

**REPORT ON COMPUTER NETWORKS LAB(U18CSI5201L)**

***Submitted by***

**Indrajit M (22BCS036)**

**B.E-COMPUTER SCIENCE AND ENGINEERING**

**KUMARAGURU COLLEGE OF TECHNOLOGY COIMBATORE-641 049**

(An Autonomous Institution Affiliated to Anna University, Chennai)

**OCTOBER 2024**

**1**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Ex.No** | **Name of the experiment** | **Page.no** |
| **1.a** | **Develop a TCP Echo Client and Server Program using UNIX Socket Programming.** | **03** |
| **1.b** | **Develop a UDP Echo Client and Server Program using UNIX Socket Programming** | **10** |
| **2.a** | **Modify the above to make it as a TCP Chat Client and Server Program.** | **17** |
| **2.b** | **Modify the above to make it as a UDP Chat Client and Server Program.** | **24** |
| **3** | **Simulation of datalink and network layer protocols(ARP and RARP)** | **30** |
| **4** | **Performance analysis of TCP and UDP using simulation tool** | **33** |
| **5** | **Performance analysis of routing protocols using simulation tool.** | **38** |
| **6** | **Demonstrate the working of network tools such as Ping, TCP Dump, Traceroute, Netstat, Ipconfig.** | **46** |
| **7** | **Analyze the network traffic using Wireshark tool/Packet tracer tool** | **62** |

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 1.a

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise:** Develop a TCP Echo Client and Server Program using UNIX Socket

Programming

**STEP 1: INTRODUCTION**

#### OBJECTIVE OF THE EXERCISE/EXPERIMENT

To develop echo client server application using TCP

**STEP 2: ACQUISITION**

1. **Facilities/material required to do the exercise/experiment:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Facilities/material required** | **Quantity** |
| **1.** | **PC with Linux Platform** | **1/Student** |
| **2.** | **LAN connection** |  |

1. **Procedure for doing the exercise/experiment: SERVER:**
   1. Start the program.
   2. Declare the variables for the socket.
   3. Specify the family, IPaddress and port number.
   4. Create a socket using socket() function.
   5. Bind the IP address and port number.
   6. Listen and accept the client’s request for the connection.
   7. Read the client’s message.
   8. Display the client’s message.
   9. Close the socket.
   10. Stop the program.

#### CLIENT:

1. Start the program.
2. Declare the variable for the socket.
3. Specify the family, protocol, IP address and port number.
4. Create a socket using socket() function.
5. Call the connect() function.
6. Read the input message.
7. Send the input message to the server.
8. Display the server’s echo message.
9. close the socket.
10. Stop the program.

**Program**

**Echo client server application using TCP**

#### SERVER

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr; socklen\_t client\_addr\_len = sizeof(client\_addr); char buffer[BUFFER\_SIZE];

// Create a socket

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0); if (server\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the server address

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error binding socket"); close(server\_socket);

exit(1);

}

// Listen for incoming connections if (listen(server\_socket, 5) == -1) {

perror("Error listening for connections"); close(server\_socket);

exit(1);

}

printf("Server listening on port %d...\n", PORT);

// Accept a connection from a client

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len); if (client\_socket == -1) {

perror("Error accepting connection"); close(server\_socket);

exit(1);

}

printf("Client connected.\n");

// Receive and echo data while (1) {

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0); if (bytes\_received <= 0) {

printf("Connection closed by client.\n"); break;

}

buffer[bytes\_received] = '\0'; printf("Received: %s", buffer);

// Echo the received data back to the client send(client\_socket, buffer, strlen(buffer), 0);

}

// Close sockets close(client\_socket); close(server\_socket);

return 0;

}

#### CLIENT

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address #define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr; char buffer[BUFFER\_SIZE];

// Create a socket

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0); if (client\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

// Connect to the server

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error connecting to server"); close(client\_socket);

exit(1);

}

printf("Connected to server at %s:%d\n", SERVER\_IP, PORT);

// Send and receive data while (1) {

printf("Enter a message to send (or 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

if (strcmp(buffer, "quit\n") == 0)

{break;

}

// Send the message to the server send(client\_socket, buffer, strlen(buffer), 0);

// Receive and print the echo from the server

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0); if (bytes\_received <= 0) {

printf("Connection closed by server.\n"); break;

}

buffer[bytes\_received] = '\0'; printf("Received: %s", buffer);

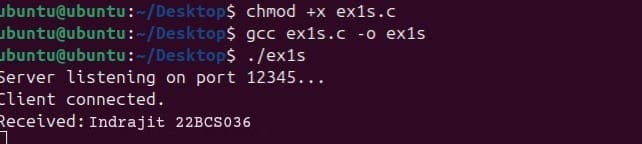
}

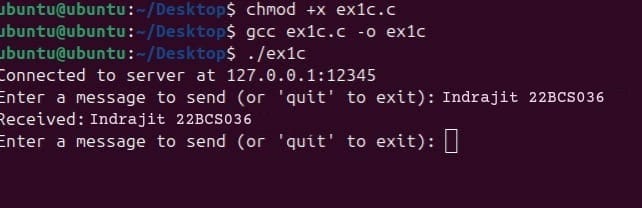
// Close the socket close(client\_socket);

return 0;

}

**OUTPUT**





**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 1.b

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise :** Develop a UDP Echo Client and Server Program using UNIX Socket

Programming

**STEP 1: INTRODUCTION**

#### OBJECTIVE OF THE EXERCISE/EXPERIMENT

To develop echo client server application using UDP

**STEP 2: ACQUISITION**

1. **Facilities/material required to do the exercise/experiment:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Facilities/material required** | **Quantity** |
| **1.** | **PC with Linux Platform** | **1/Student** |
| **2.** | **LAN connection** |  |

1. **Procedure for doing the exercise/experiment: SERVER:**
   1. Start the program.
   2. Declare the variables for the socket.
   3. Specify the family, IPaddress and port number.
   4. Create a socket using socket() function.
2. Bind the IP address and port number.
3. Listen and accept the client’s request for the connection.
4. Read the client’s message.
5. Display the client’s message.
6. Close the socket.
7. Stop the program.

#### CLIENT:

1. Start the program.
2. Declare the variable for the socket.
3. Specify the family, protocol, IP address and port number.
4. Create a socket using socket() function.
5. Call the connect() function.6)Read the input message.
6. Send the input message to the server.
7. Display the server’s echo message.
8. close the socket.
9. Stop the program.

