1.Implementation of stop and wait protocol and sliding window protocol.

Stop and wait protocol:

```
 #include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void stopAndWait(int frames) {
    for (int i = 0; i < frames; i++) {
        printf("Sending frame %d...\n", i);
        sleep(1); // Simulating delay
        printf("Acknowledgment received for frame %d\n", i);}
   printf("All frames sent successfully!\n");}
int main() {
    int frames = 5;
    stopAndWait(frames);
    return 0;}
```

Sliding window protocol:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define WINDOW_SIZE 3
#define TOTAL_FRAMES 7
void slidingWindow() {
    int base = 0;
    while (base < TOTAL_FRAMES) {
        printf("\nWindow: [%d - %d]\n", base, base + WINDOW_SIZE - 1);
        for (int i = base; i < base + WINDOW_SIZE && i < TOTAL_FRAMES; i++) {
            printf("Sending frame %d...\n", i)    }
        sleep(2); // Simulating delay
        printf("Acknowledgment received for frame %d\n", base);
        base++;     }
 printf("\nAll frames sent successfully!\n");}
int main() {  slidingWindow();
    return 0;}
```

output:

for stop and wait protocol

Sending frame 0...

Acknowledgment received for frame 0

Sending frame 1...

Acknowledgment received for frame 1

Sending frame 2...

Acknowledgment received for frame 2

Sending frame 3...

Acknowledgment received for frame 3

Sending frame 4...

Acknowledgment received for frame 4

All frames sent successfully!

For sliding window protocol:

Window: [0 - 2]

Sending frame 0...

Sending frame 1...

Sending frame 2...

Acknowledgment received for frame 0

Window: [1 - 3]

Sending frame 1...

Sending frame 2...

Sending frame 3...

Acknowledgment received for frame 1

Window: [2 - 4]

Sending frame 2...

Sending frame 3...

Sending frame 4...

Acknowledgment received for frame 2

Window: [3 - 5]

Sending frame 3...

Sending frame 4...

Sending frame 5...

Acknowledgment received for frame 3

Window: [4 - 6]

Sending frame 4...

Sending frame 5...

Sending frame 6...

Acknowledgment received for frame 4

All frames sent successfully!

2. Study of socket programming and client -server model.

Server code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY}
    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, 5);
   printf("Server is listening on port 5000...\n");
    int client_socket = accept(server_socket, NULL, NULL);
    char buffer[1024] = {0};
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Received: %s\n", buffer);
    send(client_socket, "Message received!", 18, 0);
    return 0;}
```

client code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    send(client_socket, "Hello Server!", 14, 0);
    char buffer[1024] = {0};
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Server response: %s\n", buffer);
    return 0;
}
```

Output

Server console:

Server is listening on port 5000...

Received: Hello Server!

Client console:

Server response: Message received!

3. write a code simulating ARP/RARP protocols.

```
#include <stdio.h>
#include <string.h>
#define MAX_ENTRIES 5
typedef struct {
    char ip_address[16];
    char mac_address[18];
} ARPEntry;
ARPEntry arp_table[MAX_ENTRIES] = {
    {"192.168.1.1", "00:A0:C9:14:C8:29"},
    {"192.168.1.2", "00:A0:C9:14:C8:30"},
    {"192.168.1.3", "00:A0:C9:14:C8:31"},
    {"192.168.1.4", "00:A0:C9:14:C8:32"},
    {"192.168.1.5", "00:A0:C9:14:C8:33"}
};
void resolve_ip_to_mac(char *ip) {
    for (int i = 0; i < MAX_ENTRIES; i++) {
        if (strcmp(arp_table[i].ip_address, ip) == 0) {
            printf("MAC Address: %s\n", arp_table[i].mac_address);
            return;   } }
    printf("MAC Address not found!\n");}
void resolve_mac_to_ip(char *mac) {
    for (int i = 0; i < MAX_ENTRIES; i++) {
        if (strcmp(arp_table[i].mac_address, mac) == 0) {
            printf("IP Address: %s\n", arp_table[i].ip_address);
            return  } }
    printf("IP Address not found!\n");}
int main() {
    int choice;
    char input[18];
    while (1) {
        printf("\nSelect option:\n");
        printf("1. ARP (Resolve IP to MAC)\n");
```

