C. U. Shah University , wadhwan City

NEXT GENERATION NETWORKS

(4TE08NGN1)

8TH SEMESTER B. TECH ENGG.

NGN

C. U. SHAH COLLEGE OF ENGINEERING & TECHNOLOGY

यशो सर्वदा वर्धमानम्

C. U. Shah Technical Institute of Diploma Studies

Wadhwan City - 363030

Faculty of Technology & Engineering

C. U. Shah College of Engineering & Technology

(Managed By Wardhman Bharti Trust)

Accredited by NAAC with B Grade

WADWAN CITY – 363030

Certificate

This is to certify that

Mr/Ms. _____

Enrollment No. _____ *Of* **8th Semester**

B. Tech Courses in **Computer Engineering** *has satisfactory completed his/her Term work in* **NEXT GENERATION NETWORKS (4TE08NGN1)** *with in four Walls of Laboratory / Drawing Hall of This College during the year* **2025-26**

Staff In charge.

Date of Submission : _____Head of The Department

List Of Practical

Next Generation Networks (NGN) (4TE08NGN1)

Student Name:

Student Enrollment No.:

Sr. No	Date	Practical	Page No	Marks	Sign of Faculty
01		HTTP Web Client Program to Download a Web Page Using TCP Sockets			
02		Socket Program for Echo			
03		(a) Simulation of DNS Using UDP Sockets (b) Client-Server Application for Chat			
04		File Transfer Between Client and Server			
05		(a) Simulating ARP Protocols (b) Write a Code Simulating RARP Protocols			
06		Study of Network Simulator and Simulation of Congestion Control Algorithms Using NS			
07		Study of TCP/UDP Performance Using Simulation Tool			
08		(a) Simulation of Distance Vector Routing Algorithm (b) Simulation of Link State Routing Algorithm			
09		Simulation of Error Correction Code (like CRC)			

Practical 1: HTTP Web Client Program to Download a Web Page Using TCP Sockets

AIM:

To write a web client program to download a webpage using TCP sockets.

Theory:

The HTTP protocol is used for transferring web pages. The web client establishes a TCP connection, sends an HTTP GET request to a server, and retrieves the HTML content in response.

Code :

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

void get_web_page(const char *host, const char *page) {
    int sockfd;
    struct sockaddr_in server_addr;
    char request[1024], response[4096];

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(80);
    server_addr.sin_addr.s_addr = inet_addr("93.184.216.34"); // example.com IP
```

```
    // Connect to the server
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(1);
    }

    // Prepare HTTP GET request
    sprintf(request, "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", page, host);

    // Send GET request
    send(sockfd, request, strlen(request), 0);

    // Receive response
    int n;
    while ((n = recv(sockfd, response, sizeof(response), 0)) > 0) {
        write(STDOUT_FILENO, response, n);
    }

    // Close the socket
    close(sockfd);
}

int main() {
    get_web_page("example.com", "/");
    return 0;
}
```

Question & Answers:

1. **Q:** What is the purpose of the GET request in the HTTP protocol?

- **A:** The GET request is used by the client to request a specific resource (e.g., a web page) from the server.

2. **Q:** Why is the connection closed after receiving the response?
    - **A:** The connection is closed to free up resources and follow the HTTP protocol for connection management.

Result:

- The HTML content of the requested webpage will be printed on the terminal.

Observation:

- The client successfully connects to the web server and retrieves the requested page's HTML content.

Practical 2a: Socket Program for Echo

AIM:

To create a socket program that implements a simple echo server and client.

Theory:

An echo server receives messages from a client and sends the same message back (echoes). This demonstrates basic socket programming where data is exchanged between client and server.

Code :

Echo Server:

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

void start_echo_server(int port) {
   int sockfd, clientfd;
   struct sockaddr_in server_addr, client_addr;
   socklen_t addr_len = sizeof(client_addr);
   char buffer[1024];

   // Create socket
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0) {
      perror("Socket creation failed");
      exit(1);
   }

   // Set up server address
   server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));

    // Listen for incoming connections
    listen(sockfd, 5);
    printf("Server listening...\n");

    // Accept client connection
    clientfd = accept(sockfd, (struct sockaddr *)&client_addr, &addr_len);
    if (clientfd < 0) {
        perror("Accept failed");
        exit(1);
    }

    // Echo received messages
    while (1) {
        memset(buffer, 0, sizeof(buffer));
        int n = recv(clientfd, buffer, sizeof(buffer), 0);
        if (n <= 0) break;  // End of message
        send(clientfd, buffer, n, 0);
    }

    close(clientfd);
    close(sockfd);
}

int main() {
```

```
    start_echo_server(12345);
    return 0;
}
```

Echo Client:

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

void echo_client(const char *server_ip, int port, const char *message) {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[1024];

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }

    // Set up server address
    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(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
```

```
    perror("Connection failed");
    exit(1);
  }

  // Send message
  send(sockfd, message, strlen(message), 0);

  // Receive echo
  int n = recv(sockfd, buffer, sizeof(buffer), 0);
  buffer[n] = '\0';
  printf("Echo from server: %s\n", buffer);

  close(sockfd);
}

int main() {
  echo_client("127.0.0.1", 12345, "Hello, Server!");
  return 0;
}
```

Question & Answers:

1. **Q:** What does the recv() function do in socket programming?
    - **A:** recv() is used to receive data from the connected socket.
2. **Q:** What happens when the server closes the connection?
    - **A:** The connection is terminated, and the server stops listening for further messages.

Result:

- The client sends a message to the server, and the server responds with the same message.

Observation:

- The client sends and receives the same message, showing successful echoing of data.

Practical 3a: Simulation of DNS Using UDP Sockets

AIM:

To simulate DNS (Domain Name System) using UDP sockets for querying domain names.

Theory:

DNS resolves domain names into IP addresses. UDP is used because DNS queries are lightweight and require low latency. In this simulation, a client sends a domain name, and the server responds with an IP address.

Code :

DNS Server (UDP):

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

void dns_server(int port) {
   int sockfd;
   struct sockaddr_in server_addr, client_addr;
   socklen_t addr_len = sizeof(client_addr);
   char domain[1024], ip_address[] = "192.168.1.1";  // Simulated IP for domain

   // Create UDP socket
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd < 0) {
      perror("Socket creation failed");
      exit(1);
   }

   // Set up server address
   server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));

    printf("DNS Server running...\n");

    while (1) {
        // Receive domain name
        recvfrom(sockfd, domain, sizeof(domain), 0, (struct sockaddr *)&client_addr, &addr_len);

        // Send IP address back to the client
        sendto(sockfd, ip_address, strlen(ip_address), 0, (struct sockaddr *)&client_addr, addr_len);
    }

    close(sockfd);
}

int main() {
    dns_server(53);
    return 0;
}
```

DNS Client (UDP):

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

```
void dns_client(const char *server_ip, int port, const char *domain) {
    int sockfd;
    struct sockaddr_in server_addr;
    char ip_address[1024];

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);

    // Send domain name
    sendto(sockfd, domain, strlen(domain), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr));

    // Receive IP address
    recvfrom(sockfd, ip_address, sizeof(ip_address), 0, NULL, NULL);
    printf("Resolved IP for domain '%s': %s\n", domain, ip_address);

    close(sockfd);
}

int main() {
```

```
    dns_client("127.0.0.1", 53, "example.com");
    return 0;
}
```

Question & Answers:

1. **Q:** Why does DNS use UDP instead of TCP?
    - **A:** DNS uses UDP because it is faster, connectionless, and has low overhead for short query-response communication.
2. **Q:** What is the advantage of simulating DNS using UDP?
    - **A:** UDP is well-suited for simple, lightweight, and fast query-response protocols like DNS.

Result:

- The client receives an IP address in response to the domain query.