**Program**

**Echo client server application using UDP SERVER:**

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int server\_socket;

struct sockaddr\_in server\_addr, client\_addr; socklen\_t client\_addr\_len = sizeof(client\_addr); char buffer[BUFFER\_SIZE];

// Create a socket

server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); if (server\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the server address

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error binding socket"); close(server\_socket);

exit(1);

}

printf("Server listening on port %d...\n", PORT);

// Receive and echo data while (1) {

int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (bytes\_received <= 0)

{ perror("Error receiving data");break;

}

buffer[bytes\_received] = '\0'; printf("Received: %s", buffer);

// Echo the received data back to the client

sendto(server\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&client\_addr, client\_addr\_len);

}

// Close the socket close(server\_socket);

return 0;

}

#### CLIENT:

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address #define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr; char buffer[BUFFER\_SIZE];

// Create a socket

client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); if (client\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

// Send and receive data while (1) {

printf("Enter a message to send (or 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

if (strcmp(buffer, "quit\n") == 0)

{break;

}

// Send the message to the server

sendto(client\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Receive and print the echo from the server

int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, NULL, NULL); if (bytes\_received <= 0) {

perror("Error receiving data"); break;

}

buffer[bytes\_received] = '\0'; printf("Received: %s", buffer);

}

// Close the socket close(client\_socket);

return 0;

}

**OUTPUT**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 2.a

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise :** Develop a TCP Chat Client and Server Program.

**STEP 1: INTRODUCTION**

#### OBJECTIVE OF THE EXERCISE/EXPERIMENT

To develop CHAT client server application using TCP

**STEP 2: ACQUISITION**

1. **Facilities/material required to do the exercise/experiment:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Facilities/material required** | **Quantity** |

|  |  |  |
| --- | --- | --- |
| **1.** | **PC with Linux Platform** | **1/Student** |
| **2.** | **LAN connection** |  |

1. **Procedure for doing the exercise/experiment:**
   * Start the program, declare the variables
   * Create a socket using the socket structure socket(AF\_INET, SOCK\_STREAM,0)
   * Set the socket family, IP address and the port using the server address
   * Set the socket address of 8 bytes to zero using the memset() function
   * Establish the connection to the server, and then create a child process
   * The child process send a message to the server using send function and receive themessage from the server
   * The client terminate the connection whenever it receive the bye message from theserver
   * Compile and execute the program

#### SERVER

* + Start the program, declare the variables
  + Create a socket using the socket structure socket(AF\_INET, SOCK\_STREAM,0)
  + Set the socket family, IP address and the port using the server address
  + Set the socket address of 8 bytes to zero using the memset() function
  + Bind and listen the socket structure
  + Accept the client connection using the socket descriptor and the server address
  + The child process receive the message from the client using the socket descriptor
  + The child process send the response to the client, and terminate the connectionwhenever it receive the bye message from the client
  + Compile and execute the program
  + Start the program, declare the variables

**Program SERVER:**

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr; socklen\_t client\_addr\_len = sizeof(client\_addr); char buffer[BUFFER\_SIZE];

// Create a socket

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0); if (server\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the server address

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error binding socket"); close(server\_socket);

exit(1);

}

// Listen for incoming connections if (listen(server\_socket, 5) == -1) {

perror("Error listening for connections"); close(server\_socket);

exit(1);

}

printf("Server listening on port %d...\n", PORT);

// Accept a connection from a client

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len); if (client\_socket == -1) {

perror("Error accepting connection");

close(server\_socket); exit(1);

}

printf("Client connected.\n");

// Chat loop while (1) {

// Receive a message from the client

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0); if (bytes\_received <= 0) {

printf("Connection closed by client.\n"); break;

}

buffer[bytes\_received] = '\0'; printf("Client: %s", buffer);

// Prompt for a reply

printf("Server (Type 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

// Send the reply to the client send(client\_socket, buffer, strlen(buffer), 0);

// Check if the server wants to quit if (strcmp(buffer, "quit\n") == 0) {

break;

}

}

// Close sockets close(client\_socket); close(server\_socket);

return 0;

}

#### CLIENT:

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address #define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr; char buffer[BUFFER\_SIZE];

// Create a socket

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0); if (client\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

// Connect to the server

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error connecting to server"); close(client\_socket);

exit(1);

}

printf("Connected to server at %s:%d\n", SERVER\_IP, PORT);

// Chat loop while (1) {

// Prompt for a message to send printf("Client (Type 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

// Send the message to the server send(client\_socket, buffer, strlen(buffer), 0);

// Check if the client wants to quit if (strcmp(buffer, "quit\n") == 0) {

break;

}

// Receive a message from the server

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0); if (bytes\_received <= 0) {

printf("Connection closed by server.\n"); break;

}

buffer[bytes\_received] = '\0'; printf("Server: %s", buffer);

}

// Close the socket close(client\_socket);

return 0;

}

**OUTPUT**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 2.b

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise :** Develop a UDP Chat Client and Server Program.

**STEP 1: INTRODUCTION**

#### OBJECTIVE OF THE EXERCISE/EXPERIMENT

To develop CHAT client server application using UDP

**STEP 2: ACQUISITION**

1. **Facilities/material required to do the exercise/experiment:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Facilities/material required** | **Quantity** |

|  |  |  |
| --- | --- | --- |
|  | **PC with Linux Platform** | **1/Student** |
| **1.** |  |  |
|  | **LAN connection** |  |
| **2.** |  |  |

1. **Procedure for doing the exercise/experiment:**
   * Start the program, declare the variables
   * Create a socket using the socket structure socket(AF\_INET, SOCK\_STREAM,0)
   * Set the socket family, IP address and the port using the server address
   * Set the socket address of 8 bytes to zero using the memset() function
   * Establish the connection to the server, and then create a child process
   * The child process send a message to the server using send function and receive themessage from the server
   * The client terminate the connection whenever it receive the bye message from theserver
   * Compile and execute the program

#### SERVER

* + Start the program, declare the variables
  + Create a socket using the socket structure socket(AF\_INET, SOCK\_STREAM,0)
  + Set the socket family, IP address and the port using the server address
  + Set the socket address of 8 bytes to zero using the memset() function
  + Bind and listen the socket structure
  + Accept the client connection using the socket descriptor and the server address
  + The child process receive the message from the client using the socket descriptor
  + The child process send the response to the client, and terminate the connectionwhenever it receive the bye message from the client
  + Compile and execute the program
  + Start the program, declare the variables

