PES COLLEGE OF ENGINEERG MANDYA

ಶ್ರದ್ಧಾವಾನ್ ಲಭತೇ ಜ್ಞಾನಂ

ESTD 1962

PES COLLEGE OF ENGINEERING MANDYA

An Autonomous Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ESTD 1983

PESCE-MANDYA

COMPETENT RESPONSIBLE PROFESSIONALS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Computer Networks Lab Manual

(P22CSL506)

Preface

This laboratory manual is designed to complement your theoretical understanding of computer networks by providing hands-on experience with real-world networking scenarios. The primary goal of these lab exercises is to bridge the gap between theory and practical implementation, allowing you to develop the skills necessary to design, configure, and troubleshoot computer networks. The lab exercises are crafted to give you practical exposure to key concepts in computer networking. Through hands-on activities, you will gain a deeper understanding of networking protocols, configurations, and troubleshooting techniques. The scenarios presented in the lab exercises are inspired by real-world networking challenges. By working through these scenarios, you will be better prepared to handle the complexities of designing and managing networks in professional settings. Many of the lab activities are designed to be collaborative, encouraging teamwork and communication. Effective communication is a crucial skill in the field of networking, and these exercises provide an opportunity to enhance your ability to work in a team.

Computer Networks Lab set

Sl. No	Experiments
	PART A

1. Simulate a topology with 2 LAN's each having two devices connected to switches. Switches are connected to a common router. Observe the packet flow.
2. Construct simple LAN using n nodes and understand working of Address Resolution Protocol (ARP).
3. Perform an experiment to understand the dynamic IP address allocation process observe the routing table at the end of simulation.
4. Construct a simple LAN by configuring static routing and observe the routing table at the end of simulation.
5. Simulate a topology where 3 routers are fully connected and each router connected to an end device. Observe the flow of ICMP packets from one network to other using RIP protocol.
6. Simulate a topology where 3 routers are fully connected and each router connected to an end device. Observe the flow of ICMP packets from one network to other using OSPF protocol.

Simulate a network for browsing and understand DNS protocol.

PART B

1. Write a program to implement error detection/ error correction using hamming code.
2. Write a program to show working of the Stop and wait protocol.
3. Implementation of CSMA/CD.
4. Write a program to implement Distance Vector Routing algorithm.
5. Write program to create a least cost tree using Link State Routing algorithm.
6. To write a client-server application for chat using TCP.

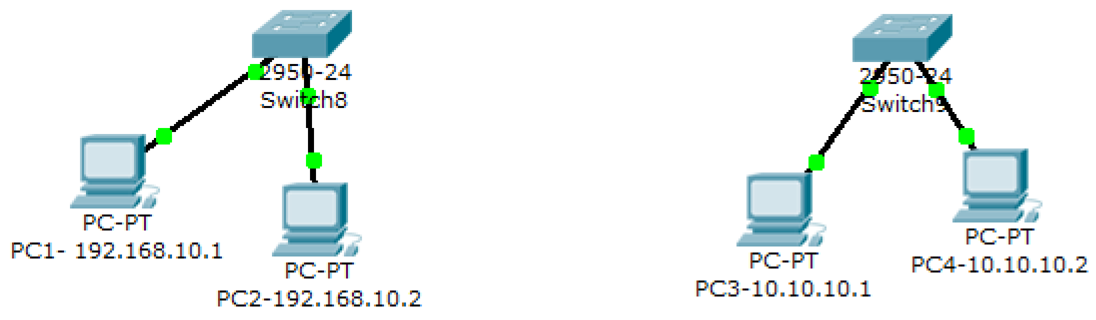
Course Outcomes: On completion of this course, students are able to:		**Bloom's Level**
Cos	**Course Outcomes with Action verbs for the Course topics**	
CO1	**Understand** the working of various networking components in the simulation environment.	**L1**
CO2	**Analyse** the working principle of the protocols in the TCP/IP protocol suite.	**L2**
CO3	**Implemen**t given networking scenarios and analyse the results.	**L3**

CO	Statement	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	**Understand** the working of various networking components in the simulation environment.	1	1	1										1		1
CO2	**Analyse** the working principle of the protocols in the TCP/IP protocol suite.	2	1	2	2	2								1		2
CO3	**Implemen**t given networking scenarios and analyse the results.	1	2	2	2	2								1		2

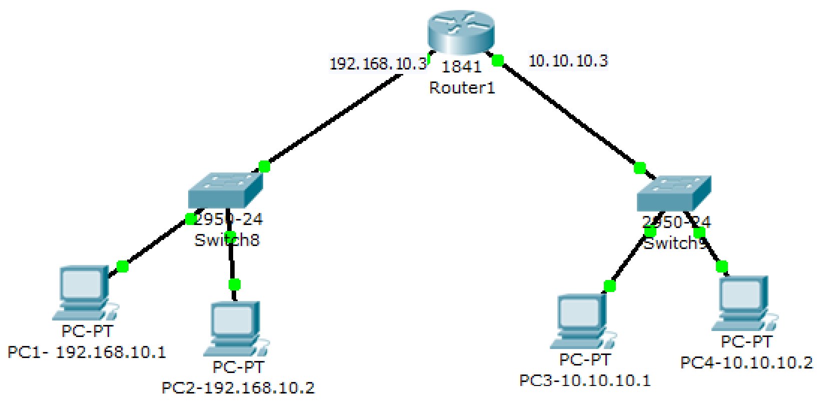
Part A

1. Simulate a topology with 2 LAN's each having two devices connected to switches. Switches are connected to a common router. Observe the packet flow.

   Step 1: Configure LAN 1 and LAN2 each connected to a switch.

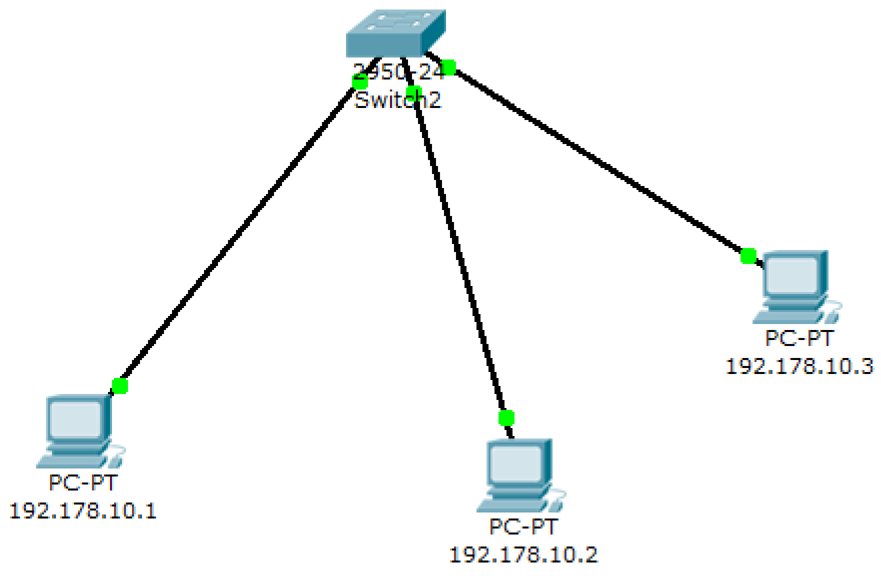


   Step 2: Check for successful packet movement with in the LANs.
   Step 3: Connect each LAN to a common router and configure with default gateway of each LAN.
   Step 4: Check for successful packet movement between the LANs and observe routing table.



