**Madanapalle Institute Of Technology And Science**

**(UGC-AUTONOMOUS)**

**(Affiliated to J.N.T.U.A, Anantapuramu and Approved by AICTE New Delhi)**

**An ISO 9001:2008 Certified Institution**

**Department of Computer Science & Engineering – Data Science**

**COMPUTER NETWORKS LABORATORY**



**Name of the Course Instructor : K Durga Charan,**

**Subject & Code : COMPUTER NETWORKS LABORATORY (20CSD115)**

**Academic Year/ Semester : 2022 – 23 / II Semester**

**Class / Section : III-CDS Section**

**Year & Semester : III & II**

**Vision**

To empower data science students with data driven insights that drive strategic decision making, optimize business operations, and create innovative solutions to complex problems.

**Mission**

* M1: To impart quality and value-based education and contribute towards the innovation of computing and Data Science to raise the satisfaction level of all stakeholders.
* M2: To enable students to get expertise in critical skills with data science education and facilitate socially responsive research and innovation.
* M3: To inculcate the best practices that lead to quality placements and personality development.

**Program Educational Objectives (PEOs)**

|  |  |
| --- | --- |
| PEO1 | To develop strong foundations in data science principles, methodologies, and tools. |
| PEO2 | To instill integrity and responsibility in handling data and making ethical decisions. |
| PEO3 | To foster innovation, problem-solving, and research contributions in the field |

**Program Outcomes (POs)**

|  |  |
| --- | --- |
| PO1 | Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and engineering specialization to the solution of complex engineering problems. |
| PO2 | Problem Analysis: Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences. |
| PO3 | Design/development of solutions: Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations. |
| PO4 | Conduct investigations of complex Problems: Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| PO5 | Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| PO6 | The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| PO9 | Individual and team work: Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings. |
| PO10 | Communication: Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions |
| PO11 | Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one’s own work, as a member and leader in a team. Manage projects in multidisciplinary environments. |
| PO12 | Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |

**Program Specific Outcomes (PSOs)**

|  |  |
| --- | --- |
| PSO1 | Demonstrate mastery in statistical analysis, machine learning, and visualization, extracting insights from complex datasets. |
| PSO2 | Apply data-driven approaches to solve intricate problems, communicate findings, and drive strategic outcomes. |

**Pre-requisite:** Nil

L T P C

0 0 3 1.5

**Course Description:**

This course helps the students to understand and implement OSI layers, simulation of various protocols, Error detection techniques, Routing and Congestion Algorithms simulation using NS, and protocol analysis, packet analysis, and traffic analysis using Wireshark.

**Course Objectives:**

1. To provide students with a theoretical and practical base in computer networks issues

2. Student will be able purse his study in advanced networking courses

3. Prepare students for easy transfer from academia into practical life

**List of Programs:**

1. Study of Basic Network Commands and Network Configuration commands.

2. Implement a program for OSI functionality to transmit data from client to server.

3. Implement a program for the following Encoding Techniques: NRZ, NRZ-I, Manchester

4. Implement a program for framing Techniques

a) Character Count b) Bit Stuffing and Destuffing c) Byte Stuffing and Destuffing

5. Implement a program for Flow control based on Sliding Window protocol

a) Go Back N ARQ b) Selective repeat ARQ

6. Implement a program for CRC polynomials.

7. Simulation of Transferring data between two nodes using NS.

8. Simulation of data transfer and packet loss using NS.

9. Simulation of Congestion Control Algorithm using NS.

10. Simulate a 3 nodes point-to-point network with duplex links between them. Set the queue size vary the bandwidth and find the number of packets dropped.

11. Simulate a 4 nodes point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant Applications over TCP and UDP agents changing the parameter and determining the number of packets sent by TCP/UDP.

12. Simulate an Ethernet LAN using N-nodes (6-10), change error rate and data rate and compare the throughput.

13. Protocol analysis with Wireshark.

14. Packet Capture & Traffic Analysis with Wireshark.

**Course Outcomes:**

After completing this course, the students should be able to

1. Understand and Implement OSI layers functionality.

2. Implement of Encoding and framing techniques

3. Implement error detection and correction techniques.

4. Simulate the various congestion control protocols using NS and learn NS tool.

5. Analyze packets and traffic using packet analyzer tool such as Wireshark.

**Textbook(s)**

1. Data Communications and Networking, Behrouz A. Forouzan, 6 , Tata McGraw Hill, 2021.

**Reference Books**

1. Computer Networking: A Top-Down Approach Featuring the Internet, James F.Kurose, K.W.Ross, Third Edition, Pearson Education
2. Understanding Communications and Networks, Third Edition, W.A.Shay, Cengage

Learning. Implementation and Management,6th Edition,2012.

**Mode of Evaluation:** Continuous Internal Evaluation and End Semester Examination

Table of Contents

[Experiment No- 1 7](#_Toc138501106)

[Experiment No- 2 18](#_Toc138501107)

[Experiment No- 3 24](#_Toc138501108)

[Experiment No- 4 27](#_Toc138501109)

[Experiment No- 5 31](#_Toc138501110)

[Experiment No- 6 45](#_Toc138501111)

[Experiment No- 7 52](#_Toc138501112)

[Experiment No- 8 55](#_Toc138501113)

[Experiment No- 9 58](#_Toc138501114)

[Experiment No- 10 61](#_Toc138501115)

[Experiment No- 11 63](#_Toc138501116)

[Experiment No- 12 65](#_Toc138501117)

[Experiment No- 13 68](#_Toc138501118)

[Experiment No- 14 84](#_Toc138501119)

# Experiment No- 1

## AIM:

Study of Basic Network Commands and Network Configuration commands.

## Procedure:

Basic Networking Commands are platform dependent. And some of the generic Commands to understanding.

## network_command_ipconfig1. IPCONFIG

**The IPCONFIG network command provides a comprehensive view of information regarding the IP address configuration of the device we are currently working on.**

**The IPConfig command also provides us with some variation in the primary command that targets specific system settings or data, which are:**

• IPConfig/all - Provides primary output with additional information about network adapters.

• IPConfig/renew - Used to renew the system’s IP address.

• IPConfig/release - Removes the system’s current IP address.

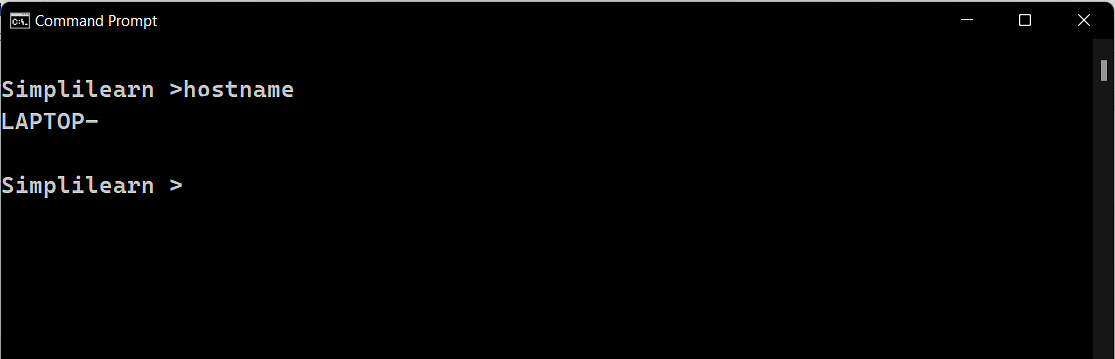
Command to enter in Prompt - ipconfig

## network_command_nslookup2. NSLOOKUP

The NSLOOKUP command is used to troubleshoot network connectivity issues in the system. Using the nslookup command, we can access the information related to our system’s DNS server, i.e., domain name and IP address.

Command to enter in Prompt – nslookup

## 3. HOSTNAME

The HOSTNAME command displays the hostname of the system. The hostname command is much easier to use than going into the system settings to search for it.

Command to enter in Prompt - hostname

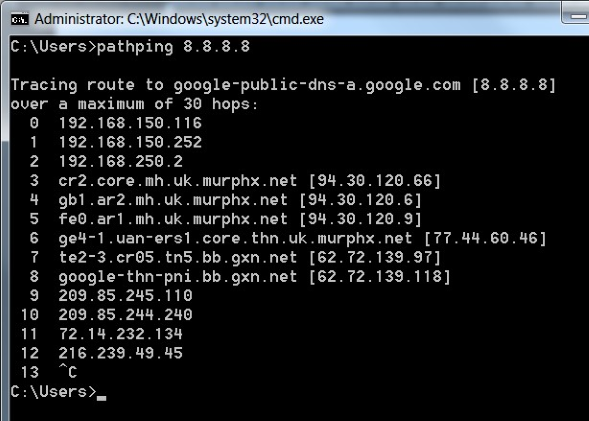
## network_command_ping4. PING

The Ping command is one of the most widely used commands in the prompt tool, as it allows the user to check the connectivity of our system to another host.

This command sends four experimental packets to the destination host to check whether it receives them successfully, if so, then, we can communicate with the destination host. But in case the packets have not been received, that means, no communication can be established with the destination host.

Command to enter in Prompt - ping

## 5. PATH PING

We discussed the Ping command and the Tracert command. There are similarities between these commands. The pathping command which provides a combination of the best aspects of Tracert and Ping. This command takes 300 seconds to gather statistics and then returns reports on latency and packet loss statistics at intermediate hops between the source and the target in more detail than those reports provided by Ping or Tracert commands.

The syntax for path ping is as follows:

path ping [-n] [-h] [-g <Hostlist>] [-p <Period>] [-q <NumQueries> [-w <timeout>] [-i <IPaddress>] [-4 <IPv4>] [-6 <IPv6>][<TargetName>]

* N: Prevents path ping functioning from attempting to resolve routers’ IP addresses to their names.
* -h MaxHops: This tracert option specifies the maximum number of hops in the search for the target. If the MaxHops option is not specified the target has not been found by 30 hops then the tracert command will stop looking.
* -w timeout: A timeout value must be specified while executing this ping command. It adjusts the amount of time in milliseconds.
* -ip <IPaddress>: Indicates the source address.
* target: This is the destination IP address or a hostname user want to ping.

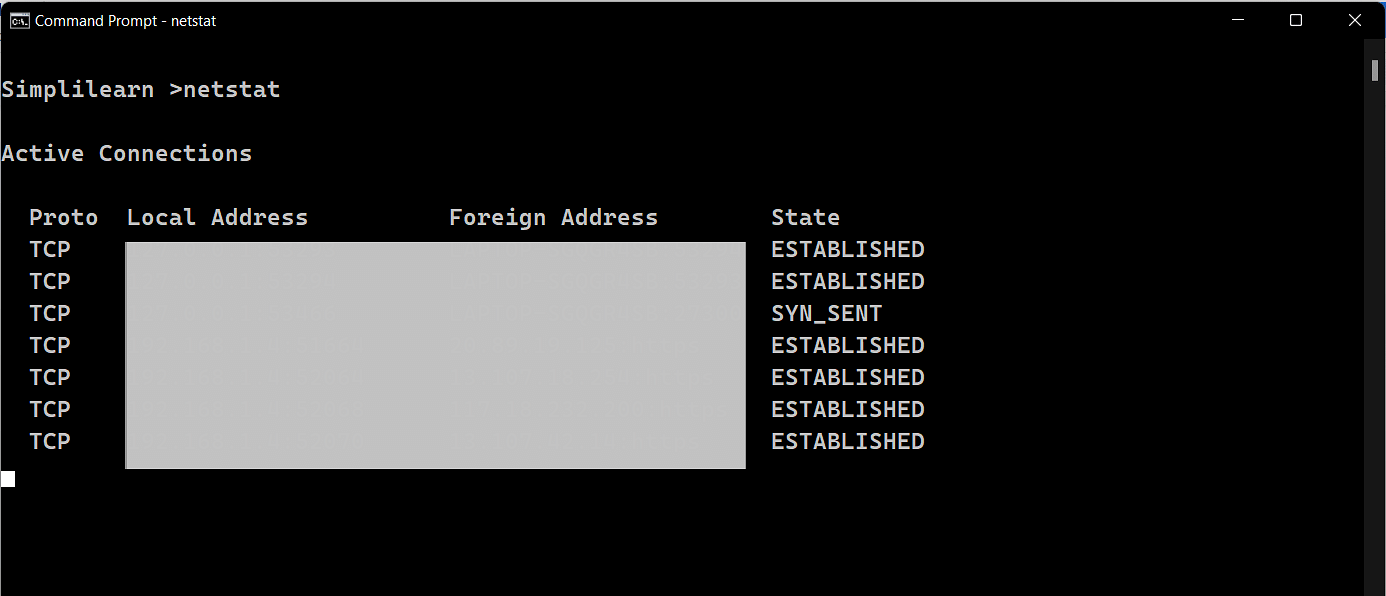
## 6. TRACERT

The TRACERT command is used to trace the route during the transmission of the data packet over to the destination host and also provides us with the “hop” count during transmission.

Using the number of hops and the hop IP address, we can troubleshoot network issues and identify the point of the problem during the transmission of the data packet.

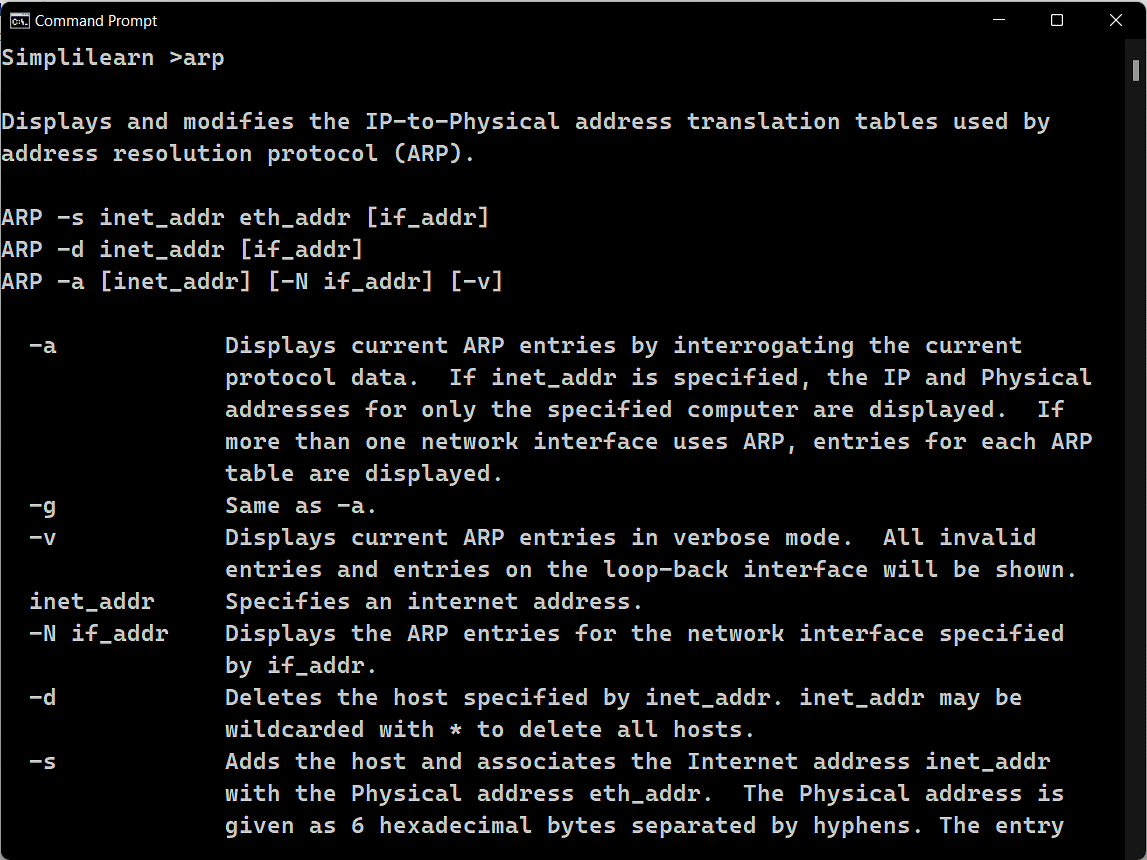
Command to enter in Prompt- tracert IP-address OR tracert www.destination\_host\_name.com

## 7. NETSTAT

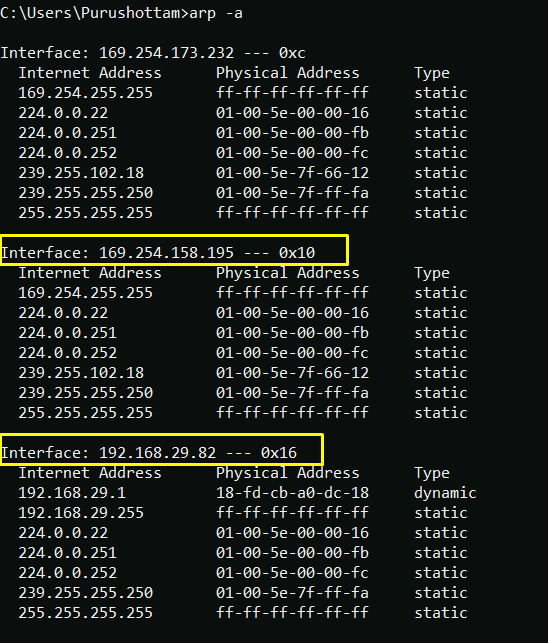
The Netstat command as the name suggests displays an overview of all the network connections in the device. The table shows detail about the connection protocol, address, and the current state of the network.

Command to enter in Prompt – netstat

## 8. ARP(Address Resolution Protocol)

The ARP command is used to access the mapping structure of IP addresses to the MAC address. This provides us with a better understanding of the transmission of packets in the network channel.