**Program**

#### SERVER

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int server\_socket;

struct sockaddr\_in server\_addr, client\_addr; socklen\_t client\_addr\_len = sizeof(client\_addr); char buffer[BUFFER\_SIZE];

// Create a socket

server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); if (server\_socket == -1) {

perror("Error creating socket");

exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the server address

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{perror("Error binding socket"); close(server\_socket);

exit(1);

}

printf("Server listening on port %d...\n", PORT);

// Receive and send data while (1) {

int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr

\*)&client\_addr, &client\_addr\_len); if (bytes\_received <= 0) {

perror("Error receiving data"); break;

}

buffer[bytes\_received] = '\0'; printf("Client: %s", buffer);

// Prompt for a reply

printf("Server (Type 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

// Send the reply to the client

sendto(server\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&client\_addr, client\_addr\_len);

// Check if the server wants to quit if (strcmp(buffer, "quit\n") == 0) {

break;

}

}

// Close the socket close(server\_socket);

return 0;

}

#### CLIENT

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address #define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr; char buffer[BUFFER\_SIZE];

// Create a socket

client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); if (client\_socket == -1) {

perror("Error creating socket"); exit(1);

}

// Set up the server address struct server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(PORT); server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

// Send and receive data while (1) {

// Prompt for a message to send printf("Client (Type 'quit' to exit): "); fgets(buffer, sizeof(buffer), stdin);

// Send the message to the server

sendto(client\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Check if the client wants to quit if (strcmp(buffer, "quit\n") == 0) {

break;

}

// Receive and print the server's reply

int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, NULL, NULL); if (bytes\_received <= 0) {

perror("Error receiving data"); break;

}

buffer[bytes\_received] = '\0'; printf("Server: %s", buffer);

}

// Close the socket close(client\_socket);

return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 3

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise : Simulation of datalink and network layer protocols**

AIM :

To write a C program to implement simulation of ARP and RARP network protocols. THEORY :

The term ARP is an abbreviation for Address resolution protocol. The ARP retrieves the receiver’s physical address in a network.

The term RARP is an abbreviation for Reverse Address Resolution Protocol. The RARP retrieves a computer’s logical address from its available server.

PROGRAM:

ARP/RARP CLIENT :

#include<stdio.h> #include<string.h> #include<sys/types.h> #include<sys/shm.h> main()

{

int shmid,a;

char \*ptr,\*shmptr;

char ptr2[51],ip[12],mac[26]; shmid=shmget(3000,10,0666); shmptr=shmat(shmid,NULL,0);

puts("The ARPtable is:"); printf("%s",shmptr); printf("\n1.ARP\n2.RARP\n3.EXIT\n"); scanf("%d",&a);

switch(a)

{

case 1:

puts("Enter ip address:"); scanf("%s",ip); ptr=strstr(shmptr,ip);

ptr-=8; sscanf(ptr,"%s%\*s",ptr2); printf("mac addr is:%s",ptr2); break; case 2:

puts("Enter mac addr"); scanf("%s",mac); ptr=strstr(shmptr,mac); sscanf(ptr,"%\*s%s",ptr2); printf("%s",ptr2).break; case 3:

exit(1);

}

}

ARP/RARP SERVER :

#include<stdio.h> #include<sys/types.h> #include<sys/shm.h> #include<string.h> main()

{

int shmid,a,i;

char \*ptr,\*shmptr; shmid=shmget(3000,10,IPC\_CREAT|0666); shmptr=shmat(shmid,NULL,0); ptr=shmptr;

for(i=0;i<3;i++)

{

puts("Enter the name:"); scanf("%s",ptr); a=strlen(ptr); printf("String length:%d",a); ptr[a]=' ';

puts("Enter ip:"); ptr=ptr+a+1; scanf("%s",ptr); ptr[a]='\n'; ptr=ptr+a+1;

}

ptr[strlen(ptr)]='\0';

printf("\nARP table at serverside is=\n%s",shmptr); shmdt(shmptr);

}

OUTPUT : ARP/RARP SERVER

A screenshot of a computer

Description automatically generated

ARP/RARP CLIENT

A screenshot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 4

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise : Performance analysis of TCP and UDP using simulation tool**

**AIM :**

To write a C program to perform analysis of TCP and UDP using simulation tool- ns2.

**THEORY :**

Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks. It simulates wired and wireless network.

**PROGRAM:**

**TCP**

set ns [new Simulator] set nf [open tcp.nam w]

$ns namtrace-all $nf set tf [open out.tr w]

$ns trace-all $tf proc finish {} { global ns nf tf

$ns flush-trace close $nf close $tf

exec nam tcp.nam & exit 0

}

set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node]

$ns duplex-link $n0 $n4 1Mb 50ms DropTail

$ns duplex-link $n1 $n4 1Mb 50ms DropTail

$ns duplex-link $n2 $n5 1Mb 1ms DropTail

$ns duplex-link $n3 $n5 1Mb 1ms DropTail

$ns duplex-link $n4 $n5 1Mb 50ms DropTail

$ns duplex-link-op $n4 $n5 queuePos 0.5 set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp set sink [new Agent/TCPSink]