Observation:

- The server successfully resolves domain names to IP addresses using a simulated DNS protocol.

Practical 3b: Client-Server Application for Chat

AIM:

To create a chat application where the client and server can send and receive messages over TCP sockets.

Theory:

This application demonstrates communication between a client and server. The server listens for incoming connections, and the client sends messages. Both client and server can send and receive messages simultaneously.

Code :

Chat Server:

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

#define PORT 12345
#define MAX 100

void chat_server() {
   int sockfd, clientfd;
   struct sockaddr_in server_addr, client_addr;
   socklen_t addr_len = sizeof(client_addr);
   char buffer[MAX];

   // Create socket
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0) {
      perror("Socket creation failed");
      exit(1);
```

```
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = INADDR_ANY;

// Bind socket
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));

// Listen for client connections
listen(sockfd, 5);
printf("Server is waiting for client connection...\n");

// Accept client connection
clientfd = accept(sockfd, (struct sockaddr *)&client_addr, &addr_len);
if (clientfd < 0) {
   perror("Accept failed");
   exit(1);
}

// Communication with client
while (1) {
   memset(buffer, 0, sizeof(buffer));
   int n = recv(clientfd, buffer, sizeof(buffer), 0);
   if (n <= 0) break;
   printf("Client: %s\n", buffer);
   printf("Server: ");
   fgets(buffer, sizeof(buffer), stdin);
   send(clientfd, buffer, strlen(buffer), 0);
}
```

```
    close(clientfd);
    close(sockfd);
}

int main() {
    chat_server();
    return 0;
}
```

Chat Client:

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

#define MAX 100

void chat_client(const char *server_ip) {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[MAX];

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }
```

```
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(12345);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);

    // Connect to server
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(1);
    }

    // Chat with server
    while (1) {
        printf("Client: ");
        fgets(buffer, sizeof(buffer), stdin);
        send(sockfd, buffer, strlen(buffer), 0);
        memset(buffer, 0, sizeof(buffer));
        int n = recv(sockfd, buffer, sizeof(buffer), 0);
        if (n <= 0) break;
        printf("Server: %s", buffer);
    }

    close(sockfd);
}

int main() {
    chat_client("127.0.0.1");
    return 0;
}
```

Question & Answers:

1. **Q:** What happens if the server crashes while the client is connected?
    - **A:** The client will encounter a connection error when trying to send or receive data.
2. **Q:** How does the server know when to stop communication with the client?
    - **A:** The server receives a message with zero bytes (EOF), indicating the end of the communication.

Result:

- The client and server exchange messages, simulating a simple chat application.

Observation:

- Messages are exchanged successfully between client and server, demonstrating basic TCP communication.

Practical 4: File Transfer Between Client and Server

AIM:

To implement a file transfer system between a client and a server using TCP sockets.

Theory:

This program enables the client to send a file to the server, which saves it on the server's side. File transfer is done in chunks to avoid large data loss.

Code :

File Server:

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

#define PORT 12345
#define FILENAME "received_file.txt"

void file_server() {
    int sockfd, clientfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[1024];

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }
```

```
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = INADDR_ANY;

// Bind the socket
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));

// Listen for incoming connections
listen(sockfd, 5);
printf("Server listening for file transfer...\n");

// Accept client connection
clientfd = accept(sockfd, (struct sockaddr *)&client_addr, &addr_len);
if (clientfd < 0) {
   perror("Accept failed");
   exit(1);
}

// Receive file and save it
FILE *fp = fopen(FILENAME, "w");
if (fp == NULL) {
   perror("File creation failed");
   exit(1);
}

while (1) {
   int n = recv(clientfd, buffer, sizeof(buffer), 0);
   if (n <= 0) break;
   fwrite(buffer, sizeof(char), n, fp);
```

```
    }

    printf("File received and saved as %s\n", FILENAME);
    fclose(fp);
    close(clientfd);
    close(sockfd);
}

int main() {
    file_server();
    return 0;
}
```

File Client:

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

#define FILENAME "file_to_send.txt"

void file_client(const char *server_ip) {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[1024];
    FILE *fp = fopen(FILENAME, "r");

    if (fp == NULL) {
        perror("File not found");
```

```
    exit(1);
  }

  // Create socket
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
    perror("Socket creation failed");
    exit(1);
  }

  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(12345);
  server_addr.sin_addr.s_addr = inet_addr(server_ip);

  // Connect to server
  if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    perror("Connection failed");
    exit(1);
  }

  // Send file content
  while (fgets(buffer, sizeof(buffer), fp)) {
    send(sockfd, buffer, strlen(buffer), 0);
  }

  printf("File sent successfully\n");
  fclose(fp);
  close(sockfd);
}
```

```
int main() {
  file_client("127.0.0.1");
  return 0;
}
```

Question & Answers:

1. **Q:** What would happen if the server does not receive the complete file?
    - **A:** The file would be incomplete, and data corruption may occur.
2. **Q:** How is the file read and sent in chunks?
    - **A:** The file is read line-by-line and sent to the server using the send() function.

Result:

- The client successfully sends a file to the server, and the server saves it as "received_file.txt."

Observation:

- The file transfer works as expected, with no data loss or corruption.

Practical 5a: Simulating ARP Protocols

AIM:

To simulate the ARP (Address Resolution Protocol) which maps IP addresses to MAC addresses.

Theory:

ARP is used to find the MAC address corresponding to an IP address within a local network. When a device needs to communicate with another device on the same network, it uses ARP to obtain the MAC address associated with the destination IP address.

Code :

```
#include <stdio.h>
#include <string.h>

#define ARP_TABLE_SIZE 5

// ARP Table (IP -> MAC mapping)
struct arp_entry {
   char ip_address[16];
   char mac_address[18];
};

struct arp_entry arp_table[ARP_TABLE_SIZE] = {
   {"192.168.1.1", "00:1A:2B:3C:4D:5E"},
   {"192.168.1.2", "00:1A:2B:3C:4D:5F"},
   {"192.168.1.3", "00:1A:2B:3C:4D:60"},
   {"192.168.1.4", "00:1A:2B:3C:4D:61"},
   {"192.168.1.5", "00:1A:2B:3C:4D:62"}
};

// Function to simulate ARP request
void arp_request(const char *ip_address) {
   for (int i = 0; i < ARP_TABLE_SIZE; i++) {
```

```
    if (strcmp(arp_table[i].ip_address, ip_address) == 0) {
      printf("ARP Response: IP Address %s -> MAC Address %s\n", ip_address,
arp_table[i].mac_address);
      return;
    }
  }
  printf("ARP Response: IP Address %s not found\n", ip_address);
}

int main() {
  arp_request("192.168.1.1");
  arp_request("192.168.1.10");  // IP not in ARP table
  return 0;
}
```

Question & Answers:

1. **Q:** What is the purpose of ARP in networking?
    - **A:** ARP is used to map IP addresses to MAC addresses to allow devices on the same local network to communicate.
2. **Q:** What happens if an IP address is not found in the ARP cache?
    - **A:** The device will broadcast an ARP request to find the corresponding MAC address.

Result:

- The ARP table is populated with the IP-MAC mapping and can be queried for an IP address to get its MAC address.

Observation:

- The ARP simulation successfully shows how IP to MAC address mapping is done within a local network.

Practical 5b: Write a Code Simulating RARP Protocols

AIM:

To simulate the RARP (Reverse Address Resolution Protocol), which maps a MAC address to an IP address.

Theory:

RARP is used by a device to request its IP address from a server when it knows its MAC address but not its IP address. This simulates the process of a device getting an IP address from a RARP server.