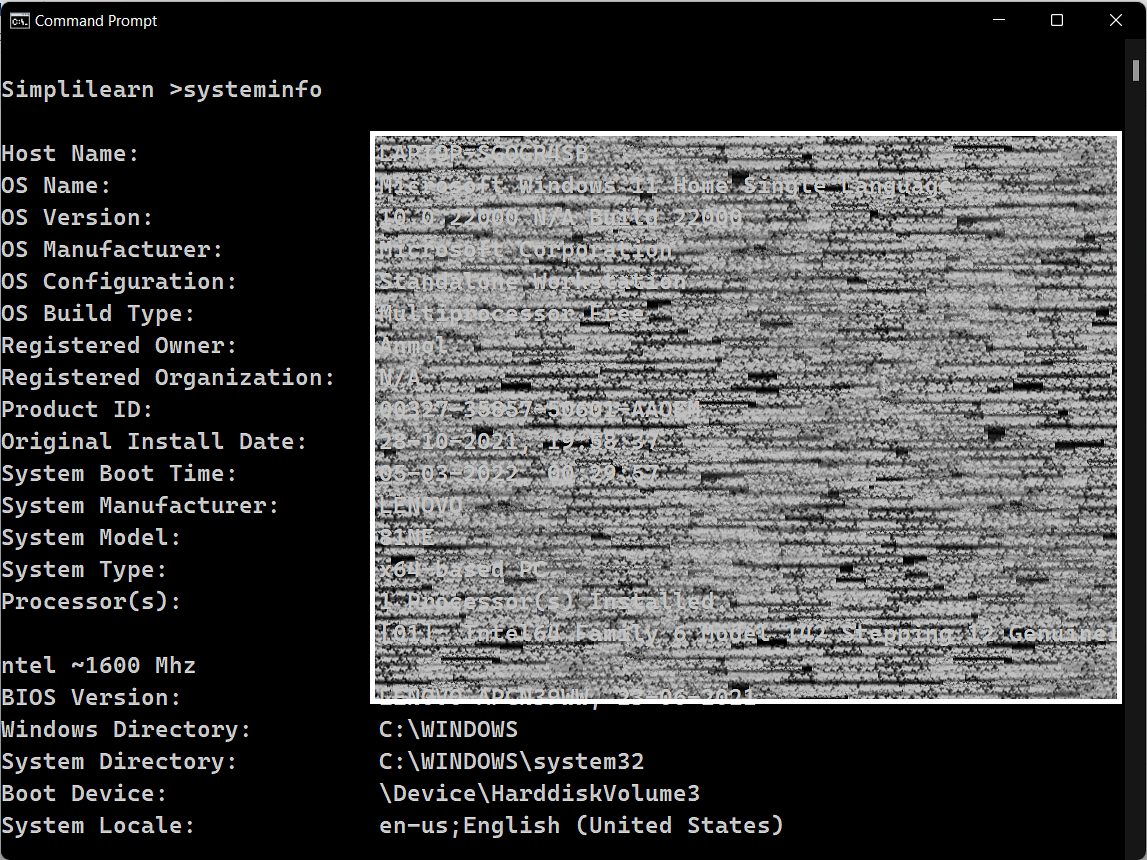
Command to enter in Prompt - arp



## 9. SYSTEMINFO

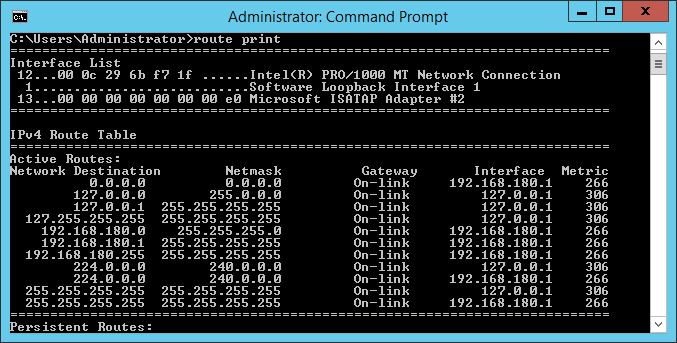
Using the SYSTEMINFO command, we can access the system’s hardware and software details, such as processor data, booting data, Windows version, etc.

Command to enter in Prompt – systeminfo



## 10 ROUTE

In IP networks, routing tables are used to direct packets from one subnet to another. The Route command provides the device’s routing tables. To get this result, just type route print. The Route command returns the routing table, and the user can make changes by Commands such as Route Add, Route Delete, and Route Change, which allows modifying the routing table as a requirement.



## 11. GETMAC COMMAND

Another very simple command that shows the MAC address of your network interfaces



## 12 NBTSTAT

Nbtstat is designed to help troubleshoot NetBIOS name resolution problems. When a network is functioning normally, NetBIOS over TCP/IP (NetBT) resolves NetBIOS names to IP addresses. It does this through several options for NetBIOS name resolution, including local cache lookup, WINS server query, broadcast, LMHOSTS lookup, Hosts lookup, and DNS server query.

The nbtstat command removes and corrects preloaded entries using a number of case-sensitive switches. The nbtstat - a <name>\*\* command performs a NetBIOS adapter status command on the computer name specified by < name> . The adapter status command returns the local NetBIOS name table for that computer as well as the MAC address of the adapter card. The nbtstat -A < IP address > command performs the same function using a target IP address rather than a name.

The nbtstat - c option shows the contents of the NetBIOS name cache, which contains NetBIOS name-to-IP address mappings.

nbtstat -n displays the names that have been registered locally on the system by NetBIOS applications such as the server and redirector.

The nbtstat -r command displays the count of all NetBIOS names resolved by broadcast and by querying a WINS server. The nbtstat -R command purges the name cache and reloads all #PRE entries from the LMHOSTS file. #PRE entries are the LMHOSTS name entries that are preloaded into the cache. For more information about the LMHOSTS file, see the appendix "LMHOSTS Files" in this book.

Nbtstat -RR sends name release packets to the WINS server and starts a refresh, thus re-registering all names with the name server without having to reboot. This is a new option in Windows NT 4.0 with Service Pack 4 as well as in Windows 2000.

You can use nbtstat -S to list the current NetBIOS sessions and their status, including statistics. Sample output looks like this:

C:\>nbtstat -S

Local Area Connection:

Node IpAddress: [172.16.0.142] Scope Id: []

NetBIOS Connection Table

Local Name State In/Out Remote Host Input Output

------------------------------------------------------------------

TESTPC1 <00> Connected Out 172.16.210.25 6MB 5MB

TESTPC1 <00> Connected Out 172.16.3.1 108KB 116KB

TESTPC1 <00> Connected Out 172.16.3.20 299KB 19KB

TESTPC1 <00> Connected Out 172.16.3.4 324KB 19KB

TESTPC1 <03> Listening

Finally, nbtstat -s provides a similar set of session listings, but provides the remote computer names, rather than their IP addresses.

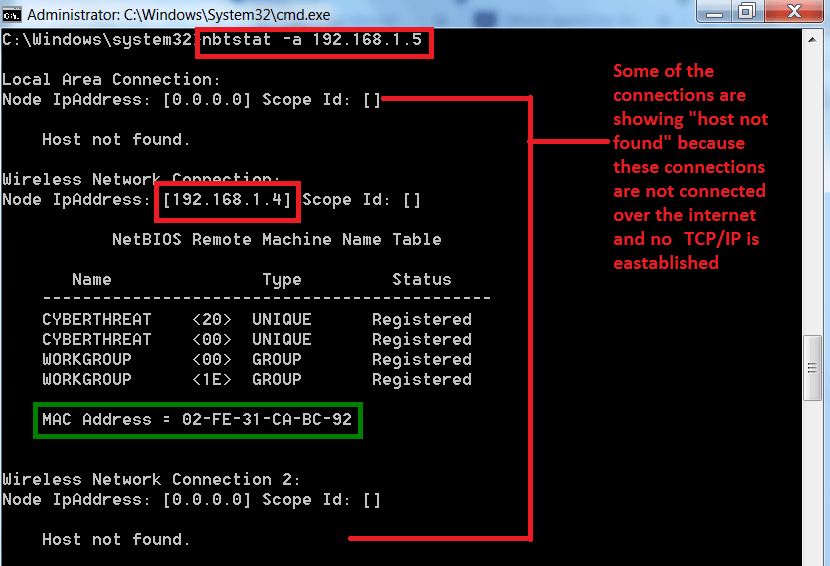
note-iconNote

The options for the Nbtstat command are case sensitive.

The Nbtstat switches are listed in Table 3.4.

**Nbtstat Switches**

| **Switch** | **Name** | **Function** |
| --- | --- | --- |
| -a < *name* > | adapter status | Returns the NetBIOS name table and MAC address of the address card for the computer name specified. |
| -A < *IP address* > | Adapter status | Lists the same information as -a when given the target's IP address. |
| -c | cache | Lists the contents of the NetBIOS name cache. |
| [ *Number* ] | Interval | Typing a numerical value tells Nbtstat to redisplay selected statistics each interval seconds, pausing between each display. Press Ctrl+C to stop redisplaying statistics. |
| -n | names | Displays the names registered locally by NetBIOS applications such as the server and redirector. |
| -r | resolved | Displays a count of all names resolved by broadcast or WINS server. |
| -R | Reload | Purges the name cache and reloads all #PRE entries from LMHOSTS. |
| -RR | ReleaseRefresh | Releases and reregisters all names with the name server. |
| -s | sessions | Lists the NetBIOS sessions table converting destination IP addresses to computer NetBIOS names. |
| -S | Sessions | Lists the current NetBIOS sessions and their status, with the IP address. |
| /? | Help | Displays this list. |



## 13.TASKLIST

Open up an Administrative level Command Prompt and run tasklist to see all of the running processes:

C:\>tasklist

Image Name PID Session Name Mem Usage

========================= ======== ================ ============

firefox.exe 26356 Console 139,352 K

regedit.exe 24244 Console 9,768 K

cmd.exe 18664 Console 2,380 K

conhost.exe 2528 Console 7,852 K

notepad.exe 17364 Console 7,892 K

notepad.exe 24696 Console 22,028 K

notepad.exe 25304 Console 5,852 K

explorer.exe 2864 Console 72,232 K

In the example above you can see the image name and the PID for each process. If you want to kill the firefox process run:

C:\>Taskkill /IM firefox.exe /F

or

C:\>Taskkill /PID 26356 /F

The /f flag is kills the process forcefully. Failure to use the /F flag will result in nothing happening in some cases. One example is whenever I want to kill the explorer.exe process I have to use the /F flag or else the process just does not terminate.

If you have multiple instances of an image open such as multiple firefox.exe processes, running the taskkill /IM firefox.exe command will kill all instances. When you specify the PID only the specific instane of firefox will be terminated.

The real power of taskkill are the filtering options that allow you to use the following variables and operators.

Variables:

STATUS, IMAGENAME,,PID,,SESSION,,CPUTIME,,MEMUSAGE,,USERNAME,,MODULES,SERVICES

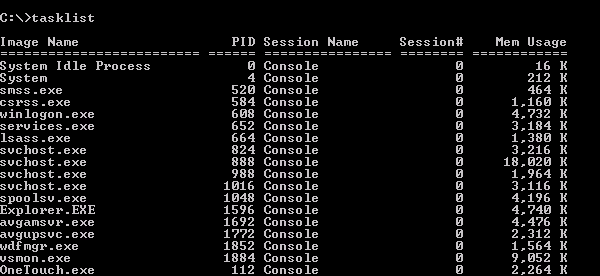
WINDOWTITLE

Operators:

eq (equals),,ne (not equal),gt (greater than),lt (less than),ge (greater than or equal),le (less than or equal)

"\*" is the wildcard.

You can use the variables and operators with the /FI filtering flag. For example, let's say you want to end all processes that have a window title that starts with "Internet":



## 14.WGET

Command wget stands for web get. The wget is a free non-interactive file downloader command. Non-interactive means it can work in background when user is not logged in. This allows user to get disconnected with the system while wget finish its work.

It can even download entire website as a local version of remote websites, fully recreating the structure of original website. In short, you can mirror an entire website with wget.

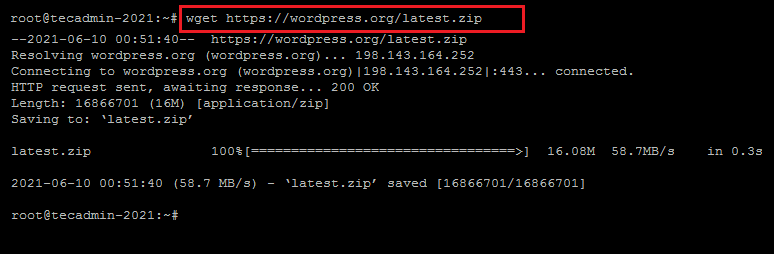
It supports HTTP, HTTPs and FTP protocol. It is a freely available package and licensed under GNU GPL. This command works on all operating systems including MAC Os and Windows.

The main feature of wget is its robustness and recursiveness. Due to its robustness, it work even in slow internet connection. It automatically starts downloading the file from where it was left in case of network failure. Due to its recursiveness, it keeps trying until the file is retrieved completely.

to use wget, you have to give location of the file over HTTP as the wget argument. File will be downloaded in the current directory.

wget Options

|  |  |
| --- | --- |
| **Option** | **Function** |
| wget <URL> | Download single file |
| wget -O <fileName> <URL> | Store with a different file name |
| wget --limit-rate=<Numberk> <URL> | Specify download rate/speed |
| wget -c <URL> | Complete the remaining downloaded file |
| wget -b <URL> | Download in background |
| wget --tries=<Number> <URL> | Set retrying attempts |
| wget -i <fileName> | Download multiple files |
| wget --mirror -p --convert-links -P ./local dir <webURL> | Download full website |
| wget --reject=<rejectingFile> <URL> | Reject a type of file |
| wget -Q<Value>m -i <fileName> | Quit downloading on exceeding certain limit |
| wget -r -A.<fileType> <webURL> | Download certain file type |



# Experiment No- 2

## AIM:

Implement a program for OSI functionality to transmit data from client to server

## Procedure:

TCP sockets are used for communication between a server and a client process. The server’s code runs first, which opens a port and listens for incoming connection requests from clients. Once a client connects to the same (server) port, the client or server may send a message. Once the message is sent, whoever receives it (server or client) will process it accordingly.

Include the header files sys/socket.h and arpa/inet.h:

#include <sys/socket.h>

#include <arpa/inet.h>

Create a socket that returns a socket descriptor; this will be used to refer to the socket later on in the code:

int socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

The server-side code keeps the address information of both the server and the client in a variable of type sockaddr\_in, which is a struct.

Initialize the server address by the port and IP:

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

Bind the socket descriptor to the server address:

bind(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr);

Turn on the socket to listen for incoming connections:

listen(socket\_desc, 1);

Store the client’s address and socket descriptor by accepting an incoming connection:

struct sockaddr client\_addr;

int client\_size = sizeof(client\_addr);

int client\_sock = accept(socket\_desc, (struct sockaddr\*)&client\_addr, &client\_size);

The server-side code stops and waits at accept() until a client calls connect().

Communicate with the client using send() and recv():

recv(client\_sock, client\_message, sizeof(client\_message), 0);

send(client\_sock, server\_message, strlen(server\_message), 0);

When recv() is called, the code stops and waits for a message from the client.

Close the server and client socket to end communication:

close(client\_sock);

close(socket\_desc);

## Server-side CODE:

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

int main(void)

{

int socket\_desc, client\_sock, client\_size;

struct sockaddr\_in server\_addr, client\_addr;

char server\_message[2000], client\_message[2000];

// Clean buffers:

memset(server\_message, '\0', sizeof(server\_message));

memset(client\_message, '\0', sizeof(client\_message));

// Create socket:

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if(socket\_desc < 0){

printf("Error while creating socket\n");

return -1;

}

printf("Socket created successfully\n");

// Set port and IP:

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Bind to the set port and IP:

if(bind(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr))<0){

printf("Couldn't bind to the port\n");

return -1;

}

printf("Done with binding\n");

// Listen for clients:

if(listen(socket\_desc, 1) < 0){

printf("Error while listening\n");

return -1;

}

printf("\nListening for incoming connections.....\n");

// Accept an incoming connection:

client\_size = sizeof(client\_addr);

client\_sock = accept(socket\_desc, (struct sockaddr\*)&client\_addr, &client\_size);

if (client\_sock < 0){

printf("Can't accept\n");

return -1;

}

printf("Client connected at IP: %s and port: %i\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

// Receive client's message:

if (recv(client\_sock, client\_message, sizeof(client\_message), 0) < 0){

printf("Couldn't receive\n");

return -1;

}

printf("Msg from client: %s\n", client\_message);

// Respond to client:

strcpy(server\_message, "This is the server's message.");

if (send(client\_sock, server\_message, strlen(server\_message), 0) < 0){

printf("Can't send\n");

return -1;

}

// Closing the socket:

close(client\_sock);

close(socket\_desc);

return 0;

}

## Client-side CODE:

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

int main(void)

{

int socket\_desc;

struct sockaddr\_in server\_addr;

char server\_message[2000], client\_message[2000];

// Clean buffers:

memset(server\_message,'\0',sizeof(server\_message));

memset(client\_message,'\0',sizeof(client\_message));

// Create socket:

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if(socket\_desc < 0){

printf("Unable to create socket\n");

return -1;

}

printf("Socket created successfully\n");

// Set port and IP the same as server-side:

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Send connection request to server:

if(connect(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0){

printf("Unable to connect\n");

return -1;

}

printf("Connected with server successfully\n");

// Get input from the user:

printf("Enter message: ");

gets(client\_message);

// Send the message to server:

if(send(socket\_desc, client\_message, strlen(client\_message), 0) < 0){

printf("Unable to send message\n");

return -1;

}

// Receive the server's response:

if(recv(socket\_desc, server\_message, sizeof(server\_message), 0) < 0){

printf("Error while receiving server's msg\n");

return -1;

}

printf("Server's response: %s\n",server\_message);

// Close the socket:

close(socket\_desc);

return 0;

}

## Procedure for Client:

Include header files, create a socket, and initialize the server’s address information in a variable of type sockaddr\_in, similar to how it was done at the server-side:

#include <sys/socket.h>

#include <arpa/inet.h>

int socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

Send a connection request to the server, which is waiting at accept():

connect(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

Communicate with the server using send() and recv():

send(socket\_desc, client\_message, strlen(client\_message),0);

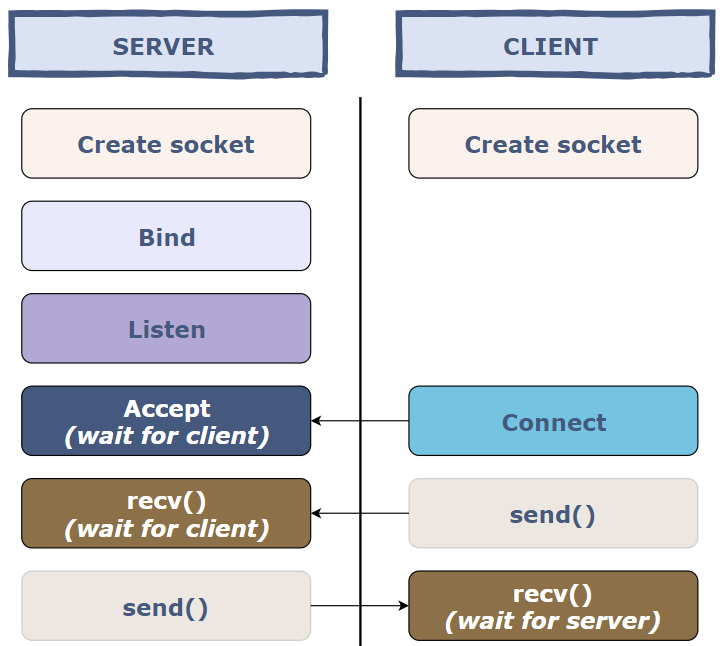
recv(socket\_desc, server\_message, sizeof(server\_message),0);

The client waits for the server to send a message when recv() is called.

Close the socket:

close(socket\_desc);

A deadlock will occur if both the client and the server are waiting for each other’s message at recv().



## Output:

A computer screen shot of a program

Description automatically generated with low confidence

A screenshot of a computer error

Description automatically generated with medium confidence

# Experiment No- 3

## AIM:

Implement a program for the following Encoding Techniques: NRZ, NRZ-I, Manchester

## Procedure:

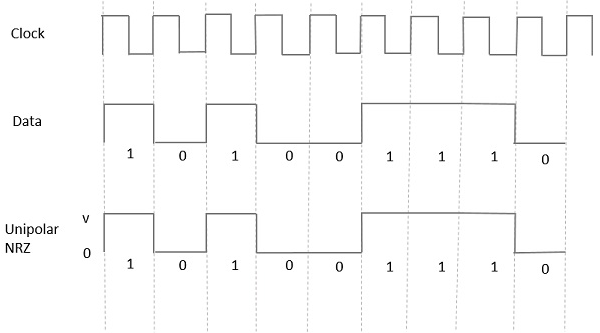
Types of Line Coding

There are 3 types of Line Coding

* Unipolar
* Polar
* Bi-polar

Unipolar Signaling

Unipolar signaling is also called as On-Off Keying or simply OOK.

The presence of pulse represents a 1 and the absence of pulse represents a 0.

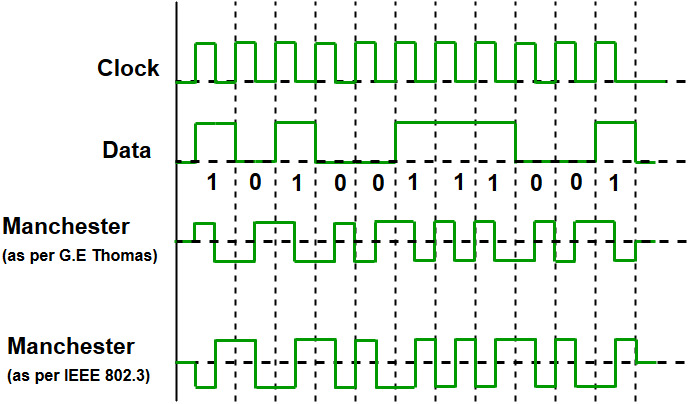
There are two variations in Unipolar signaling −

* Non Return to Zero NRZ
* Return to Zero RZ

Unipolar Non-Return to Zero NRZ

In this type of unipolar signaling, a High in data is represented by a positive pulse called as Mark, which has a duration T0 equal to the symbol bit duration. A Low in data input has no pulse.

Manchester encoding is a synchronous clock encoding technique used by the physical layer of the Open System Interconnection [OSI] to encode the clock and data of a synchronous bit stream. The idea of RZ and the idea of-L are combined in manchester

Different encoding techniques are used in data communication to ensure data security and transmission speed. Manchester encoding is an example of digital encoding. Because each data bit length is defined by default, it differs from other digital encoding schemes. The bit state is defined by the direction of the transition. Bit status is represented in various ways by different systems, although most systems use 1 bit for low to high transitions and 0 bit for high to low transitions.

In manchester duration of a bit is divided into two halves. The voltage remains the same at one level during the first half & moves to the other level. The transition at the middle of the bit provides synchronization. Differential Manchester, on the other hand, combines the idea of RZ and NRZ-I. There is always a transition at the middle of the bit, but the bit values are determined at the beginning of the bit. if next bit is zero there is transition if next bit is 1 there is none.

The binary data to be transmitted over the cable are not sent as NRZ [Non-return-to-zero].

Non-return-to-zero [NRZ] –

NRZ code’s voltage level is constant during a bit interval. When there is a long sequence of 0s and 1s, there is a problem at the receiving end. The problem is that the synchronization is lost due to a lack of transmissions.

It is of 2 types:

* NRZ-level encoding –

The polarity of signals changes when the incoming signal changes from ‘1’ to ‘0’ or from ‘0’ to ‘1’. It considers the first bit of data as polarity change.

* NRZ-Inverted/ Differential encoding –

In this, the transitions at the beginning of the bit interval are equal to 1 and if there is no transition at the beginning of the bit interval is equal to 0.

## Python Program for NRZ :

import matplotlib.pyplot as plt

data = "10110111001"

y = []

for bit in data:

if bit == '0':

y += [0]\*10

else:

y += [1]\*10

plt.plot( y )

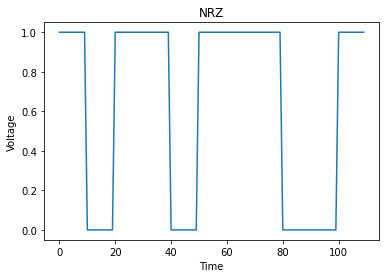
plt.title('NRZ')

plt.ylabel('Voltage')

plt.xlabel('Time')

plt.show()

## Expected Output:



## Python Program for NRZ-I:

import matplotlib.pyplot as plt

data = "101101010001"

y = [1]

for i, bit in enumerate(data):

if bit == '0':

y += [y[i]]\*10

else:

y += [int(not y[i])]\*10

plt.plot( y )

plt.title('NRZi')

plt.ylabel('Voltage')

plt.xlabel('Time')

plt.show()

## Expected Output:

Icon

Description automatically generated with medium confidence

# Experiment No- 4

## AIM:

Implement a program for framing Techniques.

a) Character Count b) Bit Stuffing and Destuffing c) Byte Stuffing and Destuffing

## Character Count

### Algorithm

1. Define a string.
2. Define and initialize a variable count to 0.
3. Iterate through the string till the end and for each character except spaces, increment the count by 1.
4. To avoid counting the spaces check the condition i.e. string[i] != ' '.

#### Python Code

string = "The best of both worlds";

count = 0;

#Counts each character except space

for i in range(0, len(string)):

if(string[i] != ' '):

count = count + 1;

#Displays the total number of characters present in the given string

print("Total number of characters in a string: " + str(count));

## Bit Stuffing

Bit Stuffing is a process of inserting an extra bit as 0, once the frame sequence encountered 5 consecutive 1’s. Given an array, arr[] of size N consisting of 0’s and 1’s, the task is to return an array after the bit stuffing.

Examples:

Input: N = 6, arr[] = {1, 1, 1, 1, 1, 1}

Output: 1111101

Explanation: During the traversal of the array, 5 consecutive 1’s are encountered after the 4th index of the given array. Hence, a zero bit has been inserted into the stuffed array after the 4th index

Input: N = 6, arr[] = {1, 0, 1, 0, 1, 0}

Output: 101010

Approach: The idea is to check if the given array consists of 5 consecutive 1’s. Follow the steps below to solve the problem:

Initialize the array brr[] which stores the stuffed array. Also, create a variable count which maintains the count of the consecutive 1’s. Traverse in a while loop using a variable i in the range [0, N) and perform the following tasks: If arr[i] is 1 then check for the next 4 bits if they are set bits as well. If they are, then insert a 0 bit after inserting all the 5 set bits into the array brr[]. Otherwise, insert the value of arr[i] into the array brr[].

#### Python Code

# Function for bit stuffing

def bitStuffing(N, arr):

# Stores the stuffed array

brr = [0 for \_ in range(30)]

# Variables to traverse arrays

k = 0

i = 0

j = 0

# Loop to traverse in the range [0, N)

while (i < N):

# If the current bit is a set bit

if (arr[i] == 1):

count = 1

# Insert into array brr[]

brr[j] = arr[i]

# Loop to check for

# next 5 bits

k = i + 1

while True:

if not (k < N and arr[k] == 1 and

count < 5):

break

j += 1

brr[j] = arr[k]

count += 1

# If 5 consecutive set bits

# are found insert a 0 bit

if (count == 5):

j += 1

brr[j] = 0

i = k

k += 1

# Otherwise insert arr[i] into

# the array brr[]

else:

brr[j] = arr[i]

i += 1

j += 1

# Print Answer

for i in range(0, j):

print(brr[i], end="")

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

N = 6

arr = [1, 1, 1, 1, 1, 1]

bitStuffing(N, arr)

**Output**

1111101

Time Complexity: O(N)

Auxiliary Space: O(N)

## Destuffing

Bit Destuffing or Bit Unstuffing is a process of undoing the changes in the array made during the bit stuffing process i.e, removing the extra 0 bit after encountering 5 consecutive 1’s.

Examples:

Input: N = 7, arr[] = {1, 1, 1, 1, 1, 0, 1}

Output: 111111

Explanation: During the traversal of the array, 5 consecutive 1’s are encountered after the 4th index of the given array. Hence, the next 0 bit must be removed to de-stuffed array.

Input: N = 6, arr[] = {1, 0, 1, 0, 1, 0}

Output: 101010

Approach: This problem can be solved similarly to the bit stuffing problem. The only required change in the above-discussed approach is whenever 5 consecutive 1’s are encountered, skip the next bit in the array arr[] in place of inserting a 0 bit in the array brr[].

#### Python Code

# Function for bit de-stuffing

def bitDestuffing(N, arr):

# Stores the de-stuffed array

brr = [0 for i in range(30)];

# Variables to traverse the arrays

k = 0;

i = 0;

j = 0;

# Stores the count of consecutive ones

count = 1;

# Loop to traverse in the range [0, N)

while (i < N):

# If the current bit is a set bit

if (arr[i] == 1):

# Insert into array brr

brr[j] = arr[i];

# Loop to check for

# the next 5 bits

for k in range(i + 1, k < N and arr[k] == 1 and count < 5,1):

j += 1;

brr[j] = arr[k];

count += 1;

# If 5 consecutive set

# bits are found skip the

# next bit in arr

if (count == 5):

k += 1;

i = k;

# Otherwise insert arr[i] into

# the array brr

else:

brr[j] = arr[i];

i += 1;

j += 1;

# Print Answer

for i in range(0, j):

print(brr[i],end="");

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

N = 7;

arr = [1, 1, 1, 1, 1, 0, 1];

bitDestuffing(N, arr);

Output

111111

Time Complexity: O(N)

Auxiliary Space: O(N)

# Experiment No- 5

## AIM:

Implement a program for Flow control based on Sliding Window protocol.

a) Go Back N ARQ b) Selective repeat ARQ

## Procedure:

Selective-repeat Automatic Repeat Request (ARQ) is one of the techniques where a data link layer may deploy to control errors.

Techniques to control ARQ

Generally, there are three types of techniques which control the errors by Automatic Repeat Request (ARQ) they are −

* Selective Repeat ARQ
* Go-Back-N ARQ
* Stop-and-wait ARQ

Requirements for Error Control

There are some requirements for error control mechanisms and they are as follows −

* Error detection − The sender and receiver, or any must ascertain that there is some error in the transit.
* Positive ACK − Whenever a receiver receives a correct frame, it should acknowledge it.
* Negative ACK − Whenever the receiver receives a damaged frame or a duplicate frame, it sends a NACK back to the sender and sender must retransmit the correct frame.
* Retransmission − The sender always maintains a clock and sets a timeout period. If an ACK of data-frame previously transmitted does not arrive before the timeout, the sender retransmits the frame, thinking that the frame or it’s ACK is lost in transit

## Selective Repeat Protocol:

The Selective Repeat protocol is another sliding window protocol used for reliable data transfer in computer networks. It is a receiver-based protocol that allows the receiver to acknowledge each packet individually, rather than a cumulative acknowledgement of a sequence of packets. The sender sends packets in a window and waits for acknowledgements for each packet in the window. If a packet is lost, the receiver sends a NACK for the lost packet, and the sender retransmits only that packet. The sender also maintains a timer for each packet, and if an acknowledgement is not received within the timer’s timeout period, the sender retransmits only that packet.

### key features include:

Receiver-based protocol

Each packet is individually acknowledged by the receiver

Only lost packets are retransmitted, reducing network congestion

Maintains a buffer to store out-of-order packets

Requires more memory and processing power than Go-Back-N

Provides efficient transmission of packets.

It is also known as Sliding Window Protocol and used for error detection and control in the data link layer.

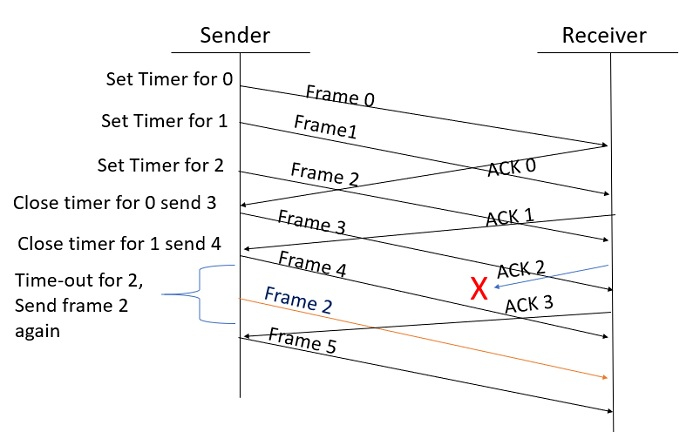
In the selective repeat, the sender sends several frames specified by a window size even without the need to wait for individual acknowledgement from the receiver as in Go-Back-N ARQ. In selective repeat protocol, the retransmitted frame is received out of sequence.

In Selective Repeat ARQ only the lost or error frames are retransmitted, whereas correct frames are received and buffered.

The receiver while keeping track of sequence numbers buffers the frames in memory and sends NACK for only frames which are missing or damaged. The sender will send/retransmit a packet for which NACK is received.

### Example

Given below is an example of the Selective Repeat ARQ –



### Explanation

Step 1 − Frame 0 sends from sender to receiver and set timer.

Step 2 − Without waiting for acknowledgement from the receiver another frame, Frame1 is sent by sender by setting the timer for it.

Step 3 − In the same way frame2 is also sent to the receiver by setting the timer without waiting for previous acknowledgement.

Step 4 − Whenever sender receives the ACK0 from receiver, within the frame 0 timer then it is closed and sent to the next frame, frame 3.

Step 5 − whenever the sender receives the ACK1 from the receiver, within the frame 1 timer then it is closed and sent to the next frame, frame 4.

Step 6 − If the sender doesn’t receive the ACK2 from the receiver within the time slot, it declares timeout for frame 2 and resends the frame 2 again, because it thought the frame2 may be lost or damaged.

## Go-Back-N Protocol:

The Go-Back-N protocol is a sliding window protocol used for reliable data transfer in computer networks. It is a sender-based protocol that allows the sender to transmit multiple packets without waiting for an acknowledgement for each packet. The receiver sends a cumulative acknowledgement for a sequence of packets, indicating the last correctly received packet. If any packet is lost, the receiver sends a negative acknowledgement (NACK) for the lost packet, and the sender retransmits all the packets in the window starting from the lost packet. The sender also maintains a timer for each packet, and if an acknowledgement is not received within the timer’s timeout period, the sender retransmits all packets in the window.

### The key features of the Go-Back-N (GBN) protocol include:

• Sliding window mechanism

• Sequence numbers

• Cumulative acknowledgements

• Timeout mechanism

• NACK mechanism

• Simple implementation.

In Go-Back-N ARQ, N is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver.

It uses the principle of protocol pipelining in which the multiple frames can be sent before receiving the acknowledgment of the first frame. If we have five frames and the concept is Go-Back-3, which means that the three frames can be sent, i.e., frame no 1, frame no 2, frame no 3 can be sent before expecting the acknowledgment of frame no 1.

In Go-Back-N ARQ, the frames are numbered sequentially as Go-Back-N ARQ sends the multiple frames at a time that requires the numbering approach to distinguish the frame from another frame, and these numbers are known as the sequential numbers.

The number of frames that can be sent at a time totally depends on the size of the sender's window. So, we can say that 'N' is the number of frames that can be sent at a time before receiving the acknowledgment from the receiver.