$ns attach-agent $n2 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

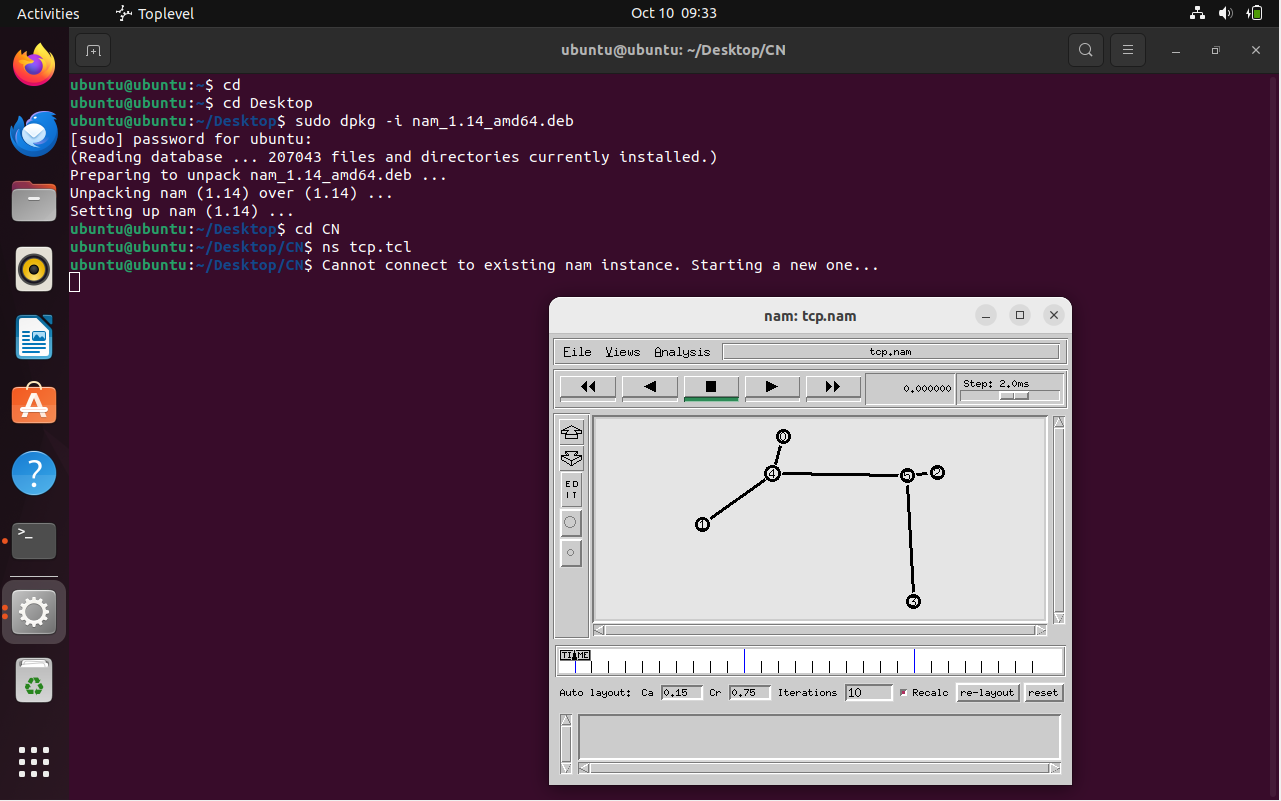
$ns at 0.0 "$ftp start"

$ns at 2.5 "$ftp stop"

$ns at 3 "finish"

$ns run

### OUTPUT :



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

UDP

set ns [new Simulator] set nf [open udp.nam w]

$ns namtrace-all $nf set tf [open out.tr w]

$ns trace-all $tf proc finish {} { global ns nf tf

$ns flush-trace close $nf close $tf

exec nam udp.nam & exit 0

}

set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node]

$ns duplex-link $n0 $n4 1Mb 50ms DropTail

$ns duplex-link $n1 $n4 1Mb 50ms DropTail

$ns duplex-link $n2 $n5 0.1Mb 1ms DropTail

$ns duplex-link $n3 $n5 1Mb 1ms DropTail

$ns duplex-link $n4 $n5 1Mb 50ms DropTail

$ns duplex-link-op $n2 $n5 queuePos 1 set tcp [new Agent/UDP]

$ns attach-agent $n0 $tcp set sink [new Agent/Null]

$ns attach-agent $n2 $sink

$ns connect $tcp $sink

set ftp [new Application/Traffic/CBR]

$ftp attach-agent $tcp

$ns at 0.0 "$ftp start"

$ns at 2.5 "$ftp stop"

$ns at 3 "finish"

$ns run

**OUTPUT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 5

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise : Performance analysis of routing protocols using simulation**

**tool.**

### LINK STATE ROUTING PROTOCOL AIM:

To simulate a link failure and to observe link state routing protocol in action.

### ALGORITHM:

1. Create a simulator object
2. Set routing protocol to link state routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create four nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a quad node.
8. Add TCP agent for node n0
9. Create FTP traffic on top of TCP and set traffic parameters.
10. Add a sink agent to node n3
11. Add UDP agent for node n2
12. Create CBR traffic on top of UDP and set traffic parameters.
13. Connect source and the sink
14. Schedule events as follows:
    1. Start traffic flow at 0.0
    2. Down the link n1-n3 at 1.0
    3. Up the link n1-n3 at 2.0
    4. Call finish procedure at 5.0
15. Start the scheduler
16. Observe the traffic route when link is up and down
17. View the simulated events and trace file analyze it
18. Stop

### PROGRAM :

set ns [new Simulator] set nf [open out.nam w]

$ns namtrace-all $nfset tr [open out.tr w]

$ns trace-all $trproc finish {} { global nf ns tr

$ns flush-traceclose $tr exec nam out.nam &exit 0

}

set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail

$ns duplex-link $n1 $n3 10Mb 10ms DropTail

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

$ns duplex-link-op $n0 $n1 orient right-down

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n1 orient right-upset tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp set ftp [new Application/FTP]

$ftp attach-agent $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sinkset udp [new Agent/UDP]

$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp set null [new Agent/Null]$ns attach-agent $n3 $null

$ns connect $tcp $sink

$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3

$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"

$ns at 0.0 "$cbr start"