Code :

```
#include <stdio.h>
#include <string.h>

#define RARP_TABLE_SIZE 5

// RARP Table (MAC -> IP mapping)
struct rarp_entry {
   char mac_address[18];
   char ip_address[16];
};

struct rarp_entry rarp_table[RARP_TABLE_SIZE] = {
   {"00:1A:2B:3C:4D:5E", "192.168.1.1"},
   {"00:1A:2B:3C:4D:5F", "192.168.1.2"},
   {"00:1A:2B:3C:4D:60", "192.168.1.3"},
   {"00:1A:2B:3C:4D:61", "192.168.1.4"},
   {"00:1A:2B:3C:4D:62", "192.168.1.5"}
};

// Function to simulate RARP request
void rarp_request(const char *mac_address) {
```

```
  for (int i = 0; i < RARP_TABLE_SIZE; i++) {
    if (strcmp(rarp_table[i].mac_address, mac_address) == 0) {
      printf("RARP Response: MAC Address %s -> IP Address %s\n", mac_address,
rarp_table[i].ip_address);
      return;
    }
  }
  printf("RARP Response: MAC Address %s not found\n", mac_address);
}

int main() {
  rarp_request("00:1A:2B:3C:4D:5E");
  rarp_request("00:1A:2B:3C:4D:99");  // MAC not in RARP table
  return 0;
}
```

Question & Answers:

1. **Q:** Why do devices use RARP instead of ARP?
    - **A:** Devices use RARP when they need to obtain an IP address based on their MAC address (e.g., when booting from a diskless setup).
2. **Q:** What happens if a MAC address is not found in the RARP table?
    - **A:** The device will not receive an IP address, and the communication will fail.

Result:

- The system simulates the process of a device requesting its IP address based on its MAC address.

Observation:

- The simulated RARP protocol successfully maps MAC addresses to IP addresses.

Practical 6: Study of Network Simulator and Simulation of Congestion Control Algorithms Using NS

AIM:

To study and simulate congestion control algorithms in networks using a network simulator like NS2 or NS3.

SYSTEM REQUIREMENTS:

PC: Pentium or higher

One LAN card onboard or on PCI slot with 10/100Mbps speed.
128MB RAM

500MB free space on Hard
drive CD ROM drive

Serial port, LPT port & USB port installed on system Operating
System: Windows 2000 or higher

THEORY:

LTS-01 Local area network / wireless local area network trainer system:

It is designed to help students understand the basic concepts, modes of operation and protocols involved in networking. The trainer has integrated hardware flow control on panel board for better understanding of different types of LAN topologies involved in networking. The trainer system is provided with windows-based user friendly software with analysis of protocols, different layers, network and measurement of error rate and throughput.

Students can easily do connections in different topologies and can learn actual data transfer either through hardware or through simulated network concept. Facility is provided into system software to introduce errors into packets being sent and analyze the effect of error on different protocols and hence find the effect on through put graph as well.

Trainer and its various types of experimentation using this system. This system works into server-client base. For any topology user has to select one server and select the network type whether it is LAN or WLAN. To understand the topology concept user can connect two or more clients to hardware. Depending on the topology selected user will have option to select protocols for the selected topology. Upon selection of protocol user can then create network of connected computers.

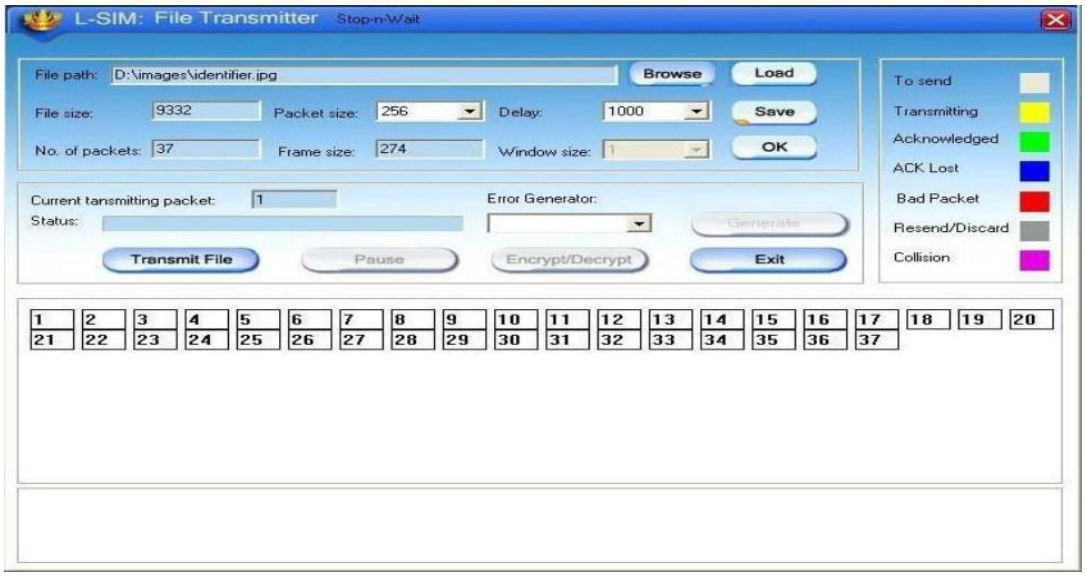
In any network which is created by user server can send or can communicate with any of the clients however clients can communicate only with server, no client to client communication is possible. Transmitter port protocol & network analysis can be done after communication is over between server and clients. Throughput v/s Packet size graph can be plotted for which at least two file transfers should be carried out. This plot can be printed to attach in the lab exercise sheet.

For the LAN network LAN cards must be installed prior to start work on this trainer. For wire less LAN USB ports should be available on the computers which are to be used for experimentation. In WLAN wireless access cards gets connected to computer USB ports and access point gets connected to hardware device.

L-SIM LAN Protocol Simulator & Analyzer Software:

It is designed to teach the basic concepts, topologies & various protocols involved in networking. The software is provided with analysis of protocols, different layers, network and measurement of error rate and throughput. Facility is provided to introduce errors into packets being sent and analyze the effect of error on different protocols and hence find the effect on throughput graph as well. Software is supported with neat operating instruction manual and online help.

MODEL WINDOW DIAGRAM FOR L-SIM



N-SIM Network simulation software:

It is developed to provide basic understanding and implementation of various advanced concepts in networking. The software provides an opportunity to understand network fundamentals through animations & simulations. The simulation provides for network experimentation with various LAN and WAN protocols, network devices, routers, encryption, decryption, file transfer, error insertion and analysis of error rate and throughput etc. This software covers Ethernet LAN, wireless LAN and router. All networking theory is explained using simulation and animation.

MODEL WINDOW DIAGRAM FOR N-SIM

Rapid advances in computer & communication technologies have resulted in the increasing merger of these two fields. The lines have blurred among computing, switching & digital transmission equipment; and the same digital techniques are being used for data, audio & video transmission. Merging & evolving technologies, coupled with increasing demands for efficient & timely collection, processing & dissemination of information, have led to the development of integrated systems that transmit & process all types of data. These intgrated systems are broadly divided as follows

- **DATA COMMUNICATION** dealing with transmission, transmission media, signal decoding, interfacing, data link control & multiplexing

- NETWORKING deals with the technology & architecture of communication network

- **COMMUNICATION PROTOCOLS** which covers the architecture as well

as analysis of individual protocols at various layers depending on the hardware & software Network laboratory is designed & developed considering the curriculum offered by Anna University. Trainers offered under network laboratory are designed for students at all level to study and understand all the concepts of data communication, data transfer using serial and parallel ports, Ethernet and wireless LAN with complete protocol understanding and actual hands on with hardware & software with ease.

Network laboratory consists of DCT-03 Data communication trainer kit, LTS- 01 LAN / Wireless LAN training system, L-SIM LAN / WLAN protocol simulator and analyzer software & N-SIM Network simulation software.

