Practical File

Of

Computer Networks

(PCC-CS-602)

submitted in partial fulfilment of the requirement for the award of degree of

Bachelor of Technology (B.Tech)

in

Computer Engineering

by

Ayush (21001003022)

Under the guidance of

Dr. Umesh Kumar

Department of Computer Engineering

J. C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA
SECTOR-6 FARIDABAD HARYANA-121006

INDEX

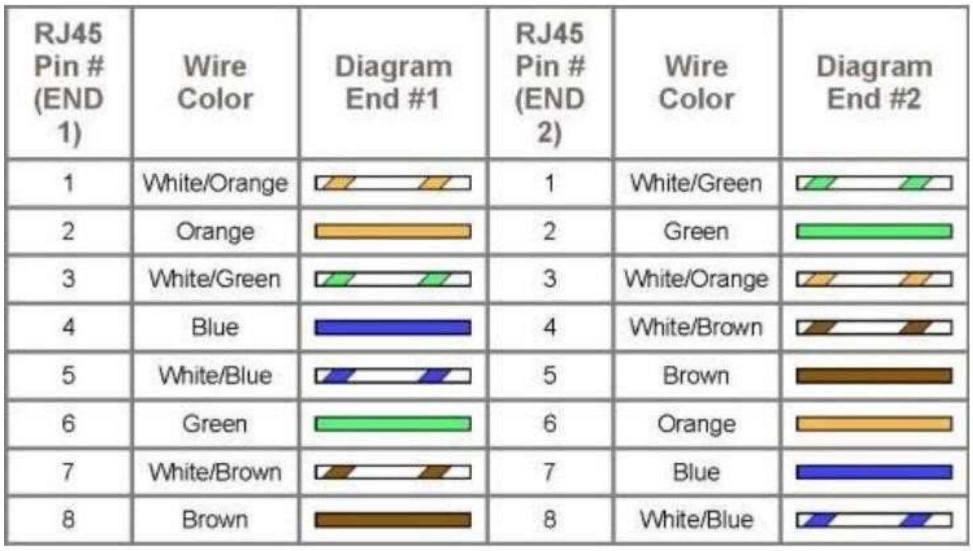
EXPERIMENT-1

Aim:
Study of different types of Network cables and practically implement the cross-wired cable and straight through cable using clamping tool.

Procedure:
The following steps should be done:

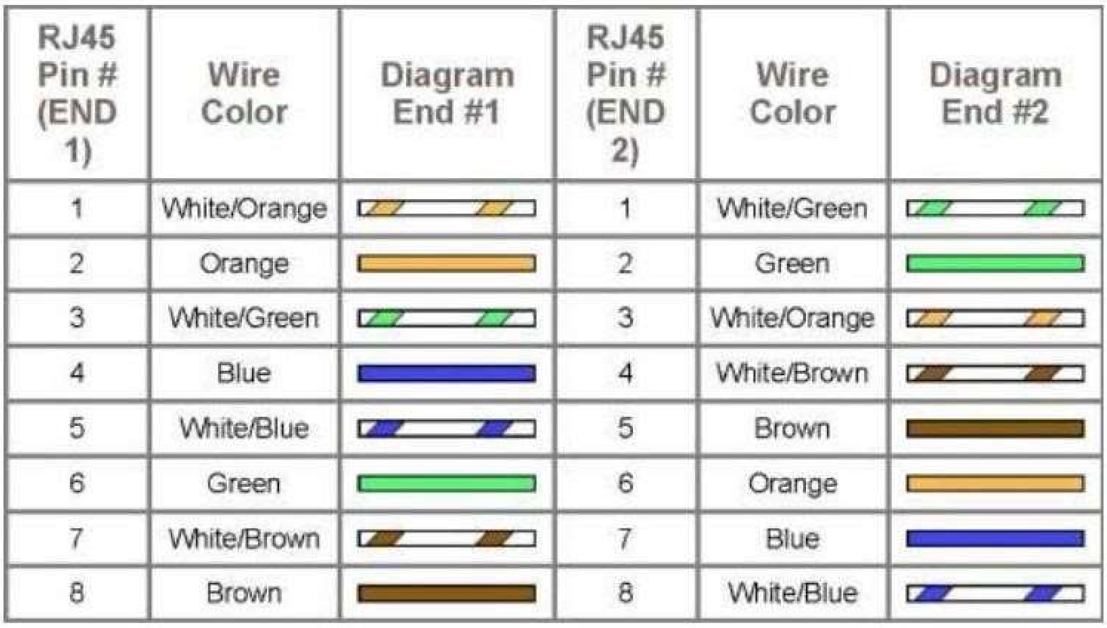
- Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be very careful at this point, as to not nick or cut into the wires, which are inside. Doing so could alter the characteristics of your cable, or even worse render is useless. Check the wires, one more time for nicks or cuts. If there are any, just whack the whole end off, and start over.
- Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand. You do not want the wires to become untwisted down inside the jacket. Category 5 cable must only have 1/2 of an inch of 'untwisted' wire at the end; otherwise, it will be 'out of spec'. At this point, you obviously have ALOT more than 1/2 of an inch of un-twisted wire.
- You have 2 end jacks, which must be installed on your cable. If you are using a premade cable, with one of the ends whacked off, you only have one end to install - the crossed over end. Below are two diagrams, which show how you need to arrange the cables for each type of cable end. Decide at this point which end you are making and examine the associated picture below.

Diagram shows you how to prepare Cross wired connection



RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange	▨	1	White/Green	▨
2	Orange	🟧	2	Green	🟩
3	White/Green	▨	3	White/Orange	▨
4	Blue	🟦	4	White/Brown	▨
5	White/Blue	▨	5	Brown	🟫
6	Green	🟩	6	Orange	🟧
7	White/Brown	▨	7	Blue	🟦
8	Brown	🟫	8	White/Blue	▨

Diagram shows you how to prepare straight through wired connection



RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange	⬜🟧	1	White/Green	⬜🟩
2	Orange	🟧	2	Green	🟩
3	White/Green	⬜🟩	3	White/Orange	⬜🟧
4	Blue	🟦	4	White/Brown	⬜🟫
5	White/Blue	⬜🟦	5	Brown	🟫
6	Green	🟩	6	Orange	🟧
7	White/Brown	⬜🟫	7	Blue	🟦
8	Brown	🟫	8	White/Blue	⬜🟦

EXPERIMENT-2

Aim:

Study of following Network Devices in Detail

- Repeater
- Hub
- Switch
- Bridge
- Router
- Gate Way

1. Repeater:
Functioning at Physical Layer. A repeater is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports, so cannot be use to connect for more than two devices.

2. Hub:
An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. Switch:
A network switch or switching hub is a computer networking device that connects network segments. The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. Bridge:
A network bridge connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. Switch or Layer 2 switch is often used interchangeably with bridge. Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.