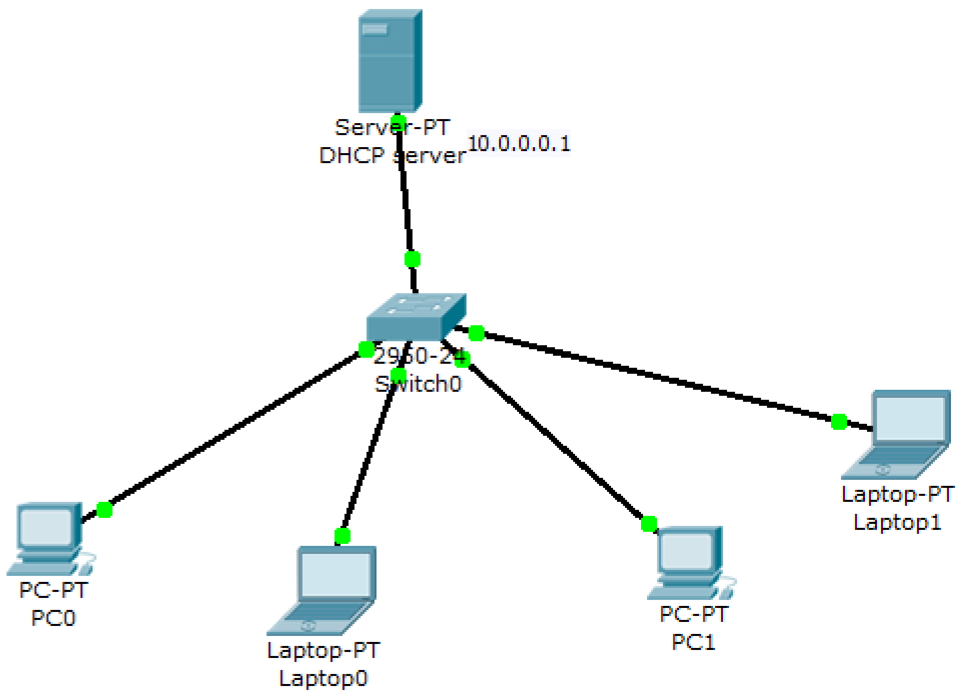
2. Construct simple LAN using 3 nodes and understand working of Address Resolution Protocol (ARP).

Step 1: Configure a LAN with 3 nodes and switch connected.
Step 2: Observer ARP table of the nodes and switch, it will be empty at this point.
Step 3: Run the simulation and send a packet from one node to another. Observer ARP table of sender, receiver and switch.

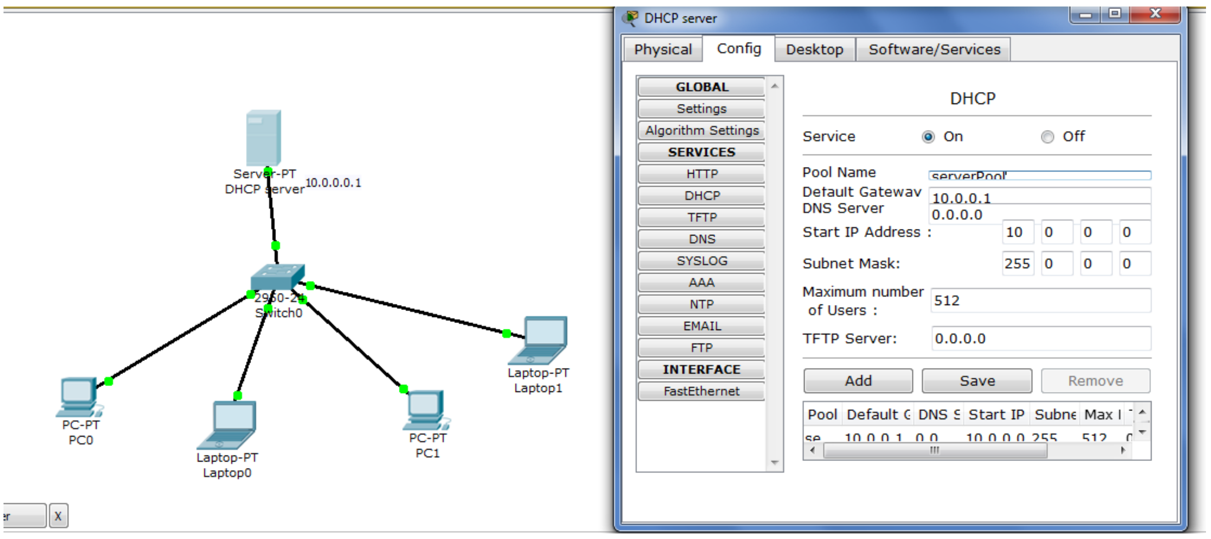


3. Perform an experiment to understand the dynamic IP address allocation process observe the routing table at beginning and the end of simulation.

Step 1: Configure a LAN with a DHCP server as shown below.



Step 2: Configure DHCP server. Go to config→ services→ select DHCP→default gateway(10.0.0.1)→start IP address→Subnet mask→ Save.



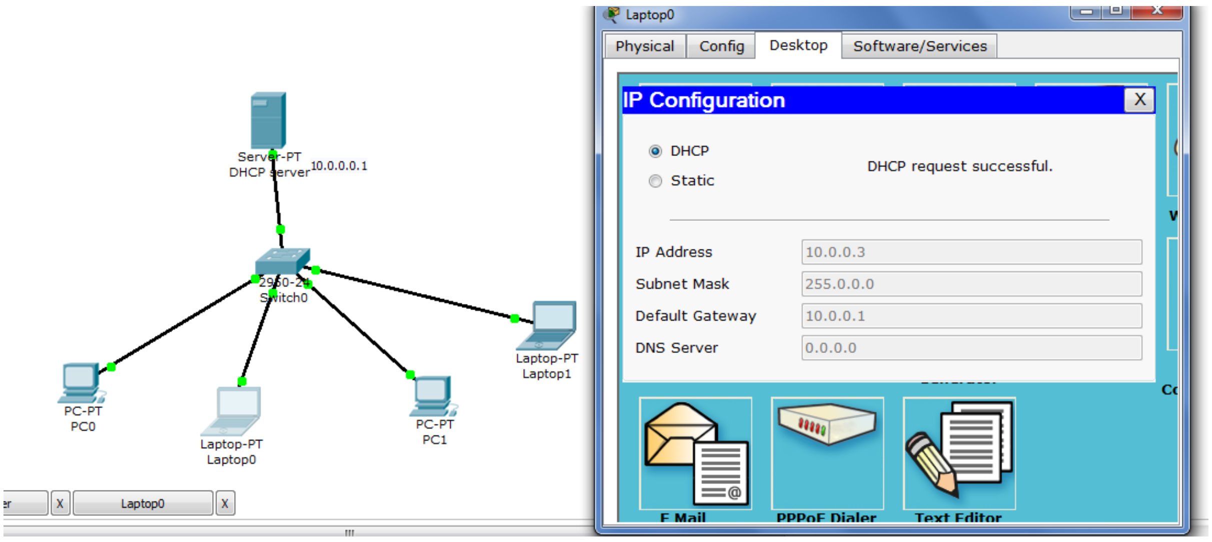
DHCP

Service [✓ On] Off

Pool Name	serverPool
Default Gateway	10.0.0.1
DNS Server	0.0.0.0
Start IP Address :	10 0 0 0
Subnet Mask:	255 0 0 0
Maximum number of Users :	512
TFTP Server:	0.0.0.0

Pool	Default G	DNS S	Start IP	Subne	Max I
se	10.0.0.1	0.0	10.0.0.0	255	512

Step 3: Observer the IP address assigned to each end node by selecting DHCP option.