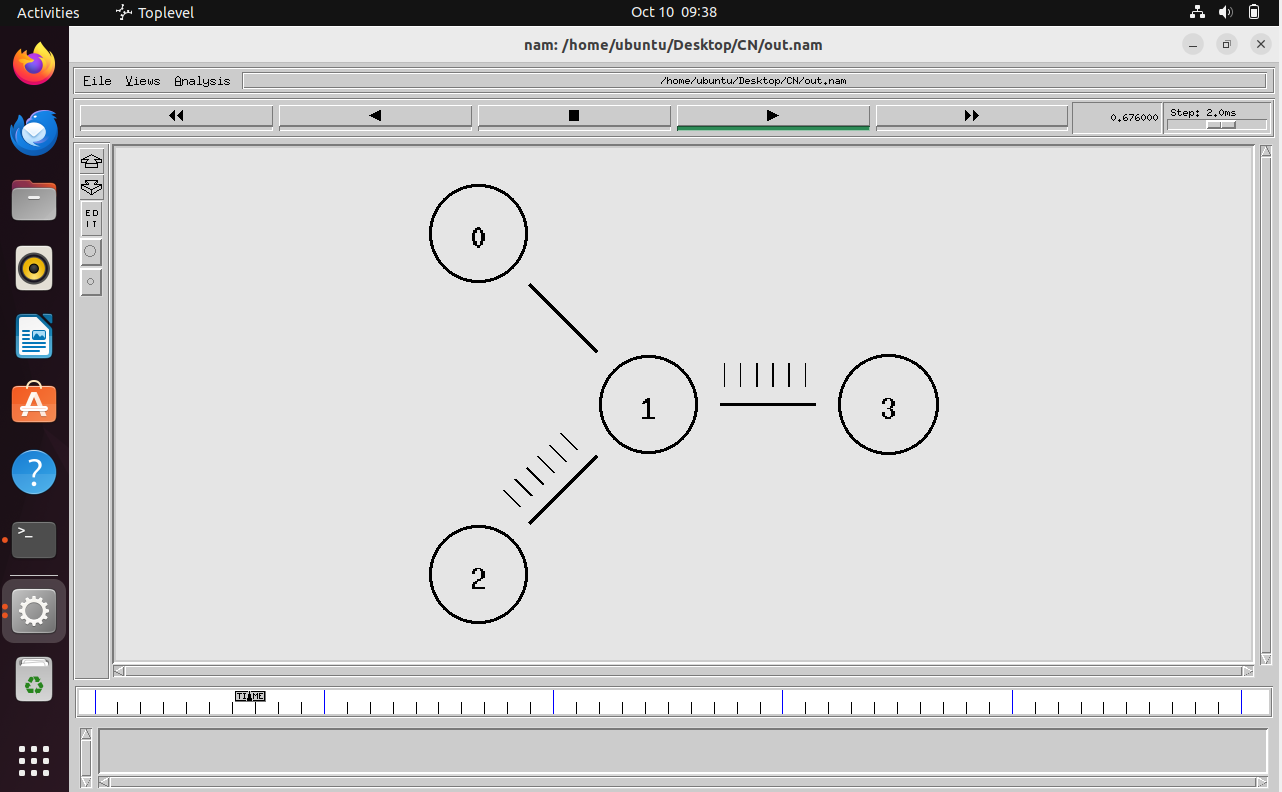
$ns at 5.0 "finish"

$ns run

### OUTPUT:

A computer screen shot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

**DISTANCE VECTOR ROUTINGPROTOCOL AIM:**

To simulate a link failure and to observe distance vector routing protocol in action.

### ALGORITHM:

1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    1. Start traffic flow at 0.5
    2. Down the link n3-n4 at 1.0
    3. Up the link n3-n4 at 2.0
    4. Stop traffic at 3.0
    5. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop the program.

### PROGRAM:

set ns [new Simulator]

$ns rtproto DV

set nf [open out.nam w]

$ns namtrace-all $nf set nt [open trace.tr w]

$ns trace-all $ntproc finish {} { global ns nf

$ns flush-traceclose $nf exec nam -a out.nam &exit 0

}

set n1 [$ns node]set n2 [$ns node]set n3 [$ns node]set n4 [$ns node]set n5 [$ns node]set n6 [$ns node]set n7 [$ns node]set n8 [$ns node]

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

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

$ns duplex-link $n3 $n4 1Mb 10ms DropTail

$ns duplex-link $n4 $n5 1Mb 10ms DropTail

$ns duplex-link $n5 $n6 1Mb 10ms DropTail

$ns duplex-link $n6 $n7 1Mb 10ms DropTail

$ns duplex-link $n7 $n8 1Mb 10ms DropTail

$ns duplex-link $n8 $n1 1Mb 10ms DropTail

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

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

$ns duplex-link-op $n3 $n4 orient right-up

$ns duplex-link-op $n4 $n5 orient right

$ns duplex-link-op $n5 $n6 orient right-down

$ns duplex-link-op $n6 $n7 orient down

$ns duplex-link-op $n7 $n8 orient left-down

$ns duplex-link-op $n8 $n1 orient leftset udp0 [new Agent/UDP]

$ns attach-agent $n1 $udp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0set null0 [new Agent/Null]

$ns attach-agent $n4 $null0

$ns connect $udp0 $null0

$ns at 0.0 "$n1 label Source"

$ns at 0.0 "$n4 label Destination"

$ns at 0.5 "$cbr0 start"

$ns rtmodel-at 1.0 down $n3 $n4

$ns rtmodel-at 2.0 up $n3 $n4

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run

**OUTPUT:**

A computer screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 6

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

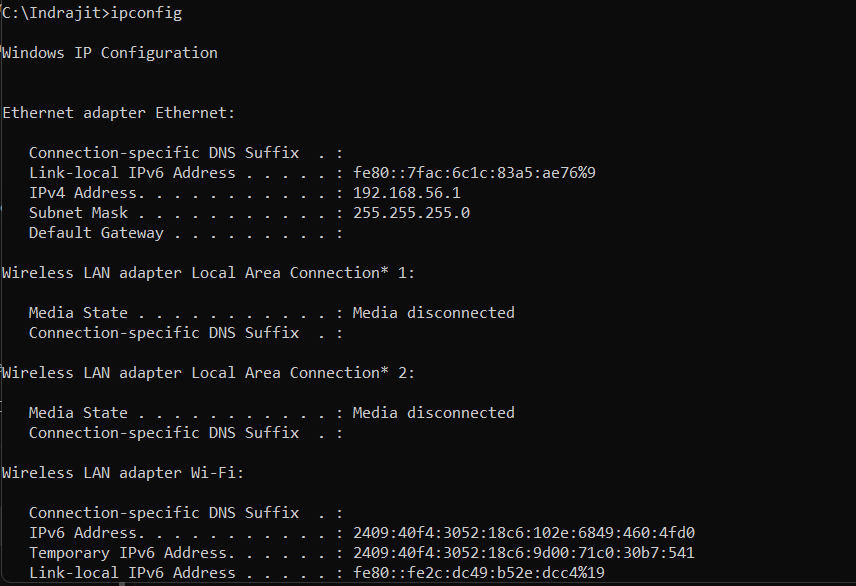
**Title of the exercise : Demonstrate the working of network tools such as Ping, TCP**