The DCT-03: Data communication trainer is a unique trainer kit for the development of exercises and theoretical-experimental courses to understand the basic concept and working of modes and protocols in serial and parallel communication.

The trainer kit consists of functional blocks for serial and parallel communication system.

The trainer kit is highly innovative from a technological as well as an educational point of view. The

trainer kit is used as “basic unit” to examine all the peculiar operating standards of serial and parallel communication system. The only external equipments required are two Computers wit

serial and parallel communication ports and an Oscilloscope. Utmost care has been laid in the design and quality control of all circuits, to ensure the repeatability of the results of the experiments.

Data communication is a term referred when the sender and receiver are digital devices, which communicate with each other by means of binary information. The objective of this trainer kit is to clear the various aspects of the data communications which comprise of

- The information source or sender.
- The medium for carrying information.
- The information receiver.
- The communication protocols, which ensure proper transfer of data.

With an increasing demand in information exchange the field of data communication technique is emerging as the only solution, to satisfy the various needs of today’s communication sector and to achieve very high bandwidth along with highest accuracy. The communication media is shifting from analog signal transfer towards digital communication.

With PC becoming the biggest storage devices in digital form, it becomes the main source and destination for information exchange. With rapid growth in both the communication technologies as well as computer hardware and software technologies, these two fields are merged to form a data communication network. Now the digital data is used for data, voice and image transmission.

Depending upon the application the communication link can be of point to point communication between two devices or a multipoint communication between at least 3 devices and data transfer can be serial or in parallel form.

Viva Questions and Answers:

1.Define NS.

A Network Simulator is an object oriented, event driven discrete simulator written in C++, with an object oriented tool command language (OTCL) interpreter as a front end.

2.What protocols does NS support?

A lot! Almost all variants of TCP, several forms of multicast, wired networking, several ad hoc routing protocols and propagation models (but not cellular phones), data diffusion, satellite, and other stuff. See the documentation (described above) for details, or download ns and look.

3.Define data communication.

Data communication is a term referred when the sender and receiver are digital devices, which communicate with each other by means of binary information.

4.List of Network Simulators:

Ns2 (Network Simulator 2). Ns3 (Network Simulator 3). OPNET.

OMNeT+ +.
NetSim.R EAL.
QualNet.
J-Sim.

5.Define OPNET.

OPNET Network Simulator. OPNET Network simulator is a tool to simulate the behavior and performance of any type of network. The main difference Opnet Network Simulator comparing to other simulators lies in its power and versatility. IT Guru provides pre-built models of protocols and devices.

Practical 7: Study of TCP/UDP Performance Using Simulation Tool

AIM:

To simulate and study the performance of TCP and UDP protocols in a network simulation tool like NS2/NS3.

Theory:

TCP and UDP protocols handle data transmission differently. TCP is reliable but slower, whereas UDP is faster but unreliable. This practical helps in comparing the performance of these protocols in various network conditions.

TOOLS USED:

Opnet Simulator

THEORY:

The transport layer protocols provide connection- oriented sessions and reliable data delivery services. This paper seeks to reflect a comparative analysis between the two transport layer protocols, which are TCP/IP and UDP/IP, as well to observe the effect of using these two protocols in a client server network. The similarities and differences between TCP and UDP over the Internet are also presented in our work. We implement a network structure using Opnet Modeler and finally, based on the practical results obtained we present the conclusions-showing the difference between these two protocols and how they work.

The transport layer is not just another layer. It is the heart of the whole protocol hierarchy. Its task is to provide reliable, cost-effective data transport from the source machine to the destination machine, independently of the physical network or networks currently in use.

TCP and UDP are transport layer components that provide the connection point through which applications access network services. TCP and UDP use IP, which is a lower-layer best effort delivery service. IP encapsulates TCP packets and UDP datagrams and delivers this information across router-connected internet works.

The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer. To achieve this goal, the transport layer makes use of the services provided by the network layer. Without the transport layer, the whole concept of layered protocols would make little sense e.g. The Transport Layer prepares applications data for transport over the network and processes network data to be used by applications. It is responsible for the end-to-end transfer of data over the network and is the four of the OSI model. The Transport layer meets a number of functions:

- enabling the applications to communicate over the network at the same time when using a single device;

- ensure that all amount of data is receive by the correct application; - responsible for fragmentation and reassembly;

- develop mechanism for handling errors.

Comparison Between TCP And UDP

Service	TCP	UDP
Flow controls	The receiver can signal the sender to slow down.	ACKs, which are used in TCP to control packet flow, are not returned.
Connection setup	It takes time, but with TCP reliability is ensured.	No connection is required.
Guaranteed message delivery	Returns acknowledgments.	UDP does not return ACKs, the receiver can't signal that packets have been successfully delivered.
Congestion controls	Network devices can take advantage of TCP ACK to control the behavior of sender.	If ACK, are missing, the network cannot signal congestion to the sender.

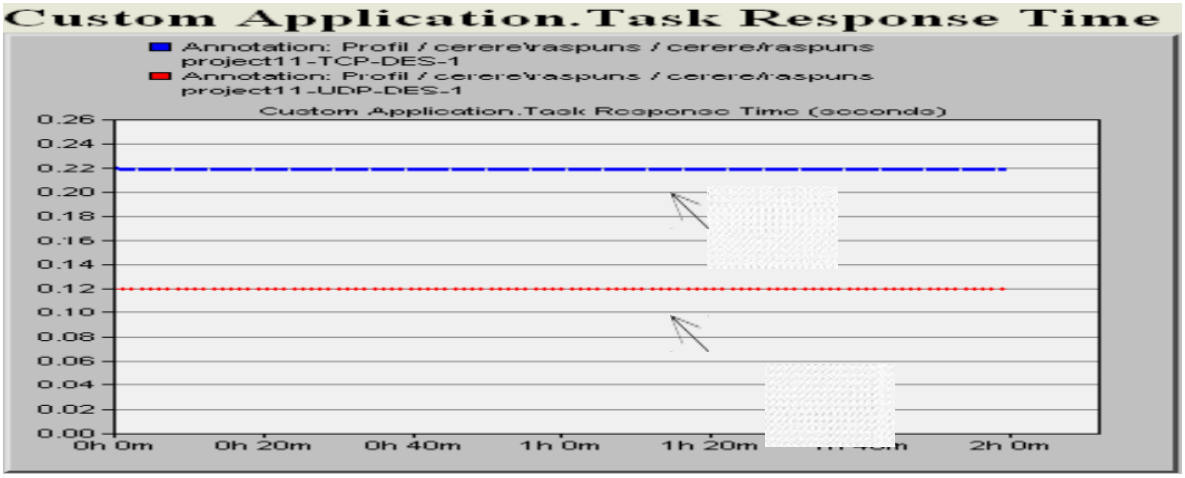
A big difference between TCP and UDP is the congestion control algorithm. For the TCP, congestion algorithm prevents the sender from overrunning the network capacity, while TCP can adapt the sender's rate with the network capacity and attempt to avoid potential congestions problems.

User Datagram Protocol (UDP), another transport protocol in IP networks, is described e.g. The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines e.g. [5]. It uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer. Is a connectionless protocol which doesn't provide flow control, reliability or error recovery and the retransmissions of data in case of errors must be ordered by other protocols. UDP is designed for applications that do not have to recompose the data segment that arrives from the sender. In another way, application-level protocols are directly responsible for the security of data transmitted.

Difference from the TCP is that there is no mechanism for error detections. If applications that use UDP doesn't have their own mechanism for information retrieval can lose those data and be forced to retransmitted again. On the other side this applications are not slow down by the confirmation process and the memory will be available for work much faster.

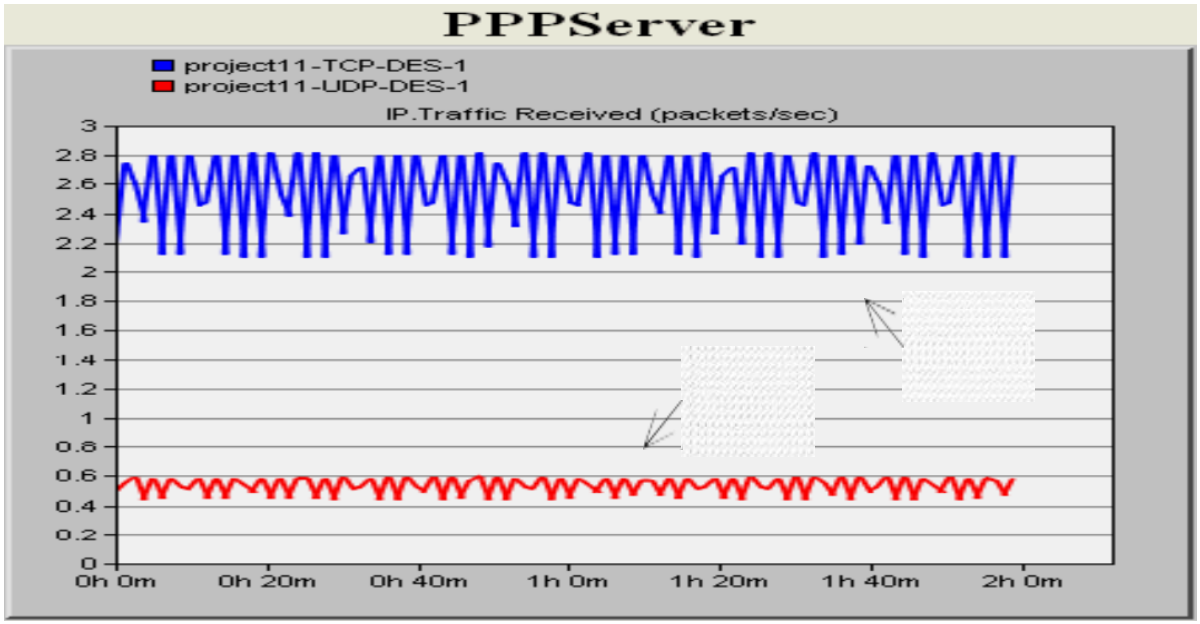
SIMULATION RESULTS:

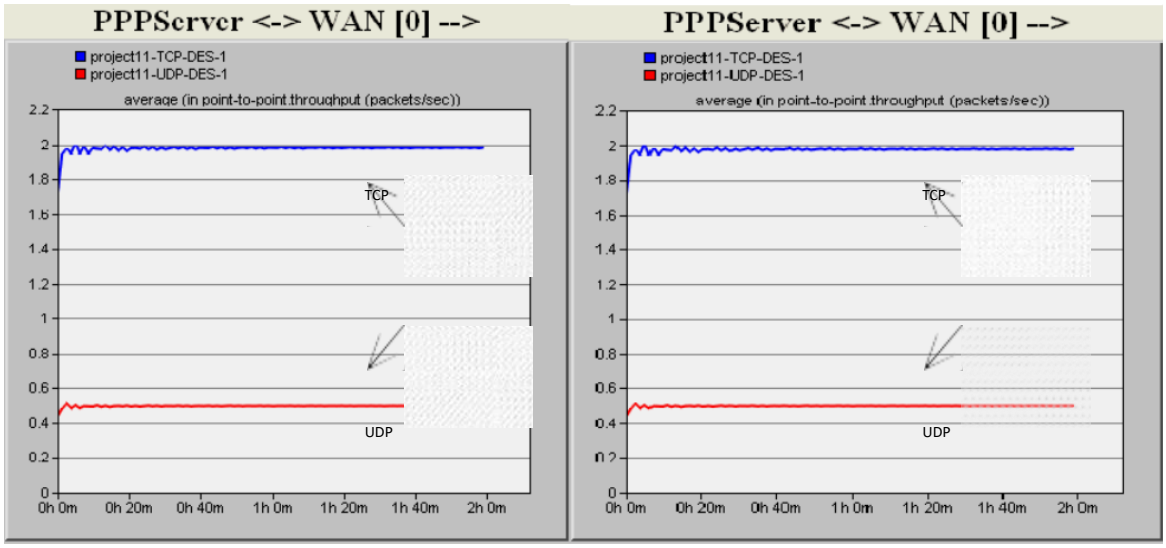
The simulation time is set for two hours data transfer between LAN network and the server with no packet latency and packet discard ratio of 0% while packets traverse thru the WAN. The task response time, in seconds, Fig. 1, shows how long the application need to be completed. The time when using TCP to complete the task is greater that the one using UDP. When using TCP, source and destination need to perform a three-way handshake before starting sending data and all amount of data need to be acknowledge by the destination when it is receive, so is taking more time than UDP, which doesn't perform this tasks.



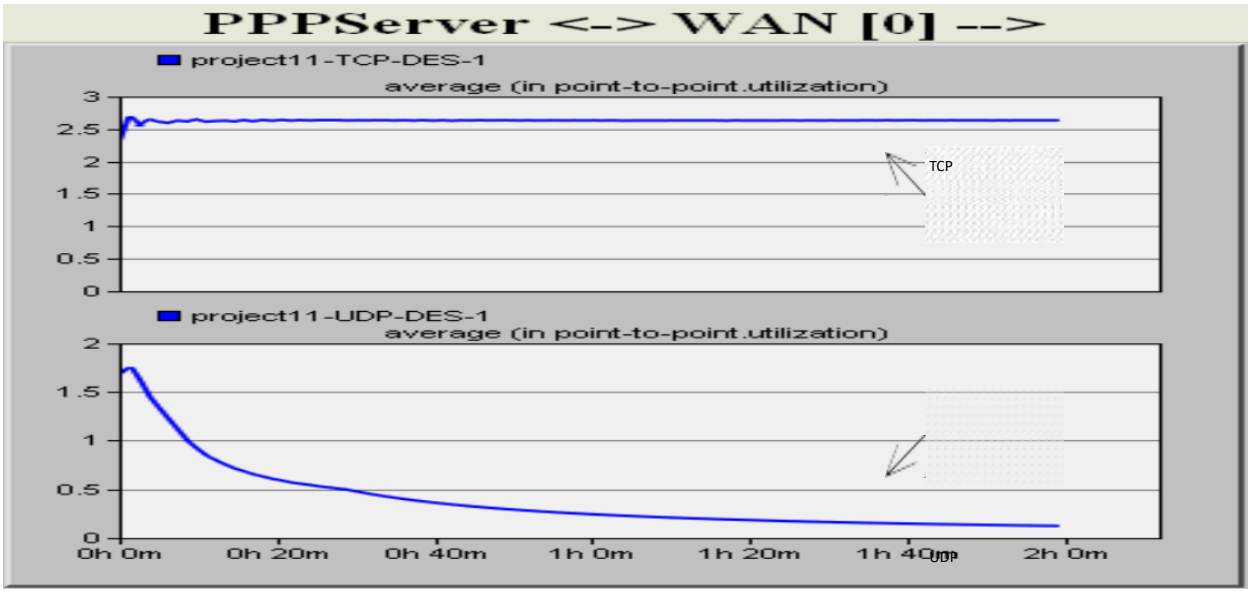
1.Response time for TCP and UDP

2.Traffic received (packets/sec) for the server





3.Traffic/Link utilization from the WAN to the server



Link utilization with a 0.5% packets discard ratio

The main difference between these two protocols is that TCP provides reliability and congestion control services, while UDP is orientated to improve performance.

The most important and common thing that TCP and UDP are using is the ability to set a host-to-host communication channel, so the packets will be delivered between processes running on two different computers. UDP is the right choice for application where reliability is not a must but the speed and performance is. Instead, TCP, even if it takes more time for the processes, has additional functions like same order delivery, reliability and flow control. As future work, we plan to conduct several studies regarding packets routing in computer networks to improve the fairness of

data transmissions using different network protocols.

RESULT:

Thus the TCP/UDP performance has bee

Practical 8a: Simulation of Distance Vector Routing Algorithm

AIM:

To simulate the Distance Vector Routing algorithm to determine the best path for packet forwarding.

APPARATUS REQUIRED:

1. VI-RTSIM software.
2. Personal computer.

THEORY:

Distance Vector Algorithm:

- A Distance vector routing, each router periodically share its knowledge about the entire network with it's neighbors.
- The three keys to under this algorithm are
    1. Knowledge about the whole network.
    2. Routing only to neighbor.
    3. Information sharing at regular intervals.

Knowledge about the whole work:

- Each router shares its knowledge about entire network. It sends all of its collected knowledge about the network to its neighbors.

Routing only to neighbor:

- Each router periodically sends its knowledge about the network only to those routers to which it has direct links. It sends whatever knowledge it has.

Information sharing at regular intervals:

- The every 30 seconds, each router sends its information about the whole network to its neighbors.

Sharing Information:

- LAN's are connected by router, represented by the assuming A, B, C, D, E and F.
- Distance vector routing simplifies the routing process by assuming a lost of one unit for every link.
- The efficiency of transmission is a function only of the number of links required to reach a destination. In this, the cost on hop count.

Routing Table:

- Each router gets its initial knowledge about the internet work and how it uses shared information to update that knowledge.
- The routing table has e columns network lost router ID.
- The first block is final destination of packet.
- The second block is no of hop count.
- The third block is that to which a packet delivers must.

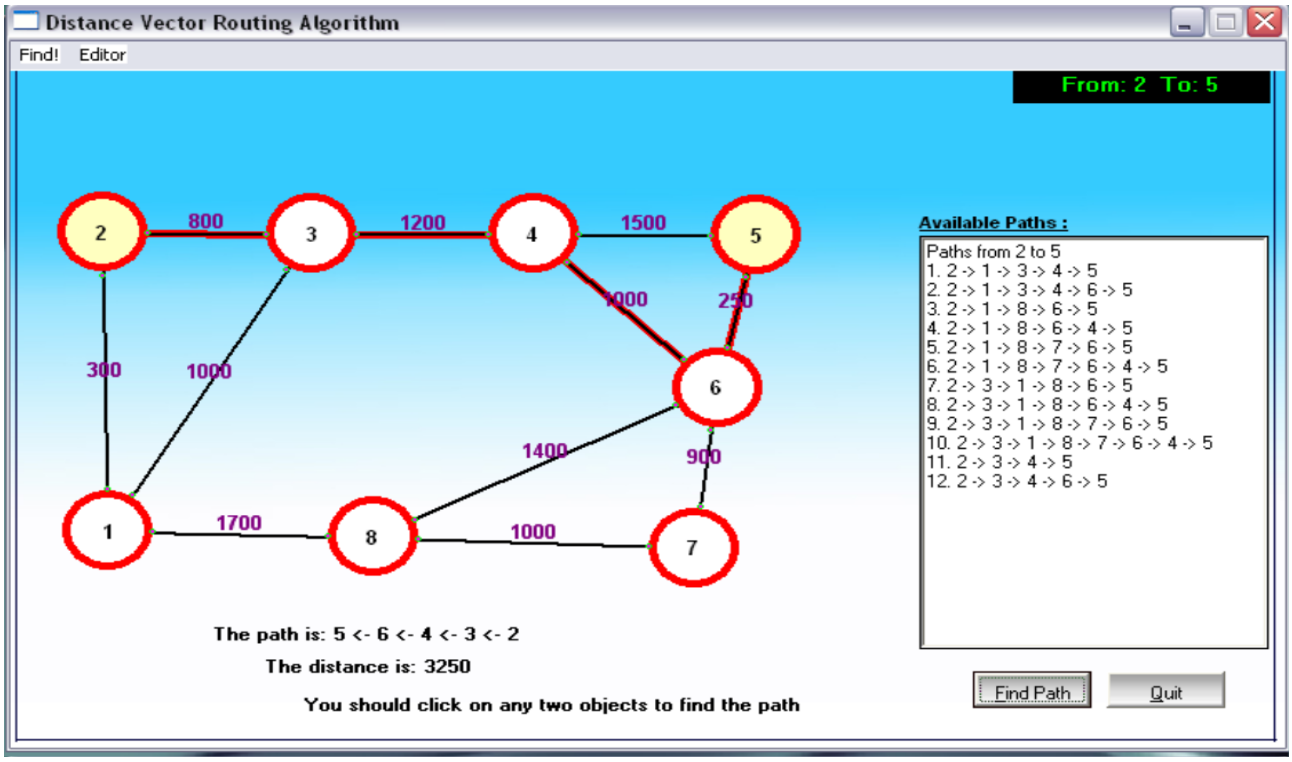
Updating algorithm:

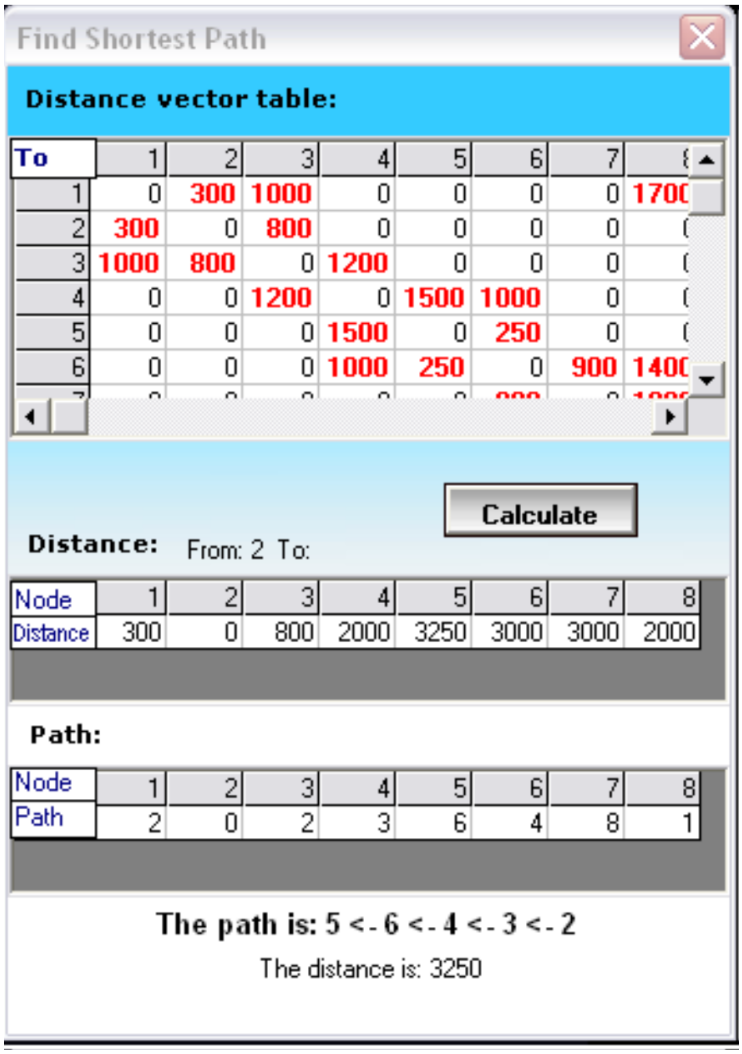
- Updating algorithm requires that the router first has one hop to the hop count field for each advertised router.
- The router should apply the below rules to each router, if the advertised destination is not in routing table
- If next hop field is same, router should replace the entry in the table with advertised one.
- If next hop field is same, router should replace the entry in the table with advertised one.
- . If next hop field is not the same, advertised hop count is smaller than the one in the table, the router should replace the entry in the table with new one.
- IF advertised hop count is not smaller, the router should do no routing.

PROCEDURE

1. Open VI-RTSIM software from desktop
2. Click the Simulation menu bar
3. Select the "Distance – Vector Routing Algorithm" option from Routing algorithm menu bar.

4. Network with routers connected through link is drawn by using option in editor(add router, join link, delete router, delete link, Add caption to link, add caption to router)
5. Select any two nodes to find the shortest distance between them.
6. Click the Find path Button to run the program.
7. Now the shortest paths between the two nodes are calculated.


Distance Vector Routing Algorithm
Find!
Editor
From: 2 To: 5
2
3
4
5
6
1
8
7
800
1200
1500
3000
250
300
1000
1400
900
1700
1000
Available Paths :
Paths from 2 to 5
1. 2 -> 1 -> 3 -> 4 -> 5
2. 2 -> 1 -> 3 -> 4 -> 6 -> 5
3. 2 -> 1 -> 8 -> 6 -> 5
4. 2 -> 1 -> 8 -> 6 -> 4 -> 5
5. 2 -> 1 -> 8 -> 7 -> 6 -> 5
6. 2 -> 1 -> 8 -> 7 -> 6 -> 4 -> 5
7. 2 -> 3 -> 1 -> 8 -> 6 -> 5
8. 2 -> 3 -> 1 -> 8 -> 6 -> 4 -> 5
9. 2 -> 3 -> 1 -> 8 -> 7 -> 6 -> 5
10. 2 -> 3 -> 1 -> 8 -> 7 -> 6 -> 4 -> 5
11. 2 -> 3 -> 4 -> 5
12. 2 -> 3 -> 4 -> 6 -> 5
The path is: 5 <- 6 <- 4 <- 3 <- 2
The distance is: 3250
You should click on any two objects to find the path
Find Path
Quit


Find Shortest Path
Distance vector table:
To 1 2 3 4 5 6 7 8
1 0 300 1000 0 0 0 0 1700
2 300 0 800 0 0 0 0
3 1000 800 0 1200 0 0 0
4 0 0 1200 0 1500 1000 0
5 0 0 0 1500 0 250 0
6 0 0 0 1000 250 0 900 1400
Calculate
Distance: From: 2 To:
Node 1 2 3 4 5 6 7 8
Distance 300 0 800 2000 3250 3000 3000 2000
Path:
Node 1 2 3 4 5 6 7 8
Path 2 0 2 3 6 4 8 1
The path is: 5 <- 6 <- 4 <- 3 <- 2
The distance is: 3250

questions and answers:

1. **what is distance vector routing protocol?**

In distance vector routing the least cost route between any two nodes is the route with minimum distance. In this each node maintains a vector table of minimum distances to every node. The table at each node also guides the packets to the desired node by showing the next stop in the route.

2. **What is advantage in distance vector routing protocol?**

In this each node shares its routing table with its immediate neighbors periodically and when there is a change.

3. **What are the routing protocols? •** Intra domain routing protocols

- Interdomain routing protocols

4. **What is intradomain routing protocols?**

Routing inside an autonomous system is referred to as intradomain routing protocols.

5. **What is RIP?**

RIP(routing information protocol) is an intradomain routing protocol used inside an autonomous system. it is very simple protocol based on distance vector routing

.

Practical 8b: Simulation of Link State Routing Algorithm

AIM:

To simulate the Link State Routing algorithm to determine the shortest paths in a network.

APPARATUS REQUIRED:

1. VI-RTSIM software.
2. Personal computer.

THEORY:

Link State Vector Algorithm:

- In Link state routing, each router share its information of its neighbors with every other router in the inter-network.

Knowledge about the neighborhood:

- Instead of sending its entire routing table, a router sends information about its neighborhood only.

To all router:

- Each router send this information to every other router on the internetworking, not just to its neighbors.
- If s does so by a process called “flooding” it means that a router sends its information.

Information sharing when there is a Change:

- Each router sends out information about the neighbors when there is a change.

Information sharing:

- Link state routing process use the same internet work as distance vector algorithm.
- Here each other sends its knowledge about is neighbors to every other router in the internet work.
- Cost is applied only by routers and not by any other station on a network, if cost was added by every station, instead of by routers alone, it would accumulate unpredictably.
- Cost is applied as a packet leaves the router rather then as if enters. Most networks are

broadcast networks. When a packet is in network every station, including the router, can pick it up, we cannot assign any cost to a packet.

Link state packet:

✞ When a router floods the network with information about its neighborhood, it is said to be advertising. The basis of this advertising is a short packet called a link state packet (LSP).

Advertiser	Network	Cost	Neighbor

Getting information about neighbors:

- ✞ A router gets its information about its neighbors by periodically sending them a short greeting packet.
- ✞ If the neighbor responds to the greeting as expected, it is assumed to be alive and functioning.

Initialization:

- ✞ Imagine that all routers in our sample internet work come up at the same time.
- ✞ Each router sends a greeting packet to its neighbors to find out the state of each link.

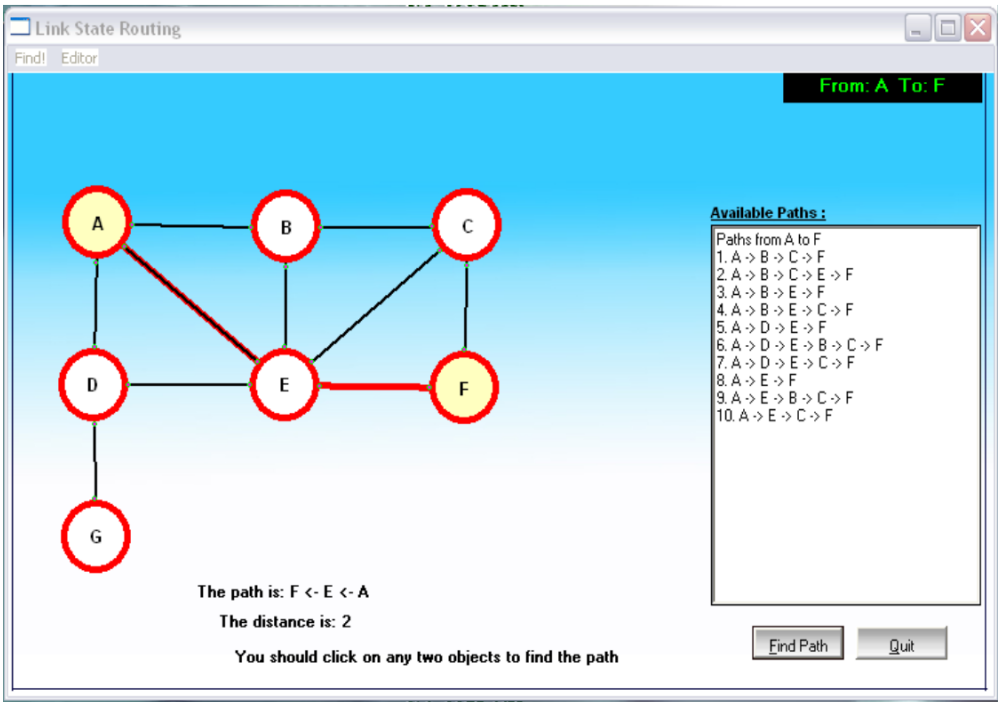
Link – State Database:

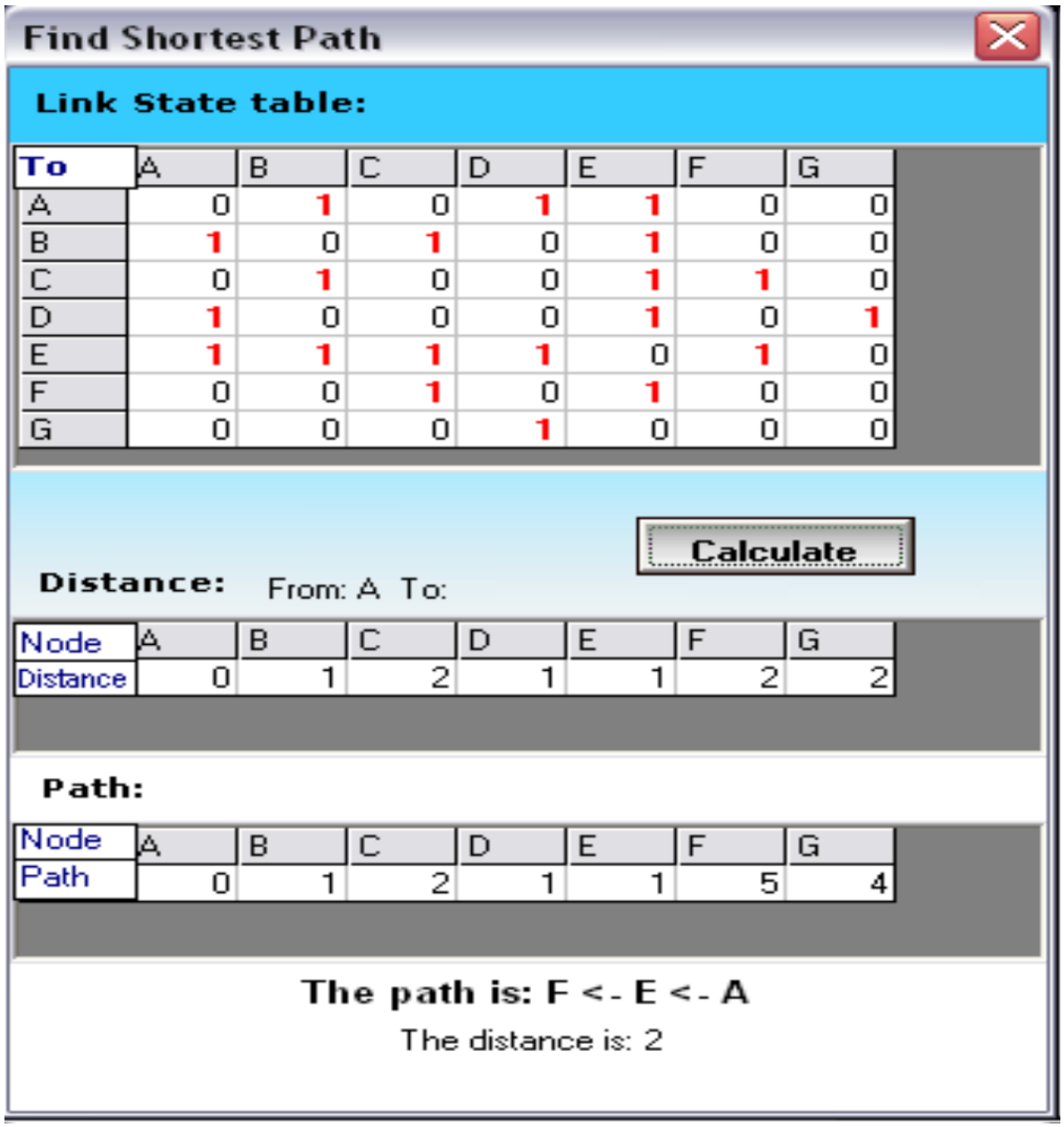
- ✞ Every router every LSP and puts the information into a link-state database.
- ✞ Because every router receives the same LSPs every router builds the same database.
- ✞ It stores this database on its disk and uses it to calculate its routing table. If a router is added to be deleted from the system, the whole database must be shared for fast updating.

PROCEDURE

1. Open VI-RTSIM software from desktop
2. Click the Simulation menu bar
3. Select the “Link State Routing Algorithm” option from Routing algorithm menu bar.

4. Network with routers connected through link is drawn by using option in editor(add router, join link, delete router, delete link, Add caption to link, add caption to router)
5. Select any two nodes to find the shortest distance between them.
6. Click the Find path Button to run the program.
7. Now the shortest paths between the two nodes using link state routing algorithm was calculated.





Questions and answers:

1. What is link state routing protocol?

It has a different concept from that of distance vector routing. In this each node in the

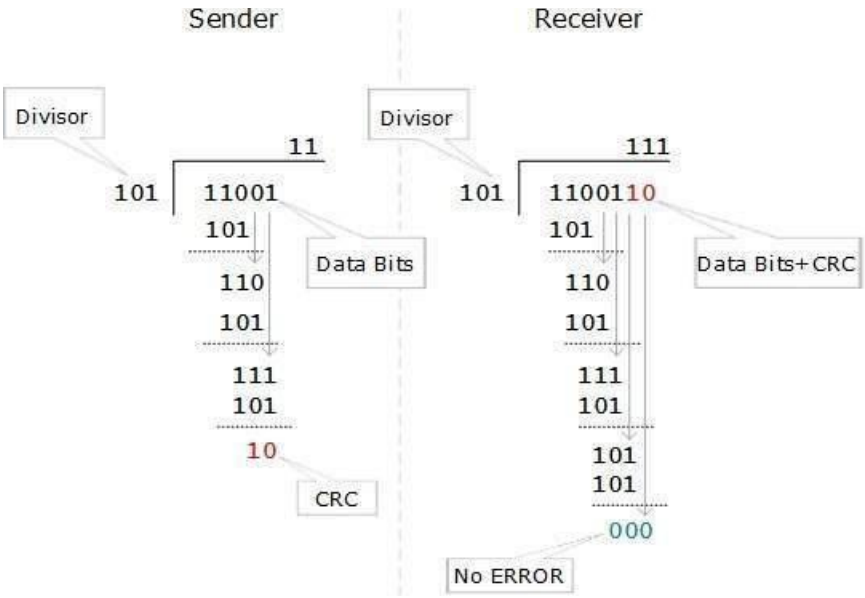
Practical 9: Simulation of Error Correction Code (like CRC)

AIM:

To simulate error correction techniques like CRC (Cyclic Redundancy Check) to detect and correct errors in data transmission.

Theory:

CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as code words



Code :

#include <stdio.h>

#include <string.h>

unsigned int crc32_table[256];

```
void init_crc32() {
   unsigned int crc, polynomial = 0xedb88320;
   for (unsigned int i = 0; i < 256; i++) {
      crc = i;
      for (unsigned int j = 8; j > 0; j--) {
         if (crc & 1)
            crc = (crc >> 1) ^ polynomial;
         else
            crc = crc >> 1;
      }
      crc32_table[i] = crc;
   }
}

unsigned int compute_crc32(const char *data) {
   unsigned int crc = 0xffffffff;
   while (*data) {
      crc = (crc >> 8) ^ crc32_table[(crc & 0xff) ^ *data++];
   }
   return crc ^ 0xffffffff;
}

int main() {
   init_crc32();
   const char *data = "Hello, CRC!";
   unsigned int crc = compute_crc32(data);
```

```
    printf("CRC32: %08x\n", crc);
    return 0;
}
```

Question & Answers:

1. **Q:** What is the purpose of the CRC algorithm?
    - **A:** CRC is used to detect errors in data transmission and ensure data integrity.
2. **Q:** How does CRC detect errors?
    - **A:** CRC compares the remainder of the division with the transmitted remainder. If they don't match, there is an error.

Result:

- CRC32 checksum is calculated for the provided data string.

Observation:

- The CRC code successfully computes a checksum that can be used for error detection.