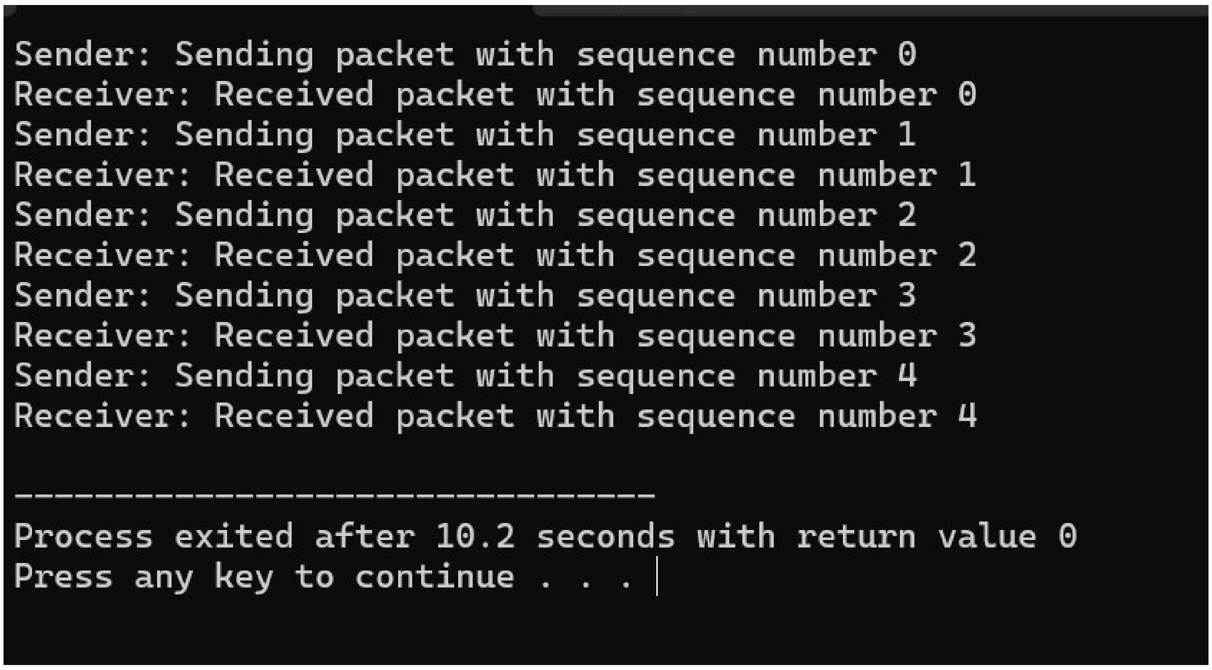
5. Router:
A router is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers

exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. Gate Way:
In a communications network, a network node equipped for interfacing with another network that uses different protocols.

- A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.
- A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

<u>OUTPUT:</u>



```
Sender: Sending packet with sequence number 0
Receiver: Received packet with sequence number 0
Sender: Sending packet with sequence number 1
Receiver: Received packet with sequence number 1
Sender: Sending packet with sequence number 2
Receiver: Received packet with sequence number 2
Sender: Sending packet with sequence number 3
Receiver: Received packet with sequence number 3
Sender: Sending packet with sequence number 4
Receiver: Received packet with sequence number 4

--------------------------------
Process exited after 10.2 seconds with return value 0
Press any key to continue . . .
```

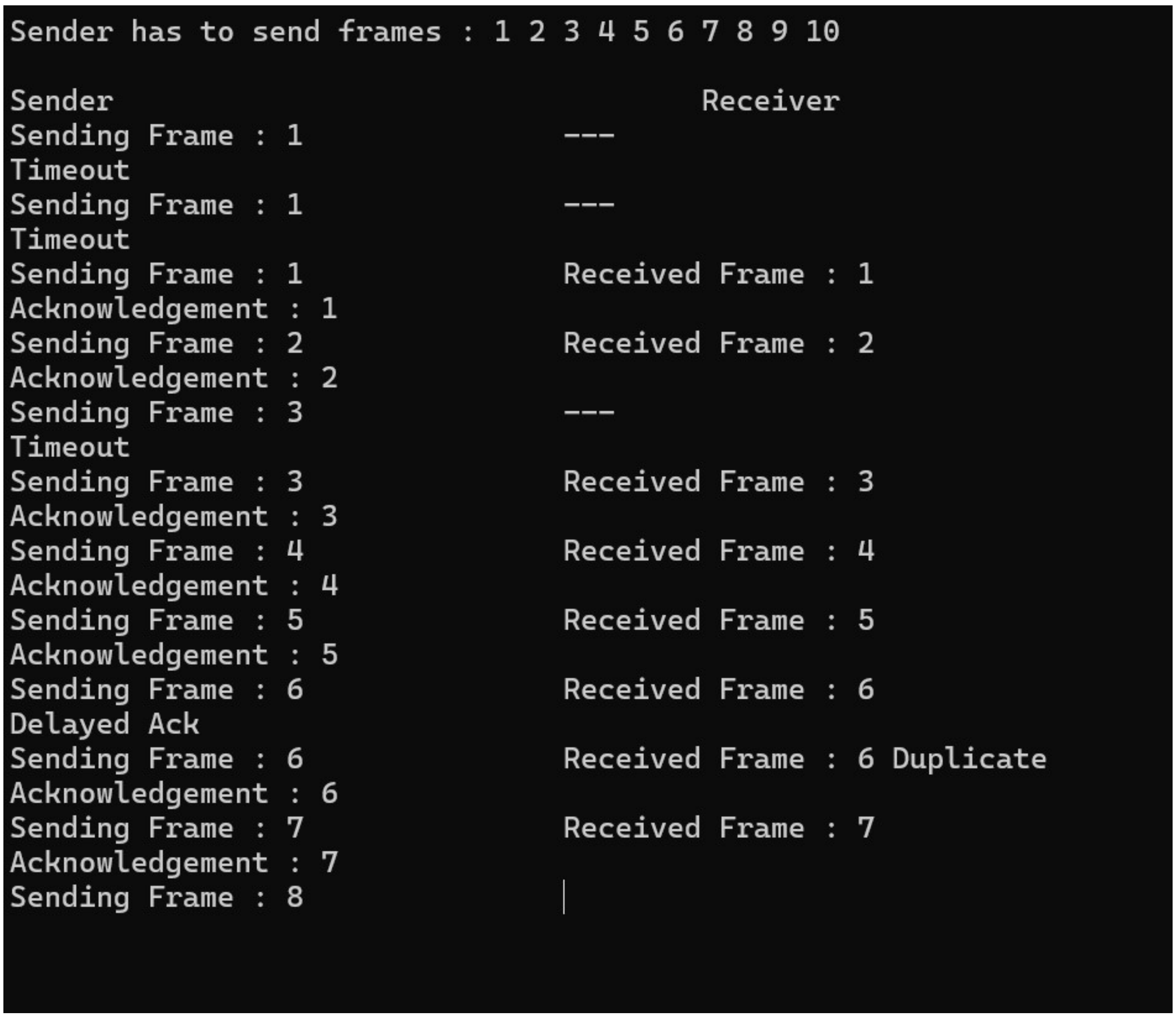
EXPERIMENT-3

Write a program to implement simplest protocol for noiseless channel.