**Dump, Traceroute, Netstat, Ipconfig.**

**Networks Commands:**

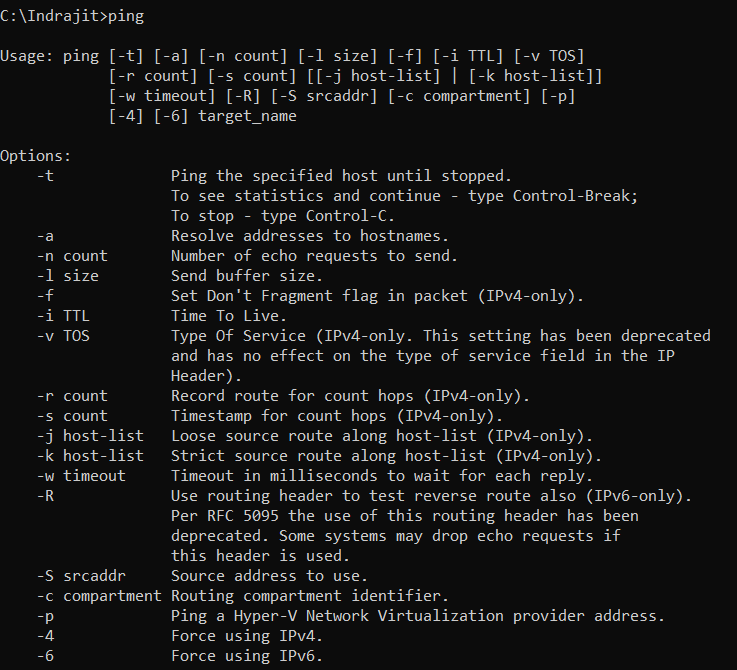
1. ipconfig

**ipconfig** (standing for "[Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) configuration") is a [console](https://en.wikipedia.org/wiki/Console_application) [application](https://en.wikipedia.org/wiki/Console_application) program of some computer [operating systems](https://en.wikipedia.org/wiki/Operating_system) that displays all current [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP) network configuration values and refreshes [Dynamic Host](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol) [Configuration Protocol](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol) (DHCP) and [Domain Name System](https://en.wikipedia.org/wiki/Domain_Name_System) (DNS) settings.[[1]](https://en.wikipedia.org/wiki/Ipconfig#cite_note-1) IPCONFIG



1. Ping

**ping** is a [computer network](https://en.wikipedia.org/wiki/Computer_network) administration [software utility](https://en.wikipedia.org/wiki/Utility_software) used to test the reachability of a [host](https://en.wikipedia.org/wiki/Host_(network)) on an [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP) network. It is available for virtually all operating systems that have networking capability, including most embedded network administration software.



1. Tracert google.com

The tracert (short for "trace route") command is a network diagnostic tool used in Windows operating systems to track the path that packets take from the source computer to a specified destination (usually another computer or server). This command is useful for identifying routing issues and understanding the network topology between two points.

A screenshot of a computer

Description automatically generated

1. nslookup

The nslookup (short for "name server lookup") command is a network utility tool used in Windows (and other operating systems) to query the Domain Name System (DNS) to obtain domain name or IP address mapping. It is useful for diagnosing DNS-related issues and for obtaining information about domain names and IP addresses.

A computer screen with white text

Description automatically generated

## netstat

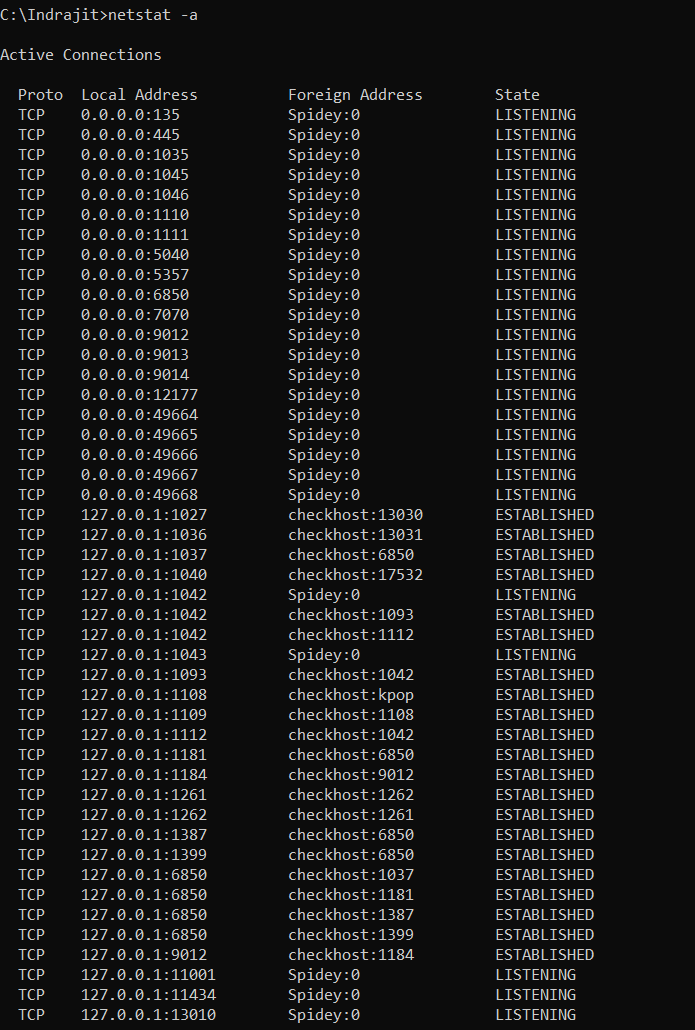
The netstat (short for "network statistics") command is a powerful network utility in Windows (and other operating systems) that displays various network- related information, including active connections, routing tables, and network interface statistics. It is a valuable tool for network administrators and users for diagnosing network issues and monitoring network activity.

A screenshot of a computer screen

Description automatically generated

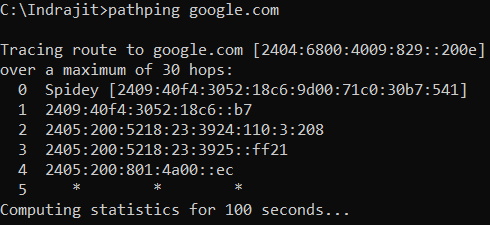
## netstat -a

The netstat -a command is a powerful tool used in Windows (and other operating systems) to display all active network connections and listening ports on the local computer. This command provides detailed information about both TCP and UDP connections, including the local and foreign addresses and the state of each connection.



## pathing

Pathping is a network diagnostic utility that combines the functionality of ping and tracert commands. It provides detailed information about the route packets take to a destination and analyzes the performance and reliability of each hop along the route. Pathping helps identify the path and pinpoint specific routers or network segments that may be causing network issues.



## Route

The route command in Windows is used to display and modify the routing table. The routing table determines the path that network traffic takes from your computer to its destination. It is an essential tool for network configuration and troubleshooting, allowing administrators to manage how packets are routed across a network.

A screenshot of a computer screen

Description automatically generated

## arp -a

The arp -a command in Windows is used to display the Address Resolution Protocol (ARP) cache, which contains mappings between IP addresses and their corresponding MAC (Media Access Control) addresses. The ARP cache is used to store IP-to-MAC address mappings that the system has discovered, making it quicker to find the MAC address for a given IP address in subsequent communications.