```
        printf("2. RARP (Resolve MAC to IP)\n");
        printf("3. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        getchar(); // Clear newline from buffer
        if (choice == 1) {
            printf("Enter IP Address: ");
            fgets(input, sizeof(input), stdin);
            input[strcspn(input, "\n")] = 0; // Remove newline character
            resolve_ip_to_mac(input);
        } else if (choice == 2) {
            printf("Enter MAC Address: ");
            fgets(input, sizeof(input), stdin);
            input[strcspn(input, "\n")] = 0; // Remove newline character
            resolve_mac_to_ip(input);
        } else {
            printf("Exiting program...\n");
            break;
        }
    }
    return 0;
}
```

Output:

Select option:

1. ARP (Resolve IP to MAC)

2. RARP (Resolve MAC to IP)

3. Exit

Enter choice: 1

Enter IP Address: 192.168.1.2

MAC Address: 00:A0:C9:14:C8:30

Select option:

1. ARP (Resolve IP to MAC)

2. RARP (Resolve MAC to IP)

3. Exit

Enter choice: 2

Enter MAC Address: 00:A0:C9:14:C8:32

IP Address: 192.168.1.4

4. write a code simulating PING and TRACEROTE commands.

a.simulating PING command:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char target[50];
    printf("Enter hostname or IP to ping: ");
    scanf("%s", target);
    printf("Pinging %s...\n", target);
    char command[100];
    sprintf(command, "ping -c 4 %s", target);
    system(command); // Execute ping command
    return 0;}
```

b. simulating TRACEROTE command:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char target[50];
    printf("Enter hostname or IP for traceroute: ");
    scanf("%s", target);
    printf("Tracing route to %s...\n", target);
    char command[100];
    sprintf(command, "traceroute %s", target);
    system(command); // Execute traceroute command
    return 0;
}
```

Output:

PING simulation:

Enter hostname or IP to ping: google.com

Pinging google.com...

PING google.com (142.250.182.142): 56 data bytes

64 bytes from 142.250.182.142: icmp_seq=1 ttl=118 time=12.3 ms

64 bytes from 142.250.182.142: icmp_seq=2 ttl=118 time=13.1 ms

64 bytes from 142.250.182.142: icmp_seq=3 ttl=118 time=11.9 ms

64 bytes from 142.250.182.142: icmp_seq=4 ttl=118 time=12.7 ms

TRACEROUTE simulatuion:

Enter hostname or IP for traceroute: google.com

Tracing route to google.com...

1 <1 ms <1 ms <1 ms 192.168.1.1

2 10 ms 12 ms 11 ms 203.0.113.1

3 35 ms 38 ms 37 ms google.com

5. create a socket for HTTP for web page upload and download.

Server code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
#define BUFFER_SIZE 4096
void handle_request(int client_socket) {
    char buffer[BUFFER_SIZE];
    read(client_socket, buffer, sizeof(buffer));
    if (strncmp(buffer, "GET", 3) == 0) {
        FILE *file = fopen("webpage.html", "r");
        if (file == NULL) {
           write(client_socket, "HTTP/1.1 404 Not Found\n\n", 24);
        } else {
            write(client_socket, "HTTP/1.1 200 OK\nContent-Type: text/html\n\n", 41);
            char line[BUFFER_SIZE];
            while (fgets(line, sizeof(line), file)) {
                write(client_socket, line, strlen(line)); }
            fclose(file);}
    } else if (strncmp(buffer, "POST", 4) == 0) {
        FILE *file = fopen("uploaded_page.html", "w");
        if (file == NULL) {
            write(client_socket, "HTTP/1.1 500 Internal Server Error\n\n", 37);
        } else {
            fwrite(buffer, strlen(buffer), 1, file);
            fclose(file);
            write(client_socket, "HTTP/1.1 200 OK\nFile Uploaded Successfully\n\n", 44); } }
    close(client_socket);}
int main() {
    int server_socket, client_socket;
```

```
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
    bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr));
    listen(server_socket, 5);
    printf("HTTP Server is running on port %d...\n", PORT);
    while (1) {
        addr_size = sizeof(client_addr);
        client_socket = accept(server_socket, (struct sockaddr*)&client_addr, &addr_size);
        handle_request(client_socket)  }
    close(server_socket);
    return 0;}
```

client code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
#define BUFFER_SIZE 4096
void send_request(const char *request) {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1")
    connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr));
    write(client_socket, request, strlen(request));
 read(client_socket, buffer, sizeof(buffer));
```