If the acknowledgment of a frame is not received within an agreed-upon time period, then all the frames available in the current window will be retransmitted. Suppose we have sent the frame no 5, but we didn't receive the acknowledgment of frame no 5, and the current window is holding three frames, then these three frames will be retransmitted.

The sequence number of the outbound frames depends upon the size of the sender's window. Suppose the sender's window size is 2, and we have ten frames to send, then the sequence numbers will not be 1,2,3,4,5,6,7,8,9,10. Let's understand through an example.

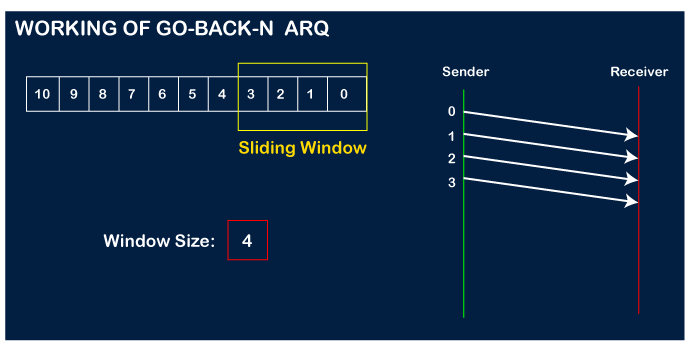
* N is the sender's window size.
* If the size of the sender's window is 4 then the sequence number will be 0,1,2,3,0,1,2,3,0,1,2, and so on.

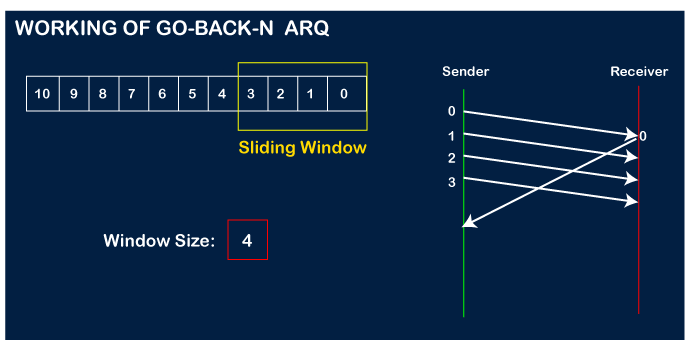
The number of bits in the sequence number is 2 to generate the binary sequence 00,01,10,11.

### Explanation

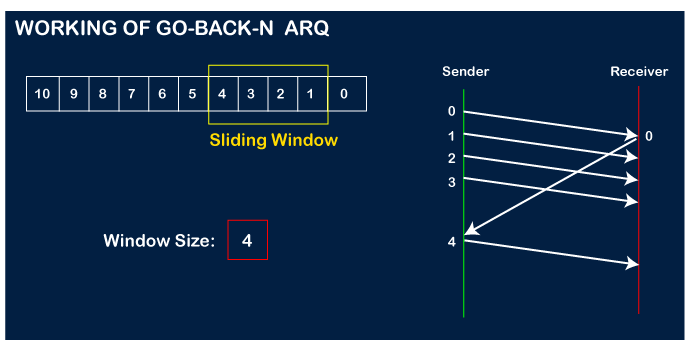
Suppose there are a sender and a receiver, and let's assume that there are 11 frames to be sent. These frames are represented as 0,1,2,3,4,5,6,7,8,9,10, and these are the sequence numbers of the frames. Mainly, the sequence number is decided by the sender's window size. But, for the better understanding, we took the running sequence numbers, i.e., 0,1,2,3,4,5,6,7,8,9,10. Let's consider the window size as 4, which means that the four frames can be sent at a time before expecting the acknowledgment of the first frame.

**Step 1:** Firstly, the sender will send the first four frames to the receiver, i.e., 0,1,2,3, and now the sender is expected to receive the acknowledgment of the 0th frame.

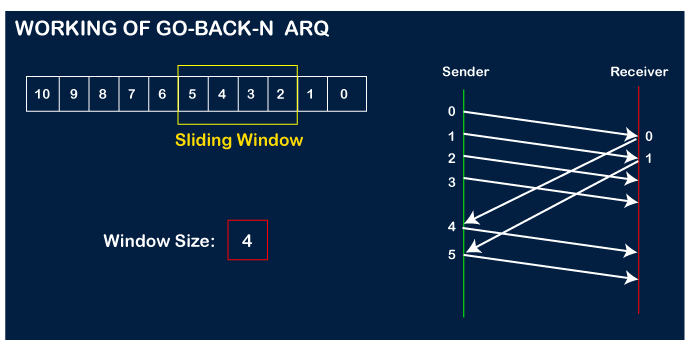
 Let's assume that the receiver has sent the acknowledgment for the 0 frame, and the receiver has successfully received it.



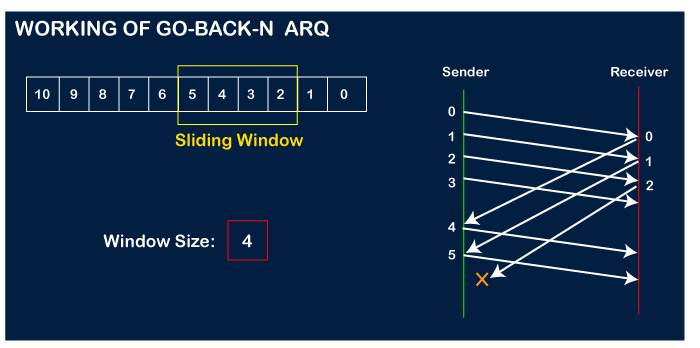
The sender will then send the next frame, i.e., 4, and the window slides containing four frames (1,2,3,4).

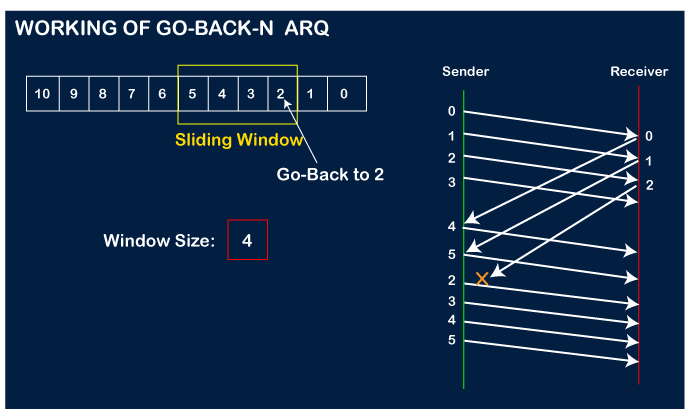


The receiver will then send the acknowledgment for the frame no 1. After receiving the acknowledgment, the sender will send the next frame, i.e., frame no 5, and the window will slide having four frames (2,3,4,5).



Now, let's assume that the receiver is not acknowledging the frame no 2, either the frame is lost, or the acknowledgment is lost. Instead of sending the frame no 6, the sender Go-Back to 2, which is the first frame of the current window, retransmits all the frames in the current window, i.e., 2,3,4,5.





## Similarities between the two protocols are:

Both protocols use a sliding window mechanism to allow the sender to transmit multiple packets without waiting for an acknowledgement for each packet.

Both protocols use sequence numbers to ensure the correct order of packets.

Both protocols use a timer mechanism to handle lost or corrupted packets.

Both protocols can retransmit packets that are not acknowledged by the receiver.

Both protocols can reduce network congestion by only retransmitting lost packets.

Both protocols are widely used in modern communication networks.

|  |  |  |
| --- | --- | --- |
| S.NO | Go-Back-N Protocol | Selective Repeat Protocol |
| 1 | [In Go-Back-N Protocol, if the sent frame are find suspected then all the frames are re-transmitted from the lost packet to the last packet transmitted.](https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/) | [In selective Repeat protocol, only those frames are re-transmitted which are found suspected.](https://www.geeksforgeeks.org/sliding-window-protocol-set-3-selective-repeat/) |
| 2 | Sender window size of Go-Back-N Protocol is N. | Sender window size of selective Repeat protocol is also N. |
| 3 | Receiver window size of Go-Back-N Protocol is 1. | Receiver window size of selective Repeat protocol is N. |
| 4 | Go-Back-N Protocol is less complex. | Selective Repeat protocol is more complex. |
| 5 | In Go-Back-N Protocol, neither sender nor at receiver need sorting. | In selective Repeat protocol, receiver side needs sorting to sort the frames. |
| 6 | In Go-Back-N Protocol, type of Acknowledgement is cumulative. | In selective Repeat protocol, type of Acknowledgement is individual. |
| 7 | In Go-Back-N Protocol, Out-of-Order packets are NOT Accepted (discarded) and the entire window is re-transmitted. | In selective Repeat protocol, Out-of-Order packets are Accepted. |
| 8 | In Go-Back-N Protocol, if Receives  a corrupt packet, then also, the entire window is re-transmitted. | In selective Repeat protocol, if Receives  a corrupt packet, it immediately sends a negative acknowledgement and hence only the selective packet is retransmitted. |
| 9 | Efficiency of Go-Back-N Protocol is | Efficiency of selective Repeat protocol is also |
| N/(1+2\*a) | N/(1+2\*a) |

## Programs

## Go-Back-N Protocol

### receiver.py

# Receiver.py

import time, socket, sys

import random

print("\nWelcome to Chat Room\n")

print("Initialising....\n")

time.sleep(1)

s = socket.socket()

shost = socket.gethostname()

ip = socket.gethostbyname(shost)

print(shost, "(", ip, ")\n")

host = input(str("Enter server address: "))

name = input(str("\nEnter your name: "))

port = 1234

print("\nTrying to connect to ", host, "(", port, ")\n")

time.sleep(1)

s.connect((host, port))

print("Connected...\n")

s.send(name.encode())

s\_name = s.recv(1024)

s\_name = s\_name.decode()

print(s\_name, "has joined the chat room\nEnter [e] to exit chat room\n")

while True:

m=s.recv(1024)

m=m.decode()

k=s.recv(1024)

k=k.decode()

k=int(k)

i=0

a=""

b=""

f=random.randint(0,1)

message=""

while i!=k:

f=random.randint(0,1)

if(f==0):

b="ACK Lost"

message = s.recv(1024)

message = message.decode()

s.send(b.encode())

elif(f==1):

b="ACK "+str(i)

message = s.recv(1024)

message = message.decode()

s.send(b.encode())

a=a+message

i=i+1

print("The message received is :", m)

### sender.py

# Sender.py

import time, socket, sys

def decimalToBinary(n):

return n.replace("0b", "")

def binarycode(s):

a\_byte\_array = bytearray(s, "utf8")

byte\_list = []

for byte in a\_byte\_array:

binary\_representation = bin(byte)

byte\_list.append(decimalToBinary(binary\_representation))

#print(byte\_list)

a=""

for i in byte\_list:

a=a+i

return a

print("\nWelcome to Chat Room\n")

print("Initialising....\n")

time.sleep(1)

s = socket.socket()

host = socket.gethostname()

ip = socket.gethostbyname(host)

port = 1234

s.bind((host, port))

print(host, "(", ip, ")\n")

name = input(str("Enter your name: "))

s.listen(1)

print("\nWaiting for incoming connections...\n")

conn, addr = s.accept()

print("Received connection from ", addr[0], "(", addr[1], ")\n")

s\_name = conn.recv(1024)

s\_name = s\_name.decode()

print(s\_name, "has connected to the chat room\nEnter [e] to exit chat room\n")

conn.send(name.encode())

while True:

message = input(str("Me : "))

conn.send(message.encode())

if message == "[e]":

message = "Left chat room!"

conn.send(message.encode())

print("\n")

break

message=binarycode(message)

f=str(len(message))

conn.send(f.encode())

i=0

j=0

j=int(input("Enter the window size -> "))

b=""

j=j-1

f=int(f)

k=j

while i!=f:

while(i!=(f-j)):

conn.send(message[i].encode())

b=conn.recv(1024)

b=b.decode()

print(b)

if(b!="ACK Lost"):

time.sleep(1)

print("Acknowledgement Received! The sliding window is in the range "+(str(i+1))+" to "+str(k+1)+" Now sending the next packet")

i=i+1

k=k+1

time.sleep(1)

else:

time.sleep(1)

print("Acknowledgement of the data bit is LOST! The sliding window remains in the range "+(str(i+1))+" to "+str(k+1)+" Now Resending the same packet")

time.sleep(1)

while(i!=f):

conn.send(message[i].encode())

b=conn.recv(1024)

b=b.decode()

print(b)

if(b!="ACK Lost"):

time.sleep(1)

print("Acknowledgement Received! The sliding window is in the range "+(str(i+1))+" to "+str(k)+" Now sending the next packet")

i=i+1

time.sleep(1)

else:

time.sleep(1)

print("Acknowledgement of the data bit is LOST! The sliding window remains in the range "+(str(i+1))+" to "+str(k)+" Now Resending the same packet")

time.sleep(1)

### Selective Repeat Protocol

### cliselect.py

#Selective Repeat ARQ - Client Side Code

import socket

def client\_program():

n = 4

win\_start = 0

win\_end = win\_start + n - 1

host = socket.gethostname() # as both code is running on same pc

port = 12344 # socket server port number

sender = []

flag = 0 #send whole sender list else 1 means send only win\_start frame

client\_socket = socket.socket() # instantiate

client\_socket.connect((host, port)) # connect to the server

print 'Window Size is ', n

print '\*\*\*\*\*\*\*\* Enter "bye" to close connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

message = raw\_input("Hit any key to start sending frames -> ") # take input

while message.lower().strip() != 'bye':

print "Sending frames..."

if (flag == 0):

for i in range(n):

sender.append(win\_start + i)

for i in sender :

print "Frame -> ", i

else:

print "Frame -> ", win\_start

msg = str(win\_start)

client\_socket.send(msg.encode()) # send message

data = client\_socket.recv(1024).decode() # receive NAK

msg = str(data)

ack = int(msg)

if ack not in sender:

win\_start = ack

win\_end = win\_start + n - 1

flag = 0 #send new frame

for i in range(n):

sender.pop()

else:

win\_start = int(msg)

flag = 1 #send old frame

print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

print 'Received ACK server: ' + data # show in terminal

message = raw\_input("Hit any key to start sending frames -> ") # again take input

client\_socket.close() # close the connection

if \_\_name\_\_ == '\_\_main\_\_':

client\_program()

######################### OUTPUT : Client Side #################################

'''sayali@sayali:~/Desktop/Saiy/MyDocs/CN $ python cliselect.py

Window Size is 4

\*\*\*\*\*\*\*\* Enter "bye" to close connection \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hit any key to start sending frames ->

Sending frames...

Frame -> 0

Frame -> 1

Frame -> 2

Frame -> 3

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

Received ACK server: 4

Hit any key to start sending frames ->

Sending frames...

Frame -> 4

Frame -> 5

Frame -> 6

Frame -> 7

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

Received ACK server: 4

Hit any key to start sending frames ->

Sending frames...

Frame -> 4

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

Received ACK server: 7

Hit any key to start sending frames ->

Sending frames...

Frame -> 7

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

Received ACK server: 8

Hit any key to start sending frames ->

Sending frames...

Frame -> 8

Frame -> 9

Frame -> 10

Frame -> 11

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

Received ACK server: 12

Hit any key to start sending frames -> bye

### serselect.py

# Selective Repeat ARQ - Server Side Code

import socket

import random

def server\_program():

# get the hostname

host = socket.gethostname()

port = 12344 # initiate port no above 1024

exp = 0

n = 4

new = 1

win\_start = 0

win\_end = win\_start + n - 1

receiver = []

server\_socket = socket.socket() # get instance

# look closely. The bind() function takes tuple as argument

server\_socket.bind((host, port)) # bind host address and port together

# configure how many client the server can listen simultaneously

server\_socket.listen(2)

conn, address = server\_socket.accept() # accept new connection

print "Connection from: ", str(address)

while True:

# receive data stream. it won't accept data packet greater than 1024 bytes

data = conn.recv(1024).decode()

if not data:

# if data is not received break

break

rec = int(data)

lim = rec + n - 1

count = 0

flag = 0

ack = rec

randy = random.randint(1, 4)

if new == 1 : #you received a new frame of a new window

while(count != randy):

temp = random.randint(rec, lim)

if temp not in receiver:

print "Received Frame -> ", temp

count+=1

flag = 1 #Atleast one new frame added in receiver buffer

receiver.append(temp)

else :

print "Received Frame -> ", rec #you received a new frame of an old window

receiver.append(rec)

flag = 1

if(flag == 1):

for i in range(rec,lim+1):

if i not in receiver:

ack = i

break

ack = i+1

print "Sending ACK -> ", ack #next expected frame

print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

data = str(ack)

conn.send(data.encode()) # send data to the client

if ack > win\_end :

win\_start = ack

win\_end = win\_start + n - 1

new = 1 # now receive a new frame of a new window

else :

new = 0 # now received a new frame of an old window

conn.close() # close the connection

if \_\_name\_\_ == '\_\_main\_\_':

server\_program()

############################# OUTPUT : Server Side ##########################

'''sayali@sayali:~/Desktop/Saiy/MyDocs/CN $ python serselect.py

Connection from: ('127.0.0.1', 40912)

Received Frame -> 0

Received Frame -> 3

Received Frame -> 2

Received Frame -> 1

Sending ACK -> 4

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

Received Frame -> 6

Received Frame -> 5

Sending ACK -> 4

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

Received Frame -> 4

Sending ACK -> 7

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

Received Frame -> 7

Sending ACK -> 8

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

Received Frame -> 9

Received Frame -> 11

Received Frame -> 8

Received Frame -> 10

Sending ACK -> 12

# Experiment No- 6

## AIM:

## Implement a program for CRC polynomials.

## Procedure:

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.   
CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Another example is x2 + 1 that represents key 101.

n : Number of bits in data to be sent

from sender side.

k : Number of bits in the key obtained

from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

The binary data is first augmented by adding k-1 zeros in the end of the data

Use modulo-2 binary division to divide binary data by the key and store remainder of division.

Append the remainder at the end of the data to form the encoded data and send the same

 Receiver Side (Check if there are errors introduced in transmission)  
Perform modulo-2 division again and if the remainder is 0, then there are no errors.

In this article we will focus only on finding the remainder i.e. check word and the code word.

Modulo 2 Division:  
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).

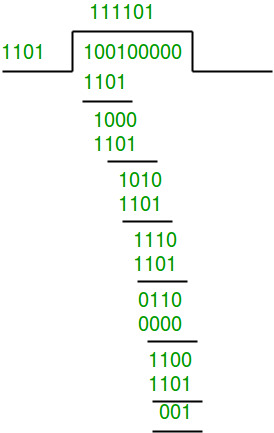
The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.

When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Illustration:  
Example 1 (No error in transmission):

Data word to be sent - 100100

Key - 1101 [ Or generator polynomial x3 + x2 + 1]

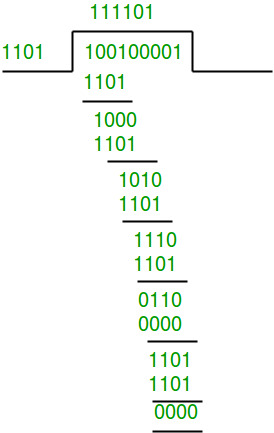
Sender Side:

Therefore, the remainder is 001 and hence the encoded

data sent is 100100001.

Receiver Side:

Code word received at the receiver side 100100001



Therefore, the remainder is all zeros. Hence, the

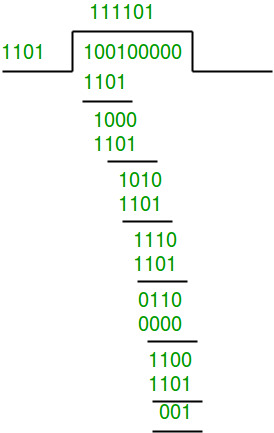
data received has no error.

Example 2: (Error in transmission)

Data word to be sent - 100100

Key - 1101

Sender Side:



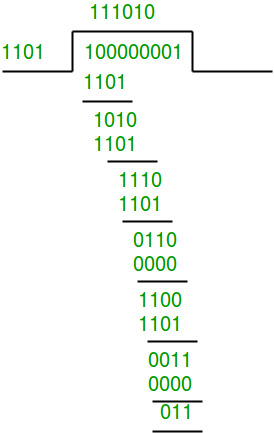
Therefore, the remainder is 001 and hence the

code word sent is 100100001.

Receiver Side

Let there be an error in transmission media

Code word received at the receiver side - 100000001



Since the remainder is not all zeroes, the error  
is detected at the receiver side.

## Program:

# Returns XOR of 'a' and 'b'

# (both of same length)

def xor(a, b):

# initialize result

result = []

# Traverse all bits, if bits are

# same, then XOR is 0, else 1

for i in range(1, len(b)):

if a[i] == b[i]:

result.append('0')

else:

result.append('1')

return ''.join(result)

# Performs Modulo-2 division

def mod2div(dividend, divisor):

# Number of bits to be XORed at a time.

pick = len(divisor)

# Slicing the dividend to appropriate

# length for particular step

tmp = dividend[0: pick]

while pick < len(dividend):

if tmp[0] == '1':

# replace the dividend by the result

# of XOR and pull 1 bit down

tmp = xor(divisor, tmp) + dividend[pick]

else: # If leftmost bit is '0'

# If the leftmost bit of the dividend (or the

# part used in each step) is 0, the step cannot

# use the regular divisor; we need to use an

# all-0s divisor.

tmp = xor('0'\*pick, tmp) + dividend[pick]

# increment pick to move further

pick += 1

# For the last n bits, we have to carry it out

# normally as increased value of pick will cause

# Index Out of Bounds.

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0'\*pick, tmp)

checkword = tmp

return checkword

# Function used at the sender side to encode

# data by appending remainder of modular division

# at the end of data.

def encodeData(data, key):

l\_key = len(key)

# Appends n-1 zeroes at end of data

appended\_data = data + '0'\*(l\_key-1)

remainder = mod2div(appended\_data, key)

# Append remainder in the original data

codeword = data + remainder

print("Remainder : ", remainder)

print("Encoded Data (Data + Remainder) : ",

codeword)

# Driver code

data = "100100"

key = "1101"

encodeData(data, key)

## Output

Sender side...

Remainder : 001

Encoded Data (Data + Remainder) :100100001

Receiver side...

correct message received

# Experiment No- 7

## AIM:

Simulation of Transferring data between two nodes using NS.

## Procedure:

This network consists of 4 nodes (n0, n1, n2, n3) as shown in above figure. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1KByte. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to generate 1 KByte packets at the rate of 1 Mbps. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec.

## Program:

#Create a simulator object

set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red

#Open the NAM trace file

set nf [open out.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish {} {

global ns nf

$ns flush-trace

#Close the NAM trace file

close $nf

#Execute NAM on the trace file

exec nam out.nam &

exit 0

}

#Create four nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

#Create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10

$ns queue-limit $n2 $n3 10

#Give node position (for NAM)

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link-op $n1 $n2 orient right-up

$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)

$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection

set tcp [new Agent/TCP]

$tcp set class\_ 2

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

#Setup a FTP over TCP connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type\_ FTP

#Setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $udp $null

$udp set fid\_ 2

#Setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type\_ CBR

$cbr set packet\_size\_ 1000

$cbr set rate\_ 1mb

$cbr set random\_ false

#Schedule events for the CBR and FTP agents

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 4.0 "$ftp stop"

$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)

$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish"

#Print CBR packet size and interval

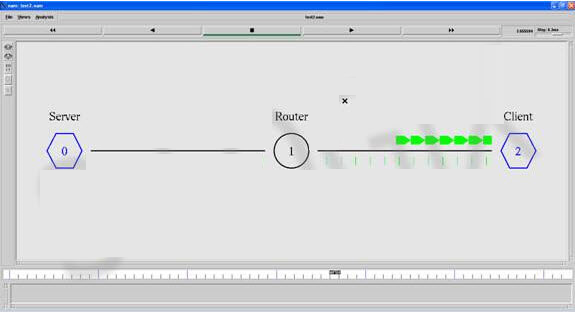
puts "CBR packet size = [$cbr set packet\_size\_]"

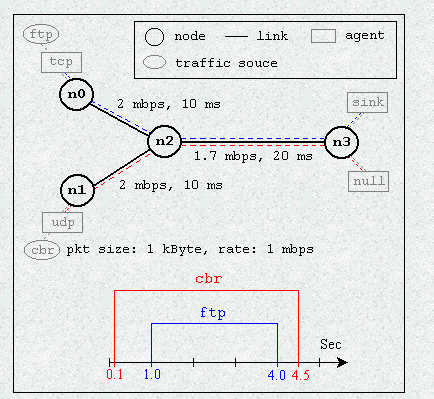
puts "CBR interval = [$cbr set interval\_]"

#Run the simulation

$ns run

## Output:





# Experiment No- 8

## AIM:

Simulation of data transfer and packet loss using NS.

## Procedure:

Packet loss in a communication is the difference between the generated and received packets. Packet Loss is calculated using awk script which processes the trace file and produces the result.

## Program:

# Simulator Instance Creation

set ns [new Simulator]

#Fixing the co-ordinate of simutaion area

set val(x) 500

set val(y) 500

# Define options

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation model

set val(netif) Phy/WirelessPhy ;# network interface type

set val(mac) Mac/802\_11 ;# MAC type

set val(ifq) Queue/DropTail/PriQueue ;# interface queue type

set val(ll) LL ;# link layer type

set val(ant) Antenna/OmniAntenna ;# antenna model

set val(ifqlen) 50 ;# max packet in ifq

set val(nn) 3 ;# number of mobilenodes

set val(rp) AODV ;# routing protocol

set val(x) 500 ;# X dimension of topography

set val(y) 500 ;# Y dimension of topography

set val(stop) 10.0 ;# time of simulation end

# set up topography object

set topo [new Topography]

$topo load\_flatgrid $val(x) $val(y)

#Nam File Creation nam – network animator

set namfile [open sample26.nam w]

#Tracing all the events and cofiguration

$ns namtrace-all-wireless $namfile $val(x) $val(y)

#Trace File creation

set tracefile [open sample26.tr w]

#Tracing all the events and cofiguration

$ns trace-all $tracefile

# general operational descriptor- storing the hop details in the network

create-god $val(nn)

# configure the nodes

$ns node-config -adhocRouting $val(rp) \

-llType $val(ll) \

-macType $val(mac) \

-ifqType $val(ifq) \

-ifqLen $val(ifqlen) \

-antType $val(ant) \

-propType $val(prop) \

-phyType $val(netif) \

-channelType $val(chan) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF \

-movementTrace ON

# Node Creation

for {set i 0} {$i < 3} {incr i} {

set node\_($i) [$ns node]

$node\_($i) color black

}

#Location fixing for a single node

$node\_(0) set X\_ 258

$node\_(0) set Y\_ 222

$node\_(0) set Z\_ 0

for {set i 1} {$i < 3} {incr i} {

$node\_($i) set X\_ [expr rand()\*$val(x)]

$node\_($i) set Y\_ [expr rand()\*$val(y)]

$node\_($i) set Z\_ 0

}

# Label and coloring

for {set i 0} {$i < 3} {incr i} {

$ns at 0.1 "$node\_($i) color blue"

$ns at 0.1 "$node\_($i) label Node$i"

}

#Size of the node

for {set i 0} {$i < 3} {incr i} {

$ns initial\_node\_pos $node\_($i) 30

}

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Defining Communication Between node0 and all nodes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

for {set i 1} {$i < 3} {incr i} {

# Defining a transport agent for sending

set udp [new Agent/UDP]

# Attaching transport agent to sender node

$ns attach-agent $node\_($i) $udp

# Defining a transport agent for receiving

set null [new Agent/Null]

# Attaching transport agent to receiver node

$ns attach-agent $node\_(0) $null

#Connecting sending and receiving transport agents

$ns connect $udp $null

#Defining Application instance

set cbr [new Application/Traffic/CBR]

# Attaching transport agent to application agent

$cbr attach-agent $udp

#Packet size in bytes and interval in seconds definition

$cbr set packetSize\_ 512

$cbr set interval\_ 0.1

# data packet generation starting time

$ns at 1.0 "$cbr start"

# data packet generation ending time

#$ns at 6.0 "$cbr stop"

}

# ending nam and the simulation

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "stop"

#Stopping the scheduler

$ns at 10.01 "puts \"end simulation\" ; $ns halt"

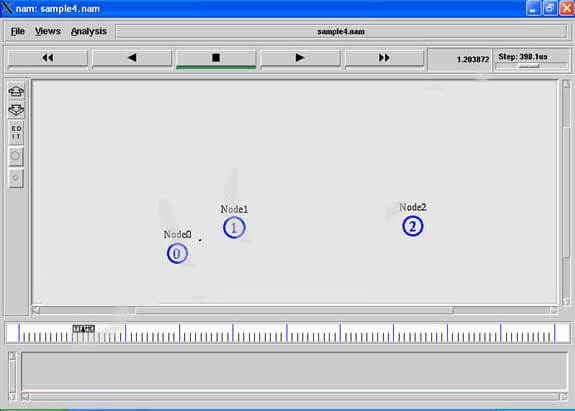
#$ns at 10.01 "$ns halt"

exec awk -f Packet\_Loss.awk sample26.tr > packet\_loss.tr &

#Starting scheduler

$ns run

## Output:



# Experiment No- 9

## AIM:

Simulation of Congestion Control Algorithm using NS.

## Program :

set ns [new Simulator]

set f [ open congestion.tr w ]

$ns trace-all $f

set nf [ open congestion.nam w ]

$ns namtrace-all $nf

$ns color 1 Red

$ns color 2 Blue

$ns color 3 White

$ns color 4 Green

#to create nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

# to create the link between the nodes with bandwidth, delay and queue

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail

$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail

$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail

# Sending node with agent as Reno Agent

set tcp1 [new Agent/TCP/Reno]

$ns attach-agent $n0 $tcp1

set tcp2 [new Agent/TCP/Reno]

$ns attach-agent $n1 $tcp2

set tcp3 [new Agent/TCP/Reno]

$ns attach-agent $n2 $tcp3

set tcp4 [new Agent/TCP/Reno]

$ns attach-agent $n1 $tcp4

$tcp1 set fid\_ 1

$tcp2 set fid\_ 2

$tcp3 set fid\_ 3

$tcp4 set fid\_ 4

# receiving (sink) node

set sink1 [new Agent/TCPSink]

$ns attach-agent $n4 $sink1

set sink2 [new Agent/TCPSink]

$ns attach-agent $n5 $sink2

set sink3 [new Agent/TCPSink]

$ns attach-agent $n3 $sink3

set sink4 [new Agent/TCPSink]

$ns attach-agent $n4 $sink4

# establish the traffic between the source and sink

$ns connect $tcp1 $sink1

$ns connect $tcp2 $sink2

$ns connect $tcp3 $sink3

$ns connect $tcp4 $sink4

# Setup a FTP traffic generator on "tcp"

set ftp1 [new Application/FTP]

$ftp1 attach-agent $tcp1

$ftp1 set type\_ FTP

set ftp2 [new Application/FTP]

$ftp2 attach-agent $tcp2

$ftp2 set type\_ FTP

set ftp3 [new Application/FTP]

$ftp3 attach-agent $tcp3

$ftp3 set type\_ FTP

set ftp4 [new Application/FTP]

$ftp4 attach-agent $tcp4

$ftp4 set type\_ FTP

# RTT Calculation Using Ping ------------------------------------------------

set p0 [new Agent/Ping]

$ns attach-agent $n0 $p0

set p1 [new Agent/Ping]

$ns attach-agent $n4 $p1

#Connect the two agents

$ns connect $p0 $p1

# Method call from ping.cc file

Agent/Ping instproc recv {from rtt} {

$self instvar node\_

puts "node [$node\_ id] received ping answer from \

$from with round-trip-time $rtt ms."

}

# ---------------------------------------------------------------------------

# start/stop the traffic

$ns at 0.2 "$p0 send"

$ns at 0.3 "$p1 send"

$ns at 0.5 "$ftp1 start"

$ns at 0.6 "$ftp2 start"

$ns at 0.7 "$ftp3 start"

$ns at 0.8 "$ftp4 start"

$ns at 66.0 "$ftp4 stop"

$ns at 67.0 "$ftp3 stop"

$ns at 68.0 "$ftp2 stop"

$ns at 70.0 "$ftp1 stop"

$ns at 70.1 "$p0 send"

$ns at 70.2 "$p1 send"

# Set simulation end time

$ns at 80.0 "finish"

# procedure to plot the congestion window

# cwnd\_ used from tcp-reno.cc file

proc plotWindow {tcpSource outfile} {

global ns

set now [$ns now]

set cwnd\_ [$tcpSource set cwnd\_]

# the data is recorded in a file called congestion.xg.

puts $outfile "$now $cwnd\_"

$ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"

}

set outfile [open "congestion.xg" w]

$ns at 0.0 "plotWindow $tcp1 $outfile"

proc finish {} {

exec nam congestion.nam &

exec xgraph congestion.xg -geometry 300x300 &

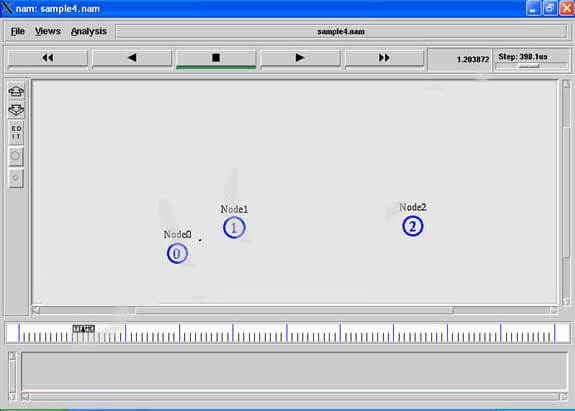
exit 0

}

# Run simulation

$ns run

## Output:



# Experiment No- 10

## AIM:

Simulate a 3 nodes point-to-point network with duplex links between them. Set the queue size vary the bandwidth and find the number of packets dropped..

## Program:

#===================================

# Simulation parameters setup

#===================================

set val(stop) 100.0 ;# time of simulation end

#===================================

# Initialization

#===================================

#Create a ns simulator

set ns [new Simulator]

#Open the NS trace file

set tracefile [open out.tr w]

$ns trace-all $tracefile

#Open the NAM trace file

set namfile [open out.nam w]

$ns namtrace-all $namfile

#===================================

# Nodes Definition

#===================================

#Create 3 nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

#===================================

# Links Definition

#===================================

#Createlinks between nodes

$ns duplex-link $n0 $n2 0.2Mb 10ms DropTail

$ns queue-limit $n0 $n2 5

$ns duplex-link $n1 $n2 0.2Mb 10ms DropTail

$ns queue-limit $n1 $n2 5

#Give node position (for NAM)

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link-op $n1 $n2 orient right-up

#===================================

# Agents Definition

#===================================

#Setup a TCP connection

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink2 [new Agent/TCPSink]

$ns attach-agent $n2 $sink2

$ns connect $tcp0 $sink2

$tcp0 set packetSize\_ 2500

#Setup a TCP connection

set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set sink3 [new Agent/TCPSink]

$ns attach-agent $n2 $sink3

$ns connect $tcp1 $sink3

$tcp1 set packetSize\_ 2500

#===================================

# Applications Definition

#===================================

#Setup a FTP Application over TCP connection

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

$ns at 1.0 "$ftp0 start"

$ns at 25.0 "$ftp0 stop"

#Setup a FTP Application over TCP connection

set ftp1 [new Application/FTP]

$ftp1 attach-agent $tcp1

$ns at 1.0 "$ftp1 start"

$ns at 25.0 "$ftp1 stop"

#===================================

# Termination

#===================================

#Define a 'finish' procedure

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam out.nam &

exit 0

}

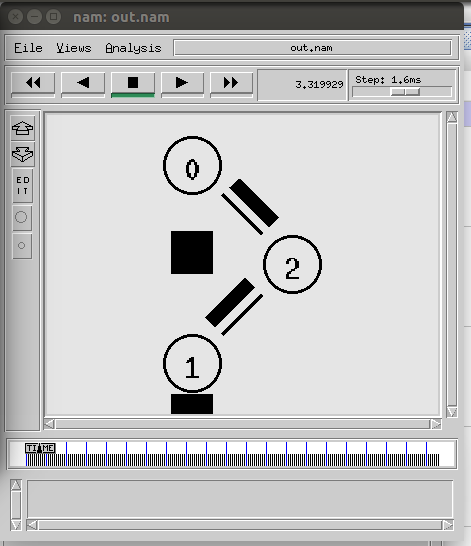
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"done\" ; $ns halt"

$ns run

## Output:



# Experiment No- 11

## AIM:

Simulate a 4 nodes point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant Applications over TCP and UDP agents changing the parameter and determining the number of packets sent by TCP/UDP.

## Program:

set ns [new Simulator]

#Open a new file for NAMTRACE

set nf [open out.nam w]

$ns namtrace-all $nf

#Open a new file to log TRACE

set tf [open out.tr w]

$ns trace-all $tf

#Body of the 'finish' procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exec awk -f exp2.awk out.tr &

exit 0

}

#Create Nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

#Create Links between Nodes

$ns duplex-link $n0 $n2 1Mb 10ms DropTail

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Set the queue limit - default is 50 packets

$ns queue-limit $n0 $n2 50

$ns queue-limit $n1 $n2 50

$ns queue-limit $n2 $n3 50

#Create TCP Agent between node 0 and node 3

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0

$ns connect $tcp0 $sink0

#Create FTP Application for TCP Agent

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

#Specify TCP packet size

Agent/TCP set packetSize\_ 1000

#Create UDP Agent between node 1 and node 3

set udp0 [new Agent/UDP]

$ns attach-agent $n1 $udp0

set null0 [new Agent/Null]

$ns attach-agent $n3 $null0

$ns connect $udp0 $null0

#Create CBR Application for UDP Agent

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

#Start and Stop FTP Traffic

$ns at 0.75 "$ftp0 start"

$ns at 4.75 "$ftp0 stop"

#Start and Stop CBR traffic

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

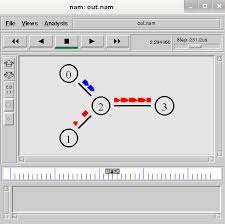
#Stop the simulation

$ns at 5.0 "finish"

#Run the simulation

$ns run

## Output:



# Experiment No- 12

## AIM:

Simulate an Ethernet LAN using N-nodes (6-10), change error rate and data rate and compare the throughput.

## Program:

#Create Simulator

set ns [new Simulator]

#Use colors to differentiate the traffic

$ns color 1 Blue

$ns color 2 Red

#Open trace and NAM trace file

set ntrace [open prog5.tr w]

$ns trace-all $ntrace

set namfile [open prog5.nam w]

$ns namtrace-all $namfile

#Finish Procedure

proc Finish {} {

global ns ntrace namfile

#Dump all trace data and close the files

$ns flush-trace

close $ntrace

close $namfile

#Execute the nam animation file

exec nam prog5.nam &

#Calculate the throughput = (number of packets received/time taken for simulation)

set TcpSize [exec grep "^r" prog5.tr | grep "tcp" | tail -n 1 | cut -d " " -f 6]

set numTcp [exec grep "^r" prog5.tr | grep -c "tcp"]

set tcpTime 123.0

set UdpSize [exec grep "^r" prog5.tr | grep "cbr" | tail -n 1 | cut -d " " -f 6]

set numUdp [exec grep "^r" prog5.tr | grep -c "cbr"]

set udpTime 124.4

puts "The throughput of FTP is"

puts "[expr ($numTcp\*$TcpSize)/$tcpTime] bytes per second"

puts "The throughput of CBR is"

puts "[expr ($numUdp\*$UdpSize)/$udpTime] bytes per second"

exit 0

}

#Create 6 nodes

for {set i 0} {$i < 6} {incr i} {

set n($i) [$ns node]

}

#Create duplex links between the nodes

$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail

$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail

$ns simplex-link $n(2) $n(3) 0.3Mb 100ms DropTail

$ns simplex-link $n(3) $n(2) 0.3Mb 100ms DropTail

#Node n(3), n(4) and n(5) are considered in a LAN

set lan [$ns newLan "$n(3) $n(4) $n(5)" 0.5Mb 40ms LL Queue/DropTail MAC/802\_3 Channel]

#Orientation to the nodes

$ns duplex-link-op $n(0) $n(2) orient right-down

$ns duplex-link-op $n(1) $n(2) orient right-up

$ns simplex-link-op $n(2) $n(3) orient right

#Setup queue between n(2) and n(3) and monitor the queue

$ns queue-limit $n(2) $n(3) 20

$ns simplex-link-op $n(2) $n(3) queuePos 0.5

#Set error model on link n(2) and n(3) and insert the error model

set loss\_module [new ErrorModel]

$loss\_module ranvar [new RandomVariable/Uniform]

$loss\_module drop-target [new Agent/Null]

$ns lossmodel $loss\_module $n(2) $n(3)

#Setup TCP Connection between n(0) and n(4)

set tcp0 [new Agent/TCP/Newreno]

$tcp0 set fid\_ 1

$tcp0 set window\_ 8000

$tcp0 set packetSize\_ 552

$ns attach-agent $n(0) $tcp0

set sink0 [new Agent/TCPSink/DelAck]

$ns attach-agent $n(4) $sink0

$ns connect $tcp0 $sink0

#Apply FTP Application over TCP

set ftp0 [new Application/FTP]

$ftp0 set type\_ FTP

$ftp0 attach-agent $tcp0

#Setup UDP Connection between n(1) and n(5)

set udp0 [new Agent/UDP]

$udp0 set fid\_ 2

$ns attach-agent $n(1) $udp0

set null0 [new Agent/Null]

$ns attach-agent $n(5) $null0

$ns connect $udp0 $null0

#Apply CBR Traffic over UDP

set cbr0 [new Application/Traffic/CBR]

$cbr0 set type\_ CBR

$cbr0 set packetSize\_ 1000

$cbr0 set rate\_ 0.1Mb

$cbr0 set random\_ false

$cbr0 attach-agent $udp0

#Schedule events

$ns at 0.1 "$cbr0 start"

$ns at 1.0 "$ftp0 start"

$ns at 124.0 "$ftp0 stop"

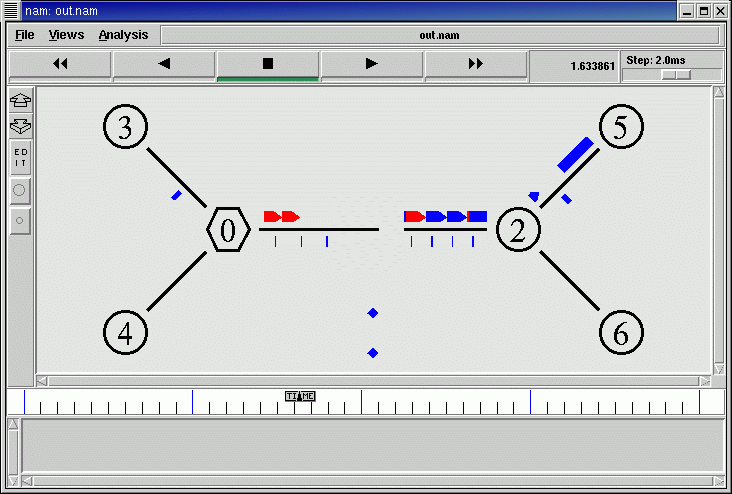
$ns at 124.5 "$cbr0 stop"

$ns at 125.0 "Finish"

#Run Simulation

$ns run

## Output:



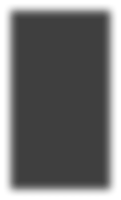
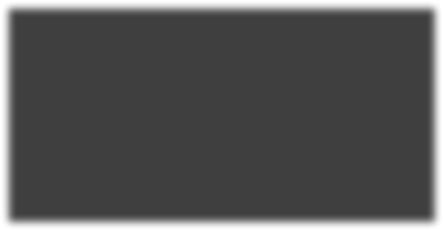
# Experiment No- 13

## AIM:

Protocol analysis with Wireshark.

## Procedure:

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



packet sniffer

application (e.g., browser,

ssh client, skype)

application

Physical (CAT5, Radio)

frames sent/recieved

Link (Ethernet)

Network (IP)

Transport (TCP/UDP)

copy of all Ethernet

operating system

packet capture (pcap)

packet analyzer

The figure above shows the structure of a packet sniﬀer. At the right are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniﬀer, shown within the dashed rectangle, is an addition to the usual software in your computer, and consists of two parts. The packet.

capture library receives a copy of every link-layer frame that is sent from or received by your computer. As you know, messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In the figure, the assumed physical media is an Ethernet, and so all upper layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/ received from/by all protocols and applications executing in your computer.

The existence of the packet capture box in this figure should give you cause to pause and think, particularly down two trains of thought. Firstly, it shows that any packet in a shared medium (Ethernet, Wi-Fi, etc) can be captured and examined without notification of the sender or receiver. You cannot rely on common link-layer protocols to protect your secrets or your privacy online. At a minimum, you should be using encryption protocols (generally buried in the application layer, though sometimes found elsewhere) to protect all network traﬃc you generate or receive. Secondly, you have the ability to act as the “bad guy” and capture the network traﬃc of other people, examine it and exploit what you find. You need to learn to use this tool in a responsible fashion. Remember the movie quote: “With great power comes great responsibility!” We will use a filter to ensure Wireshark doesn’t display traﬃc other than your own, but this is purely a voluntary measure. Please act ethically and responsibly in your use of Wireshark.

The second component of a packet sniﬀer is the packet analyzer, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must “understand” the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment.

Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string “GET,” “POST,” or “HEAD,”.

We will be using the Wireshark packet sniﬀer [wireshark.org] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at diﬀerent levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Macintosh, Windows, and Linux/Unix computers. It’s an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user- guide (wireshark.org/docs/wsug\_html\_chunked), man pages (wireshark.org/docs/man-pages), and a detailed FAQ (wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, Token-Ring, FDDI, serial (PPP and SLIP), 802.11 wireless LANs, and ATM connections (if the OS on which it's running allows Wireshark to do so).

### 1. Get Wireshark

1. In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the libpcap or WinPCap packet capture library. The libpcap software will be installed for you, if it is not installed within your operating system, when you install Wireshark.. See wireshark.org/download.html for a list of supported operating systems and download sites

2. Download the Wireshark binary from wireshark.org/download.html and install it. Make sure to also download the Wireshark user guide.

3. The Wireshark FAQ has a number of helpful hints and interesting tidbits of information, particularly if you have trouble installing or running Wireshark.

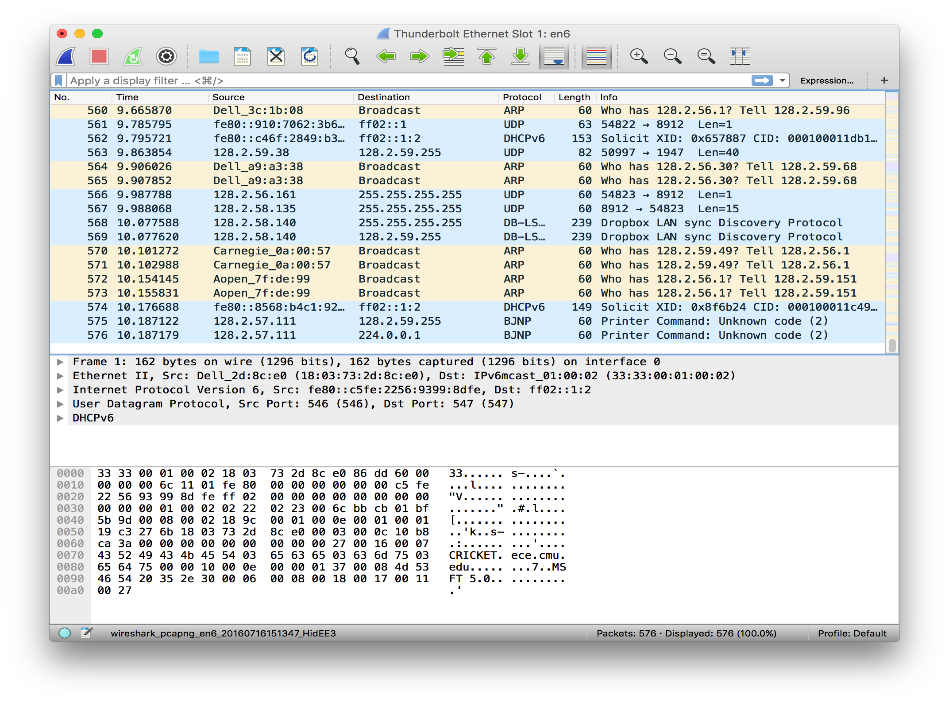
4. You may need to disable anti-virus protection software (McAﬀee, I'm looking at you!) before your own IP address will show up in captured data.

5. You should be connected to an Ethernet connection. If you only have WiFi, you'll need to figure out how to set your WiFi physical layer into monitor mode, which may be diﬃcult or impossible, depending on your operating system. Failure to follow this instruction will mean you only see traﬃc originating or being sent to your own computer, which is sub-optimal for these labs.

### 2. Run Wireshark

1. When you run the Wireshark program, the Wireshark graphical user interface will be displayed. Initially, no data will be displayed in the various windows.

By the way, the pictures I show in this lab guide may command menus listing of captured packets details of selected packet header packet content in ASCII and hexadecimal display filter specification diﬀer, perhaps substantially, from the interface you see on your computer, depending on your installed version and operating system. Be flexible.



**command**

menus

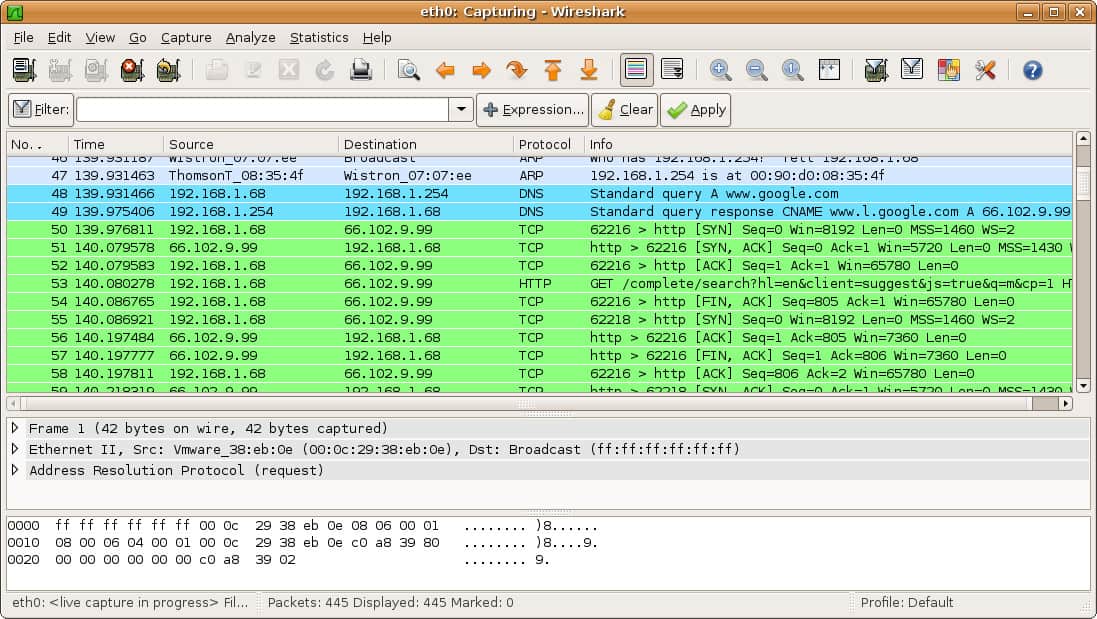
**details** of selected packet header

**listing** of captured packets

packet **content** in ASCII and hexadecimal

Once you’ve captured your network data, you’ll want to look at your captured packets. In the screenshot below you’ll see three panes, the **packet list** pane, the **packet bytes** pane, and the **packet details** pane.

If you want more information, you can click on any of the fields in each packet to see more. When you click on a packet, you’re shown a breakdown of its internal bytes in the byte view section.



## What can you do with Wireshark?

Over the past few years, Wireshark has developed a reputation as **one of the most reliable network protocol analyzers available on the market**. Users across the globe have been using this open-source application as **a complete network analysis tool**. Through Wireshark, users can troubleshoot network problems, examine network security issues, debug protocols, and learn network processes.

## How to Use Wireshark

As mentioned above, Wireshark is a network protocol analysis tool. At its core, Wireshark was designed to break down packets of data being transferred across different networks. The user can search and filter for specific packets of data and analyze how they are transferred across their network. These packets can be used for analysis on a real-time or offline basis.

The user can use this information to generate statistics and graphs. Wireshark was originally known as Ethereal but has since established itself as one of the key network analysis tools on the market. This is the go-to tool for users who want to view data generated by different networks and protocols.