IP Configuration

[✓ DHCP] Static

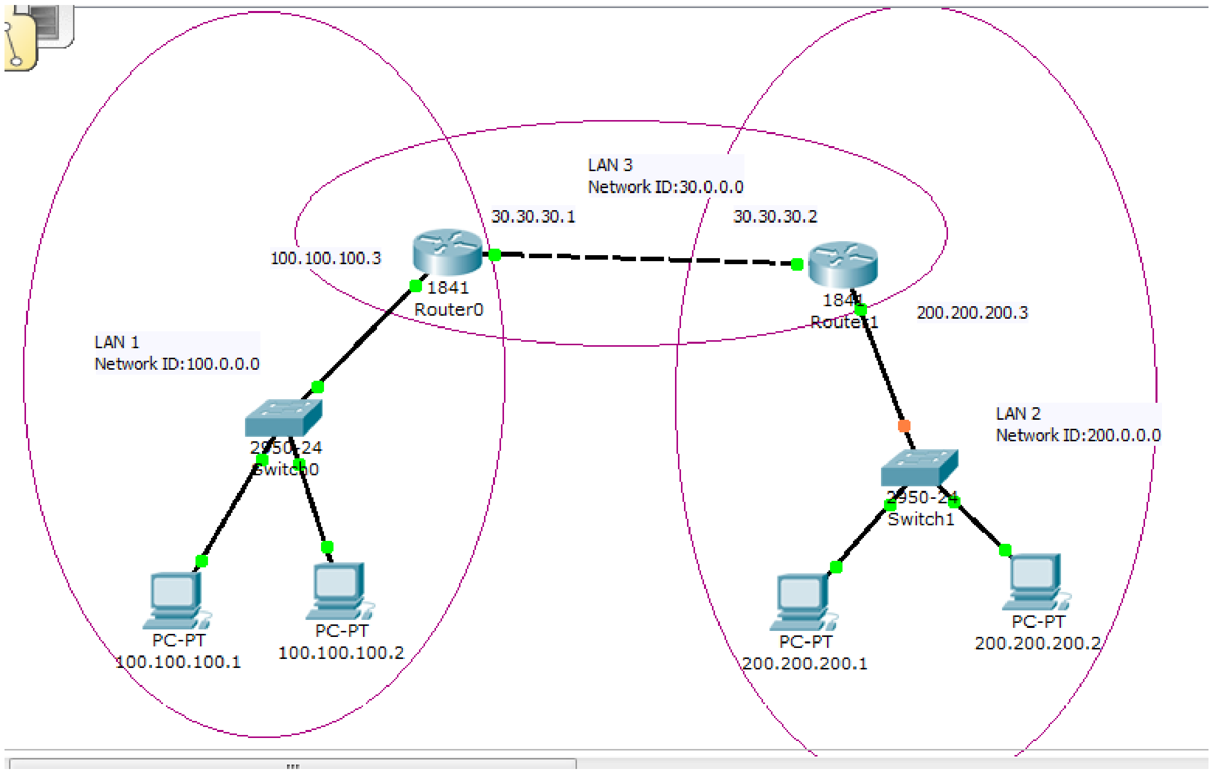
DHCP request successful.

IP Address	10.0.0.3
Subnet Mask	255.0.0.0
Default Gateway	10.0.0.1
DNS Server	0.0.0.0

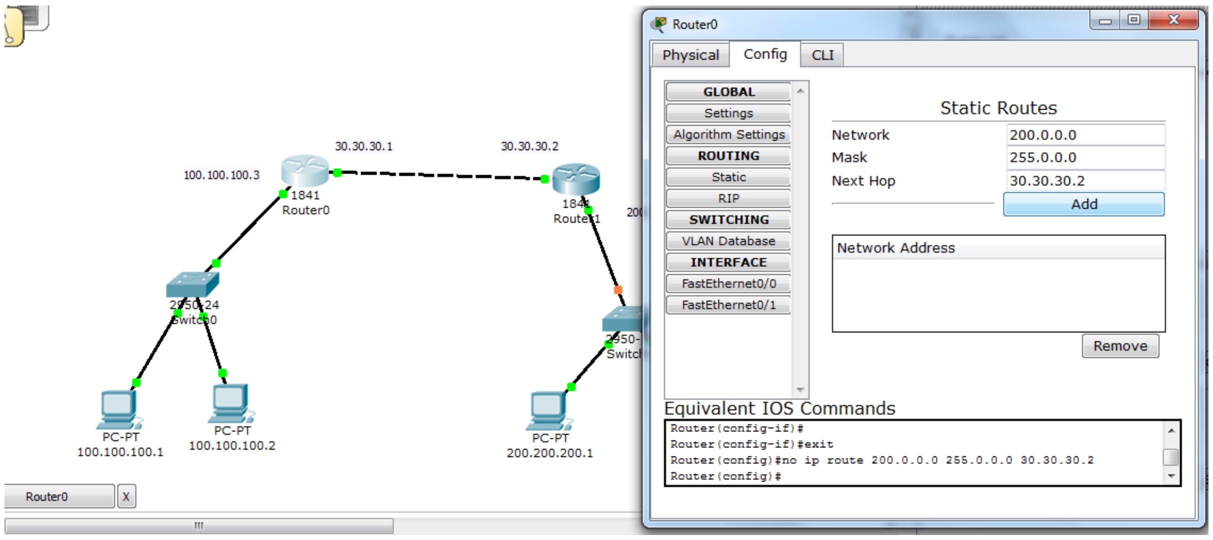
Step 4: Observer the packet movement between the nodes.

4. Construct a simple LAN by configuring static routing and observe the routing table at the beginning and at the end of simulation.

Step 1: Configure the LANs as shown below.



Step 2: Set static IP address for each router



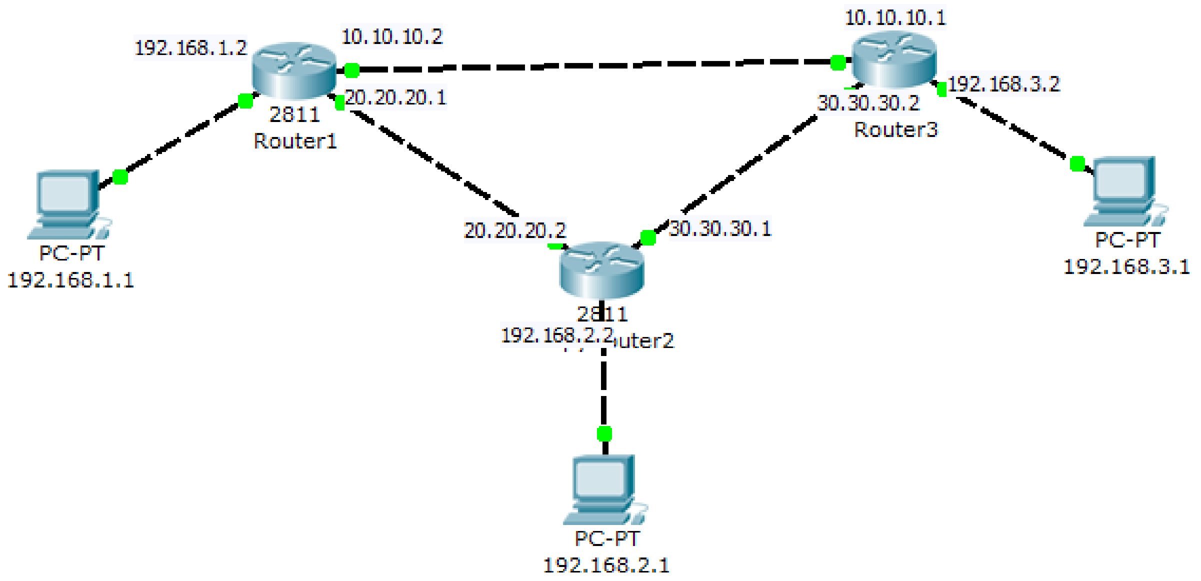
Network	200.0.0.0
Mask	255.0.0.0
Next Hop	30.30.30.2

```
Router(config-if)#
Router(config-if)#exit
Router(config)#no ip route 200.0.0.0 255.0.0.0 30.30.30.2
Router(config)#
```

Step 3: Observe the packet movement from a node in a LAN1 to another node in LAN2.

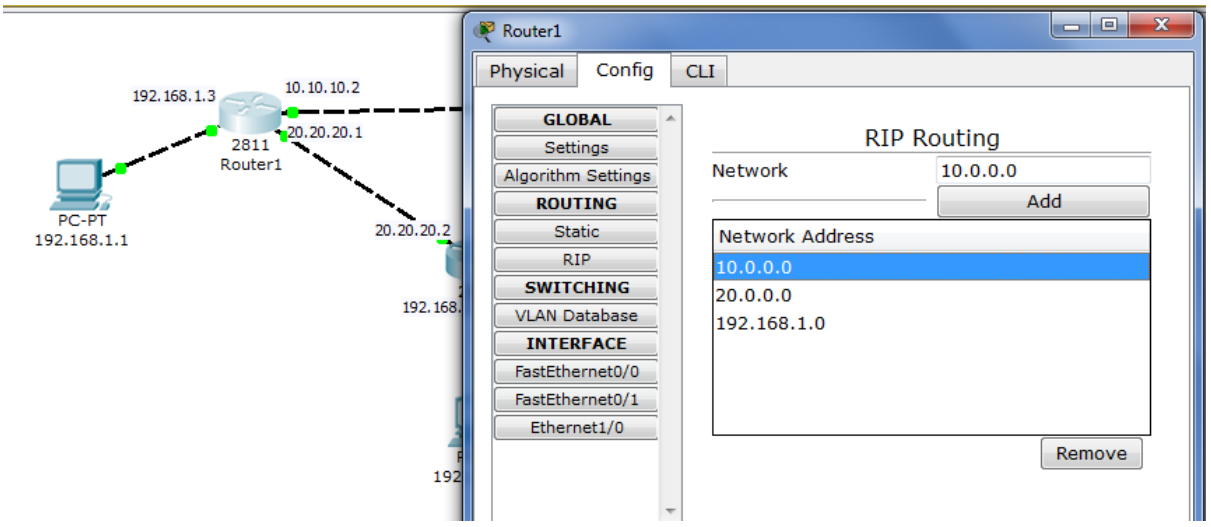
5. Simulate a topology where 3 routers are fully connected and each router connected to end end devices. Observe the flow of ICMP packets from one network to other using RIP protocol.