```
    printf("Server response:\n%s\n", buffer);
  close(client_socket);}
int main() {
    printf("1. Download Webpage (GET)\n");
    printf("2. Upload Webpage (POST)\n");
    printf("Enter choice: ");
    int choice;
    scanf("%d", &choice);
    if (choice == 1) {
        send_request("GET /webpage.html HTTP/1.1\n");
    } else if (choice == 2) {
        send_request("POST /upload HTTP/1.1\n<html><body><h1>New Page</h1></body></html>\n");
    }
    return 0;
}
```

Output:

Server console:

HTTP Server is running on port 8080...

Client connected.

Serving webpage.html...

File uploaded successfully!

Client console(GET request):

Server response:

HTTP/1.1 200 OK

Content-Type: text/html

<html><body><h1>Welcome!</h1></body></html>

Client console (POST request):

Server response:

HTTP/1.1 200 OK

File Uploaded Successfully

6. write a program to implement RPC(remote procedure call).

Define RPC interface:

```
 program RPCPROG {
    version RPCVERS {
        string rpcFunction(string) = 1;
    } = 1;
} = 0x20000001;
```

Implement the RPC server:

```
#include <stdio.h>
#include "rpc_interface.h"

char **rpcFunction_1_svc(char **name, struct svc_req *req) {
    static char response[50];
    sprintf(response, "Hello, %s! RPC call successful!", *name);
    return &response;
}
```

Implement the RPC client:

```
#include <stdio.h>
#include "rpc_interface.h"
int main(int argc, char *argv[]) {
    CLIENT *client;
    char *server = "localhost";
    char *name = argv[1];
    client = clnt_create(server, RPCPROG, RPCVERS, "tcp");
    if (client == NULL) {
        printf("Error creating RPC client.\n");
        return 1;
    }
    char **response = rpcFunction_1(&name, client);
    printf("Server response: %s\n", *response);
    return 0;
}
```

Output:

Server console:

RPC Server is running...

Client console:

Server response: Hello, Alice! RPC call successful!

7. Implementation of subnetting.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void calculateSubnet(char *ip, char *subnet) {

    int ipParts[4], subnetParts[4], networkAddress[4], broadcastAddress[4];

    sscanf(ip, "%d.%d.%d.%d", &ipParts[0], &ipParts[1], &ipParts[2], &ipParts[3]);

    sscanf(subnet, "%d.%d.%d.%d", &subnetParts[0], &subnetParts[1], &subnetParts[2], &subnetParts[3]);

   for (int i = 0; i < 4; i++) {

        networkAddress[i] = ipParts[i] & subnetParts[i];

        broadcastAddress[i] = networkAddress[i] | (~subnetParts[i] & 255);

    }

    printf("\nNetwork Address: %d.%d.%d.%d\n", networkAddress[0], networkAddress[1],
networkAddress[2], networkAddress[3]);

    printf("Broadcast Address: %d.%d.%d.%d\n", broadcastAddress[0], broadcastAddress[1],
broadcastAddress[2], broadcastAddress[3]);

    printf("First Usable IP: %d.%d.%d.%d\n", networkAddress[0], networkAddress[1], networkAddress[2],
networkAddress[3] + 1);

    printf("Last Usable IP: %d.%d.%d.%d\n", broadcastAddress[0], broadcastAddress[1],
broadcastAddress[2], broadcastAddress[3] - 1);

}

int main() {

    char ip[16], subnet[16];

    printf("Enter IP Address (e.g., 192.168.1.10): ");

    scanf("%s", ip);

    printf("Enter Subnet Mask (e.g., 255.255.255.0): ");

    scanf("%s", subnet);


    calculateSubnet(ip, subnet);


    return 0;

}
```

Output:

Enter IP Address (e.g., 192.168.1.10): 192.168.1.10

Enter Subnet Mask (e.g., 255.255.255.0): 255.255.255.0

Network Address: 192.168.1.0

Broadcast Address: 192.168.1.255

First Usable IP: 192.168.1.1

Last Usable IP: 192.168.1.254

8. Application using tcp sockets like

a. echo client and echo server b. chat c. file transfer

A.Echo client and echo server:

Echo server code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, 5);
    printf("Echo Server is running on port 5000...\n");
    int client_socket = accept(server_socket, NULL, NULL);
    char buffer[1024] = {0};
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Received: %s\n", buffer);
    send(client_socket, buffer, strlen(buffer), 0); // Echo back
    return 0;}
```

echo client code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    send(client_socket, "Hello Server!", 14, 0);
    char buffer[1024] = {0};
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Server response: %s\n", buffer);
```

```
    return 0;}
```

B. chat server:

Chat server code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>
#define MAX_CLIENTS 5
int clients[MAX_CLIENTS];
void *handle_client(void *client_socket) {
    int socket = *(int *)client_socket;
    char buffer[1024] = {0};
    while (recv(socket, buffer, sizeof(buffer), 0) > 0) {
        printf("Client says: %s\n", buffer);
        send(socket, buffer, strlen(buffer), 0); // Echo chat message }
    return NULL;
int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, MAX_CLIENTS);
    printf("Chat Server is running on port 5000...\n");
    for (int i = 0; i < MAX_CLIENTS; i++) {
        clients[i] = accept(server_socket, NULL, NULL);
        pthread_t thread;
        pthread_create(&thread, NULL, handle_client, &clients[i]);  }
    return 0;}
```

chat client code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    char message[1024];
    while (1) {
        printf("You: ");
        fgets(message, sizeof(message), stdin);
        send(client_socket, message, strlen(message), 0);
        recv(client_socket, message, sizeof(message), 0);
        printf("Server: %s\n", message)  }
    return 0;}
```

C.file trabsfer server:

File transfer server code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
void send_file(int client_socket) {
    FILE *file = fopen("sample.txt", "r");
    if (file == NULL) {
        send(client_socket, "File not found!", 15, 0);
        return;  }
    char buffer[1024];
    while (fgets(buffer, sizeof(buffer), file)) {
        send(client_socket, buffer, strlen(buffer), 0); }
    fclose(file);
int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, 5);
```

```
    printf("File Transfer Server running on port 5000...\n")
 int client_socket = accept(server_socket, NULL, NULL);
    send_file(client_socket);
    return 0;}
```

file transfer client code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    char buffer[1024];
    FILE *file = fopen("received.txt", "w");
    while (recv(client_socket, buffer, sizeof(buffer), 0) > 0) {
        fputs(buffer, file);
   }
    fclose(file);

    printf("File received successfully!\n");

    return 0;
}
```

Output:

Echo server:

Echo Server is running on port 5000...

Received: Hello Server!

Echo Client:

Server response: Hello Server!

Chat application:

Client says: Hey, how are you?

Client says: I'm doing well!

File transfer:

Server:

File Transfer Server running on port 5000...

Sending file: sample.txt

Client:

File received successfully!

9. application using TCP and UDP socket like d. DNS e. SNMP f. file transfer.

D. DNS resolver using UDP:

DNS client code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define DNS_PORT 53
#define BUFFER_SIZE 512
int main() {
    int client_socket = socket(AF_INET, SOCK_DGRAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(DNS_PORT), INADDR_ANY};
    printf("Enter domain to resolve: ");
    char domain[50];
    scanf("%s", domain);
    sendto(client_socket, domain, strlen(domain), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
    char buffer[BUFFER_SIZE];
    socklen_t addr_len = sizeof(server_addr);
    recvfrom(client_socket, buffer, sizeof(buffer), 0, (struct sockaddr *)&server_addr, &addr_len);
    printf("Resolved IP: %s\n", buffer);
    close(client_socket);
    return 0;}
```

E. SNMP Client for Network Monitoring Using UDP:

SNMP client code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h
#define SNMP_PORT 161
#define BUFFER_SIZE 512
int main() {
```

```
    int client_socket = socket(AF_INET, SOCK_DGRAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(SNMP_PORT), INADDR_ANY};
    char snmpRequest[] = {0x30, 0x29, 0x02, 0x01, 0x00}; // Sample SNMP request packet
    sendto(client_socket, snmpRequest, sizeof(snmpRequest), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    char buffer[BUFFER_SIZE];
    socklen_t addr_len = sizeof(server_addr);
    recvfrom(client_socket, buffer, sizeof(buffer), 0, (struct sockaddr *)&server_addr, &addr_len);
    printf("SNMP Response: %s\n", buffer);
    close(client_socket);
    return 0;}
```