A screenshot of a computer program

Description automatically generated

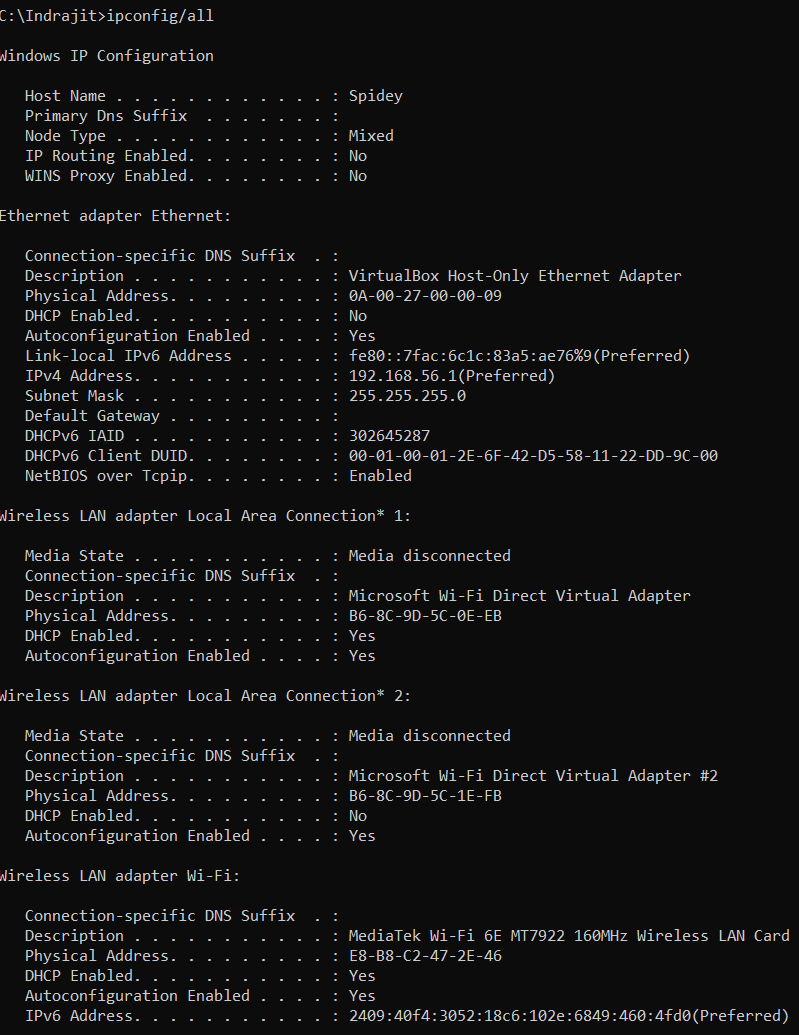
## hostname

The hostname command in Windows is used to display the name of the current machine or host. This command is simple and straightforward, providing only the hostname of the computer on which it is run.



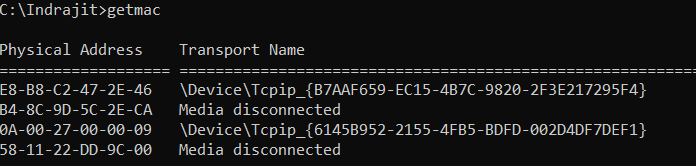
## ipconfig / all

The ipconfig /all command in Windows displays detailed information about the network configuration of all network interfaces on the computer. This includes IP addresses, subnet masks, default gateways, DNS servers, and much more. It provides a comprehensive view of the network settings, making it a valuable tool for troubleshooting and configuring network connections.



## getmac

The getmac command in Windows is used to display the MAC (Media Access Control) addresses for the network adapters on the system. The MAC address is a unique identifier assigned to network interfaces for communications on the physical network segment. This command can be useful for network management, troubleshooting, and inventory purposes.



## pathping

The pathping command in Windows combines the functionality of ping and tracert to provide detailed information about network latency and packet loss at each hop between a source and destination. It helps diagnose network issues by identifying problematic nodes along the route to a target host.

A screenshot of a computer program

Description automatically generated

## netsh interface show interface

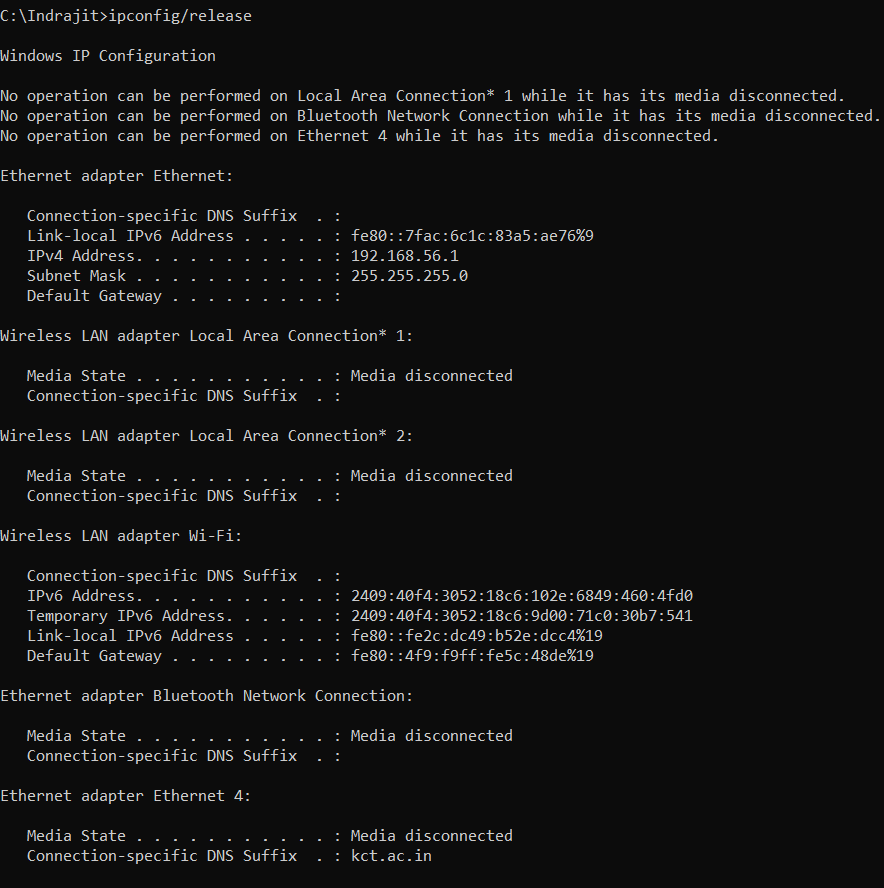
In Windows, the equivalent to nmcli connection show to display network connections and their details can be achieved using several commands and tools. Here are a few ways to get detailed information about network connections in Windows:

A screenshot of a computer

Description automatically generated

## ipconfig / release

The ipconfig /release command in Windows is used to release the current IP address configuration for all network adapters. This means it will release the DHCP lease, effectively removing the current IP address assigned to the network interfaces.