Step 1: Configure the nodes and routers as show below.



Step 2: Configure each router with RIP protocol.

Go to config→Rotuing→ RIP→ Add the network address of each network connected to a router.



RIP Routing

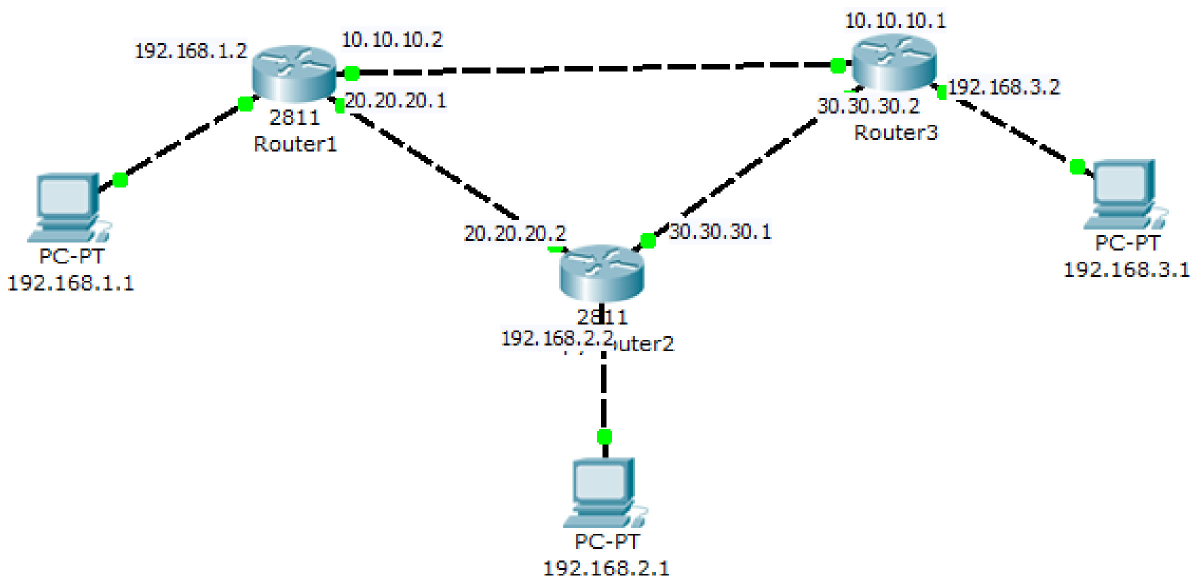
Network 10.0.0.0

Network Address
10.0.0.0
20.0.0.0
192.168.1.0

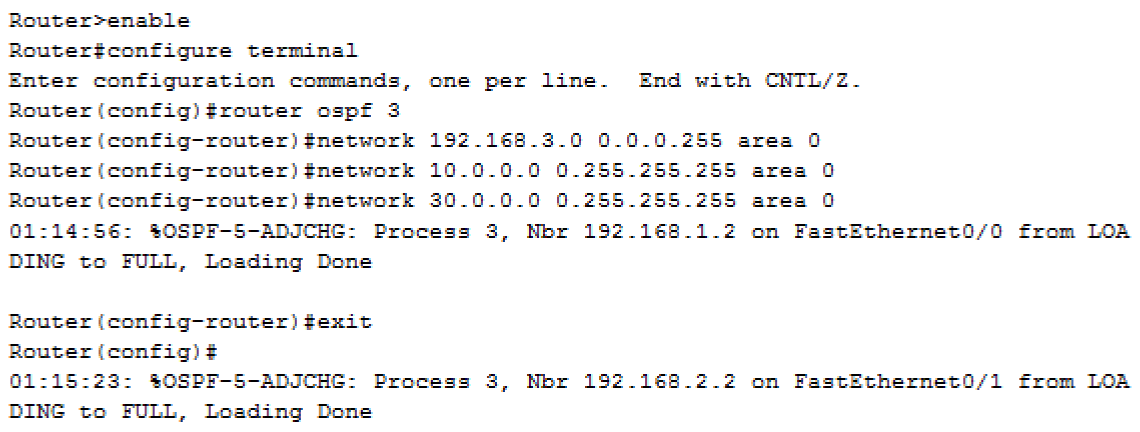
Step 3: Observe the packet movement from a node in a LAN1 to another node in LAN2

6. Simulate a topology where 3 routers are fully connected and each router connected to an end device. Observe the flow of ICMP packets from one network to other using OSPF protocol.

Step 1: Configure the nodes and routers as show below.



Step 2: Configure each router with OSPF protocol using CLI.
For each router, configure network connected to it with its wild card mask (is the inverse of subnetmask i.e if subnet mask = 255.0.0.0 wildcard mask = 0.255.255.255).
Step 3: Observe the packet movement from a node in a LAN1 to another node in LAN2



```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router ospf 3
Router(config-router)#network 192.168.3.0 0.0.0.255 area 0
Router(config-router)#network 10.0.0.0 0.255.255.255 area 0
Router(config-router)#network 30.0.0.0 0.255.255.255 area 0
01:14:56: %OSPF-5-ADJCHG: Process 3, Nbr 192.168.1.2 on FastEthernet0/0 from LOA
DING to FULL, Loading Done

Router(config-router)#exit
Router(config)#
01:15:23: %OSPF-5-ADJCHG: Process 3, Nbr 192.168.2.2 on FastEthernet0/1 from LOA
DING to FULL, Loading Done
```

Step 3: Observe the packet movement from a node in a LAN1 to another node in LAN2

7. Simulate a network for browsing and understand DNS protocol.

Part B

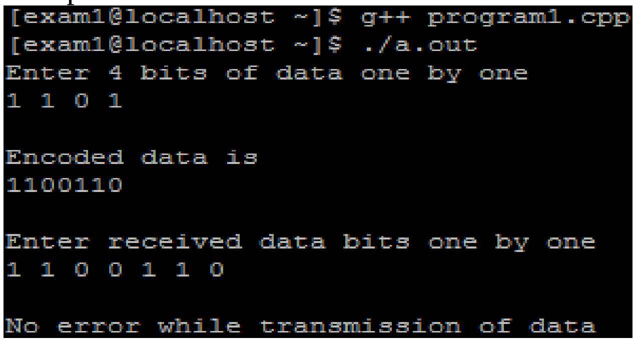
1. **Write a program to implement error detection/ error correction using hamming code.**

   Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction. In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

```
#include<stdio.h>
void main() {
 int data[10];
 int dataatrec[10],c,c1,c2,c3,i;

printf("Enter 4 bits of data one by one\n");
scanf("%d",&data[0]);
scanf("%d",&data[1]);
scanf("%d",&data[2]);
scanf("%d",&data[4]);
 // calculate even parity
data[6]=data[0]^data[2]^data[4];
data[5]=data[0]^data[1]^data[4];
data[3]=data[0]^data[1]^data[2];
 printf("\nEncoded data is\n");

for(i=0;i<7;i++)
        printf("%d",data[i]);
 printf("\n\nEnter received data bits one by one\n");
 for(i=0;i<7;i++)
        scanf("%d",&dataatrec[i]);

c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
c=c3*4+c2*2+c1 ;

    if(c==0) {
printf("\nNo error while transmission of data\n");
    }
else {
printf("\nError on position %d",c);

printf("\nData sent : ");
        for(i=0;i<7;i++)
```

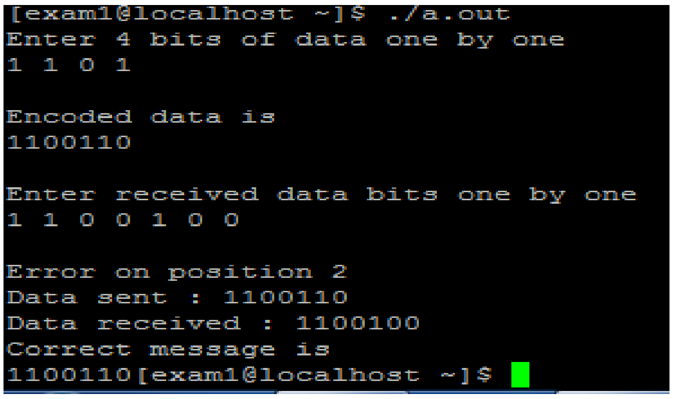
```
            printf("%d",data[i]);

  printf("\nData received : ");
        for(i=0;i<7;i++)
            printf("%d",dataatrec[i]);

  printf("\nCorrect message is\n");
    //if errorneous bit is 0 we complement it else vice versa
  if(dataatrec[7-c]==0)
  dataatrec[7-c]=1;
   else
  dataatrec[7-c]=0;
  for (i=0;i<7;i++) {
  printf("%d",dataatrec[i]);
  }
  }
  return 0;
  }
```

Output: Case 1 without error



```
[exam1@localhost ~]$ g++ program1.cpp
[exam1@localhost ~]$ ./a.out
Enter 4 bits of data one by one
1 1 0 1

Encoded data is
1100110

Enter received data bits one by one
1 1 0 0 1 1 0

No error while transmission of data
```

Case 2: with error



```
[exam1@localhost ~]$ ./a.out
Enter 4 bits of data one by one
1 1 0 1

Encoded data is
1100110

Enter received data bits one by one
1 1 0 0 1 0 0

Error on position 2
Data sent : 1100110
Data received : 1100100
Correct message is
1100110[exam1@localhost ~]$
```

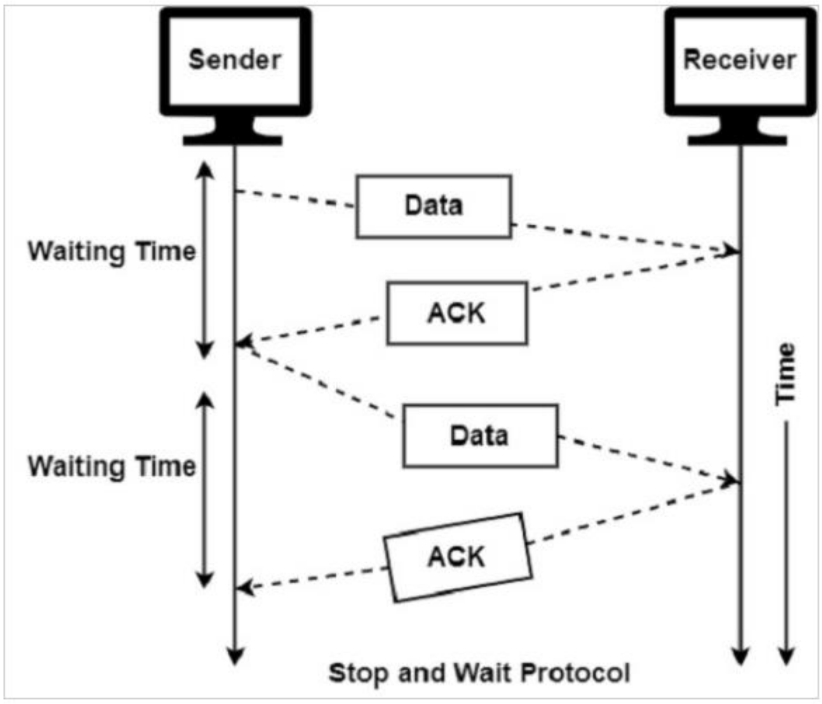
2. **Write a program to show working of the Stop and wait protocol.**

Stop and wait protocol is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will **stop and wait** for the acknowledgement from the receiver. This time (i.e. the time joining message transmitting and acknowledgement receiving) is the sender's waiting time, and the sender is idle during this time. When the sender gets the acknowledgement (ACK), it will send the next data packet to the receiver and wait for the disclosure again, and this process will continue as long as the sender has the data to send.

Characteristics

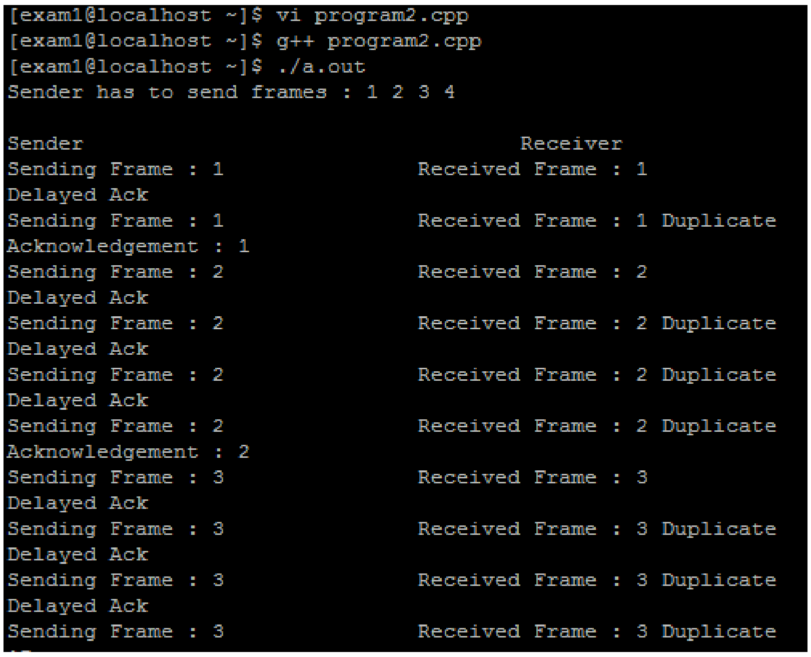
It is used in Connection-oriented communication.

- It offers error and flows control.
- It can be used in data Link and transport Layers.
- Stop and Wait ARQ executes Sliding Window Protocol with Window Size 1



```
#include<iostream>
#include <time.h>
#include <cstdlib>
#include<ctime>
#include <unistd.h>
using namespace std;
class timer {
    private:
     unsigned long begTime;
    public:
     void start() {
      begTime = clock();
     }
  unsigned long elapsedTime() {
      return ((unsigned long) clock() - begTime) / CLOCKS_PER_SEC;
    }
   bool isTimeout(unsigned long seconds) {
      return seconds >= elapsedTime();
     }
};
int main()
{
 int frames[] = {1,2,3,4};
 unsigned long seconds = 5;
 srand(time(NULL));
 timer t;
 cout<<"Sender has to send frames : ";
 for(int i=0;i<4;i++)
     cout<<frames[i]<<" ";
 cout<<endl;
 int count = 0;
 bool delay = false;
 cout<<endl<<"Sender\t\t\t\t\tReceiver"<<endl;
do
 {
     bool timeout = false;
     cout<<"Sending Frame : "<<frames[count];
     cout.flush();
     cout<<"\t\t";
     t.start();
     if(rand()%2)
     {
         int to = 2450 + rand()%(6400 - 2450)  + 1;
             for(int i=0;i<6400;i++)
             for(int j=0;j<to;j++) {}
     }
     if(t.elapsedTime() <= seconds)
     {
         cout<<"Received Frame : "<<frames[count]<<" ";
         if(delay)
         {
             cout<<"Duplicate";
             delay = false;
         }
         cout<<endl;
         count++;
     }
     else
     {
         cout<<"---"<<endl;
         cout<<"Timeout"<<endl;
         timeout = true;
     }
t.start();
     if(rand()%2 || !timeout)
     {
         int to = 24500 + rand()%(64000 - 24500)  + 1;
         for(int i=0;i<64000;i++)
             for(int j=0;j<to;j++) {}
         if(t.elapsedTime() > seconds )
         {
             cout<<"Delayed Ack"<<endl;
             count--;
             delay = true;
         }
         else if(!timeout)
             cout<<"Acknowledgement : "<<frames[count]-1<<endl;
     }
 }while(count!=4);
 return 0;
}
```

Output:



```
[exam1@localhost ~]$ vi program2.cpp
[exam1@localhost ~]$ g++ program2.cpp
[exam1@localhost ~]$ ./a.out
Sender has to send frames : 1 2 3 4

Sender                                  Receiver
Sending Frame : 1               Received Frame : 1
Delayed Ack
Sending Frame : 1               Received Frame : 1 Duplicate
Acknowledgement : 1
Sending Frame : 2               Received Frame : 2
Delayed Ack
Sending Frame : 2               Received Frame : 2 Duplicate
Delayed Ack
Sending Frame : 2               Received Frame : 2 Duplicate
Delayed Ack
Sending Frame : 2               Received Frame : 2 Duplicate
Acknowledgement : 2
Sending Frame : 3               Received Frame : 3
Delayed Ack
Sending Frame : 3               Received Frame : 3 Duplicate
Delayed Ack
Sending Frame : 3               Received Frame : 3 Duplicate
Delayed Ack
Sending Frame : 3               Received Frame : 3 Duplicate
```

3. Implementation of CSMA/CD.

Consider a scenario where there are ‘n’ stations on a link and all are waiting to transfer data through that channel. In this case, all ‘n’ stations would want to access the link/channel to transfer their own data. The problem arises when more than one station transmits the data at the moment. In this case, there will be collisions in the data from different stations.

CSMA/CD is one such technique where different stations that follow this protocol agree on some terms and collision detection measures for effective transmission. This protocol decides which station will transmit when so that data reaches the destination without corruption.

Working principal:

Step 1: Check if the sender is ready for transmitting data packets.

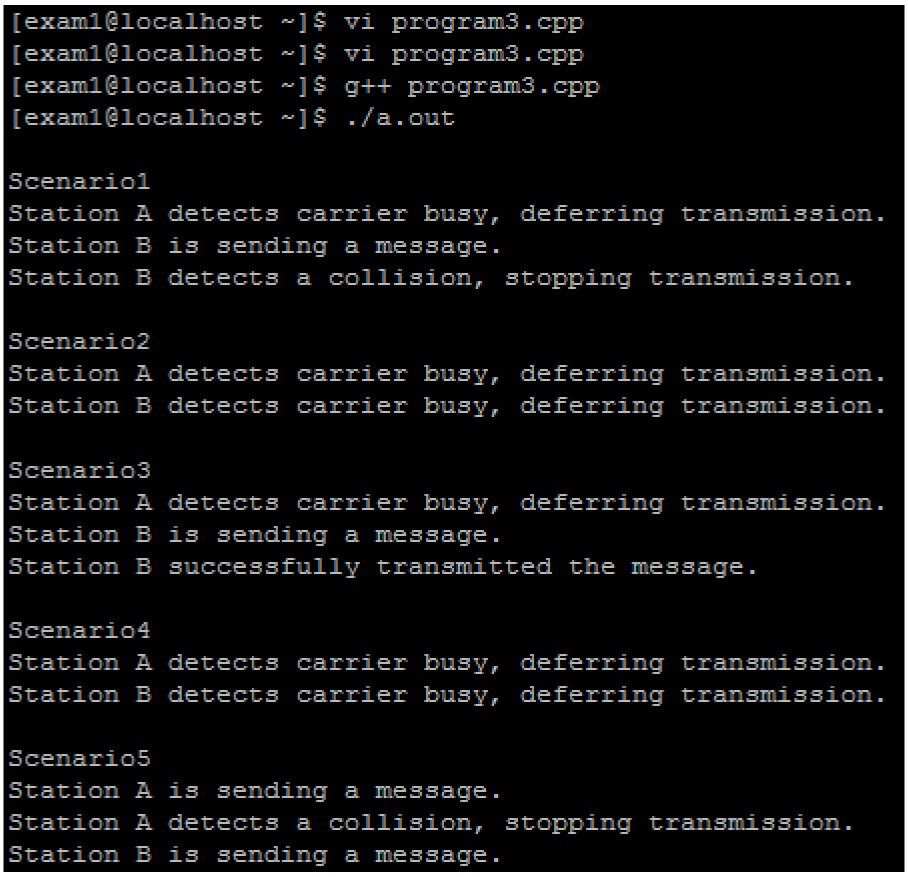
Step 2: Check if the transmission link is idle.
Sender has to keep on checking if the transmission link/medium is idle. For this, it continuously senses transmissions from other nodes. Sender sends dummy data on the link. If it does not receive any collision signal, this means the link is idle at the moment. If it senses that the carrier is free and there are no collisions, it sends the data. Otherwise, it refrains from sending data.

Step 3: Transmit the data & check for collisions.
Sender transmits its data on the link. CSMA/CD does not use an ‘acknowledgment’ system. It checks for successful and unsuccessful transmissions through collision signals. During transmission, if a collision signal is received by the node, transmission is stopped. The station then transmits a jam signal onto the link and waits for random time intervals before it resends the frame. After some random time, it again attempts to transfer the data and repeats the above process.

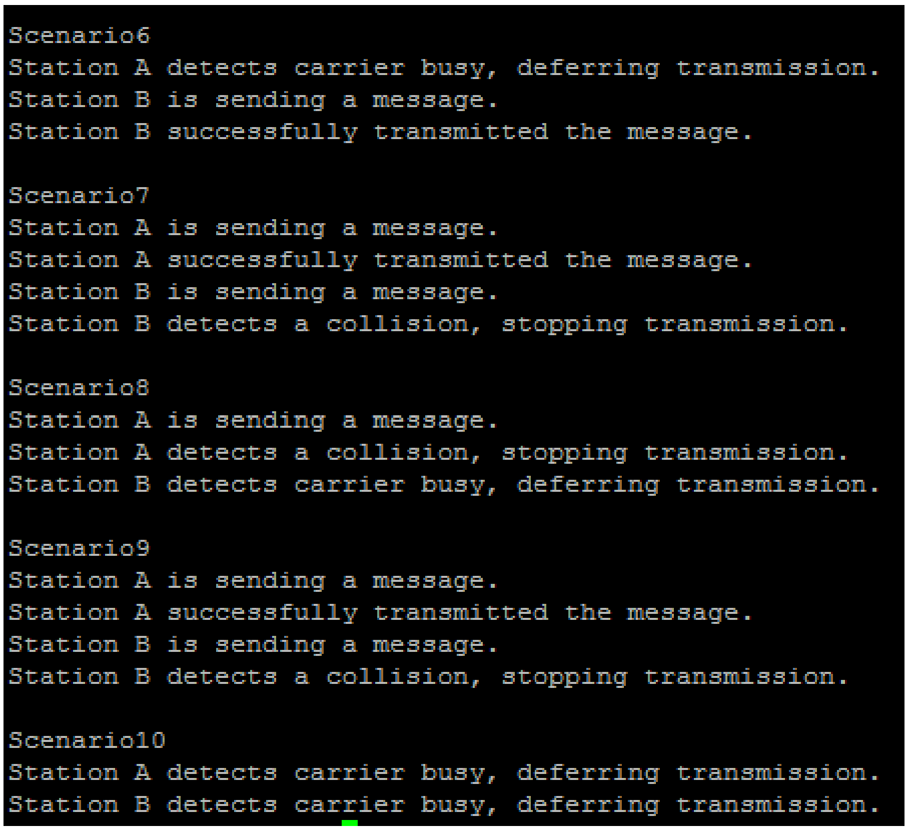
Step 4: If no collision was detected in propagation, the sender completes its frame transmission and resets the counters.

```
#include <iostream>
#include <cstdlib> // for rand()
#include <ctime>   // for time()

class Station {
public:
    Station(std::string name) : name(name) {}

    void send() {

        if (carrierBusy()) {
            std::cout << name << " detects carrier busy, deferring transmission." << std::endl;
            return;
        }

        std::cout << name << " is sending a message." << std::endl;

        if (collisionDetected()) {
            std::cout << name << " detects a collision, stopping transmission." << std::endl;
        } else {
            std::cout << name << " successfully transmitted the message." << std::endl;
        }
    }
   bool carrierBusy() {
        // Simulate a random chance of the carrier being busy
         return rand() % 2 == 1;
          }
            bool collisionDetected() {
                                 // Simulate a random chance of a collision
             return rand()*% 2 == 1;
                               }
                    private:
                                   std::string name;
    };


          int main() {

             Station stationA("Station A");
            Station stationB("Station B");
             int i=1;
          // Simulate both stations trying to send messages
          while(i<=10){
                std::cout<<"Scenario"<<i<<std::endl;
                stationA.send();
                 stationB.send();
                i++;
            }
                return 0;
}
```

Output:



```
[exam1@localhost ~]$ vi program3.cpp
[exam1@localhost ~]$ vi program3.cpp
[exam1@localhost ~]$ g++ program3.cpp
[exam1@localhost ~]$ ./a.out

Scenario1
Station A detects carrier busy, deferring transmission.
Station B is sending a message.
Station B detects a collision, stopping transmission.

Scenario2
Station A detects carrier busy, deferring transmission.
Station B detects carrier busy, deferring transmission.

Scenario3
Station A detects carrier busy, deferring transmission.
Station B is sending a message.
Station B successfully transmitted the message.

Scenario4
Station A detects carrier busy, deferring transmission.
Station B detects carrier busy, deferring transmission.

Scenario5
Station A is sending a message.
Station A detects a collision, stopping transmission.
Station B is sending a message.
```



```
Scenario6
Station A detects carrier busy, deferring transmission.
Station B is sending a message.
Station B successfully transmitted the message.

Scenario7
Station A is sending a message.
Station A successfully transmitted the message.
Station B is sending a message.
Station B detects a collision, stopping transmission.

Scenario8
Station A is sending a message.
Station A detects a collision, stopping transmission.
Station B detects carrier busy, deferring transmission.

Scenario9
Station A is sending a message.
Station A successfully transmitted the message.
Station B is sending a message.
Station B detects a collision, stopping transmission.

Scenario10
Station A detects carrier busy, deferring transmission.
Station B detects carrier busy, deferring transmission.
```

4. **Write a program to implement Distance Vector Routing algorithm**.

A **distance-vector routing (DVR)** protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

Bellman Ford Basics – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances,based on a chosen metric, are computed using information from the neighbors' distance vectors.

Distance Vector Algorithm –

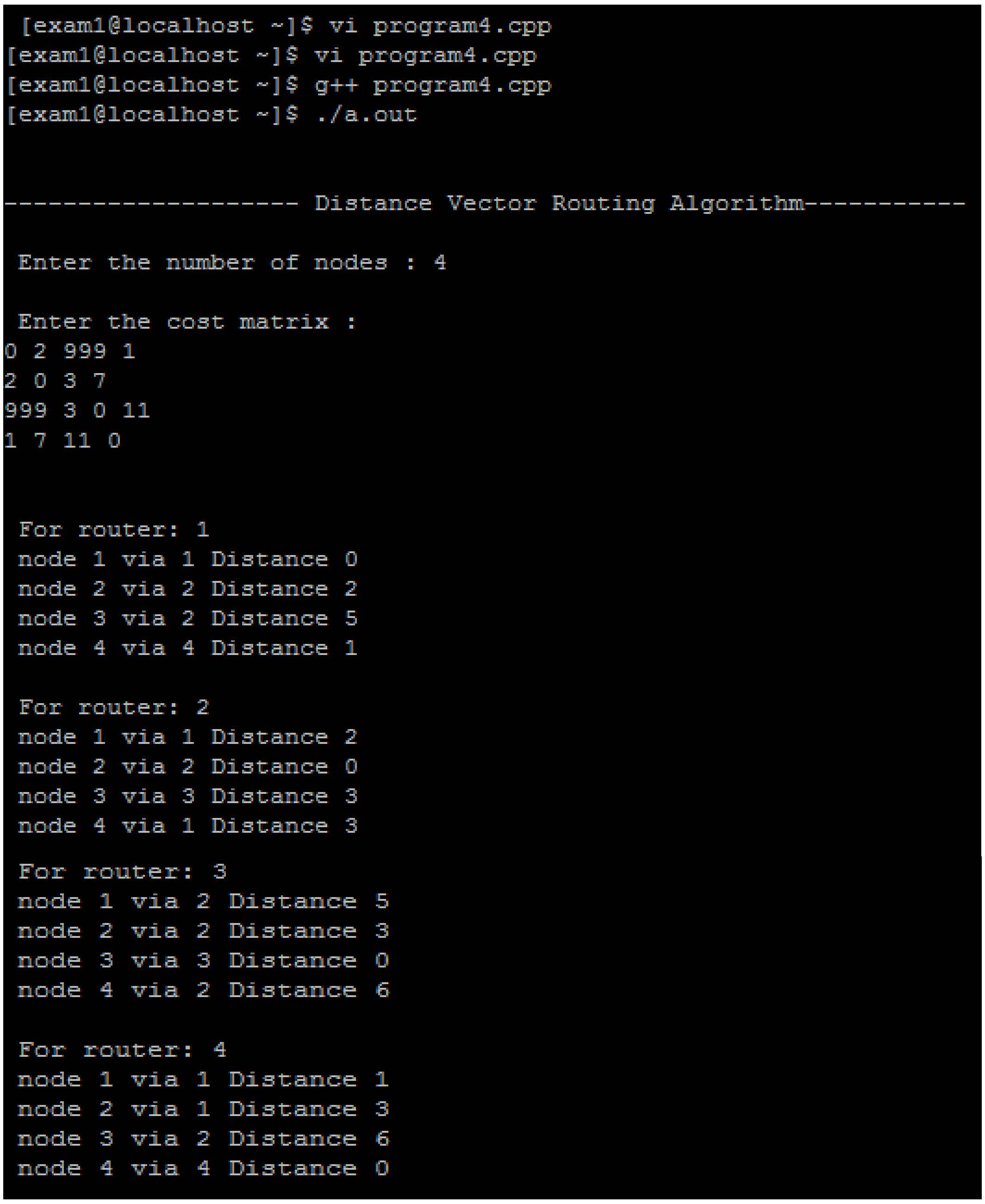
1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
   - It receives a distance vector from a neighbor containing different information than before.

- It discovers that a link to a neighbor has gone down.

```
#include<stdio.h>
#include<iostream>
using namespace std;
struct node
{
   unsigned dist[6];
   unsigned from[6];
}DVR[10];
int main()
{
   cout<<"\n\n-------------------- Distance Vector Routing Algorithm----------- ";
   int costmat[6][6];
   int nodes, i, j, k;
   cout<<"\n\n Enter the number of nodes : ";
   cin>>nodes; //Enter the nodes
   cout<<"\n Enter the cost matrix : \n" ;
   for(i = 0; i < nodes; i++)
    {
      for(j = 0; j < nodes; j++)
      {
         cin>>costmat[i][j];
         costmat[i][i] = 0;
         DVR[i].dist[j] = costmat[i][j]; //initialise the distance equal to cost matrix
         DVR[i].from[j] = j;
      }
   }
         for(i = 0; i < nodes; i++) //We choose arbitary vertex k and we calculate the
         //direct distance from the node i to k using the cost matrix and add the distance from k to
node j
         for(j = i+1; j < nodes; j++)
         for(k = 0; k < nodes; k++)
            if(DVR[i].dist[j] > costmat[i][k] + DVR[k].dist[j])
            {   //We calculate the minimum distance
               DVR[i].dist[j] = DVR[i].dist[k] + DVR[k].dist[j];
               DVR[j].dist[i] = DVR[i].dist[j];
               DVR[i].from[j] = k;
               DVR[j].from[i] = k;
            }
      for(i = 0; i < nodes; i++)
      {
         cout<<"\n\n For router: "<<i+1;
         for(j = 0; j < nodes; j++)
            cout<<"\t\n node "<<j+1<<" via "<<DVR[i].from[j]+1<<" Distance "<<DVR[i].dist[j];
      }
   cout<<" \n\n ";
   return 0;
```

}

Output:



```
 [exam1@localhost ~]$ vi program4.cpp
[exam1@localhost ~]$ vi program4.cpp
[exam1@localhost ~]$ g++ program4.cpp
[exam1@localhost ~]$ ./a.out


-------------------- Distance Vector Routing Algorithm-----------

 Enter the number of nodes : 4

 Enter the cost matrix :
0 2 999 1
2 0 3 7
999 3 0 11
1 7 11 0


 For router: 1
 node 1 via 1 Distance 0
 node 2 via 2 Distance 2
 node 3 via 2 Distance 5
 node 4 via 4 Distance 1

 For router: 2
 node 1 via 1 Distance 2
 node 2 via 2 Distance 0
 node 3 via 3 Distance 3
 node 4 via 1 Distance 3
 For router: 3
 node 1 via 2 Distance 5
 node 2 via 2 Distance 3
 node 3 via 3 Distance 0
 node 4 via 2 Distance 6

 For router: 4
 node 1 via 1 Distance 1
 node 2 via 1 Distance 3
 node 3 via 2 Distance 6
 node 4 via 4 Distance 0
```

Least cost tree using link state protocol

5. **Creating a C++ program for the least cost tree using the Link State Routing algorithm requires simulating a network of nodes and performing the Link State algorithm. Below is a simplified example of how you can implement it:**

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

const int INF = INT_MAX;

class Network {
public:
    int numNodes;
    vector<vector<int>> costMatrix;

    Network(int nodes) : numNodes(nodes) {
        costMatrix.resize(nodes, vector<int>(nodes, INF));
    }

    void addLink(int node1, int node2, int cost) {
        costMatrix[node1][node2] = cost;
        costMatrix[node2][node1] = cost;
    }

    void printLeastCostTree(int source, const vector<int>& parent) {
        cout << "Least Cost Tree:" << endl;
        for (int i = 0; i < numNodes; ++i) {
            if (i != source) {
                cout << "Node " << i << " -> Node " << parent[i] << " (Cost: " <<
costMatrix[i][parent[i]] << ")" << endl;
            }
        }
    }

    void linkStateRouting(int source) {
        vector<int> distance(numNodes, INF);
        vector<bool> inTree(numNodes, false);
        vector<int> parent(numNodes, -1);
        distance[source] = 0;

        for (int i = 0; i < numNodes - 1; ++i) {
            int u = getMinDistanceVertex(distance, inTree);
            inTree[u] = true;

            for (int v = 0; v < numNodes; ++v) {
                if (!inTree[v] && costMatrix[u][v] != INF && distance[u] + costMatrix[u][v]
< distance[v]) {
                    parent[v] = u;
                    distance[v] = distance[u] + costMatrix[u][v];
                }
            }
        }

        printLeastCostTree(source, parent);
    }

    int getMinDistanceVertex(const vector<int>& distance, const vector<bool>& inTree)
{
        int minDistance = INF;
        int minVertex = -1;

        for (int v = 0; v < numNodes; ++v) {
            if (!inTree[v] && distance[v] < minDistance) {
                minDistance = distance[v];
                minVertex = v;
            }
        }

        return minVertex;
    }
};

int main() {
    int numNodes = 4;
    Network network(numNodes);

    // Add links with their costs
    network.addLink(0, 1, 4);
    network.addLink(0, 2, 2);
    network.addLink(1, 2, 5);
    network.addLink(1, 3, 10);
    network.addLink(2, 3, 1);
    int sourceNode = 0;
    network.linkStateRouting(sourceNode);

    return 0;
}
```

6. To write echo client-server application using TCP.

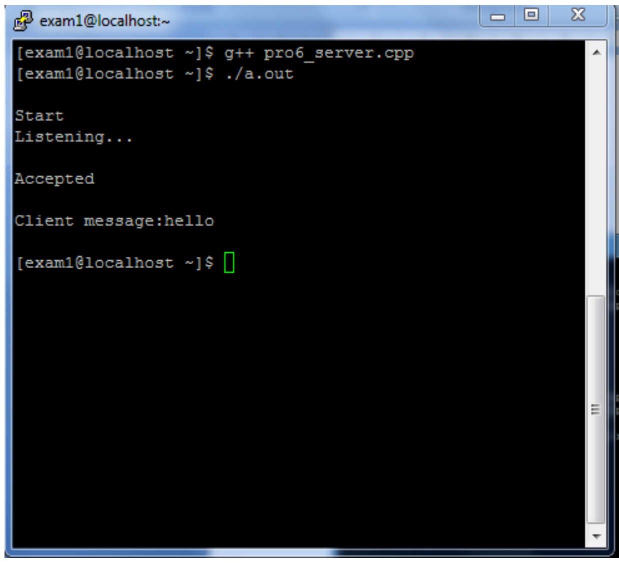
```
// TCP server side

//Include headers
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<unistd.h>
//Define server port
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
  //variable declaration
      int sockfd,newsockfd;
      socklen_t clength;
      struct sockaddr_in serv_addr,cli_addr;
      char buffer[4096];


   // create socket
      sockfd=socket(AF_INET,SOCK_STREAM,0);

   //Initialize server addres structure
      serv_addr.sin_family=AF_INET;
      serv_addr.sin_addr.s_addr=INADDR_ANY;
      serv_addr.sin_port=htons(SERV_TCP_PORT);

      printf("\nStart");
 // bind socket
      bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
      printf("\nListening...");
      printf("\n");
//Listen incoming connection
      listen(sockfd,5);
```

```
// accept connection from client
        clength=sizeof(cli_addr);
        newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
        printf("\nConnection Accepted");
        printf("\n");
//Read client message
        read(newsockfd,buffer,4096);
        printf("\nClient message:%s",buffer);
//echo back to client
        write(newsockfd,buffer,4096);
        printf("\n");
//close sockets
        close(sockfd);
        close(newsockfd);
        return 0;
}


//TCP client side
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<unistd.h>
#include<arpa/inet.h>
#define SERV_TCP_PORT 5035
int main(int argc,char*argv[])
{
        int sockfd;
        struct sockaddr_in serv_addr;
        struct hostent *server;
        char buffer[4096];

        sockfd=socket(AF_INET,SOCK_STREAM,0);
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        serv_addr.sin_port=htons(SERV_TCP_PORT);

        printf("\nReady for sending...");
        connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
        printf("\nEnter the message to send\n");
        printf("\nClient: ");
        fgets(buffer,4096,stdin);
        write(sockfd,buffer,4096);
        printf("Serverecho:%s",buffer);
        printf("\n");
        close(sockfd);
        return 0;
}
```

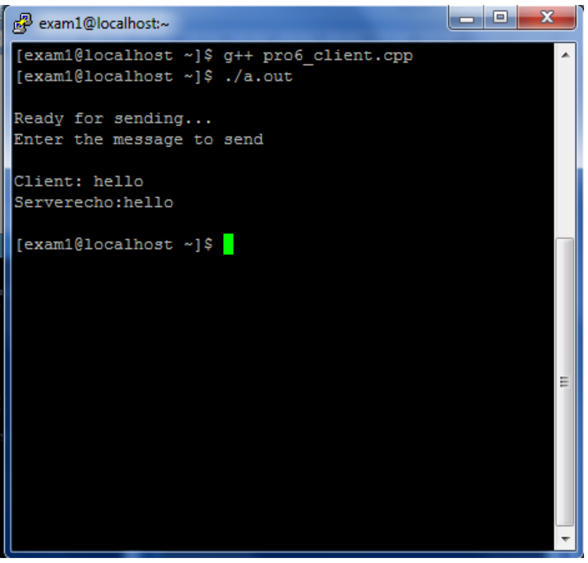
Output:

At server terminal

At client terminal



```
[exam1@localhost ~]$ g++ pro6_server.cpp
[exam1@localhost ~]$ ./a.out

Start
Listening...

Accepted

Client message:hello

[exam1@localhost ~]$
```



```
[exam1@localhost ~]$ g++ pro6_client.cpp
[exam1@localhost ~]$ ./a.out

Ready for sending...
Enter the message to send

Client: hello
Serverecho:hello

[exam1@localhost ~]$
```