F. file transfer using TCP:

File transfer server:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
void send_file(int client_socket) {
    FILE *file = fopen("file.txt", "r");
    if (file == NULL) {
        send(client_socket, "File not found!", 15, 0);
        return;
    } char buffer[1024];
    while (fgets(buffer, sizeof(buffer), file)) {
        send(client_socket, buffer, strlen(buffer), 0) }
    fclose(file);}
int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, 5);
    printf("File Transfer Server running on port 5000...\n");
    int client_socket = accept(server_socket, NULL, NULL);
    send_file(client_socket);
```

```
    return 0;}
```

file transfer client:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr = {AF_INET, htons(5000), INADDR_ANY};
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    char buffer[1024];
    FILE *file = fopen("received.txt", "w");
    while (recv(client_socket, buffer, sizeof(buffer), 0) > 0) {
        fputs(buffer, file) }
    fclose(file);
    printf("File received successfully!\n");
    return 0;}
```

Output:

D: Enter domain to resolve: google.com

Resolved IP: 142.250.190.46

E: SNMP Response: [SNMP device information]

F. server:

File Transfer Server running on port 5000...

Sending file: file.txt

Client:

File received successfully!

10. Study of network simulator (NS). and simulation of congestion control algorithms using NS.

Network Simulator (NS) is a widely used tool for simulating network protocols and scenarios. It's particularly useful for studying congestion control algorithms. Here's an overview of NS :

i. NS (from network simulator) refers to a series of discrete-event network simulators, including ns-1, ns-2, and ns-3.

ii. These simulators are primarily used in research and teaching.

iii. ns-3 is free software available under the GNU GPLv2 license for research, development, and use.

iv. The goal of the ns-3 project is to create an open simulation environment for networking research that aligns with modern networking needs.

v. It encourages community contribution, peer review, and validation of the software.

* **Simulation of Network Congestion Control Algorithms using NS :**

To simulate congestion control algorithms in NS, follow these steps:

1. Install NS: Begin by installing NS on your system. NS-2 is a popular version of NS used for network simulation.

2. Create Network Topology: Define the network topology you want to simulate. This includes nodes, links, and traffic sources.

3. Configure Nodes: Configure the nodes in your network, including their IP addresses, routing protocols, and congestion control algorithms.

4. Define Traffic: Specify the traffic patterns in your network. This can include TCP or UDP traffic, application-level traffic, and congestion scenarios.

5. Implement Congestion Control Algorithms: Implement the congestion control algorithms you want to simulate. Common algorithms include TCP Reno, TCP Vegas, TCP New Reno, TCP Cubic, etc.

6. Run the Simulation: Execute the simulation using NS. Monitor performance metrics such as throughput, packet loss, delay, and congestion window size.

7. Evaluate Results: Analyze the simulation results to evaluate the performance of the congestion control algorithms under different network conditions and loads.

* Example Scenario :

Let's say you want to simulate TCP Reno and TCP Cubic congestion control algorithms in a simple network topology with two nodes connected by a link. Here's a simplified example of how you might do it:

a) Define the network topology in NS, including two nodes and a link between them.

b) Configure TCP Reno on one node and TCP Cubic on the other node.

c) Define traffic sources (e.g., FTP traffic) between the nodes.

d) Run the simulation and collect performance metrics.

e) Analyze the throughput, packet loss, and congestion window evolution for

both TCP Reno and TCP Cubic under varying network conditions (e.g., different bandwidths, delays, and packet loss rates

11. Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer. i. Link State routing ii. Flooding iii. Distance vector.

Case Study: Routing Algorithms for Optimum and Economical Network Path Selection

In this case study, we will explore three different routing algorithms: Link State Routing, Flooding, and Distance Vector Routing. We will examine how each algorithm works, its advantages and disadvantages, and when it is most suitable to use.

1. Link State Routing

How it Works:

- Each router maintains a map of the entire network.

- Routers periodically exchange information about their links with other routers.

- Using this information, each router calculates the shortest path to every other router in the network using Dijkstra's algorithm.

- The shortest path is then used for routing packets.

Advantages:

- Provides an optimal path from source to destination.

- Adapts well to network changes as it recalculates routes when the network topology changes.

Disadvantages:

- Requires more memory and processing power on routers to maintain and calculate the network map.

- Sensitive to network failures, as a single link failure can trigger the recalculation of routes by all routers.

Suitability:

- Best suited for networks where reliability and optimal paths are crucial, such as in large-scale networks with high traffic.

2. Flooding

How it Works:

- When a router receives a packet, it forwards the packet to all of its neighbors, except the one from which it received the packet.

- Each router that receives the packet does the same until the packet reaches its destination or its TTL (Time To Live) expires.

Advantages:

- Simple and easy to implement.

- Does not require routers to maintain routing tables.

Disadvantages:

- Can lead to network congestion and wasted bandwidth as packets are forwarded to all neighbors.

- Does not guarantee that the packet will reach its destination.

Suitability:

- Suitable for small, simple networks or for broadcasting information to all nodes in a network.

3. Distance Vector Routing

How it Works:

- Each router maintains a table that contains the distance (cost) to all other routers inthe network.

- Routers periodically exchange information about their routing tables with their neighbors.

- Based on this information, each router updates its routing table to reflect the shortest path to each destination.

Advantages:

- Simple and easy to implement.

- Consumes less bandwidth compared to flooding as routers only exchange routing information with neighbors.

Disadvantages:

- Slow to converge, especially in large networks or networks with high traffic.

-May suffer from routing loops or count-to-infinity problems if not properly implemented.

Suitability:

- Suitable for small to medium-sized networks with moderate traffic where simplicity is more important than optimal routing.

Conclusion

- **Link State Routing** is suitable for large-scale networks where reliability and optimal paths are critical.

- Flooding is suitable for small, simple networks or for broadcasting information.

- Distance Vector Routing is suitable for small to medium-sized networks where

simplicity is more important than optimal routing.The choice of routing algorithm depends on the specific requirements of the network, such as size, traffic load, and the importance of optimal routing and simplicity.

12. To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable,crimping tool, etc.

Learning about handling and configuring networking hardware like RJ-45 connectors, CAT-6 cables, and crimping tools is essential for network setup and maintenance.

Here's a basic overview of these components and how to work with them:

1. RJ-45 Connector:

- The RJ-45 connector is used for Ethernet networking and is commonly found on the ends of Ethernet cables.
- To terminate an Ethernet cable with an RJ-45 connector, you will need a crimping tool and a cable stripper.
- Strip the outer jacket of the cable using the cable stripper, exposing the inner twisted pairs of wires.
- Arrange the wires according to the T568A or T568B wiring standard.
- Insert the wires into the RJ-45 connector and use the crimping tool to crimp the connector onto the cable.

2. CAT-6 Cable:

- CAT-6 cable is a type of Ethernet cable with enhanced performance specifications.
- It consists of four twisted pairs of copper wires, enclosed in a plastic jacket.
- CAT-6 cables are used for Gigabit Ethernet and other high-speed networking applications.

3. Crimping Tool:

- A crimping tool is used to attach connectors to the ends of cables.
- It has a built-in mechanism that crimps (squeezes) the connector onto the cable, creating a secure connection.

4. Steps for Crimping an RJ-45 Connector onto a CAT-6 Cable:

1. Use a cable stripper to remove about 1-1.5 inches of the outer jacket from the end of the CAT-6 cable.
2. Separate the twisted pairs of wires and arrange them in the desired order (T568A or T568B).
3. Trim the ends of the wires to ensure they are all the same length.
4. Insert the wires into the RJ-45 connector, making sure they are fully inserted and in the correct order.
5. Place the connector and cable into the crimping tool and squeeze the handle firmly to crimp the connector onto the cable.
6. Repeat the process for the other end of the cable if necessary.

5. Testing:

- After crimping the connectors, it's a good practice to test the cables using a cable tester to ensure they are properly wired and functional.

6. Additional Tips:

- Always use quality connectors and cables to ensure reliable connections.

- Follow the T568A or T568B wiring standard consistently for all connections in your network.

- Properly label and organize your cables to facilitate troubleshooting and maintenance.Practicing with these components and tools will help you become more proficient in handling and configuring networking hardware.

13. Configuration of router, hub, switch etc. (using real devices or simulators).

Configuring routers, hubs, switches, and other networking devices is a fundamental skill for network administrators.

Here's how you can configure these devices using real devices or simulators:

1. Router Configuration:

Using Real Devices (Example: Cisco Router):

1. Connect to the router using a console cable and terminal emulation software (e.g., PuTTY).

2. Enter privileged EXEC mode by typing enable and entering the enable password.

3. Enter global configuration mode by typing configure terminal.

4. Configure interfaces, IP addresses, routing protocols, and other settings.

5. Save the configuration using write memory or `copy running-config startup-config` to make the changes permanent.

Using Simulators (Example: Cisco Packet Tracer):

1. Open Cisco Packet Tracer and add a router to the workspace.

2. Double-click on the router to open the configuration window.

3. Configure interfaces, IP addresses, routing protocols, and other settings using the graphical interface.

4. Save the configuration using the save icon to make the changes permanent.

2. Hub Configuration:

Using Real Devices (Example: Ethernet Hub):

- Hubs are simple devices and do not require configuration. They operate at the physical layer and simply forward packets to all connected devices.

Using Simulators:

- Since hubs do not have configurations, there's no need to configure them in simulators.

3. Switch Configuration:

Using Real Devices (Example: Cisco Switch):

1. Connect to the switch using a console cable and terminal emulation software.

2. Enter privileged EXEC mode and then enter global configuration mode.

3. Configure VLANs, interfaces, IP addresses, and other settings.

4. Save the configuration using write memory or `copy running-config startup-config`.

Using Simulators (Example: Cisco Packet Tracer):

1. Open Cisco Packet Tracer and add a switch to the workspace.

2. Double-click on the switch to open the configuration window.

3. Configure VLANs, interfaces, IP addresses, and other settings using the graphical interface.

4. Save the configuration using the save icon.

4. Other Devices:

- For other devices like firewalls, load balancers, and access points, the configuration process will vary depending on the device and manufacturer.

- Generally, the process involves connecting to the device, entering configuration mode, and then configuring settings such as interfaces, IP addresses, security policies, etc.

It's important to note that configuring real devices should be done with caution, as incorrect configurations can disrupt network operations. Using simulators or virtual labs is a safe way to practice and learn about network device configurations.

14. Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.

Running and using networking services and commands like ping, traceroute, nslookup, arp, telnet, and ftp is essential for network troubleshooting and administration.

Here's a each command/service and how to use them:

1. Ping:

- Description: Ping is used to test the reachability of a host on an Internet Protocol (IP) network.

- Usage: ping [host]

- Example: ping www.example.com

2. Traceroute (or Tracert on Windows):

- Description: Traceroute is used to trace the path packets take from one networked device to another.

-Usage: traceroute [host]

- Example: traceroute www.example.com

3. Nslookup (or Dig on Linux):

- Description: Nslookup is used to query Domain Name System (DNS) servers to obtain domain name or IP address mapping.

- Usage: nslookup [domain]

- Example: nslookup www.example.com

4. Arp:

- Description: Arp displays and modifies the IP-to-physical address translation tables used by the Address Resolution Protocol (ARP).

- Usage: arp -a (to display ARP cache)

- Example: arp -a

5. Telnet:

Description: Telnet is used to establish a connection to a remote system over the network.

- Usage: telnet [host] [port]

- Example: telnet www.example.com 80 (to connect to a web server)

6. FTP (File Transfer Protocol):

- **Description:** FTP is used to transfer files between a client and a server on a network.
- **Usage:** ftp [host]
- **Example:** ftp ftp.example.com

These commands are available on most operating systems and can be used from the command line or terminal. They are valuable for diagnosing network connectivity issues, checking DNS records, examining network paths, and transferring files over a network.

15.Network packet analysis using tools like Wireshark, tcpdump, etc.

Network packet analysis tools like Wireshark and tcpdump are essential for troubleshooting network issues, analyzing network traffic, and understanding network protocols.

Here's how you can use these tools:

1. Wireshark:

Downloading and Installing:

- Visit the Wireshark website (https://www.wireshark.org/) and download the appropriate version for your operating system.

- Install Wireshark by following the on-screen instructions.

Capturing Packets:

- Open Wireshark and select the network interface you want to capture packets on.

- Click the "Start" button to begin capturing packets.

Analyzing Packets:

- Wireshark will display a list of captured packets. You can analyze individual packets to view details such as source and destination addresses, protocols, and packet contents.

Filtering Packets:

- Use Wireshark's display filters to focus on specific types of packets or traffic patterns.

For example, you can filter packets by protocol (e.g., "tcp") or IP address (e.g., "ip.addr == 192.168.1.1").

Saving Captured Packets:

- You can save the captured packets to a file for further analysis or sharing.

2. tcpdump:

Using tcpdump on Linux:

- Open a terminal window.

Use the following command to start capturing packets on a specific network interface:

sudo tcpdump -i [interface].

- To save captured packets to a file, use: `sudo tcpdump -i [interface] -w

[filename.pcap]`.

- Use Ctrl+C to stop capturing packets.

Analyzing Captured Packets:

- After capturing packets, you can analyze them using Wireshark or tcpdump itself.

- To analyze packets using tcpdump, use: tcpdump -r [filename.pcap].

Filtering Packets:

- You can use tcpdump's filtering capabilities to focus on specific packets. For example, to filter packets from a specific IP address, use: `tcpdump -i [interface] src host

[ip_address]`.

Note: Be cautious when using packet capture tools on live networks, as capturing and

analyzing packets without proper authorization may violate privacy and security

policies.

16. Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc.

Network simulation tools are valuable for designing, analyzing, and simulating network scenarios without the need for physical hardware.

Here's an overview of some popular network simulation tools:

1. Cisco Packet Tracer:

Features:

- Cisco Packet Tracer is a network simulation tool primarily used for Cisco networking equipment.

- It allows users to simulate network configurations, visualize packet flow, and practice networking concepts.

Usage:

- Design and configure virtual networks with Cisco devices (routers, switches, etc.).

- Simulate network topologies and test configurations.

- Practice troubleshooting and network management tasks.

2. NetSim:

Features:

- NetSim is a network simulation and emulation software for network engineers.

- It provides a platform to simulate complex networks and test various network protocols and technologies.

Usage:

- Design and simulate large-scale networks with diverse devices and configurations.

- Test and validate network designs and protocols.

- Analyze network performance and behavior under different conditions.

3. OMNeT++:

Features:

- OMNeT++ is a discrete event simulation environment for modeling complex systems, including networks.

- It supports various network protocols and technologies and provides a modular framework for simulation development.

Usage:

- Model and simulate communication networks with different protocols and architectures.

- Analyze network performance, scalability, and reliability.

- Develop and test new network protocols and algorithms.

4. NS2 (Network Simulator 2):

Features:

- NS2 is a widely used open-source network simulation tool.

- It provides a simulation environment for modeling and analyzing networking protocols and scenarios.

Usage:

- Simulate wired and wireless networks with different topologies and protocols.

- Evaluate network performance metrics such as throughput, delay, and packet loss.

- Develop and test new networking algorithms and protocols.

5. NS3 (Network Simulator 3):

Features:

- NS3 is the successor to NS2 and provides a more modern and flexible simulation environment.

- It supports a wide range of network technologies and protocols and offers a modular architecture for simulation development.

Usage:

- Model and simulate complex network scenarios with realistic behavior.

- Analyze the performance of network protocols and applications.

- Develop and test new networking concepts, protocols, and algorithms.

17. Socket Programming using UDP and TCP (e.g., simple DNS, date & time client/server, echo client/server, iterative & concurrent servers).

*** Socket Programming :**

Socket programming involves communication between processes over a network using sockets. Here are examples of simple client-server applications using UDP and TCP protocols:

a) UDP (User Datagram Protocol):

i. Connectionless protocol.

ii. Suitable for scenarios where low overhead and speed are crucial.

iii. No guarantee of message delivery or order.

iv. Ideal for real-time applications like video streaming, online gaming,

and DNS.

v. Example: Simple DNS server/client.

vi. UDP server listens on a specific port, receives DNS queries, and

responds with corresponding IP addresses.

vii. UDP client sends DNS queries to the server and processes the

responses.

b) TCP (Transmission Control Protocol):

i. Connection-oriented protocol.

ii. Ensures reliable data transfer and proper sequencing.

iii. Slower than UDP due to additional overhead.

iv. Ideal for applications requiring data integrity, such as file transfer and email.

v. Examples:

- Date & Time server/client:
  - o TCP server provides the current date and time to connected clients.
  - o Clients connect, request the date/time, and receive the response.
- Echo server/client:
  - o TCP server echoes back any data received from clients.
  - o Clients send data, and the server echoes it back.

INDEX