## ipconfig /renew

The ipconfig /renew command in Windows is used to renew the DHCP lease for all network adapters. This means it will request a new IP address from the DHCP server for the network interfaces, effectively updating the current IP address configuration.

A screenshot of a computer

Description automatically generated

## tasklist / svc

The tasklist /svc command in Windows is used to display a list of active processes and the services that are running within each process. This is useful for identifying which services are associated with which processes, providing a detailed view of the system's activity.

A screenshot of a computer program

Description automatically generated

## netsh interface ip show config

The netsh interface ip show config command in Windows is used to display detailed configuration information for all network interfaces (both IPv4 and IPv6) on the system. This includes IP addresses, subnet masks, default gateways, DNS servers, and more.

A computer screen shot of white text

Description automatically generated

## netstat -s

The netstat -s command in Windows displays statistics for a variety of network protocols and services. It provides a comprehensive summary of network activity and performance metrics, which can be useful for diagnosing network issues and monitoring network usage.

A screen shot of a computer

Description automatically generated

**KUMARAGURU COLLEGE OF TECHNOLOGY**

# Exercise/Experiment Number: 7

**Lab Code / Lab : U18CSI5201L/ COMPUTER NETWORKS LABORATORY**

**Course / Branch : III BE CSE**

**Title of the exercise : Analyze the network traffic using Wireshark tool/Packet**

**tracer tool.**

**AIM :** To know how to capture packets in wireshark

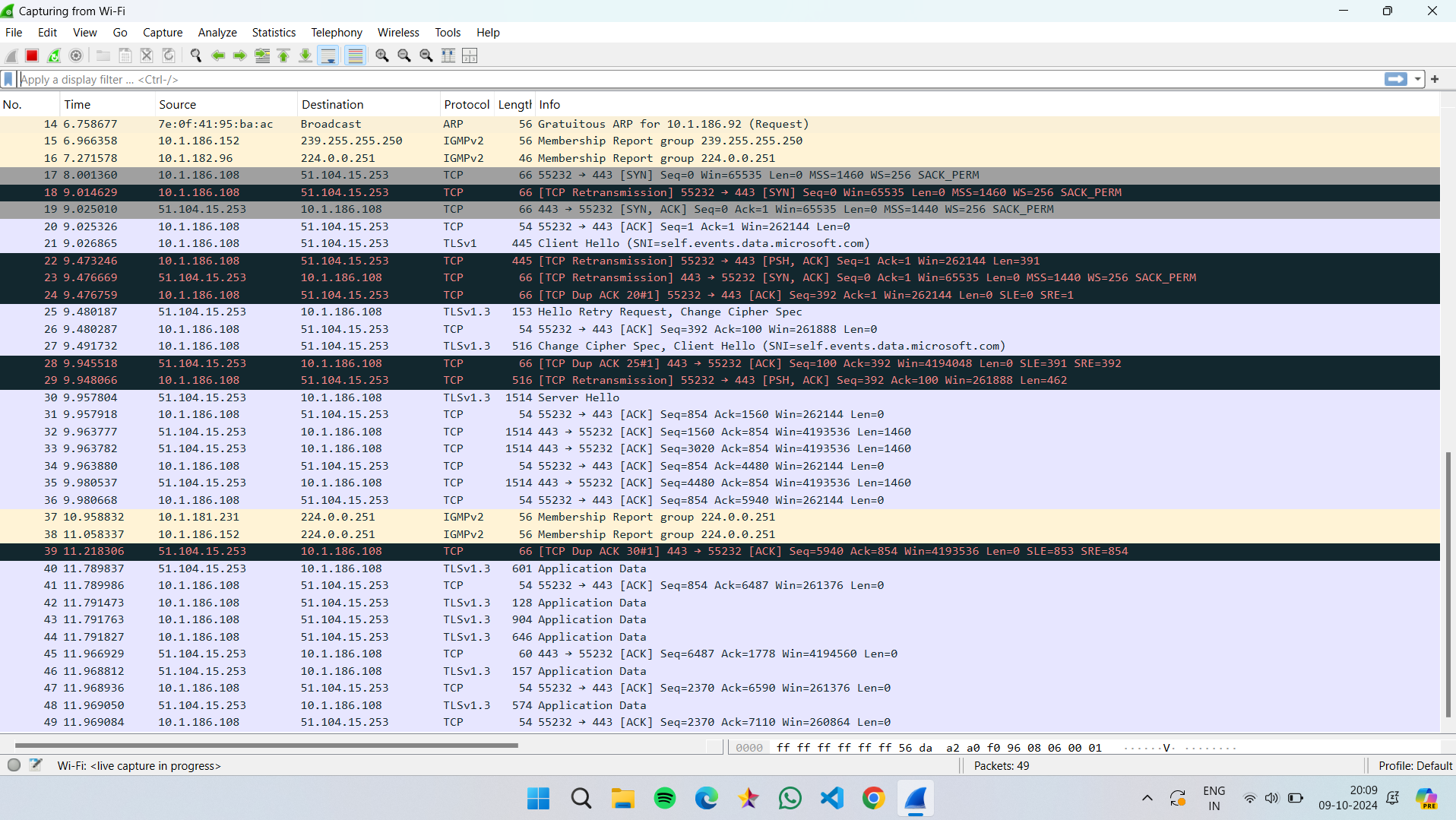
### THEORY:

Wireshark is the world’s foremost and widely used network protocol analyser. It lets you see what is happening on your network at a microscopic level.

Wireshark has a rich feature set which includes the following:

* Deep inspection of hundreds of protocols, with more being added all the time
* Live capture and offline analysis
* Capture files compressed with gzip can be decompressed on the fly
* Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others

**OUTPUT :**

****

**A screenshot of a computer

Description automatically generated**

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated