1. What is CRC?
2. What is the use of the CRC?
3. Name the CRC standards?
4. Define Checksum?
5. **Define generator polynomial?**

# EXPERIMENT NO: 6

**NAME OF THE EXPERIMENT:** Program to implement Sliding Window Protocol for

Goback N

**OBJECTIVE: **Go back n:****Sender transmits all frames present in the window that occurs after the error bit including error bit also.

**RESOURCE:** Turbo C

**PROGRAM LOGIC:** Go-Back-N protocol, also called Go-Back-N Automatic Repeat reQuest, is a data link layer protocol that uses a sliding window method for reliable and sequential delivery of data frames. It is a case of sliding window protocol having to send window size of N and receiving window size of 1.

## Working Principle

Go – Back – N ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. The frames are sequentially numbered and a finite number of frames. The maximum number of frames that can be sent depends upon the size of the sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

**CODE IN C:**

#include<stdio.h>

int main()

{

int windowsize,sent=0,ack,i;

printf("enter window size\n");

scanf("%d",&windowsize);

while(1)

{

for( i = 0; i < windowsize; i++)

{

printf("Frame %d has been transmitted.\n",sent);

sent++;

if (sent = = windowsize)

break;

}

printf("\nPlease enter the last Acknowledgement received.\n");

scanf("%d",&ack);

if(ack = = windowsize)

break;

else

sent = ack;

}

return 0;

}

**OUTPUT:-**

Enter window size

8

Frame 0 has been transmitted.

Frame 1 has been transmitted.

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.

Please enter the last Acknowledgement received.

2

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.

Please enter the last Acknowledgement received.

8

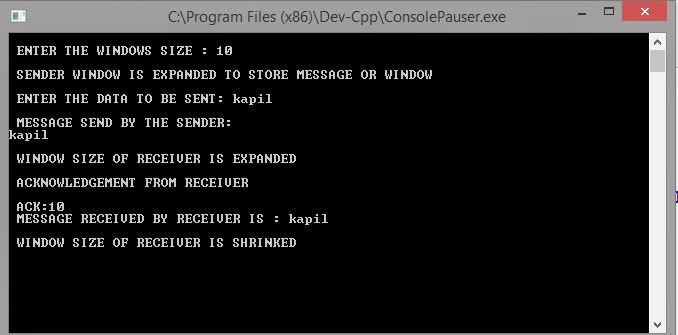
# 

\*\* \*\* \*\*

**EXAMPLE-2 :**

#### #include<stdio.h> #include<conio.h> void main() { char sender[50],receiver[50]; int i,winsize;   printf("\n ENTER THE WINDOWS SIZE : "); scanf("%d",&winsize);   printf("\n SENDER WINDOW IS EXPANDED TO STORE MESSAGE OR WINDOW \n");   printf("\n ENTER THE DATA TO BE SENT: "); fflush(stdin); gets(sender); for(i=0;i<winsize;i++) receiver[i]=sender[i]; receiver[i]=NULL;  printf("\n MESSAGE SEND BY THE SENDER:\n");  puts(sender);   printf("\n WINDOW SIZE OF RECEIVER IS EXPANDED\n");  printf("\n ACKNOWLEDGEMENT FROM RECEIVER \n"); for(i=0;i<winsize;i++); printf("\n ACK:%d",i);   printf("\n MESSAGE RECEIVED BY RECEIVER IS : ");   puts(receiver);   printf("\n WINDOW SIZE OF RECEIVER IS SHRINKED \n"); getch(); }

------------------------------------------------------------------------------------------------------------------ Output.…

[](http://3.bp.blogspot.com/-YHo9mGOMp3U/VB5tF-jq67I/AAAAAAAAAO0/8aOX4W7KoVE/s1600/go+back.JPG)

# EXPERIMENT NO: 7

**NAME OF THE EXPERIMENT:** Program to implement Sliding Window Protocol for

Selective Repeat

**OBJECTIVE: **Selective Repeat:****Sender transmits only that frame which is erroneous or is lost.

**RESOURCE:** Turbo C

# **PROGRAM LOGIC:** The sliding window protocol is a flow control protocol that allows both

# link nodes A and B to send and receive data and acknowledgments simultaneously.

* Here, the sender can send multiple frames without having to wait for acknowledgments.
* If no new data frames are ready for transmission in a specified time, a separate acknowledgment frame is generated to avoid time-out.
* Each outbound frame contains a sequence number ranging from 0 to 2(

## **Sender Window:**

## Sender Window is a set of sequence numbers maintained by the sender corresponding to the frame sequence numbers of frames sent out but not yet acknowledged.

* The sender can transmit a maximum number of frames before receiving any acknowledgment without blocking (Pipelining).
* All the frames in a sending window can be lost or damaged and hence must be saved in memory or buffer till they are acknowledged.

## Receiving Window :

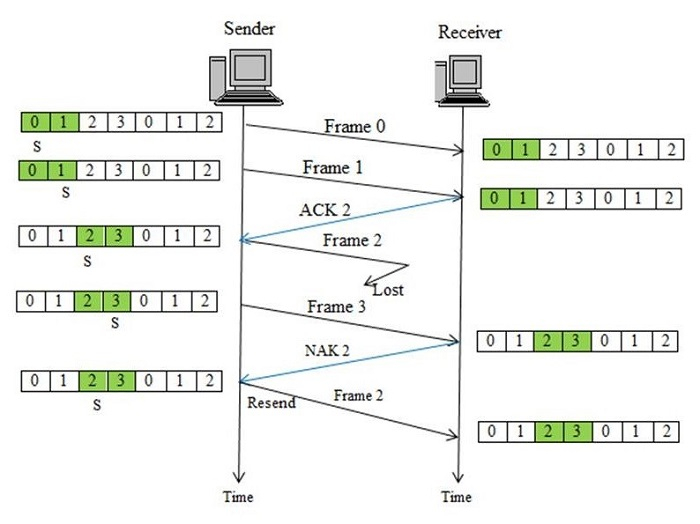
A Receiving Window is a set of sequence numbers that is maintained by the receiver. It allows receiving and acknowledging of multiple frames.

* The size of the receiving window is fixed at a specified initial size.
* Any frame received with a sequence number outside the receiving window is discarded.
* The sending and receiving window may not have the same size or any upper or lower limits.

## Selective Repeat Protocol :

The selective repeat protocol is an implementation of the sliding window protocol. In the selective repeat protocol, both the sender and the receiver maintain a window of outstanding and acceptable sequence numbers.

* In SRP, the sender's window size starts at 0 and it grows to some predefined maximum.
* The receiver's window is always fixed in size and equal to the predetermined maximum.
* The receiver has the buffer reserved for each sequence number within its fixed window.
* The sender and the receiver maintain a buffer of their window size.
* If there is an error, the receiver checks the lower edge to the last sequence number before the lost frame sequence number.
* The receiver continues to receive and acknowledge incoming frames.
* The sender maintains a timeout clock for the unacknowledged frame number and retransmits that frame after the timeout.
* The acknowledgment will be piggybacked to the sender. But when there is no traffic in the reverse direction, piggyback is impossible, a special timer will time out for the ACK so that the ACK is sent back as an independent packet. If the receiver suspects that the transmission has an error, it immediately sends back a negative acknowledgment (NAK) to the sender.



****Note**** − SRP works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. In selective repeat protocol, the size of the sender and receiver windows must be at most one-half of 2.

## Sender Site Algorithm of Selective Repeat Protocol

begin

    frame s; //s denotes frame to be sent

    frame t; //t is temporary frame

    S\_window = power(2,m-1); //Assign maximum window size

    SeqFirst = 0; // Sequence number of first frame in window

    SeqN = 0; // Sequence number of Nth frame window

    while (true) //check repeatedly

      do

          Wait\_For\_Event(); //wait for availability of packet

          if ( Event(Request\_For\_Transfer)) then

            //check if window is full

            if (SeqN–SeqFirst >= S\_window) then

                doNothing();

            end if;

            Get\_Data\_From\_Network\_Layer();

            s = Make\_Frame();

            s.seq = SeqN;

            Store\_Copy\_Frame(s);

            Send\_Frame(s);

            Start\_Timer(s);

            SeqN = SeqN + 1;

          end if;

          if ( Event(Frame\_Arrival) then

            r = Receive\_Acknowledgement();

            // Resend frame whose sequence number is with ACK

            if ( r.type = NAK) then

                if ( NAK\_No > SeqFirst && NAK\_No < SeqN ) then

                  Retransmit( s.seq(NAK\_No));

                  Start\_Timer(s);

                end if

                  //Remove frames from sending window with positive ACK

                else if ( r.type = ACK ) then

                  Remove\_Frame(s.seq(SeqFirst));

                  Stop\_Timer(s);

                  SeqFirst = SeqFirst + 1;

                end if

          end if

          // Resend frame if acknowledgement haven’t been received

          if ( Event(Time\_Out)) then

            Start\_Timer(s);

            Retransmit\_Frame(s);

          end if

end

## Receiver Site Algorithm of Selective Repeat Protocol

Begin

    frame f;

    RSeqNo = 0; // Initialise sequence number of expected frame

    NAKsent = false;

   ACK = false;

    For each slot in receive\_window

    Mark(slot)=false;

    while (true) //check repeatedly

      do

          Wait\_For\_Event(); //wait for arrival of frame

          if ( Event(Frame\_Arrival) then

            Receive\_Frame\_From\_Physical\_Layer();

            if ( Corrupted ( f.SeqNo ) AND NAKsent = false) then

               SendNAK(f.SeqNo);

               NAKsent = true;

            end if

            if ( f.SeqNo != RSeqNo AND NAKsent = false ) then

               SendNAK(f.SeqNo);

               NAKsent = true;

               if ( f.SeqNo is in receive\_window ) then

                  if ( Mark(RSeqNo) = false ) then

                     Store\_frame(f.SeqNo);

                     Mark(RSeqNo) = true;

                  end if

               end if

               else

               while ( Mark(RSeqNo))

                  Extract\_Data(RSeqNo);

                  Deliver\_Data\_To\_Network\_Layer();

                  RSeqNo = RSeqNo + 1;

                  Send\_ACK(RSeqNo);

               end while

            end if

         end if

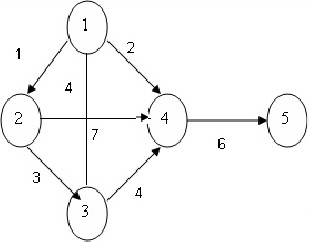
   end while

end

# EXPERIMENT NO: 10

**NAME OF THE EXPERIMENT:** Shortest Path.

**OBJECTIVE:** Implement Dijkstra‘ s algorithm to compute the Shortest path thru a

given graph.

**RESOURCE:** Turbo C

**PROGRAM LOGIC: Dijkstra's algorithm** is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.

**PROCEDURE:** Go to debug -> run or press CTRL + F9 to run the program.

## SOURCE CODE:

// .PROGRAM FOR FINDING SHORTEST PATH FOR A GIVEN GRAPH//

#include<stdio.h> #include<conio.h> void main()

{

int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index; clrscr();

printf("enter the cost matrix\n"); for(i=1;i<=5;i++) for(j=1;j<=5;j++) scanf("%d",&a[i][j]); printf("enter the paths\n"); scanf("%d",&p);

printf("enter possible paths\n"); for(i=1;i<=p;i++) for(j=1;j<=5;j++) scanf("%d",&path[i][j]);

for(i=1;i<=p;i++)

{ t[i]=0;

stp=st; for(j=1;j<=5;j++)

{

edp=path[i][j+1]; t[i]=t[i]+a[stp][edp]; if(edp==ed)

break; else stp=edp;

}

}min=t[st]; index=st; for(i=1;i<=p;i++)

{

if(min>t[i])

{

min=t[i]; index=i;

}

}

printf("minimum cost %d",min); printf("\n minimum cost path "); for(i=1;i<=5;i++)

{

printf("--> %d",path[index][i]); if(path[index][i]==ed)

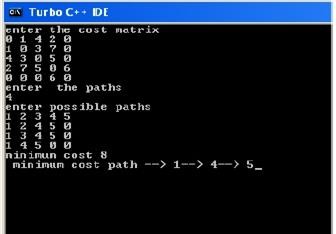
break;

}

getch();

}

**Output:**



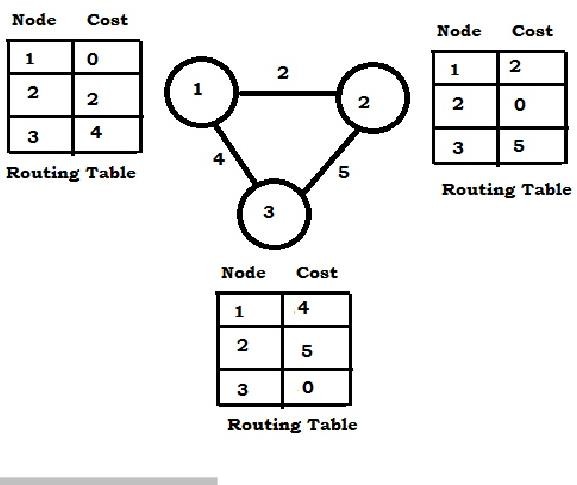
## Viva questions:

1. Define Dijkstra’s algorithm?
2. What is the use of Dijkstra’s algorithm?
3. What is path?
4. What is minimum cost path?
5. How to find shortest path using Dijkstra’s algorithm?

# EXPERIMENT NO: 11

**NAME OF THE EXPERIMENT:** Distance Vector routing.

**OBJECTIVE:** Obtain Routing table at each node using distance vector routing algorithm for a given subnet.



**RESOURCE:** Turbo C

## PROGRAM LOGIC: Distance Vector Routing Algorithms calculate a best route to reach a destination based solely on distance. E.g. RIP. RIP calculates the reach ability based on hop count. It’ s different from link state algorithms which consider some other factors like bandwidth and other metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.

**PROCEDURE:** Go to debug -> run or press CTRL + F9 to run the program.

## SOURCE CODE:

#include<stdio.h> #include<conio.h> struct node

{

unsigned dist[20]; unsigned from[20];

}rt[10];

int main()

{

int dmat[20][20]; int n,i,j,k,count=0; clrscr();

printf("\nEnter the number of nodes : "); scanf("%d",&n);printf("Enter the cost matrix :\n"); for(i=0;i<n;i++)

for(j=0;j<n;j++)

{

scanf("%d",&dmat[i][j]); dmat[i][i]=0; rt[i].dist[j]=dmat[i][j]; rt[i].from[j]=j;

}

Do

{

count=0; for(i=0;i<n;i++) for(j=0;j<n;j++) for(k=0;k<n;k++)

if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])

{

rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j]; rt[i].from[j]=k;count++;

}

}while(count!=0); for(i=0;i<n;i++)

{

printf("\nState value for router %d is \n",i+1); for(j=0;j<n;j++)

{

printf("\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);

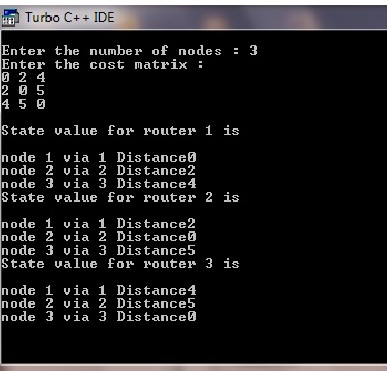
}

}

printf(“\n”);

}

**Output:**



## Viva Questions:

1. What is routing?
2. What is best algorithm among all routing algorithms?
3. What is static routing?
4. Differences between static and dynamic?
5. What is optimality principle?

# EXPERIMENT No :12

**NAME OF THE EXPERIMENT:** Broadcast Tree.

**OBJECTIVE:** Implement broadcast tree for a given subnet of hosts

**RESOURCE:** Turbo C

## PROGRAM LOGIC:

## This technique is widely used because it is simple and easy to understand. The idea of this algorithm is to bui ld a graph of the subnet with each node of the graph representing a router and each arc of the graph representing a communication line.

To choose a route between a given pair of routers the algorithm just finds the broadcast between them on the graph.

**PROCEDURE:** Go to debug -> run or press CTRL + F9 to run the program.

## SOURCE CODE:

// Write a ‘c’ program for Broadcast tree from subnet of host #include<stdio.h>

#include<conio.h> int p,q,u,v,n;

int min=99,mincost=0; int t[50][2],i,j;

int parent[50],edge[50][50]; main()

{

clrscr();

printf("\n Enter the number of nodes"); scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("%c\t",65+i); parent[i]=-1;

}

printf("\n"); for(i=0;i<n;i++)

{

printf("%c",65+i); for(j=0;j<n;j++) scanf("%d",&edge[i][j]);

}

for(i=0;i<n;i++)

{

for(j=0;j<n;j++) if(edge[i][j]!=99)

if(min>edge[i][j])

{

min=edge[i][j]; u=i;

v=j;

}

p=find(u); q=find(v); if(p!=q)

{ t[i][0]=u;

t[i][1]=v; mincost=mincost+edge[u][v]; sunion(p,q);

}

Else

{

t[i][0]=-1;t[i][1]=-1;

}

min=99;

}

printf("Minimum cost is %d\n Minimum spanning tree is\n" ,mincost); for(i=0;i<n;i++)

if(t[i][0]!=-1 && t[i][1]!=-1)

{

printf("%c %c %d", 65+t[i][0],65+t[i][1],edge[t[i][0]][t[i][1]]);printf("\n");

}

getch();

}

sunion(int l,int m)

{

parent[l]=m;

}

find(int l)

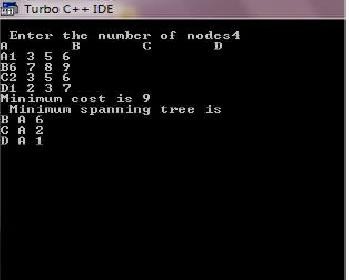
{

if(parent[l]>0)

i=parent[i]; return i;

}

**Output:**



## Viva questions:

1. What is spanning tree?
2. What is broad cast tree?
3. What are the advantages of broadcast tree?
4. What is flooding?
5. What is subnet?

# EXPERIMENT No :13

**13 ( i )**

**NAME OF THE EXPERIMENT:** Packet Capture Using Wire Shark

**OBJECTIVE:** It can capture, dissect, and decode various protocols

**RESOURCE:** Wireshark, Linux sys admin, Turbo C

## PROGRAM LOGIC:

Wireshark is an open source network packet analyzer.

It can capture, dissect, and decode various protocols. This helps Linux sysadmin to troubleshoot network issues.

Apart from using wirehshark as a standlone application for debugging network packets, you can also write your own extension or plugin using wireshark libraries for your custom application.

This tutorial explains how to use wireshark libraries to write custom code to debug network packets using a C example program.  
  
 The code explains two parts. First, to capture network packets. Second, to decode packets using libwireshark.

As a prerequisite, your system should have both libpcap and wireshark libraries installed.

To capture a packet, refer to [How to Perform Packet Sniffing Using Libpcap with C Example](https://www.thegeekstuff.com/2012/10/packet-sniffing-using-libpcap/)

[Code](https://www.thegeekstuff.com/2012/10/packet-sniffing-using-libpcap/).

You can also open an existing pcap file using the following api inside your C program:

pd = pcap\_open\_offline(pcap\_path, errbuf);

Wireshark code uses its own dissection engine (epan module library) to dissect the network packets.

The following code shows the necessary steps to initialize it properly.

The functions mentioned below are from the wireshark open source code, which will initialize the packet dissection engine, required data structures, variables, GUID mapping, memory allocation subsystem, registering all the protocol dissector handles, host name lookup, that are necessary for dissection process.

### An example

The code below makes use of the libpcap functions to achieve a basic packet capture. After capturing the packets, inside the callback function, the length of each packet is printed on stdout.

#include <pcap.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netinet/if\_ether.h>

#include <netinet/tcp.h>

#include <netinet/ip.h>

#include <string.h>

void callback(u\_char \*useless,const struct pcap\_pkthdr\* pkthdr,const u\_char\*

packet)

{

static int count = 1;

printf("\nPacket number [%d], length of this packet is: %d\n", count++, pkthdr->len);

}

int main(int argc,char \*\*argv)

{

char \*dev;

char errbuf[PCAP\_ERRBUF\_SIZE];

pcap\_t\* descr;

struct bpf\_program fp; /\* to hold compiled program \*/

bpf\_u\_int32 pMask; /\* subnet mask \*/

bpf\_u\_int32 pNet; /\* ip address\*/

pcap\_if\_t \*alldevs, \*d;

char dev\_buff[64] = {0};

int i =0;

// Check if sufficient arguments were supplied

if(argc != 3)

{

printf("\nUsage: %s [protocol][number-of-packets]\n",argv[0]);

return 0;

}

// Prepare a list of all the devices

if (pcap\_findalldevs(&alldevs, errbuf) == -1)

{

fprintf(stderr,"Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

// Print the list to user

// so that a choice can be

// made

printf("\nHere is a list of available devices on your system:\n\n");

for(d=alldevs; d; d=d->next)

{

printf("%d. %s", ++i, d->name);

if (d->description)

printf(" (%s)\n", d->description);

else

printf(" (Sorry, No description available for this device)\n");

}

// Ask user to provide the interface name

printf("\nEnter the interface name on which you want to run the packet sniffer : ");

fgets(dev\_buff, sizeof(dev\_buff)-1, stdin);

// Clear off the trailing newline that

// fgets sets

dev\_buff[strlen(dev\_buff)-1] = '';

// Check if something was provided

// by user

if(strlen(dev\_buff))

{

dev = dev\_buff;

printf("\n ---You opted for device [%s] to capture [%d] packets---\n\n Starting capture...",dev, (atoi)(argv[2]));

}

// If something was not provided

// return error.

if(dev == NULL)

{

printf("\n[%s]\n", errbuf);

return -1;

}

// fetch the network address and network mask

pcap\_lookupnet(dev, &pNet, &pMask, errbuf);

// Now, open device for sniffing

descr = pcap\_open\_live(dev, BUFSIZ, 0,-1, errbuf);

if(descr == NULL)

{

printf("pcap\_open\_live() failed due to [%s]\n", errbuf);

return -1;

}

// Compile the filter expression

if(pcap\_compile(descr, &fp, argv[1], 0, pNet) == -1)

{

printf("\npcap\_compile() failed\n");

return -1;

}

// Set the filter compiled above

if(pcap\_setfilter(descr, &fp) == -1)

{

printf("\npcap\_setfilter() failed\n");

exit(1);

}

// For every packet received, call the callback function

// For now, maximum limit on number of packets is specified

// by user.

pcap\_loop(descr,atoi(argv[2]), callback, NULL);

printf("\nDone with packet sniffing!\n");

return 0;

}

**Output:**

In the code above :

* The function pcap\_findalldevs() is used to fetch a list of all available interface devices. This list can be shown to the user so that the intended interface can be selected to sniff packets on. Please note that these is exists a function pcap\_lookupdev() that also returns an interface device but the problem with this function is that it returns the first available non loop-back device. So in case I am using wireless network connection and the interface device for my connection is ‘wlan0’ but pcap\_lookupdev() function would still return ‘eth0’ as it encounters this interface first. So using pcap\_findalldevs() is a better option as it  produces a list of interface devices to choose from.
* The list returned by the function pcap\_findalldevs() is given to user and the user’s input is taken from stdin.
* Then the function pcap\_lookupnet() is used to fetch the ip address and network mask.
* Through the function pcap\_open\_live() the pcap library is initialized with the interface device selected.
* Through pcap\_compile() function , we can compile any filter on protocol etc set by the user.
* Through pcap\_setfilter(), this filter is applied.
* Finally through function pcap\_loop() the library starts packet capture on the selected device with the filter applied and with every relevant packet captured, the callback function is called.

Here is the output of above program :

$ sudo ./pcap tcp 10

[sudo] password for himanshu:

Here is a list of available devices on your system:

1. eth0 (Sorry, No description available for this device)

2. wlan0 (Sorry, No description available for this device)

3. usbmon1 (USB bus number 1)

4. usbmon2 (USB bus number 2)

5. usbmon3 (USB bus number 3)

6. usbmon4 (USB bus number 4)

7. usbmon5 (USB bus number 5)

8. usbmon6 (USB bus number 6)

9. usbmon7 (USB bus number 7)

10. any (Pseudo-device that captures on all interfaces)

11. lo (Sorry, No description available for this device)

Enter the interface name on which you want to run the packet sniffer : wlan0

---You opted for device [wlan0] to capture [10] packets---

Starting capture...

Packet number [1], length of this packet is: 496

Packet number [2], length of this packet is: 66

Packet number [3], length of this packet is: 357

Packet number [4], length of this packet is: 66

Packet number [5], length of this packet is: 238

Packet number [6], length of this packet is: 66

Packet number [7], length of this packet is: 403

Packet number [8], length of this packet is: 66

Packet number [9], length of this packet is: 121

Packet number [10], length of this packet is: 66

Done with packet sniffing!

If you are not executing the above program as root, you should use sudo to run the program as the actions done by libpcap library requires super user privileges.

# EXPERIMENT No : 13

**13 ( i i )**

## ii) Start Capturing

The following methods can be used to start capturing packets with Wireshark:

* You can double-click on an interface in the [welcome screen](https://www.wireshark.org/docs/wsug_html_chunked/ChCapInterfaceSection.html" \o "4.4. The \“Capture\” Section Of The Welcome Screen).
* You can select an interface in the [welcome screen](https://www.wireshark.org/docs/wsug_html_chunked/ChCapInterfaceSection.html" \o "4.4. The \“Capture\” Section Of The Welcome Screen), then select Capture → Start or click the first

toolbar button.

* You can get more detailed information about available interfaces using [Section 4.5, “The](https://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureOptions.html" \o "4.5. The \“Capture Options\” Dialog Box)

[“Capture Options” Dialog Box”](https://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureOptions.html" \o "4.5. The \“Capture Options\” Dialog Box) (Capture → Options…​).

* If you already know the name of the capture interface you can start Wireshark from the

command line:

$ wireshark -i eth0 -k

This will start Wireshark capturing on interface eth0. More details can be found at [Section 11.2,](https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html" \o "11.2. Start Wireshark from the command line)

[“Start Wireshark from the command line”](https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html" \o "11.2. Start Wireshark from the command line).

# EXPERIMENT No : 13

**13 ( i ii )**

## Viewing Packets You Have Captured

Once you have captured some packets or you have opened a previously saved capture file, you can view the packets that are displayed in the packet list pane by simply clicking on a packet in the packet list pane, which will bring up the selected packet in the tree view and byte view panes.

You can then expand any part of the tree to view detailed information about each protocol in each packet. Clicking on an item in the tree will highlight the corresponding bytes in the byte view.

An example with a TCP packet selected is shown in [Figure 6.1, “Wireshark with a TCP packet selected for viewing”](https://www.wireshark.org/docs/wsug_html_chunked/ChapterWork.html" \l "ChWorkSelPack1" \o "Figure 6.1. Wireshark with a TCP packet selected for viewing). It also has the Acknowledgment number in the TCP header selected,

which shows up in the byte view as the selected bytes.

**Figure 6.1. Wireshark with a TCP packet selected for viewing**

|  |
| --- |
| IMG_256 |

You can also select and view packets the same way while Wireshark is capturing if you selected “Update list of packets in real time” in the “Capture Preferences” dialog box.

In addition you can view individual packets in a separate window as shown in [Figure 6.2, “Viewing a packet in a separate window”](https://www.wireshark.org/docs/wsug_html_chunked/ChapterWork.html" \l "ChWorkPacketSepView" \o "Figure 6.2. Viewing a packet in a separate window). You can do this by double-clicking on an item in the packet list or by selecting the packet in which you are interested in the packet list pane and selecting View → Show Packet in New Window. This allows you to easily compare two or more packets, even across multiple files.

**Figure 6.2. Viewing a packet in a separate window**

|  |
| --- |
| IMG_257 |

Along with double-clicking the packet list and using the main menu there are a number of other ways to open a new packet window:

# EXPERIMENT No : 13

**13 ( i v)**

## (iv) The “Statistics” Menu

The Wireshark Statistics menu contains the fields shown in [Table 3.9, “Statistics menu items”](https://www.wireshark.org/docs/wsug_html_chunked/ChUseStatisticsMenuSection.html" \l "ChUseStatistics" \o "Table 3.9. Statistics menu items).

**Figure 3.9. The “Statistics” Menu**

|  |
| --- |
| IMG_256 |

Each menu item brings up a new window showing specific statistics.

**Table 3.9. Statistics menu items**

| **Menu Item** | **Accelerator** | **Description** |
| --- | --- | --- |
| Capture File Properties |  | Show information about the capture file, see [Section 8.2, “The “Capture File Properties” Dialog”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatSummary.html" \o "8.2. The \“Capture File Properties\” Dialog). |
| Resolved Addresses |  | See [Section 8.3, “Resolved Addresses”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatResolvedAddresses.html" \o "8.3. Resolved Addresses) |
| Protocol Hierarchy |  | Display a hierarchical tree of protocol statistics, see [Section 8.4, “The “Protocol Hierarchy” Window”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatHierarchy.html" \o "8.4. The \“Protocol Hierarchy\” Window). |
| Conversations |  | Display a list of conversations (traffic between two endpoints), see [Section 8.5.1, “The “Conversations” Window”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatConversations.html" \l "ChStatConversationsWindow" \o "8.5.1. The \“Conversations\” Window). |
| Endpoints |  | Display a list of endpoints (traffic to/from an address), see [Section 8.6.1, “The “Endpoints” Window”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatEndpoints.html" \l "ChStatEndpointsWindow" \o "8.6.1. The \“Endpoints\” Window). |
| Packet Lengths |  | See [Section 8.7, “Packet Lengths”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatPacketLengths.html" \o "8.7. Packet Lengths) |
| I/O Graphs |  | Display user specified graphs (e.g., the number of packets in the course of time), see [Section 8.8, “The “I/O Graphs” Window”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatIOGraphs.html" \o "8.8. The \“I/O Graphs\” Window). |
| Service Response Time |  | Display the time between a request and the corresponding response, see [Section 8.9, “Service Response Time”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatSRT.html" \o "8.9. Service Response Time). |
| DHCP (BOOTP) |  | See [Section 8.10, “DHCP (BOOTP) Statistics”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatDHCPBOOTP.html" \o "8.10. DHCP (BOOTP) Statistics) |
| NetPerfMeter |  | See [Section 8.11, “NetPerfMeter Statistics”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatNetPerfMeter.html" \o "8.11. NetPerfMeter Statistics) |
| ONC-RPC Programs |  | See [Section 8.12, “ONC-RPC Programs”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatONCRPC.html" \o "8.12. ONC-RPC Programs) |
| 29West |  | See [Section 8.13, “29West”](https://www.wireshark.org/docs/wsug_html_chunked/ChStat29West.html" \o "8.13. 29West) |
| ANCP |  | See [Section 8.14, “ANCP”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatANCP.html" \o "8.14. ANCP) |
| BACnet |  | See [Section 8.15, “BACnet”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatBACnet.html" \o "8.15. BACnet) |
| Collectd |  | See [Section 8.16, “Collectd”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatCollectd.html" \o "8.16. Collectd) |
| DNS |  | See [Section 8.17, “DNS”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatDNS.html" \o "8.17. DNS) |
| Flow Graph |  | See [Section 8.18, “Flow Graph”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatFlowGraph.html" \o "8.18. Flow Graph) |
| HART-IP |  | See [Section 8.19, “HART-IP”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatHARTIP.html" \o "8.19. HART-IP) |
| HPFEEDS |  | See [Section 8.20, “HPFEEDS”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatHPFEEDS.html" \o "8.20. HPFEEDS) |
| HTTP |  | HTTP request/response statistics, see [Section 8.21, “HTTP Statistics”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatHTTP.html" \o "8.21. HTTP Statistics) |
| HTTP2 |  | See [Section 8.22, “HTTP2”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatHTTP2.html" \o "8.22. HTTP2) |
| Sametime |  | See [Section 8.23, “Sametime”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatSametime.html" \o "8.23. Sametime) |
| TCP Stream Graphs |  | See [Section 8.24, “TCP Stream Graphs”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatTCPStreamGraphs.html" \o "8.24. TCP Stream Graphs) |
| UDP Multicast Streams |  | See [Section 8.25, “UDP Multicast Streams”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatUDPMulticastStreams.html" \o "8.25. UDP Multicast Streams) |
| Reliable Server Pooling (RSerPool) |  | See [Section 8.26, “Reliable Server Pooling (RSerPool)”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatRSerPool.html" \o "8.26. Reliable Server Pooling (RSerPool)) |
| F5 |  | See [Section 8.27, “F5”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatF5.html" \o "8.27. F5) |
| IPv4 Statistics |  | See [Section 8.28, “IPv4 Statistics”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatIPv4.html" \o "8.28. IPv4 Statistics) |
| IPv6 Statistics |  | See [Section 8.29, “IPv6 Statistics”](https://www.wireshark.org/docs/wsug_html_chunked/ChStatIPv6.html" \o "8.29. IPv6 Statistics) |

# EXPERIMENT No : 14

## NAME OF THE EXPERIMENT: **[How to Run a Simple Nmap Scan](https://www.wikihow.com/Run-a-Simple-Nmap-Scan)**.

**OBJECTIVE**: Ensuring that your router is protected from unwanted intruders is one of the

foundations of a secure network.

**RESOURCE:** Nmap Installer, Linux, Windows

**PROGRAM LOGIC**: Ensuring that your router is protected from unwanted intruders is one of the foundations of a secure network. One of the basic tools for this job is Nmap, or Network Mapper. This program will scan a target and report which ports are open and which are closed, among other things. Security specialists use this program to test the security of a network. To learn how to use it yourself, see Step 1 below.

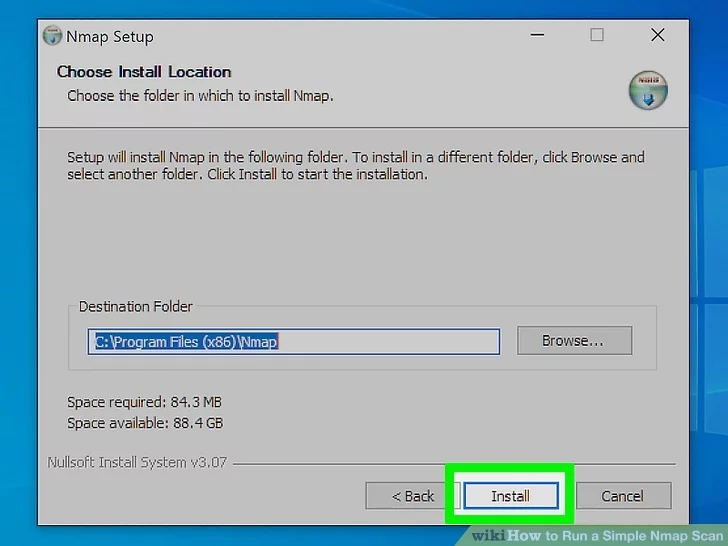
# **[How to Run a Simple Nmap Scan](https://www.wikihow.com/Run-a-Simple-Nmap-Scan)**

1. Download the installer.
2. Run the installer.
3. Open NMap/Zenmap.
4. Enter the target address.
5. Choose a profile.
6. Click Scan.
7. Review the results.

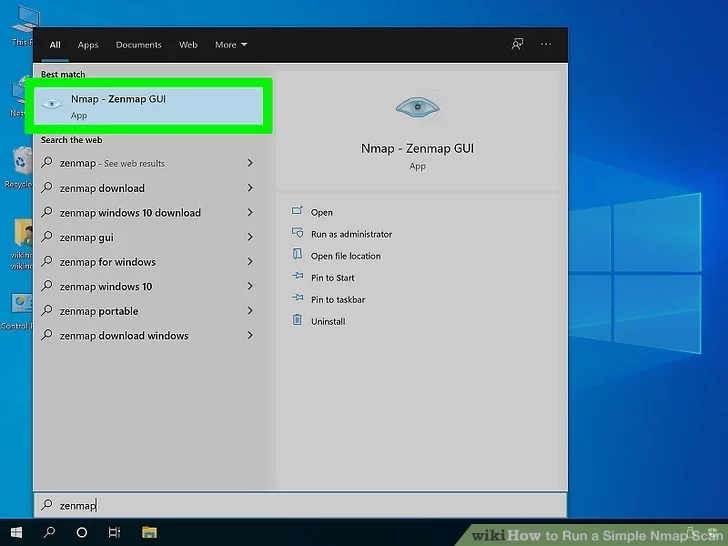
### **Using Zenmap[Download Article](https://www.wikihow.com/Run-a-Simple-Nmap-Scan)**

**1. Download the Nmap installer.** This can be found for free from the developer’s website. It is highly recommended that you download directly from the developer to avoid any potential viruses or fake files. Downloading the Nmap installer includes Zenmap, the graphical interface for Nmap which makes it easy for newcomers to perform scans without having to learn command lines.

* The Zenmap program is available for Windows, Linux, and Mac OS X. You can find the installation files for all operating systems on the Nmap website.

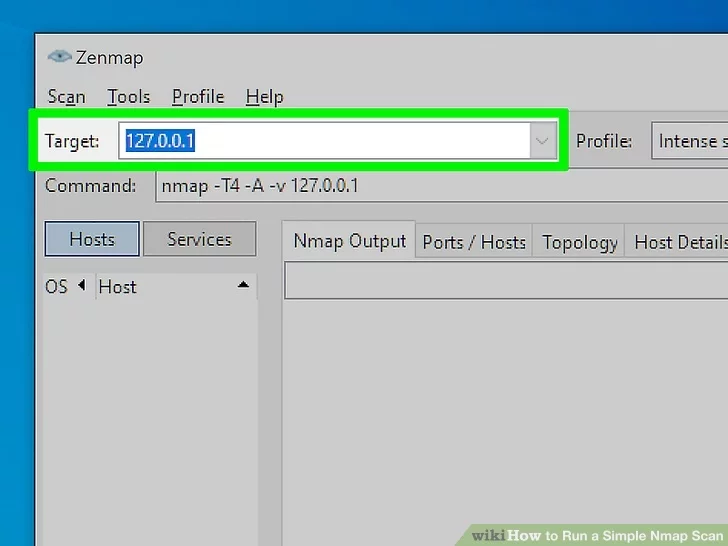
1. [](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-2-Version-3.jpg)

**2. Install Nmap.** Run the installer once it is finished downloading. You will be asked which components you would like to install. In order to get the full benefit of Nmap, keep all of these checked. Nmap will not install any adware or spyware.

[](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-3-Version-3.jpg)

1. **Run the "Nmap – Zenmap" GUI program.** If you left your settings at default during

installation, you should be able to see an icon for it on your desktop. If not, look in your Start menu. Opening Zenmap will start the program.

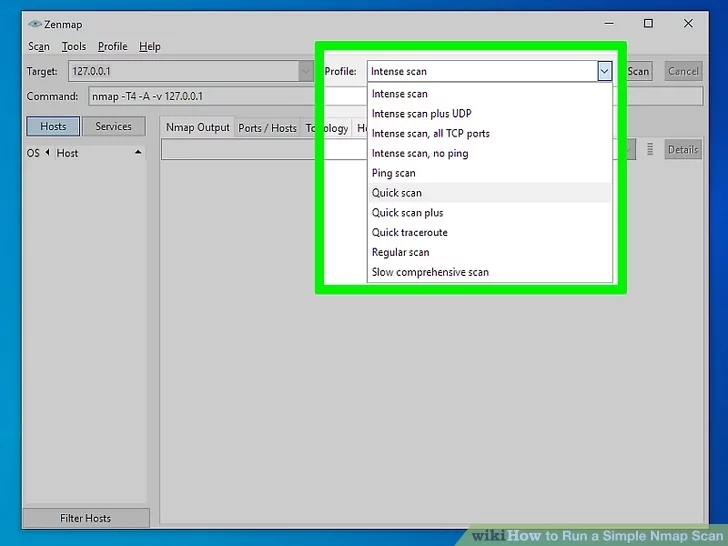


**4. Enter in the target for your scan.** The Zenmap program makes scanning a fairly simple  
 process. The first step to running a scan is choosing your target. You can enter a domain

(example.com), an IP address (127.0.0.1), a network (192.168.1.0/24), or a combination of

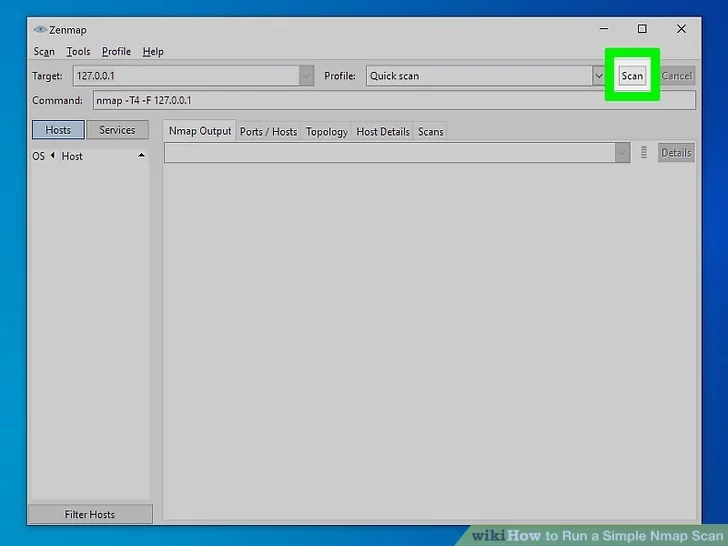
those.

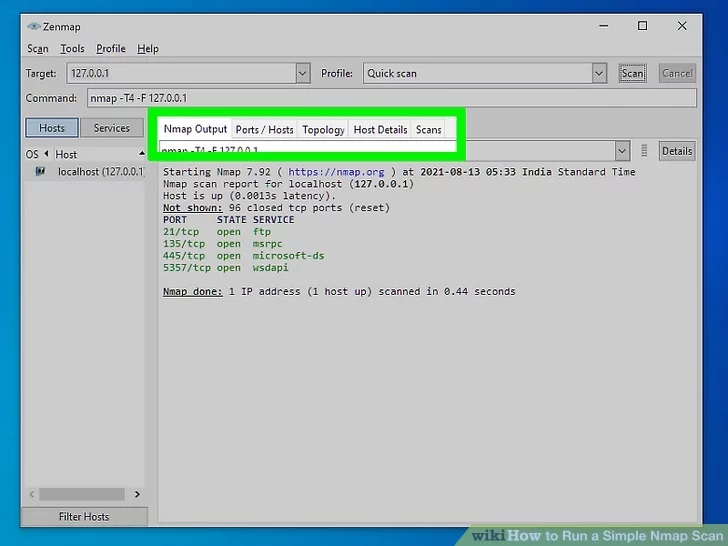
* Depending on the intensity and target of your scan, running an Nmap scan may be against the terms of your internet service provider, and may land you in hot water. Always check your local laws and your ISP contract before performing Nmap scans on targets other than your own network.



**5.Choose your Profile.** Profiles are preset groupings of modifiers that change what is scanned. The profiles allow you to quickly select different types of scans without having to type in the modifiers on the command line. Choose the profile that best fits your needs:[[1]](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "_note-1)

* **Intense scan** - A comprehensive scan. Contains Operating System (OS) detection, version detection, script scanning, traceroute, and has aggressive scan timing. This is considered an intrusive scan.
* **Ping scan** - This scan simply detects if the targets are online, it does not scan any ports.
* **Quick scan** - This is quicker than a regular scan due to aggressive timing and only scanning select ports.
* **Regular scan** - This is the standard Nmap scan without any modifiers. It will return ping and return open ports on the target.

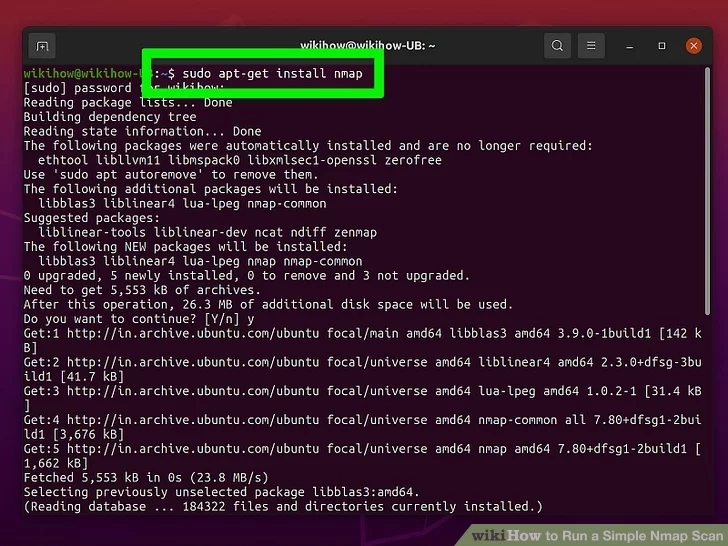
[](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-6-Version-3.jpg)

1. **Click Scan to start scanning.** The active results of the scan will be displayed in the Nmap Output tab. The time the scan takes will depend on the scan profile you chose, the physical distance to the target, and the target’s network configuration.
2. 
3. **Read your results.** Once the scan is finished, you’ll see the message "Nmap done" at the bottom of the Nmap Output tab. You can now check your results, depending on the type of scan you performed. All of the results will be listed in the main Nmap Output tab, but you can use the other tabs to get a better look at specific data.[[2]](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "_note-2)

* **Ports/Hosts** - This tab will show the results of your port scan, including the services for those ports.
* **Topology** - This shows the traceroute for the scan you performed. You can see how many hops your data goes through to reach the target.
* **Host Details** - This shows a summary of your target learned through scans, such as the number of ports, IP addresses, hostnames, operating systems, and more.
  + **Scans** - This tab stores the commands of your previously-run scans. This allows you to quickly re-scan with a specific set of parameters.

**METHOD-2**

**Using the Command Line**

1. [](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-8-Version-2.jpg)

**1. Install Nmap.** Before using Nmap, you will need to install it so that you can run it from the

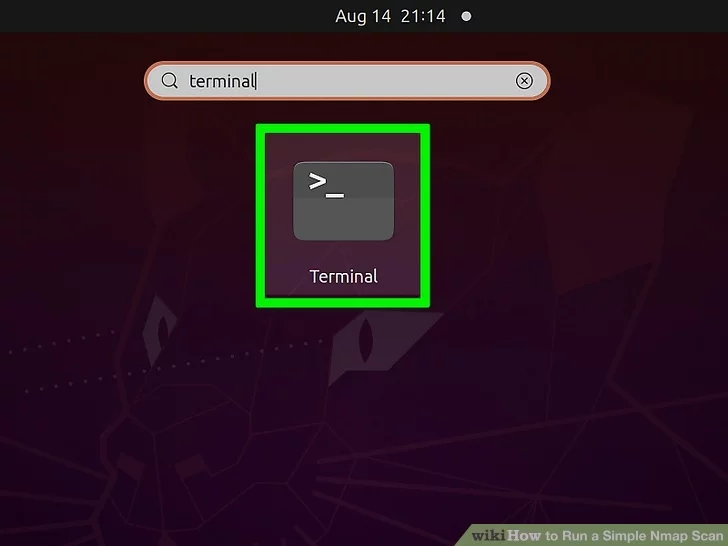
command line of your operating system. Nmap is small and available for free from the

developer. Follow the instructions below for your operating system:

* **Linux** - Download and install Nmap from your repository. Nmap is available through most of the major Linux repositories. Enter in the command below based on your distribution:

Red Hat, Fedora, SUSE  
rpm -vhU http://nmap.org/dist/nmap-6.40-1.i386.rpm (32-bit) OR  
rpm -vhU http://nmap.org/dist/nmap-6.40-1.x86\_64.rpm (64-bit)

* + - Debian, Ubuntu  
      sudo apt-get install nmap
* **Windows** - Download the Nmap installer. This can be found for free from the developer’s website. It is highly recommended that you download directly from the developer to avoid any potential viruses or fake files. Using the installer allows you to quickly install the command line Nmap tools without having to worry about extracting to the right folder.
* If you don’t want the Zenmap graphical user interface, you can uncheck it during the installation process.
* **Mac OS X** – Download the Nmap disk image. This can be found for free from the developer’s website. It is highly recommended that you download directly from the developer to avoid any potential viruses or fake files. Use the included installer to install Nmap on your system. Nmap requires OS X 10.6 or later.

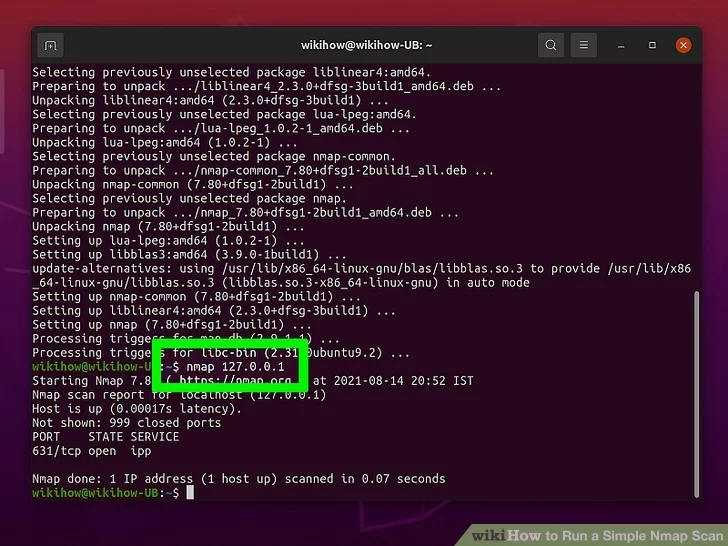


**2. Open your command line.** Nmap commands are run from the command line, and the

results are displayed beneath the command. You can use variables to modify the scan.

You can run the scan from any directory on the command line.

* **Linux** - Open the terminal if you are using a GUI for your Linux distribution. The location of the terminal varies by distribution
* **Windows** - This can be accessed by pressing the Windows key + R and then typing "cmd" into the Run field. Windows 8 users can press Windows key + X and select Command Prompt from the menu. You can run an Nmap scan from any directory.
* **Mac OS X** - Open the Terminal application located in the Utility subfolder of your Applications folder.

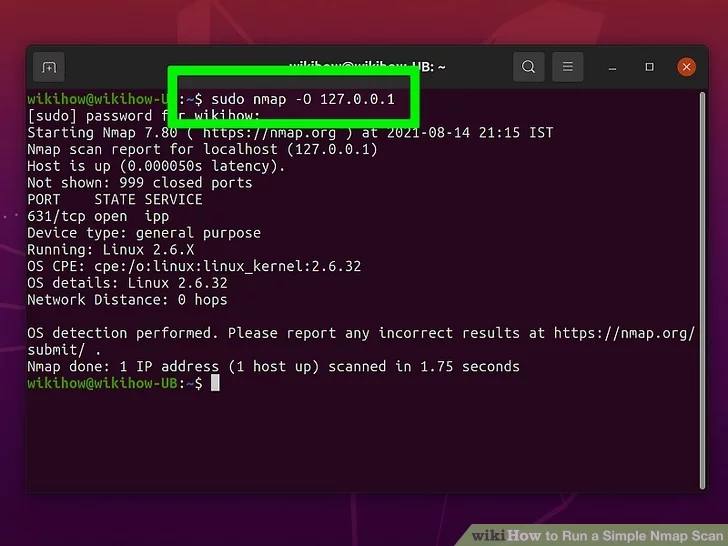
[](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-10-Version-2.jpg)

**3. Run a scan of you target’s ports.** To start a basic scan, type nmap <target>. This will

ping the target and scan the ports. This is an easily-detected scan. The results will be

displayed on your screen. You may need to scroll back up to see all of the results.

* Depending on the intensity and target of your scan, running an Nmap scan may be against the terms of your internet service provider, and may land you in hot water. Always check your local laws and your ISP contract before performing Nmap scans on targets other than your own network.

[](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-11-Version-2.jpg)

1. **Run a modified scan.** You can use command line variables to change the parameters

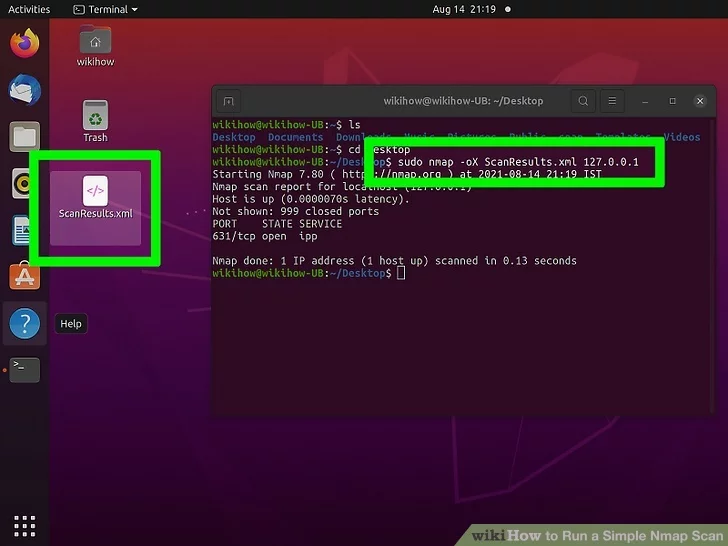
of the scan, resulting in more detailed or less detailed results. Changing the scan

variables will change the intrusiveness of the scan. You can add multiple variables by

placing a space between each one. Variables come before the target: nmap <variable>

<variable> <target>[[3]](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "_note-3)

* **-sS** - This is a SYN stealth scan. It is less detectable than a standard scan, but may take longer. Many modern firewalls can detect an –sS scan.
* **-sn** - This is a ping scan. This will disable port scanning, and will only check to see if the host is online.
* **-O** - This is an operating system scan. The scan will attempt to determine the operating system of the target.
* **-A** - This variable enables several of the most commonly used scans: OS detection, version detection, script scanning, and traceroute.
* **-F** - This enables fast mode, and will reduce the number of ports scanned.
* **-v** - This will show more information in your results, making them easier to read.

[](https://www.wikihow.com/Run-a-Simple-Nmap-Scan" \l "/Image:Run-a-Simple-Nmap-Scan-Step-12-Version-2.jpg)

**5. Output the scan to an XML file.** You can set your scan results to be outputted as an

XML file so that you can easily read them in any web browser. To do this, you will need to

use the **-oX** variable, as well as set a filename for the new XML file. A completed command

would look similar to nmap –oX Scan Results.xml <target>.

* The XML file will be saved to whatever your current working location is.

# EXPERIMENT No : 15

## NAME OF THE EXPERIMENT: **Operating System Detection using Nmap**

**OBJECTIVE**: Nmap **uses TCP/IP stack fingerprinting** for OS detection

**RESOURCE:** Nmap Installer, Linux, Windows

**PROGRAM LOGIC**: OS Detection

One of Nmap's best-known features is remote OS detection using TCP/IP stack fingerprinting. Nmap sends a series of TCP and UDP packets to the remote host and examines practically every bit in the responses. After performing dozens of tests such as TCP ISN sampling, TCP options support and ordering, IP ID sampling, and the initial window size check, Nmap compares the results to its nmap-os-db database of more than 2,600 known OS fingerprints and prints out the OS details if there is a match. Each fingerprint includes a freeform textual description of the OS, and a classification which provides the vendor name (e.g. Sun), underlying OS (e.g. Solaris), OS generation (e.g. 10), and device type (general purpose, router, switch, game console, etc). Most fingerprints also have a Common Platform Enumeration (CPE) representation, like cpe:/o:linux:linux\_kernel:2.6.

If Nmap is unable to guess the OS of a machine, and conditions are good (e.g. at least one open port and one closed port were found), Nmap will provide a URL you can use to submit the fingerprint if you know (for sure) the OS running on the machine. By doing this you contribute to the pool of operating systems known to Nmap and thus it will be more accurate for everyone.

OS detection enables some other tests which make use of information that is gathered during the process anyway. One of these is TCP Sequence Predictability Classification. This measures approximately how hard it is to establish a forged TCP connection against the remote host. It is useful for exploiting source-IP based trust relationships (rlogin, firewall filters, etc) or for hiding the source of an attack. This sort of spoofing is rarely performed any more, but many machines are still vulnerable to it. The actual difficulty number is based on statistical sampling and may fluctuate. It is generally better to use the English classification such as “worthy challenge” or “trivial joke”. This is only reported in normal output in verbose (-v) mode. When verbose mode is enabled along with -O, IP ID sequence generation is also reported. Most machines are in the “incremental” class, which means that they increment the ID field in the IP header for each packet they send. This makes them vulnerable to several advanced information gathering and spoofing attacks.

Another bit of extra information enabled by OS detection is a guess at a target's uptime. This uses the TCP timestamp option ([RFC 1323](http://www.rfc-editor.org/rfc/rfc1323.txt" \t "https://nmap.org/book/_top)) to guess when a machine was last rebooted. The guess can be inaccurate due to the timestamp counter not being initialized to zero or the counter overflowing and wrapping around, so it is printed only in verbose mode.

OS detection is covered in [Chapter 8, Remote OS Detection](https://nmap.org/book/osdetect.html" \o "Chapter 8. Remote OS Detection).

OS detection is enabled and controlled with the following options:

-O (Enable OS detection)

Enables OS detection, as discussed above. Alternatively, you can use -A to enable OS detection along with other things.

--osscan-limit (Limit OS detection to promising targets)

OS detection is far more effective if at least one open and one closed TCP port are found. Set this option and Nmap will not even try OS detection against hosts that do not meet this criteria. This can save substantial time, particularly on -Pn scans against many hosts. It only matters when OS detection is requested with -O or -A.

--osscan-guess; --fuzzy (Guess OS detection results)

When Nmap is unable to detect a perfect OS match, it sometimes offers up near-matches as possibilities. The match has to be very close for Nmap to do this by default. Either of these (equivalent) options make Nmap guess more aggressively. Nmap will still tell you when an imperfect match is printed and display its confidence level (percentage) for each guess.

--max-os-tries (Set the maximum number of OS detection tries against a target)

When Nmap performs OS detection against a target and fails to find a perfect match, it usually repeats the attempt. By default, Nmap tries five times if conditions are favorable for OS fingerprint submission, and twice when conditions aren't so good.

Specifying a lower --max-os-tries value (such as 1) speeds Nmap up, though you miss out on retries which could potentially identify the OS. Alternatively, a high value may be set to allow even more retries when conditions are favorable. This is rarely done, except to generate better fingerprints for submission and integration into the Nmap OS database.

## Introduction

While Nmap has supported OS detection since 1998, this chapter describes the 2nd generation system released in 2006.

### Reasons for OS Detection

While some benefits of discovering the underlying OS and device types on a network are obvious, others are more obscure. This section lists the top reasons I hear for discovering this extra information.

#### Determining vulnerability of target hosts

It is sometimes very difficult to determine remotely whether an available service is susceptible or patched for a certain vulnerability. Even obtaining the application version number doesn't always help, since OS distributors often back-port security fixes without changing the version number. The surest way to verify that a vulnerability is real is to exploit it, but that risks crashing the service and can lead to wasted hours or even days of frustrating exploitation efforts if the service turns out to be patched.

OS detection can help reduce these false positives. For example, the Rwho daemon on unpatched Sun Solaris 7 through 9 may be remotely exploitable (Sun alert #57659). Remotely determining vulnerability is difficult, but you can rule it out by finding that a target system is running Solaris 10.

Taking this from the perspective of a systems administrator rather than a pen-tester, imagine you run a large Sun shop when alert #57659 comes out. Scan your whole network with OS detection to find machines which need patching before the bad guys do.

#### Tailoring exploits

Even after you discover a vulnerability in a target system, OS detection can be helpful in exploiting it. Buffer overflows, format-string exploits, and many other vulnerabilities often require custom-tailored shellcode with offsets and assembly payloads generated to match the target OS and hardware architecture. In some cases, you only get one try because the service crashes if you get the shellcode wrong. Use OS detection first or you may end up sending Linux shellcode to a FreeBSD server.

#### Network inventory and support

While it isn't as exciting as busting root through a specially crafted format string exploit, there are many administrative reasons to keep track of what is running on your network. Before you renew that IRIX support contract for another year, scan to see if anyone still uses such machines. An inventory can also be useful for IT budgeting and ensuring that all company equipment is accounted for.

#### Detecting unauthorized and dangerous devices

With the ubiquity of mobile devices and cheap commodity networking equipment, companies are increasingly finding that employees are extending their networks in undesirable ways. They may install a $20 wireless access point (WAP) in their cubicle without realizing (or caring) that they just opened up the protected corporate network to potential attackers in the parking lot or nearby buildings. WAPs can be so dangerous that Nmap has a special category for detecting them, as demonstrated in [the section called “SOLUTION: Detect Rogue Wireless Access Points on an Enterprise Network”](https://nmap.org/book/osdetect-find-rogue-ap.html" \o "SOLUTION: Detect Rogue Wireless Access Points on an Enterprise Network). Users may also cause sysadmins grief by connecting insecure and/or worm-infected laptops to the corporate network. Regular scanning can detect unauthorized devices for investigation and containment.

#### Social engineering

Another possible use is social engineering. Lets say that you are scanning a target company and Nmap reports a “Datavoice TxPORT PRISM 3000 T1 CSU/DSU 6.22/2.06”. You could call up the target pretending to be Datavoice support and discuss some issues with their PRISM 3000. Tell them you are about to announce a big security hole, but are first providing the patch to valued customers. Some naive administrators might assume that only an authorized engineer from Datavoice would know so much about their CSU/DSU. Of course the patch you send them is a Trojan horse that gives you remote access to sniff and traipse through their network. Be sure to read the rest of this chapter for detection accuracy and verification advice before trying this. If you guess the target system wrong and they call the police, that will be an embarrassing story to tell your cellmates.

## Usage and Examples

The inner workings of OS detection are quite complex, but it is one of the easiest features to use. Simply add -O to your scan options. You may want to also increase the verbosity with - v for even more OS-related details. This is shown in [Example 8.1](https://nmap.org/book/osdetect-usage.html" \l "osdetect-ex-scanme1" \o "Example 8.1. OS detection with verbosity (-O -v)).

Example 8.1. OS detection with verbosity (-O -v)

# nmap -O -v scanme.nmap.org

Starting Nmap ( https://nmap.org )

Nmap scan report for scanme.nmap.org (74.207.244.221)

Not shown: 994 closed ports

PORT STATE SERVICE

22/tcp open ssh

80/tcp open http

646/tcp filtered ldp

1720/tcp filtered H.323/Q.931

9929/tcp open nping-echo

31337/tcp open Elite

Device type: general purpose

Running: Linux 2.6.X

OS CPE: cpe:/o:linux:linux\_kernel:2.6.39

OS details: Linux 2.6.39

Uptime guess: 1.674 days (since Fri Sep 9 12:03:04 2011)

Network Distance: 10 hops

TCP Sequence Prediction: Difficulty=205 (Good luck!)

IP ID Sequence Generation: All zeros

Read data files from: /usr/local/bin/../share/nmap

Nmap done: 1 IP address (1 host up) scanned in 5.58 seconds

Raw packets sent: 1063 (47.432KB) | Rcvd: 1031 (41.664KB)

Including the -O -v options caused Nmap to generate the following extra line items:

**Device type**

All fingerprints are classified with one or more high-level device types, such as router, printer, firewall, or (as in this case) general purpose. These are further described in [the section called “Device and OS classification (Class lines)”](https://nmap.org/book/osdetect-fingerprint-format.html" \l "osdetect-class" \o "Device and OS classification (Class lines)). Several device types may be shown, in which case they will be separated with the pipe symbol as in “Device Type: router|firewall”.

**Running**

This field is also related to the OS classification scheme described in [the section called “Device and OS classification (Class lines)”](https://nmap.org/book/osdetect-fingerprint-format.html" \l "osdetect-class" \o "Device and OS classification (Class lines)). It shows the OS Family (Linux in this case) and OS generation (2.6.X) if available. If there are multiple OS families, they are separated by commas. When Nmap can't narrow down OS generations to one specific choice, options are separated by the pipe symbol ('|') Examples include OpenBSD 3.X, NetBSD 3.X|4.X and Linux 2.4.X|2.5.X|2.6.X.

If Nmap finds too many OS families to print concisely, it will omit this line. When there are no perfect matches, Nmap changes the field to Running (JUST GUESSING) and adds an accuracy percentage (100% is a perfect match) in parentheses after each candidate family name. If no fingerprints are close matches, the line is omitted.

**OS CPE**

This shows a Common Platform Enumeration (CPE) representation of the operating system when available. It may also have a CPE representation of the hardware type. OS CPE begins with cpe:/o and hardware CPE begins with cpe:/h. For more about CPE see [the section called “Common Platform Enumeration (CPE)”](https://nmap.org/book/output-formats-cpe.html" \o "Common Platform Enumeration (CPE)).

**OS details**

This line gives the detailed description for each fingerprint that matches. While the Device type and Running lines are from predefined enumerated lists that are easy to parse by a computer, the OS details line contains free-form data which is useful to a human reading the report. This can include more exact version numbers, device models, and architectures specific to a given fingerprint. In this example, the only matching fingerprint was Linux 2.6.20-1 (Fedora Core 5). When there are multiple exact matches, they are comma-separated. If there aren't any perfect matches, but some close guesses, the field is renamed Aggressive OS guesses and fingerprints are shown followed by a percentage in parentheses which specifies how close each match was.

**Uptime guess**

As part of OS detection, Nmap receives several SYN/ACK TCP packets in a row and checks the headers for a timestamp option. Many operating systems use a simple counter for this which starts at zero at boot time then increments at a constant rate such as twice per second. By looking at several responses, Nmap can determine the current values and rate of increase. Simple linear extrapolation determines boot time. The timestamp algorithm is used for OS detection too (see [the section called “TCP timestamp option algorithm (TS)”](https://nmap.org/book/osdetect-methods.html" \l "osdetect-ts" \o "TCP timestamp option algorithm (TS))) since the increment rate on different systems varies from 2 Hz to 1,000 Hz.

The uptime guess is labeled a “guess” because various factors can make it completely inaccurate. Some operating systems do not start the timestamp counter at zero, but initialize it with a random value, making extrapolation to zero meaningless. Even on systems using a simple counter starting at zero, the counter eventually overflows and wraps around. With a 1,000 Hz counter increment rate, the counter resets to zero roughly every 50 days. So a host that has been up for 102 days will appear to have been up only two days. Even with these caveats, the uptime guess is accurate much of the time for most operating systems, so it is printed when available, but only in verbose mode. The uptime guess is omitted if the target gives zeros or no timestamp options in its SYN/ACK packets, or if it does not reply at all. The line is also omitted if Nmap cannot discern the timestamp increment rate or it seems suspicious (like a 30-year uptime).

**Network Distance**

A side effect of one of the OS detection tests allows Nmap to compute how many routers are between it and a target host. The distance is zero when you are scanning localhost, and one for a machine on the same network segment. Each additional router on the path adds one to the hop count. The Network Distance line is not printed in this example, since Nmap omits the line when it cannot be computed (no reply to the relevant probe).

**TCP Sequence Prediction**

Systems with poor TCP initial sequence number generation are vulnerable to blind TCP spoofing attacks. In other words, you can make a full connection to those systems and send (but not receive) data while spoofing a different IP address. The target's logs will show the spoofed IP, and you can take advantage of any trust relationship between them. This attack was all the rage in the mid-nineties when people commonly used rlogin to allow logins to their account without any password from trusted IP addresses. Kevin Mitnick is alleged to have used this attack to break into Tsutomu Shimomura's computers in December 1994.

The good news is that hardly anyone uses rlogin anymore, and many operating systems have been fixed to use unpredictable initial sequence numbers as proposed by [RFC 1948](http://www.rfc-editor.org/rfc/rfc1948.txt" \t "https://nmap.org/book/_top). For these reasons, this line is only printed in verbose mode. Sadly, many vendors still ship [vulnerable operating systems and devices](http://lcamtuf.coredump.cx/newtcp/" \t "https://nmap.org/book/_top). Even the fixed ones often vary in implementation, which leaves them valuable for OS detection purposes. The class describes the ISN generation algorithm used by the target, and difficulty is a rough estimate of how hard the system makes blind IP spoofing (0 is the easiest). The parenthesized comment is based on the difficulty index and ranges from Trivial joke to Easy, Medium, Formidable, Worthy challenge, and finally Good l uck! Further details about sequence tests are provided in [the section called “TCP ISN greatest common divisor (GCD)”](https://nmap.org/book/osdetect-methods.html" \l "osdetect-gcd" \o "TCP ISN greatest common divisor (GCD)).

While the rlogin family is mostly a relic of the past, clever attackers can still find effective uses for blind TCP spoofing. For example, it allows for spoofed HTTP requests. You don't see the results, but just the URL (POST or GET request) can have dramatic side effects. The spoofing allows attackers to hide their identity, frame someone else, or exploit IP address restrictions.

IP ID sequence generation

Many systems unwittingly give away sensitive information about their traffic levels based on how they generate the lowly 16-bit ID field in IP packets. This can be abused to spoof a port scan against other systems and for other mischievous purposes discussed in [the section called “TCP Idle Scan (-sI)”](https://nmap.org/book/idlescan.html" \o "TCP Idle Scan (-sI)). This field describes the ID generation algorithm that Nmap was able to discern. More information on how it classifies them is available in [the section called “IP ID sequence generation algorithm (TI, CI, II)”](https://nmap.org/book/osdetect-methods.html" \l "osdetect-ti" \o "IP ID sequence generation algorithm (TI, CI, II)). Note that many systems use a different IP ID space for each host they communicate with. In that case, they may appear vulnerable (such as showing the Incremental class) while still being secure against attacks such as the idle scan. For this reason, and because the issue is rarely critical, the IP ID sequence generation line is only printed in verbose mode. If Nmap does not receive sufficient responses during OS detection, it will omit the whole line. The best way to test whether a host is vulnerable to being an idle scan zombie is to test it with -sI.

While TCP fingerprinting is a powerful method for OS detection, interrogating open ports for clues is another effective approach. Some applications, such as Microsoft IIS, only run on a single platform (thus giving it away), while many other apps divulge their platform in overly verbose banner messages. Adding the -sV option enables Nmap version detection, which is trained to look for these clues (among others). In [Example 8.2](https://nmap.org/book/osdetect-usage.html" \l "osdetect-ex-hpux" \o "Example 8.2. Using version scan to detect the OS), Nmap catches the platform details from an FTP server.

**Example 8.2. Using version scan to detect the OS**

# nmap -sV -O -v 129.128.X.XX

Starting Nmap ( https://nmap.org )

Nmap scan report for [hostname] (129.128.X.XX)

Not shown: 994 closed ports

PORT STATE SERVICE VERSION

21/tcp open ftp HP-UX 10.x ftpd 4.1

22/tcp open ssh OpenSSH 3.7.1p1 (protocol 1.99)

111/tcp open rpc

445/tcp filtered microsoft-ds

1526/tcp open oracle-tns Oracle TNS Listener

32775/tcp open rpc

No exact OS matches for host

TCP Sequence Prediction: Class=truly random

Difficulty=9999999 (Good luck!)

IP ID Sequence Generation: Incremental

Service Info: OS: HP-UX

In this example, the line “No exact OS matches for host” means that TCP/IP fingerprinting failed to find an exact match. Fortunately, the Service Info field a few lines down discloses that the OS is HP-UX. If several operating systems were detected (which can happen with NAT gateway boxes that redirect ports to several different machines), the field would be OSs and the values would be comma separated. The Service Info line can also contain hostnames and device types found during the version scan. The focus of this chapter is on TCP/IP fingerprinting though, since version detection was covered in **[Chapter 7, Service and Application Version Detection](https://nmap.org/book/vscan.html" \o "Chapter 7. Service and Application Version Detection).**

With two effective OS detection methods available, which one should you use? The best answer is usually both. In some cases, such as a proxy firewall forwarding to an application on another host, the answers may legitimately differ. TCP/IP fingerprinting will identify the proxy while version scanning will generally detect the server running the proxied application. Even when no proxying or port forwarding is involved, using both techniques is beneficial. If they come out the same, that makes the results more credible. If they come out wildly different, investigate further to determine what is going on before relying on either. Since OS and version detection go together so well, the -A option enables them both.

OS detection is far more effective if at least one open and one closed TCP port are found. Set the --osscan-limit option and Nmap will not even try OS detection against hosts which do not meet this criteria. This can save substantial time, particularly on -Pn scans against many hosts. You still need to enable OS detection with -O (or -A) for the --osscan-limit option to have any effect.

Another OS detection option is --osscan-guess. When Nmap is unable to detect a perfect OS match, it sometimes offers up near-matches as possibilities. The match has to be very close for Nmap to do this by default. If you specify this option (or the equivalent --fuzzy option), Nmap will guess more aggressively. Nmap still tells you when an imperfect match is found and display its confidence level (percentage) for each guess.

When Nmap performs OS detection against a target and fails to find a perfect match, it usually repeats the attempt. By default, Nmap tries five times if conditions are favorable for OS f ingerprint submission, and twice when conditions aren't so good. The --max-os-tries option lets you change this maximum number of OS detection tries. Lowering it (usually to 1) speeds Nmap up, though you miss out on retries which could potentially identify the OS. Alternatively, a high value may be set to allow even more retries when conditions are favorable. This is rarely done, except to generate better fingerprints for submission and integration into the Nmap OS database.

Like just about every other part of Nmap, results ultimately come from the target machine itself. While rare, systems are occasionally configured to confuse or mislead Nmap. Several programs have even been developed specifically to trick Nmap OS detection (see [the section called “OS Spoofing”](https://nmap.org/book/nmap-defenses-trickery.html" \l "nmap-defenses-os-spoofing" \o "OS Spoofing)). Your best bet is to use numerous reconnaissance methods to explore a network, and don't trust any one of them.

TCP/IP fingerprinting requires collecting detailed information about the target's IP stack. The most commonly useful results, such as TTL information, are printed to Nmap output whenever they are obtained. Slightly less pertinent information, such as IP ID sequence generation and TCP sequence prediction difficulty, is only printed in verbose mode. But if you want all of the IP stack details that Nmap collected, you can find it in a compact form called a subject fingerprint. Nmap sometimes prints this (for user submission purposes) when it doesn't recognize a host. You can also force Nmap to print it (in normal, interactive, and XML formats) by enabling debugging with (-d). Then read **[the section called “Understanding an Nmap Fingerprint”](https://nmap.org/book/osdetect-fingerprint-format.html" \o "Understanding an Nmap Fingerprint)**to interpret it.