Wireshark is suitable for novice and expert users alike. The user interface is incredibly simple to use once you learn the initial steps to capture packets. More advanced users can use the platform’s decryption tools to break down encrypted packets as well.

## Wireshark Core Features

Below is a breakdown of Wireshark’s core features:

* Capture live packet data
* Import packets from text files
* View packet data and protocol information
* Save captured packet data
* Display packets
* Filter packets
* Search packets
* Colorize packets
* Generate Statistics

Most users use Wireshark to detect network problems and test their software. As an open-source project, Wireshark is maintained by a unique team keeping service standards high. In this guide, we break down how to use Wireshark. Further information can be found on [Wireshark’s official user guide](https://www.wireshark.org/docs/wsug_html/#ChapterUsing).

### Get The Ultimate Wireshark eBook for FREE

Learn everything there is to know about Wireshark. From getting started to getting the most out of it (inc. handy cheatsheet)

Top of Form

First Name

Email

Please give consent to receive emails

SEND ME THE EBOOK!



Bottom of Form

## How to Download and Install Wireshark

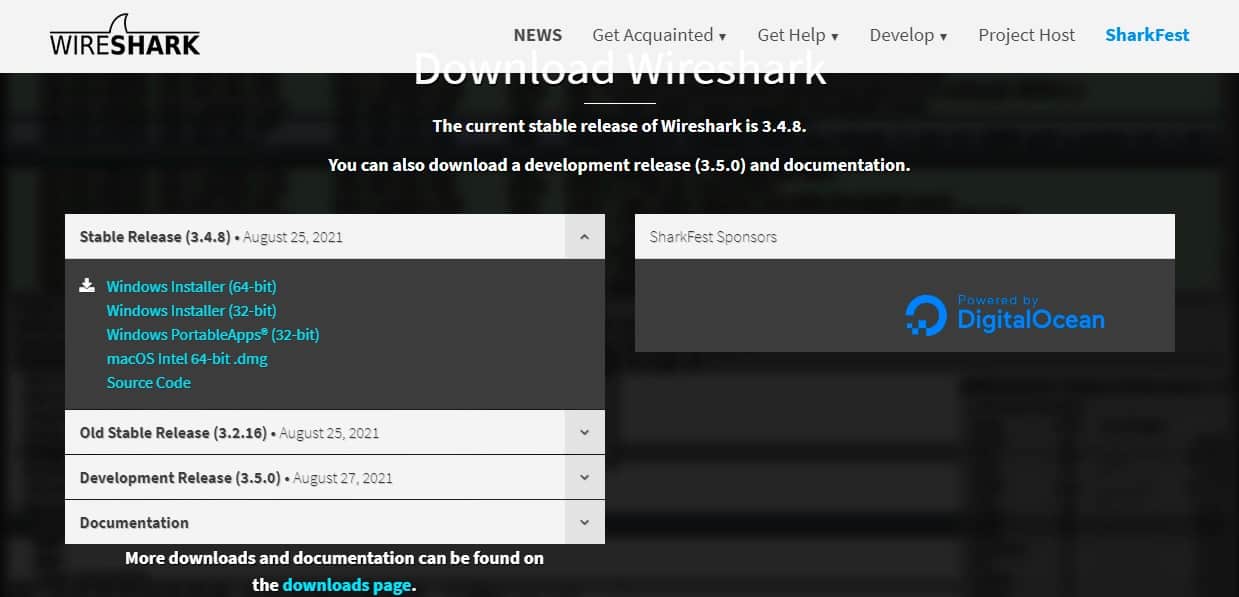
Before using Wireshark, the first thing you need to do is download and install it. You can [download Wireshark for free](https://www.wireshark.org/download.html) off of the company website. To have the smoothest running experience, it is advised that you download the latest version available on your platform from the “stable release” section.

### Install on Windows

Once you’ve downloaded the program you can start the setup process. During installation, you may be prompted to install WinPcap. It’s important to install WinPcap as without it you will be unable to capture live network traffic. Without WinPcap you will only be able to open saved capture files. To install, simply check the **Install WinPcap** box.

Let’s look at this in more detail.

Select the installer for your Windows architecture (64-bit or 32-bit) click on the link to download the package.

[](https://cdn.comparitech.com/wp-content/uploads/2021/09/Wireshark-download.jpg)

Once the installer is on your computer, follow these steps:

1. Click on the downloaded file to run it. Click Yes in the User Account Control window.
2. Click Next in the opening screen of the installer.
3. In the License Agreement screen, click the Noted button.
4. Leave all of the defaults in the Choose Components screen. Click the Next button.
5. Leave all settings as they are in the Additional Tasks screen and click on Next.
6. In Choose Install Location, just click on Next.
7. In the Packet Capture screen, leave the consent box checked and click the Next button.
8. In the USB Capture screen, check the Install USBPcap box and press the Install button.
9. Let the installation progress. During the process, you will be presented with a consent screen for Npcap. Click on I Agree.
10. In the NPcap Installation options screen check Restrict Npcap driver’s access to Administrators only, Support raw 802.11 traffic (and monitor mode) for wireless adapters, and Install Npcap in WinPcap API-compatible Mode. Click on Install.
11. When the Npcap setup has finished. Click on Next and then Finish to dismiss that dialogue window. The Wireshark installation will continue.
12. In the Installation Complete screen, click on Next and then Finish in the next screen.

Look in your Start menu for the Wireshark icon. Click on it to run the utility.

### Install on Mac

To install Wireshark on Mac you first need to download an installer. To do this, download an installer such as [exquartz](http://www.xquartz.org/). Once you’ve done this, open the Terminal and input the following command:

<% /Applications/Wireshark.app/Contents/Mac0S/Wireshark>

Then wait for Wireshark to start.

### Install on Unix

In order to run Wireshark on Unix, you need a couple of other tools installed on your system first. These are:

* [**GTK+**](https://gitlab.gnome.org/GNOME/gtk), The GIMP Tool Kit and Glib, both from the same source.
* You will also need [Glib](https://gitlab.gnome.org/GNOME/glib). You can get familiar with both tools at <https://www.gtk.org/>
* **libpcap**, which you get from <http://www.tcpdump.org/>.

After installing the above supporting software, and [downloading the software for Wireshark](https://www.wireshark.org/download.html), you need to extract it from the tar file.

gzip -d wireshark-1.2-tar.gz

tar xvf wireshark-1.2-tar

Change to the Wireshark directory and then issue the following commands:

./configure

make

make install

You can now run the Wireshark program on your Unix computer.

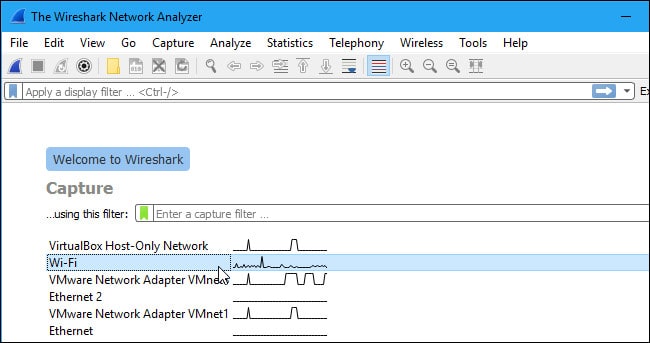
## How to Capture Data Packets

One of the core functions of Wireshark as a network analysis tool is to capture packets of data. Learning how to set up Wireshark to capture packets is essential to conducting detailed network analysis. However, it’s important to note that it can be difficult to capture packets when you’re new to Wireshark. Before you start to capture packets, there are three things you need to do:

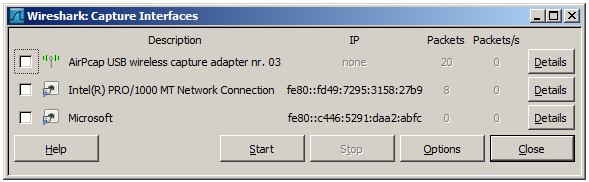
1. **Make sure that you have the administrative privileges** to start a live capture on your device
2. **Choose the correct network interface** to capture packet data from
3. **Capture** packet data **from the correct location** in your network

Once you’ve done these three things, you’re ready to start the capture process. When you use Wireshark to capture packets, they are displayed in a human-readable format to make them legible to the user. You can also **break packets down with filters and color-coding** if you wish to see more specific information.

When you first open up Wireshark, you’ll be met by the following launch screen:

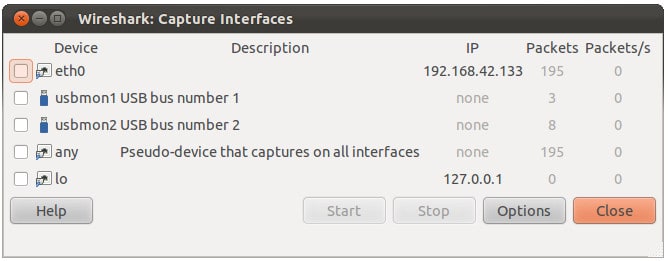
[[](https://cdn.comparitech.com/wp-content/uploads/2018/04/1-WIRESHARK-Network-Analyzer-launch-screen-.jpg)](https://cdn.comparitech.com/wp-content/uploads/2018/04/1-WIRESHARK-Network-Analyzer-launch-screen-.jpg)

The first thing you need to do is look at the available interfaces to capture. To do this, select **Capture > Options**. The “Capture Interfaces” dialog box will then open as shown below:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/2-capture-interfaces-1-.jpg)

Check the box of the interface you want to capture and press the **Start** button to start. You can select multiple interfaces if you want to capture data from multiple sources simultaneously.

On Unix or Linux, the dialog box is shown in a similar style like this:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/3-unix-linux-capture-interface.jpg)

You can also start Wireshark by using the following command line:

<¢ wireshark -i   eth0 —k>

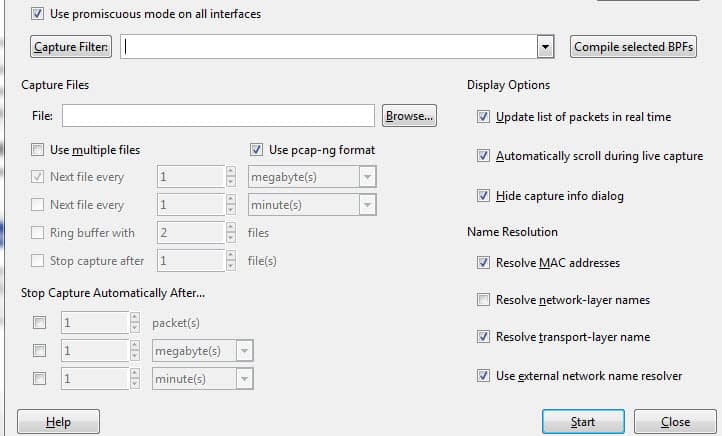
You can also use the **shark fin button**on the toolbar as a shortcut to initiate packet capturing. Once you click this button, Wireshark will start the live capture process.

If you want to stop capturing, click the red **stop**button next to the shark fin.

### Promiscuous Mode

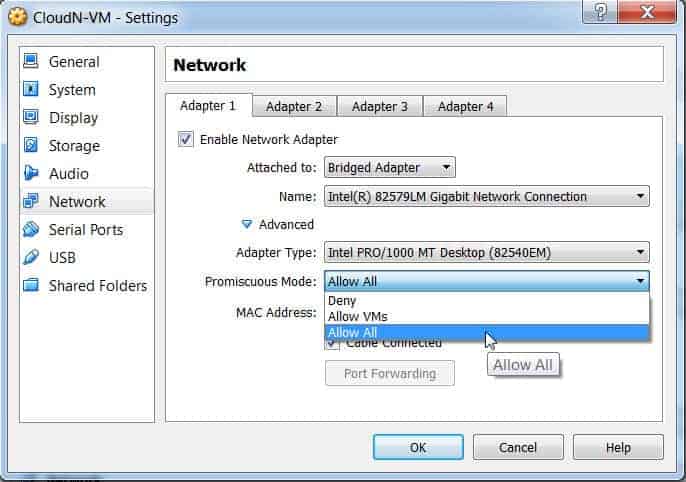
If you want to develop an overhead view of your network packet transfers, then you need to activate ‘promiscuous mode’. Promiscuous mode is **an interface mode where Wireshark details every packet it sees**. When this mode is deactivated, you lose transparency over your network and only develop a limited snapshot of your network (this makes it more difficult to conduct any analysis).

To activate promiscuous mode, click on the **Capture Options** dialog box and click **promiscuous mode**. In theory, this should show you all the traffic active on your network. The promiscuous mode box is shown below:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/4-Start-promiscuous-mode.jpg)

However, this often isn’t the case. Many network interfaces are resistant to promiscuous mode, so you need to check the Wireshark website for information on your specific hardware.

On Windows, it’s useful to open **Device Manager** and check whether you have your settings configured to reject promiscuous mode. For example:

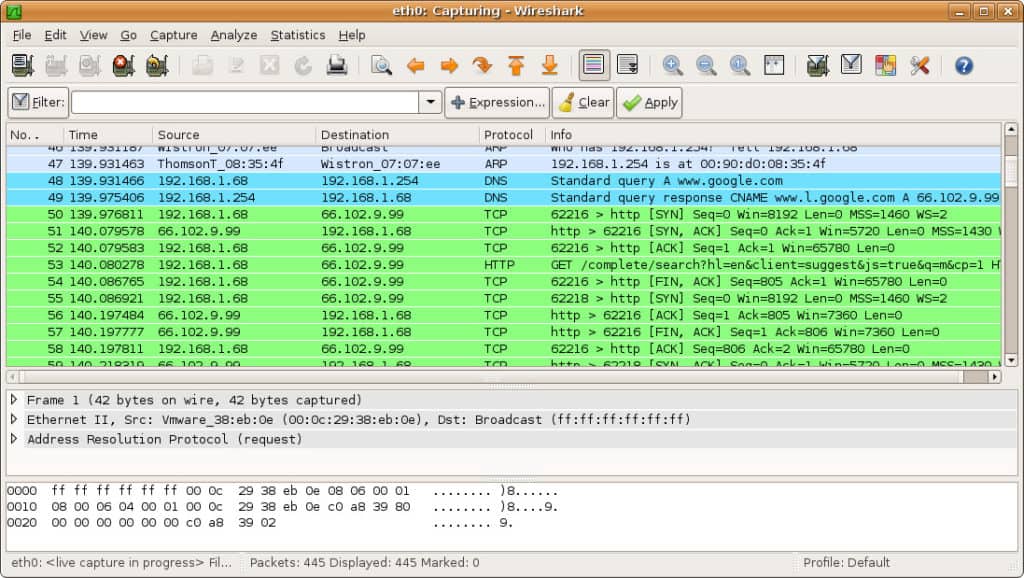
[](https://cdn.comparitech.com/wp-content/uploads/2018/04/5-Device-manager-.jpg)

(Simply click on network and then make sure that your promiscuous mode setting are set to **Allow All).**

If you have your settings set to “reject” promiscuous mode, then you’re going to limit the number of packets Wireshark captures. So even if you have promiscuous mode enabled on Wireshark check your Device Manager to make sure that your interface isn’t blocking any data from coming through. Taking the time to check through your network infrastructure will ensure Wireshark receives all the necessary packets of data.

## How to Analyze Captured Packets

Once you’ve captured your network data, you’ll want to look at your captured packets. In the screenshot below you’ll see three panes, the **packet list** pane, the **packet bytes** pane, and the **packet details** pane.

If you want more information, you can click on any of the fields in each packet to see more. When you click on a packet, you’re shown a breakdown of its internal bytes in the byte view section.  
[](https://cdn.comparitech.com/wp-content/uploads/2018/04/6-captured-packets-screenshot-.jpg)

### Packet List

The packet list pane is shown at the top of the screenshot. Each piece is broken down to a number with time, source, destination, protocol and support information.

### Packet Details

Packet details can be found in the middle, showing the protocols of the chosen packet. You can expand each section by clicking on the arrow next to your row of choice. You can also apply additional filters by right-clicking on the chosen item.

### Packet Bytes

The packet bytes pane is shown at the bottom of the page. This pane shows the internal data of your selected packet. If you highlight part of the data in this section, its corresponding information is also highlighted in the packet details pane. By default, all data is shown in hexadecimal format. If you want to change it to bit format, right-click the pane and select this option from the context menu.

## How to use Wireshark to Analyze Network Performance

If you want to use Wireshark to inspect your network and analyze all active traffic, then you need to close down all active applications on your network. This will reduce traffic to a minimum so you can see what is happening on your network more clearly. However, even if you turn off all of your applications, you’ll still have a mass of packets being sent and received.

Using Wireshark to filter these packets is the best way to take stock of your network data. When your connection is active, thousands of packets are transferring through your network every second. This means it’s vital that you filter out the information you don’t need to get a clear picture of what’s going on.

### Capture Filters and Display Filters

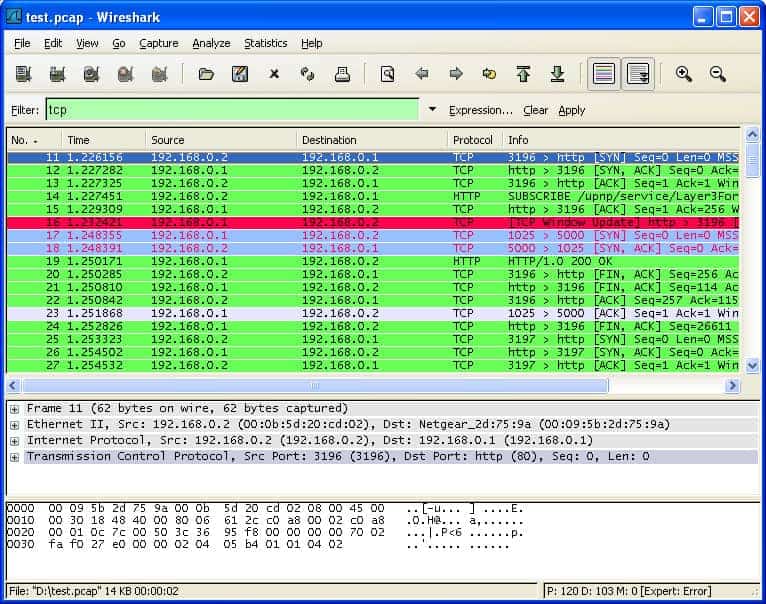
**Capture Filters** and **Display Filters** are two types of distinct filters that can be used on Wireshark.

Capture Filters and Display Filters are two types of distinct filters that can be used on Wireshark. Capture Filters are used to reduce the size of incoming packet capture, essentially filtering out other packets during live packet capturing. As a result, capture filters are set before you begin the live capture process.

Capture Filters are used to reduce the size of incoming packet capture, essentially filtering out other packets during live the packet capturing. As a result, capture filters are set before you begin the live capture process.

**Capture Filters can’t be modified once a capture has been started**. On the other hand, **Display Filters** can be used to filter data that has already been recorded. Capture Filters determine what data you capture from live network monitoring, and Display Filters dictate the data you see when looking through previously captured packets.

If you want to start filtering your data, one of the easiest ways to do this is to use the filter box below the toolbar. For example, if you type in HTTP in the filter box, you will be provided with a list of all HTTP packets captured. When you start typing, you’ll be met with an autocomplete field. The filter box is shown below:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/7-FILTERING-.jpg)

You can use hundreds of different filters to break down your packet information, from 104apci to zvt. An extensive list can be found on the Wireshark website [here](https://www.wireshark.org/docs/dfref/). You can also choose a filter by clicking on the bookmark icon to the left of the entry field. This will raise a menu of popular filters.

If you choose to set a capture filter, then your changes will come into effect once you start recording live network traffic. To activate a display filter, simply click on the arrow to the right of the entry field. Alternatively, you can click Analyze > Display Filters and choose a filter from the list of defaults.

After choosing a filter, you can view the TCP conversation behind a packet. To do this, right click on the packet and click Follow > TCP stream. This will show you the TCP exchange between the client and server.

If you want more information about Wireshark filtering, [Wireshark’s guide to display filters](https://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html) is a good point of reference.

### Using Color Coding

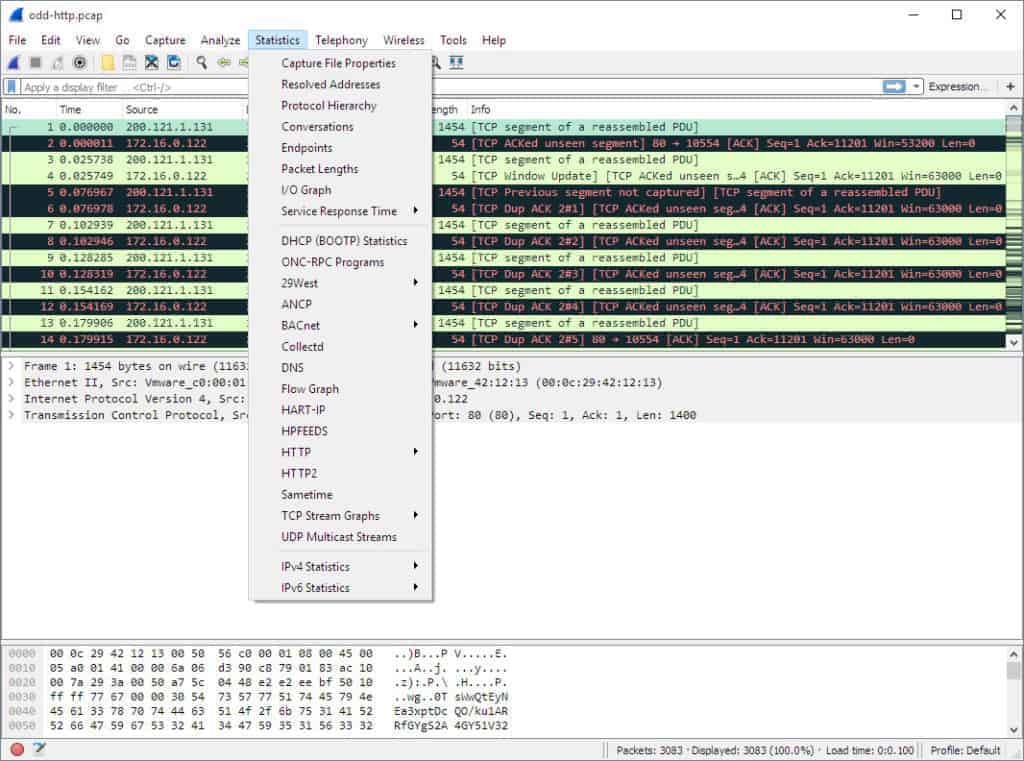
In addition to filtering which packets are shown or recorded, Wireshark’s color-coding facility makes it easier for the user to identify different packet types according to their color. For example, TCP traffic is denoted by light purple and UDP traffic is denoted by light blue. It’s important to note that black is used to highlight packets with errors.

On Wireshark’s default settings, there are around 20 colors you can choose from. You may edit, disable or delete these. If you want to turn off colorization, click on the **View**menu and click **Colorize Packet List** field to turn it off. If you’d like to view more information about the color-coding on Wireshark, click**View >Coloring Rules.**

### Viewing Network Statistics

To view more information on your network, the statistics drop-down menu is incredibly useful. The statistics menu can be located at the top of the screen and will provide you with several metrics from size and timing information to plotted charts and graphs. You can also apply display filters to these statistics to narrow down important information.

The Wireshark statistics menu is shown below:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/8-ws-statistics-menu.jpg)

In this menu are a variety of options to help you break down your network information.

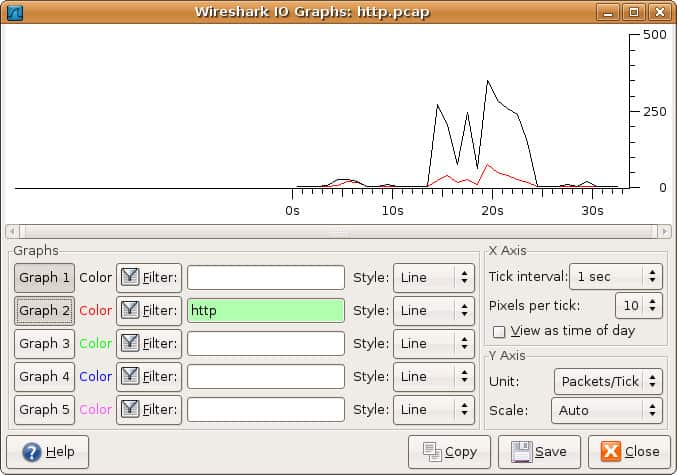
### Statistics Menu Selections

Here are some of the core sections:

* **Protocol Hierarchy** – The Protocol Hierarchy option raises a window with a complete table of all captured protocols. Active display filters are also displayed at the bottom.
* **Conversations** – Reveals the network conversation between two endpoints (For example exchange of traffic from one IP address to another).
* **Endpoints** – Displays a list of endpoints (a network endpoint is where protocol traffic of a specific protocol layer ends).
* **IO Graphs** – Displays user-specific graphs, visualizing the number of packets throughout the data exchange.
* **RTP\_statistics** – Allows the user to save the content of an RTP audio stream directly to an Au-file.
* **Service Response Time** – Displays the response time between a request and the network’s response.
* **TcpPduTime** – Displays the time taken to transfer data from a Protocol Data Unit. Can be used to find TCP retransmissions.
* **VoIP\_Calls** – Shows VoIP calls obtained from live captures.
* **Multicast Stream** – Detects multicast streams and measures the size of bursts and the output buffers of certain speeds.

## Visualizing Network Packets With IO Graphs

If you want to create a visual representation of your data packets, then you need to open IO graphs. Simply click on the **statistics**menu and select **IO graphs**. You’ll then be met by a graph window:

[](https://cdn.comparitech.com/wp-content/uploads/2018/04/9-IO-graphs-window.jpg)

You can **configure IO graphs with your own settings** according to the data you want to display. By default only graph 1 is enabled, so if you want to activate 2-5 you need to click on them. Likewise, if you want to apply a display filter for a graph, click the filter icon next to the graph you want to interact with. The style column allows you to change how your graph is structured. You can choose between **Line**, **FBar**, **Dot**, or **Impulse**.

You can also interact with the X and Y axis metrics on your graph as well. On the X-axis, the tick interval sections allow you to dictate how long the interval is, from minutes to seconds. You can also check the view as time of day checkbox to change the time of the X-axis.

Under the Y-axis section, you can change the unit of measurement from any of the following options: **Packets/Tick**, **Bytes/Tick**, **Bits/Tick**, or **Advanced**. The scale allows you to choose the scale of measurement for the Y-axis of the graph.

Once you press save the graph is then stored in a file format of your choice

# Experiment No- 14

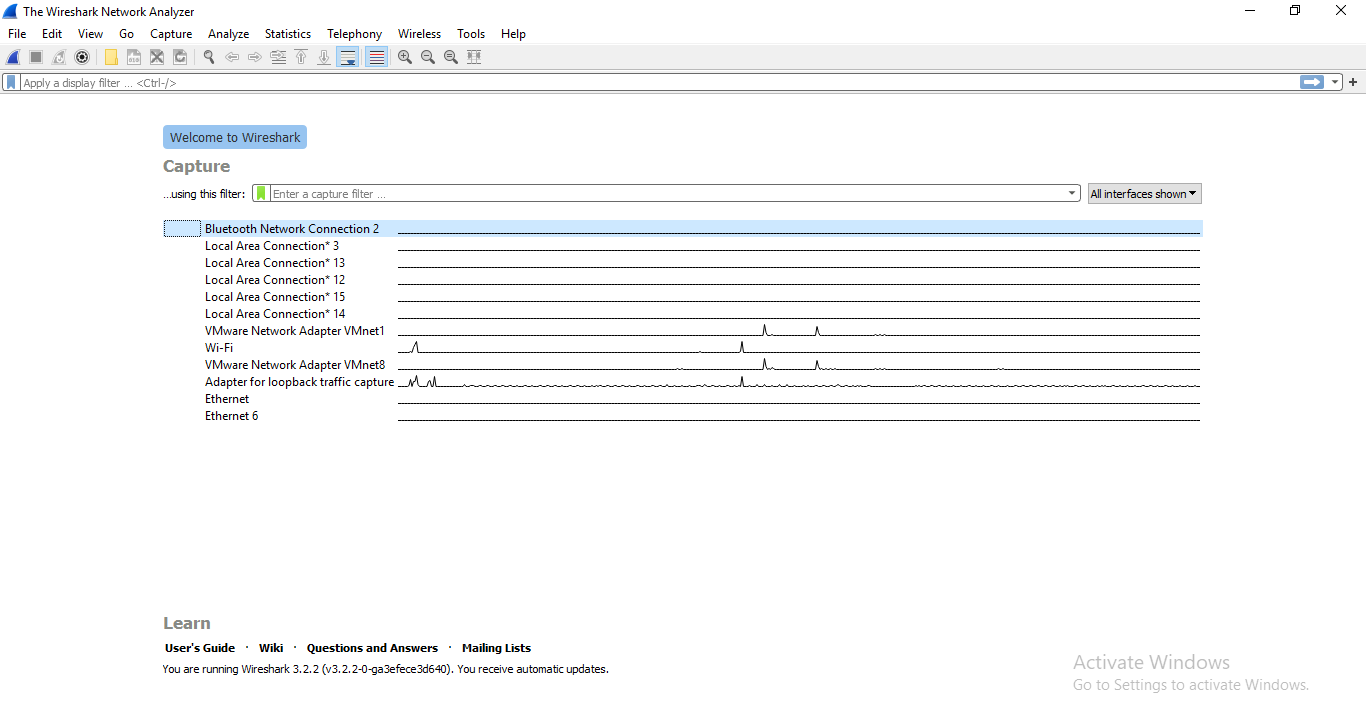
## AIM:

Packet Capture & Traffic Analysis with Wireshark.

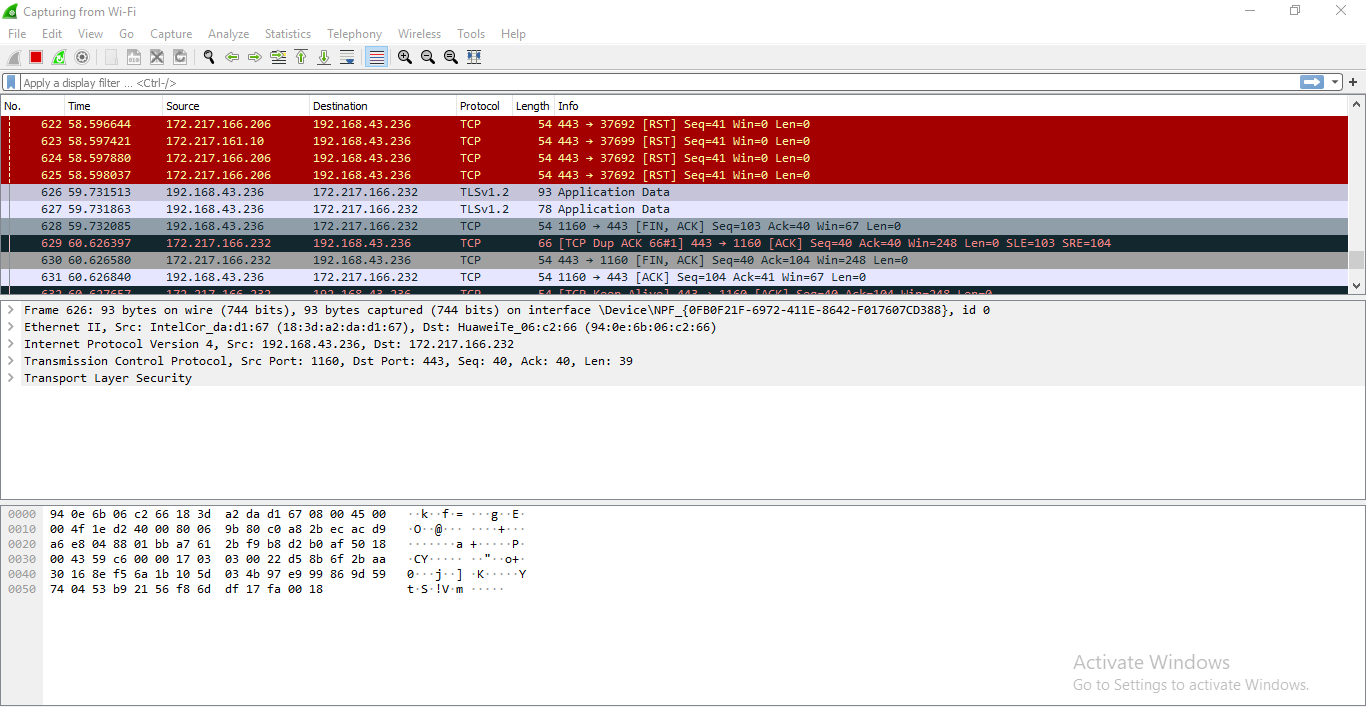
## Procedure:

Wireshark captures the data coming or going through the NICs on its device by using an underlying packet capture library. By default, Wireshark captures on-device data only, but it can capture almost all the data on its LAN if run in promiscuous mode. Currently, Wireshark uses NMAP’s Packet Capture library(called npcap).

**Getting Up and Running:** After installation launch Wireshark, approve the administrator or superuser privileges and you will be presented with a window that looks like this:



This window shows the interfaces on your device. To start sniffing select one interface and click on the bluefin icon on the top left. The data capture screen has three panes. The top pane shows real-time traffic, the middle one shows information about the chosen packet and the bottom pane shows the raw packet data. The top pane shows source address(IPv4 or IPv6) destination address, source and destination ports, protocol to which the packet belongs to and additional information about the packet.



Since there are a lot of packets going in and out every second, looking at all of them or searching for one type of packets will be tedious. This is why packet filters are provided. Packets can be filtered based on many parameters like IP address, port number or protocol at capture level or at display level. As obvious a display level filter will not affect the packets being captured.

**Some of the general capture filters are:**

 host (capture the traffic through a single target)

net( capture the traffic through a network or sub-network). “net” can be prefixed with “src” or “dst” to indicate whether the data coming from or going to the target host(s).)

port (capture the traffic through or from a port). “port” can be prefixed with “src” or “dst” to indicate whether the data coming from or going to the target port.

“and”, “not” and “or” logical connectives.(Used to combine multiple filters together).

There are some more basic filters and they can be combined very creatively. Another range of filters, display filters are used to create abstraction on captured data. These basic examples should provide a basic idea of their syntax:   
 tcp.port==80/udp.port==X shows the tcp/udp traffic at port X.

http.request.uri matches “parameter=value$” shows packets that are HTTP requests at the application layer level and their URI ends with a parameter with some value. The logical connective and or and not work here too.

ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16 will show traffic to and from workstations and servers. There is also a concept of coloring rules. Each protocol/port/other element is provided a unique color to make it easily visible for quick analysis. More details on coloring rules is [here](https://wiki.wireshark.org/ColoringRules)

**Plugins** are extra pieces of codes that can be embedded into the native Wireshark. Plugins help in analysis by:   Showing parameter specific statistics and insights. Handling capture files and issues related to their formats. Collaborating with other tools and frameworks to set up an all-in-one network monitoring solution. With just the basic capability to see all the traffic going through your device or in your LAN and the tools and plugins to help you in analysis, you can do a great deal of things with your device. Like:   
 Troubleshooting Internet connectivity problems with your device or WiFi.

Monitoring your device for unwanted traffic that may be an indication of a malware infection. Testing the working of your application that involve networking. Using it to just understand how computer networks work.