CODE:

```
#include <iostream>
#include <cstdlib>
#include <ctime> #include <unistd.h>
using namespace std;

class Packet { public:
    int sequenceNumber;

    Packet(int seq) : sequenceNumber(seq) {}
};

class Sender { public:     void send(Packet packet) {
        cout << "Sender: Sending packet with sequence number " <<
packet.sequenceNumber
<< endl;
        sleep(1);
    }
};

class Receiver { public:     void receive(Packet packet) {
        cout << "Receiver: Received packet with sequence number " <<
packet.sequenceNumber <<endl;
        sleep(1);
    }
};

int main() {
    srand(time(NULL));

    Sender sender;
    Receiver receiver;

    int sequenceNumber = 0;     int totalPackets = 5;
    while (sequenceNumber < totalPackets) {         Packet
packet(sequenceNumber);
        sender.send(packet);         receiver.receive(packet);
        ++sequenceNumber;
    }
}
```

OUTPUT:



```
Sender has to send frames : 1 2 3 4 5 6 7 8 9 10

Sender                                  Receiver
Sending Frame : 1               ---
Timeout
Sending Frame : 1               ---
Timeout
Sending Frame : 1               Received Frame : 1
Acknowledgement : 1
Sending Frame : 2               Received Frame : 2
Acknowledgement : 2
Sending Frame : 3               ---
Timeout
Sending Frame : 3               Received Frame : 3
Acknowledgement : 3
Sending Frame : 4               Received Frame : 4
Acknowledgement : 4
Sending Frame : 5               Received Frame : 5
Acknowledgement : 5
Sending Frame : 6               Received Frame : 6
Delayed Ack
Sending Frame : 6               Received Frame : 6 Duplicate
Acknowledgement : 6
Sending Frame : 7               Received Frame : 7
Acknowledgement : 7
Sending Frame : 8
```

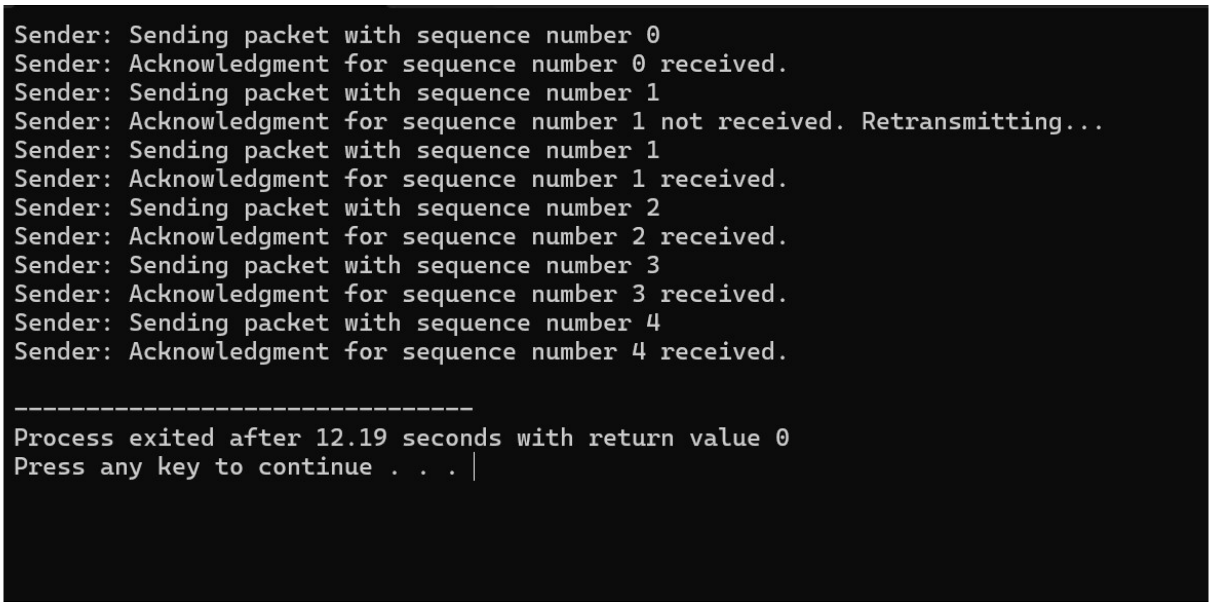
EXPERIMENT-4

Write a program to implement stop and wait protocol for noiseless channel.

CODE:

```
#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[] = {1,2,3,4,5,6,7,8,9,10};  unsigned long seconds = 5;
 srand(time(NULL));
 timer t;
 cout<<"Sender has to send frames : ";  for(int i=0;i<10;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 = 24600 + rand()%(64000 - 24600)  + 1;
for(int i=0;i<64000;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 = 24600 + rand()%(64000 - 24600)  + 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!=10);  return 0; }
```

OUTPUT:



```
Sender: Sending packet with sequence number 0
Sender: Acknowledgment for sequence number 0 received.
Sender: Sending packet with sequence number 1
Sender: Acknowledgment for sequence number 1 not received. Retransmitting...
Sender: Sending packet with sequence number 1
Sender: Acknowledgment for sequence number 1 received.
Sender: Sending packet with sequence number 2
Sender: Acknowledgment for sequence number 2 received.
Sender: Sending packet with sequence number 3
Sender: Acknowledgment for sequence number 3 received.
Sender: Sending packet with sequence number 4
Sender: Acknowledgment for sequence number 4 received.

--------------------------------
Process exited after 12.19 seconds with return value 0
Press any key to continue . . .
```

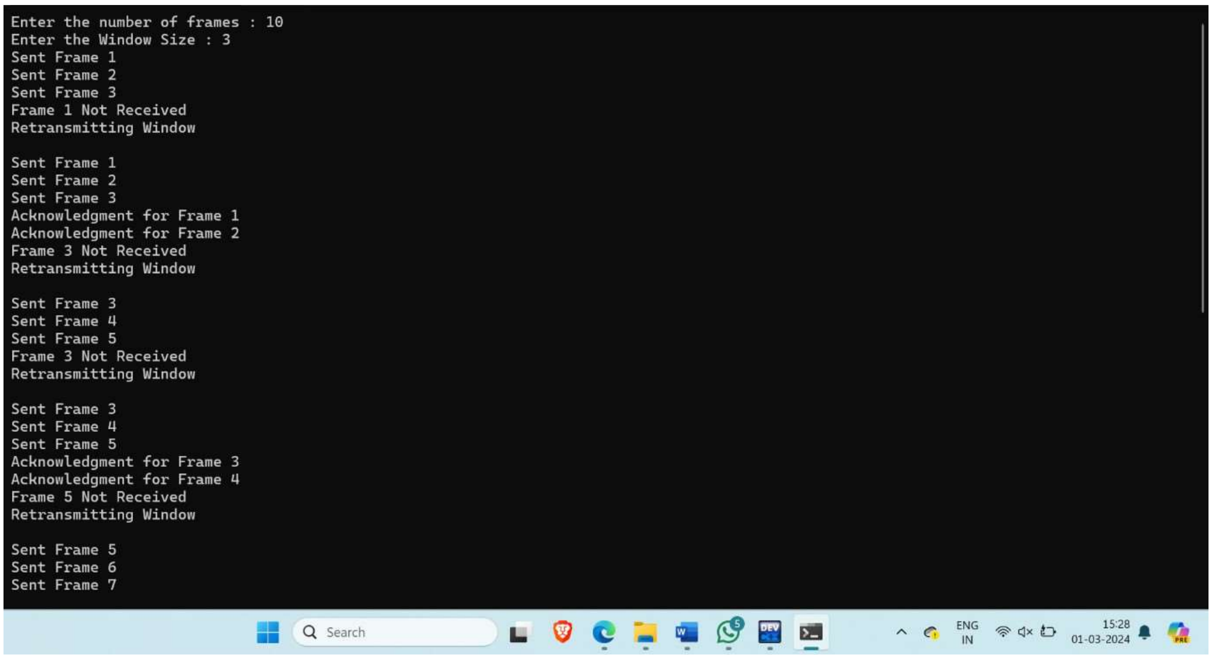
EXPERIMENT-5

Write a program to implement stop and wait ARQ protocol for noisy channel.

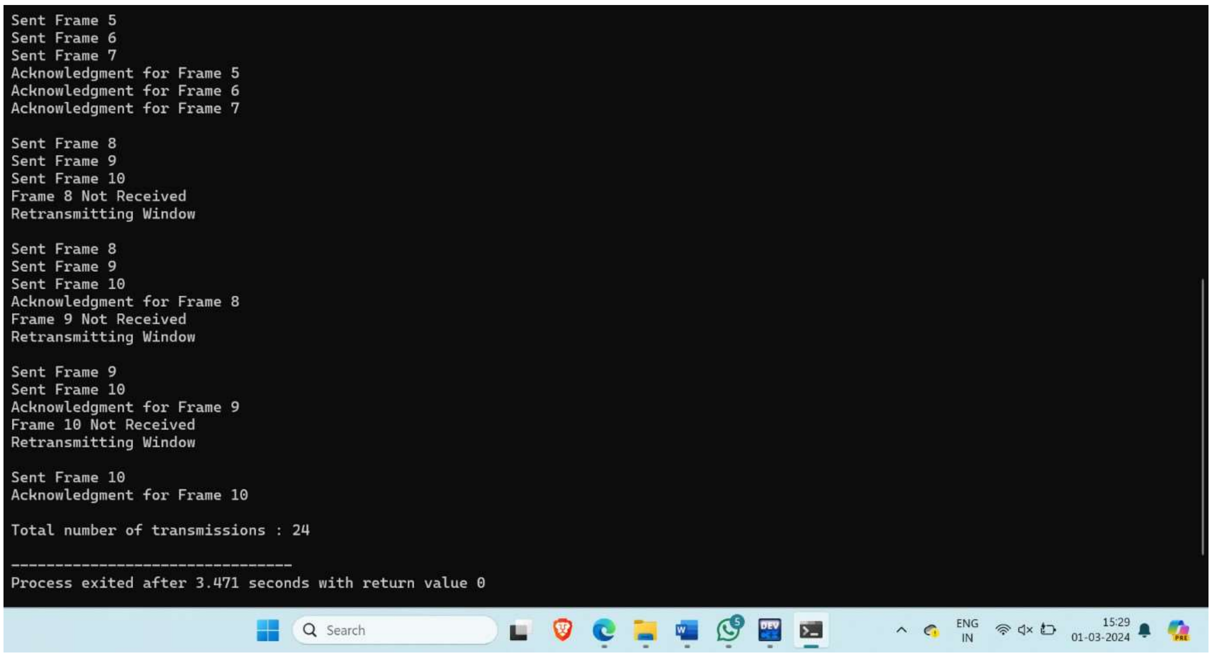
CODE:

```
#include <iostream>
#include <cstdlib>
#include <ctime> #include <unistd.h>
using namespace std;

class Packet { public:
    int sequenceNumber;

    Packet(int seq) : sequenceNumber(seq) {}
};


class Sender { public:     void send(Packet packet) {
        cout << "Sender: Sending packet with sequence number " <<
packet.sequenceNumber
<<endl;
        sleep(1);
    }

    bool waitForAck(int expectedSeqNum) {         sleep(1);
        bool ackReceived = (rand() % 2 == 0);
        if (ackReceived) {
            cout << "Sender: Acknowledgment for sequence number " <<
expectedSeqNum << " received." <<endl;
        } else {
            cout << "Sender: Acknowledgment for sequence number " <<
expectedSeqNum << " not received. Retransmitting..." <<endl;
        }
        return ackReceived;
    }
};

class Receiver { public:     void receive(Packet packet) {
        cout << "Receiver: Received packet with sequence number " <<
packet.sequenceNumber << endl;
        sleep(1);
    }

    Packet generateAck(int seqNumber) {
        cout << "Receiver: Generating acknowledgment for sequence
number " << seqNumber << endl;
        return Packet(seqNumber);
    }
};

int main() {     srand(time(NULL));

    Sender sender;
    Receiver receiver;

    int sequenceNumber = 0;     int totalPackets = 5;

    while (sequenceNumber < totalPackets) {         Packet
packet(sequenceNumber);
        sender.send(packet);

        if (sender.waitForAck(sequenceNumber)) {
            ++sequenceNumber;
        }
    }

    return 0;
}
```

OUTPUT:



```
Enter the number of frames : 10
Enter the Window Size : 3
Sent Frame 1
Sent Frame 2
Sent Frame 3
Frame 1 Not Received
Retransmitting Window

Sent Frame 1
Sent Frame 2
Sent Frame 3
Acknowledgment for Frame 1
Acknowledgment for Frame 2
Frame 3 Not Received
Retransmitting Window

Sent Frame 3
Sent Frame 4
Sent Frame 5
Frame 3 Not Received
Retransmitting Window

Sent Frame 3
Sent Frame 4
Sent Frame 5
Acknowledgment for Frame 3
Acknowledgment for Frame 4
Frame 5 Not Received
Retransmitting Window

Sent Frame 5
Sent Frame 6
Sent Frame 7
```



```
Sent Frame 5
Sent Frame 6
Sent Frame 7
Acknowledgment for Frame 5
Acknowledgment for Frame 6
Acknowledgment for Frame 7

Sent Frame 8
Sent Frame 9
Sent Frame 10
Frame 8 Not Received
Retransmitting Window

Sent Frame 8
Sent Frame 9
Sent Frame 10
Acknowledgment for Frame 8
Frame 9 Not Received
Retransmitting Window

Sent Frame 9
Sent Frame 10
Acknowledgment for Frame 9
Frame 10 Not Received
Retransmitting Window

Sent Frame 10
Acknowledgment for Frame 10

Total number of transmissions : 24

--------------------------------
Process exited after 3.471 seconds with return value 0
```

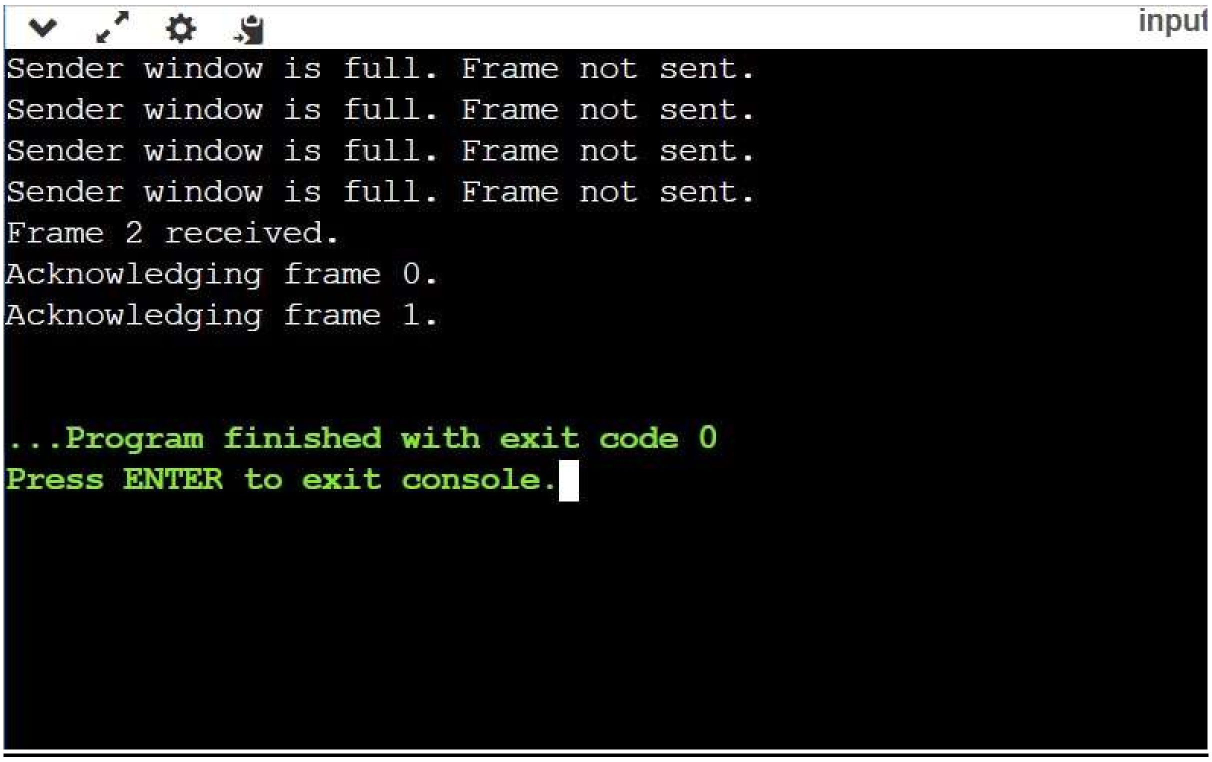
EXPERIMENT-6

Write a program to implement GO Back-N Protocol for noisy channel.

CODE:

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;
int main() {
     int nf,N;
     int no_tr=0;
     srand(time(NULL));
     cout<<"Enter the number of frames : ";
     cin>>nf;
     cout<<"Enter the Window Size : ";
     cin>>N;
     int i=1;
     while(i<=nf)
     {
     int x=0;
     for(int j=i;j<i+N && j<=nf;j++)
     {
         cout<<"Sent Frame "<<j<<endl;
         no_tr++;
     }
     for(int j=i;j<i+N && j<=nf;j++)
     {
         int flag = rand()%2;
         if(!flag)
             {
                 cout<<"Acknowledgment for Frame "<<j<<endl;
                  x++;
             }else
             {   cout<<"Frame "<<j<<" Not Received"<<endl;
                 cout<<"Retransmitting Window"<<endl;
                 break;
             }
     }
     cout<<endl;
     i+=x;
 }
 cout<<"Total number of transmissions : "<<no_tr<<endl;  return 0;
}
```

OUTPUT:



```
Sender window is full. Frame not sent.
Sender window is full. Frame not sent.
Sender window is full. Frame not sent.
Sender window is full. Frame not sent.
Frame 2 received.
Acknowledging frame 0.
Acknowledging frame 1.


...Program finished with exit code 0
Press ENTER to exit console.
```

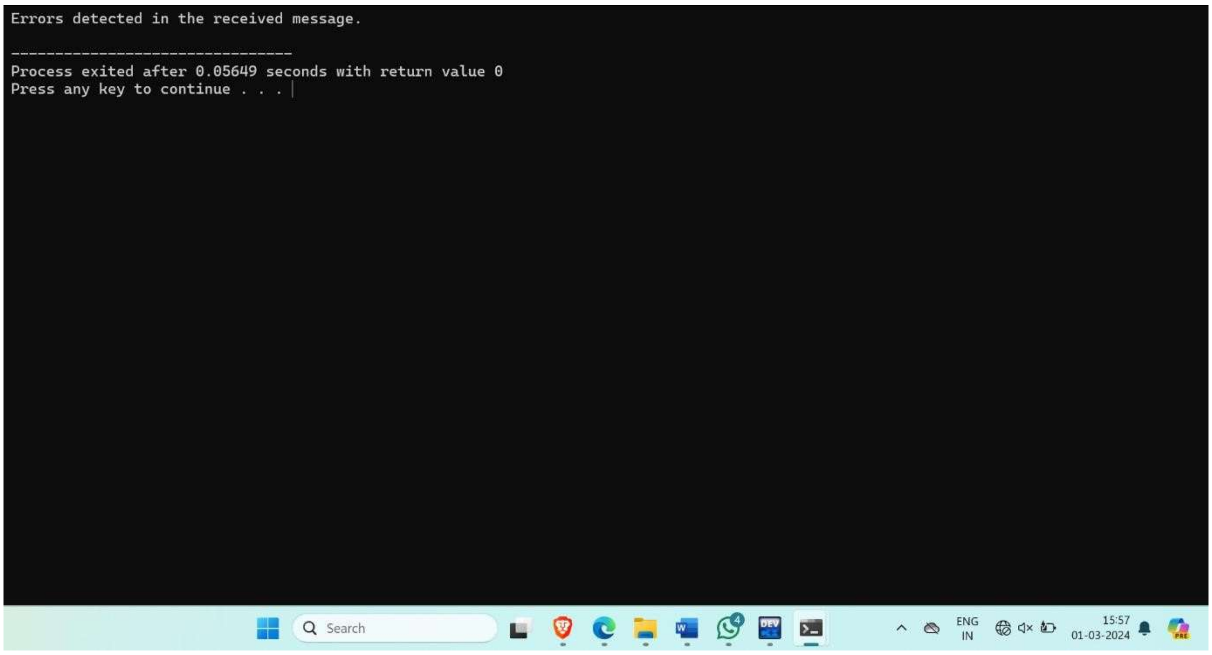
EXPERIMENT-7

Write a program to implement selective repeat ARQ for noisy channel.

CODE:

```
#include <iostream> #include <unordered_map> using namespace std; class
SelectiveRepeatARQ { public:
    SelectiveRepeatARQ(int windowSize) : windowSize(windowSize),
sendBase(0), nextSeqNum(0) {}

    void sendFrame(const std::string& frame) {         if (nextSeqNum <
sendBase + windowSize) {             receiverWindow[nextSeqNum] =
frame;
            ++nextSeqNum;
        } else {
            cout << "Sender window is full. Frame not sent." <<endl;
        }
    }

    void receiveFrame(int frameSeqNum) {         auto it =
receiverWindow.find(frameSeqNum);
        if (it != receiverWindow.end()) {
            std::cout << "Frame " << frameSeqNum << " received." <<
std::endl;             receiverWindow.erase(it);

            while (receiverWindow.find(sendBase) !=
receiverWindow.end()) {                 cout << "Acknowledging frame "
<< sendBase << "." <<endl;
receiverWindow.erase(sendBase);
                ++sendBase;
            }
        } else {
            cout << "Frame " << frameSeqNum << " not in sender window.
Discarding." << endl;
        }} private:
    int windowSize;     int sendBase;     int nextSeqNum;
    unordered_map<int, string> receiverWindow;
}; int main() {
    int windowSize = 4;
    SelectiveRepeatARQ srArq(windowSize);
    for (int i = 0; i < 8; ++i) {
        srArq.sendFrame("Frame " + std::to_string(i));}
srArq.receiveFrame(2);

    return 0;
}
```

OUTPUT:



```
Errors detected in the received message.

--------------------------------
Process exited after 0.05649 seconds with return value 0
Press any key to continue . . .
```

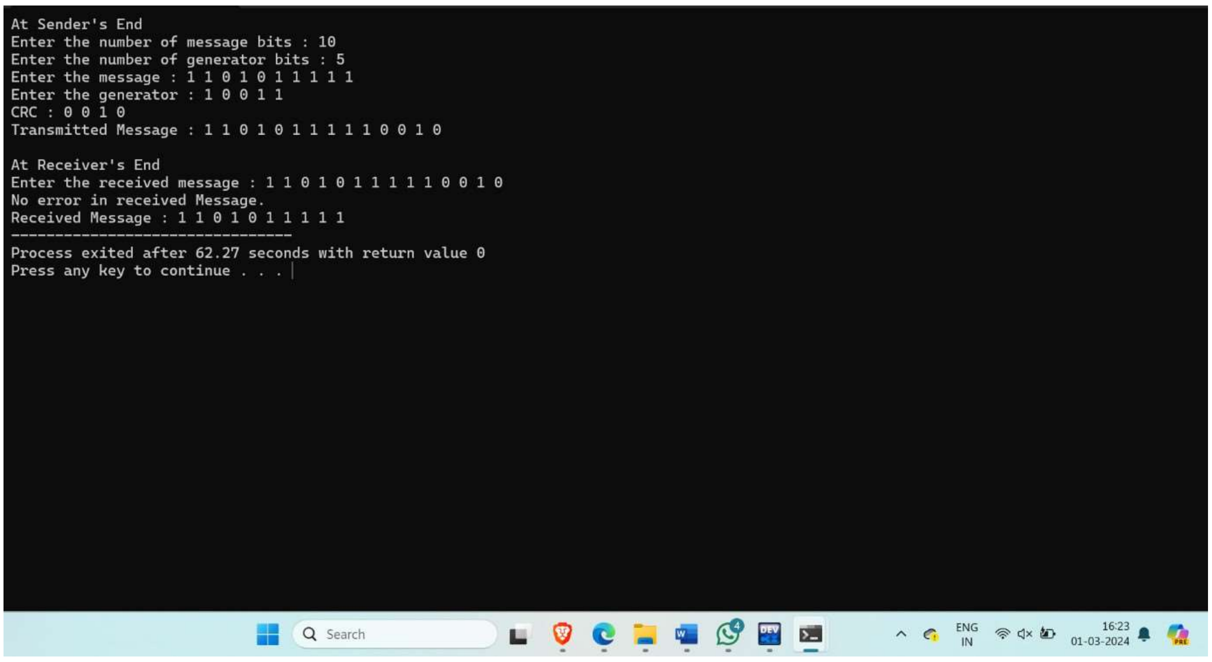
EXPERIMENT-8

Write a program to implement parity check using minimum hamming distance.

CODE:

```
#include <iostream>
#include <vector>
#include<string>

using namespace std;
int hammingDistance(string& s1,string& s2) {
    int distance = 0;
    for (int i = 0; i < s1.length(); ++i) {
        if (s1[i] != s2[i]) {
            distance++;
        }
    }
    return distance;
}
bool detectErrors(vector<string>& str1,string& str2) {
    int minDistance = str2.length();      for (int
i=0;i<str1.size();i++) {
        int distance = hammingDistance(str1[i], str2);         if
(distance < minDistance) {
            minDistance = distance;
        }
    }
    return minDistance > 0;
}

int main() {     vector<string> str1;
     str1.push_back("000");     str1.push_back("011");
str1.push_back("101");     str1.push_back("110");
    string str2 = "100";
    bool errorsDetected = detectErrors(str1, str2);     if
(errorsDetected) {
        cout << "Errors detected in the received message." << endl;
    } else {
        cout << "No errors detected in the received message." << endl;
    }

    return 0;
}
```

OUTPUT:



```
At Sender's End
Enter the number of message bits : 10
Enter the number of generator bits : 5
Enter the message : 1 1 0 1 0 1 1 1 1 1
Enter the generator : 1 0 0 1 1
CRC : 0 0 1 0
Transmitted Message : 1 1 0 1 0 1 1 1 1 1 0 0 1 0

At Receiver's End
Enter the received message : 1 1 0 1 0 1 1 1 1 1 0 0 1 0
No error in received Message.
Received Message : 1 1 0 1 0 1 1 1 1 1
--------------------------------
Process exited after 62.27 seconds with return value 0
Press any key to continue . . .
```

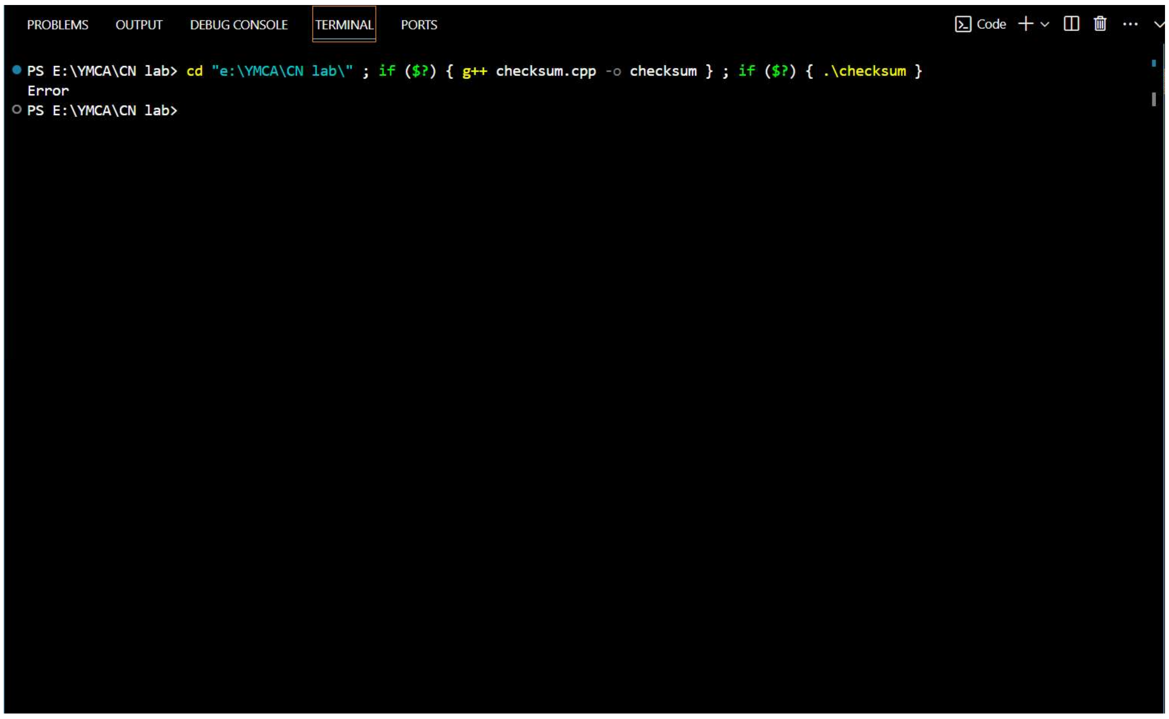
EXPERIMENT – 9

Write a program to implement cyclic redundancy code for error detection.

CODE:

```
#include<iostream> using namespace std;
void division(int temp[],int gen[],int n,int r)
{
for(int i=0;i<n;i++)
{
if (gen[0]==temp[i])
{
for(int j=0,k=i;j<r+1;j++,k++) if(!(temp[k]^gen[j]))
temp[k]=0; else
temp[k]=1;
} }}
int main()
{int n,r,message[50],gen[50],temp[50]; cout<<"At Sender's End "<<endl;
cout<<"Enter the number of message bits : "; cin>>n;
cout<<"Enter the number of generator bits : "; cin>>r;
cout<<"Enter the message : "; for(int i=0;i<n;i++)
cin>>message[i]; cout<<"Enter the generator : "; for(int i=0;i<r;i++)
cin>>gen[i]; r--;
for(int i=0;i<r;i++) message[n+i] = 0;
for(int i=0;i<n+r;i++) temp[i] = message[i];
division(temp,gen,n,r); cout<<"CRC : "; for(int i=0;i<r;i++)
{
cout<<temp[n+i]<<" "; message[n+i] = temp[n+i];
}
cout<<endl<<"Transmitted Message : "; for(int i=0;i<n+r;i++)
cout<<message[i]<<" "; cout<<endl<<endl<<"At Receiver's End "<<endl;
cout<<"Enter the received message : ";
for(int i=0;i<n+r;i++) cin>>message[i];

for(int i=0;i<n+r;i++) temp[i] = message[i];
division(temp,gen,n,r); for(int i=0;i<r;i++)
{
if(temp[n+i])
{
cout<<"Error detected in received message."; return 0;
} }
cout<<"No error in received Message.\nReceived Message : "; for(int
i=0;i<n;i++)
cout<<message[i]<<" "; return 0; }
```

OUTPUT:



```
PS E:\YMCA\CN lab> cd "e:\YMCA\CN lab\" ; if ($?) { g++ checksum.cpp -o checksum } ; if ($?) { .\checksum }
Error
PS E:\YMCA\CN lab>
```

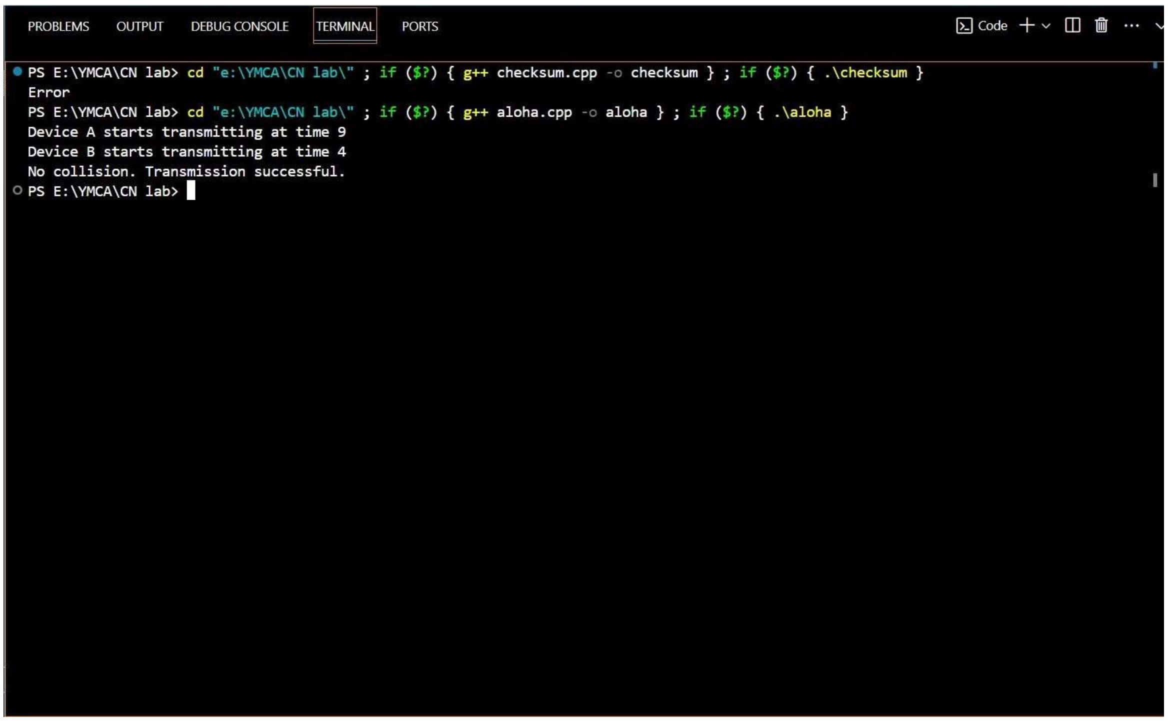
EXPERIMENT -10

Write a program to implement checksum.

CODE:

```
 #include <bits/stdc++.h> using namespace std;
string Ones_complement(string data) { for (int i = 0; i <
data.length(); i++) {
if (data[i] == '0')
data[i] = '1'; else
data[i] = '0';
}
return data;
}

string checkSum(string data, int block_size) { int n = data.length();
if (n % block_size != 0) {
int pad_size = block_size - (n % block_size); for (int i = 0; i <
pad_size; i++) {
data = '0' + data;
}
}
string result = "";
for (int i = 0; i < block_size; i++) { result += data[i];
}
for (int i = block_size; i < n; i += block_size) {

string next_block = "";
for (int j = i; j < i + block_size; j++) { next_block += data[j];
}
string additions = ""; int sum = 0, carry = 0;
for (int k = block_size - 1; k >= 0; k--) {
sum += (next_block[k] - '0') + (result[k] - '0'); carry = sum / 2;
if (sum == 0) {
additions = '0' + additions; sum = carry;
}
else if (sum == 1) { additions = '1' + additions; sum = carry;
}
else if (sum == 2) { additions = '0' + additions; sum = carry;
}
else {
additions = '1' + additions; sum = carry;
}
}
string final = ""; if (carry == 1) {
for (int l = additions.length() - 1; l >= 0; l--) {

if (carry == 0) {
final = additions[l] + final;
}
else if (((additions[l] - '0') + carry) % 2 == 0) { final = "0" +
final;
carry = 1;
}
else {
final = "1" + final; carry = 0;
}
}
result = final;
}
else {
result = additions;
}
}
return Ones_complement(result);
}


bool checker(string sent_message, string rec_message, int block_size) {
string sender_checksum = checkSum(sent_message, block_size);
string receiver_checksum = checkSum(rec_message + sender_checksum,
block_size); if (count(receiver_checksum.begin(),
receiver_checksum.end(), '0') == block_size) {
return true;
}
else {
return false;

}
}


int main() {
string sent_message = "10000101011000111001010011101101"; string
recv_message = "10000101011000111001010011101101"; int block_size = 8;
if (checker(sent_message, recv_message, block_size)) { cout << "No
Error";
}
else {
cout << "Error";
}
return 0;
}
```

OUTPUT:



```
PS E:\YMCA\CN lab> cd "e:\YMCA\CN lab\" ; if ($?) { g++ checksum.cpp -o checksum } ; if ($?) { .\checksum }
Error
PS E:\YMCA\CN lab> cd "e:\YMCA\CN lab\" ; if ($?) { g++ aloha.cpp -o aloha } ; if ($?) { .\aloha }
Device A starts transmitting at time 9
Device B starts transmitting at time 4
No collision. Transmission successful.
PS E:\YMCA\CN lab>
```

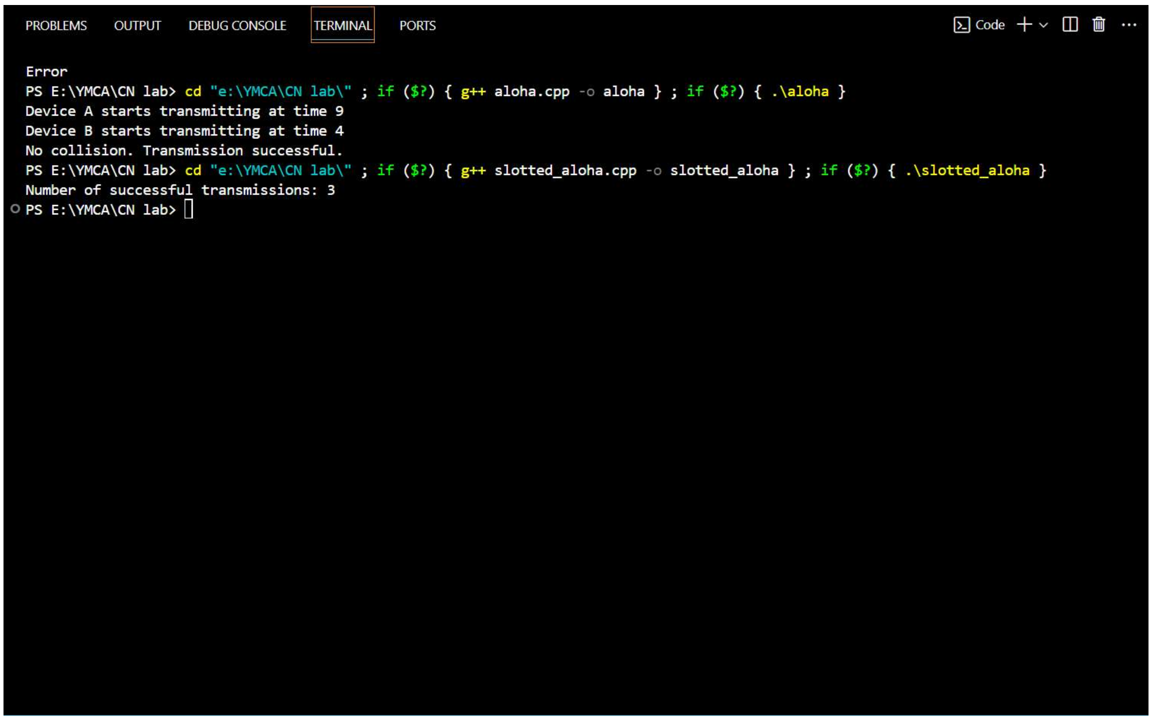
EXPERIMENT - 11:

Write a program to implement pure aloha.

CODE:

```
 #include <iostream> #include <cstdlib> #include <ctime>

using namespace std;


bool isCollision(int timeA, int timeB, int tp) { if (timeA < timeB &&
timeB < timeA + tp) {
return true;
}
if (timeB < timeA && timeA < timeB + tp) { return true;
}
return false;
}


int main() { srand(time(0));

int tp = 1;
int backoffTime = 0;


int timeA = rand() % 10; int timeB = rand() % 10;

cout << "Device A starts transmitting at time " << timeA << endl;

cout << "Device B starts transmitting at time " << timeB << endl;


if (isCollision(timeA, timeB, tp)) {
cout << "Collision detected!" << endl; backoffTime = rand() % (2 * tp +
1);
cout << "Device A waits for " << backoffTime << " time units before
retransmitting."
<< endl;
cout << "Device B waits for " << backoffTime << " time units before
retransmitting."
<< endl;
} else {
cout << "No collision. Transmission successful." << endl;
}


return 0;
}
```

OUTPUT:



```
Error
PS E:\YMCA\CN lab> cd "e:\YMCA\CN lab\" ; if ($?) { g++ aloha.cpp -o aloha } ; if ($?) { .\aloha }
Device A starts transmitting at time 9
Device B starts transmitting at time 4
No collision. Transmission successful.
PS E:\YMCA\CN lab> cd "e:\YMCA\CN lab\" ; if ($?) { g++ slotted_aloha.cpp -o slotted_aloha } ; if ($?) { .\slotted_aloha }
Number of successful transmissions: 3
PS E:\YMCA\CN lab>
```

EXPERIMENT - 12

Write a program to implement slotted aloha.

CODE:

```
#include <iostream> #include <vector> #include <cstdlib> #include
<ctime>

using namespace std;


bool simulateSlot(int numNodes, double p) { vector<bool>
transmissionAttempts(numNodes, false); int successfulTransmissions = 0;

for (int i = 0; i < numNodes; ++i) { if (rand() < p * RAND_MAX) {
transmissionAttempts[i] = true;
}
}


for (int i = 0; i < numNodes; ++i) { if (transmissionAttempts[i]) {
successfulTransmissions++;
}
}


if (successfulTransmissions > 1) { return false;
}

return true;
}


int main() { srand(time(0));

int numNodes = 3; double p = 0.5;
int numSlots = 10;


int successfulTransmissions = 0;


for (int i = 0; i < numSlots; ++i) { if (simulateSlot(numNodes, p)) {
successfulTransmissions++;
}
}


cout << "Number of successful transmissions: " <<
successfulTransmissions << endl;
return 0;
}
```

EXPERIMENT-13

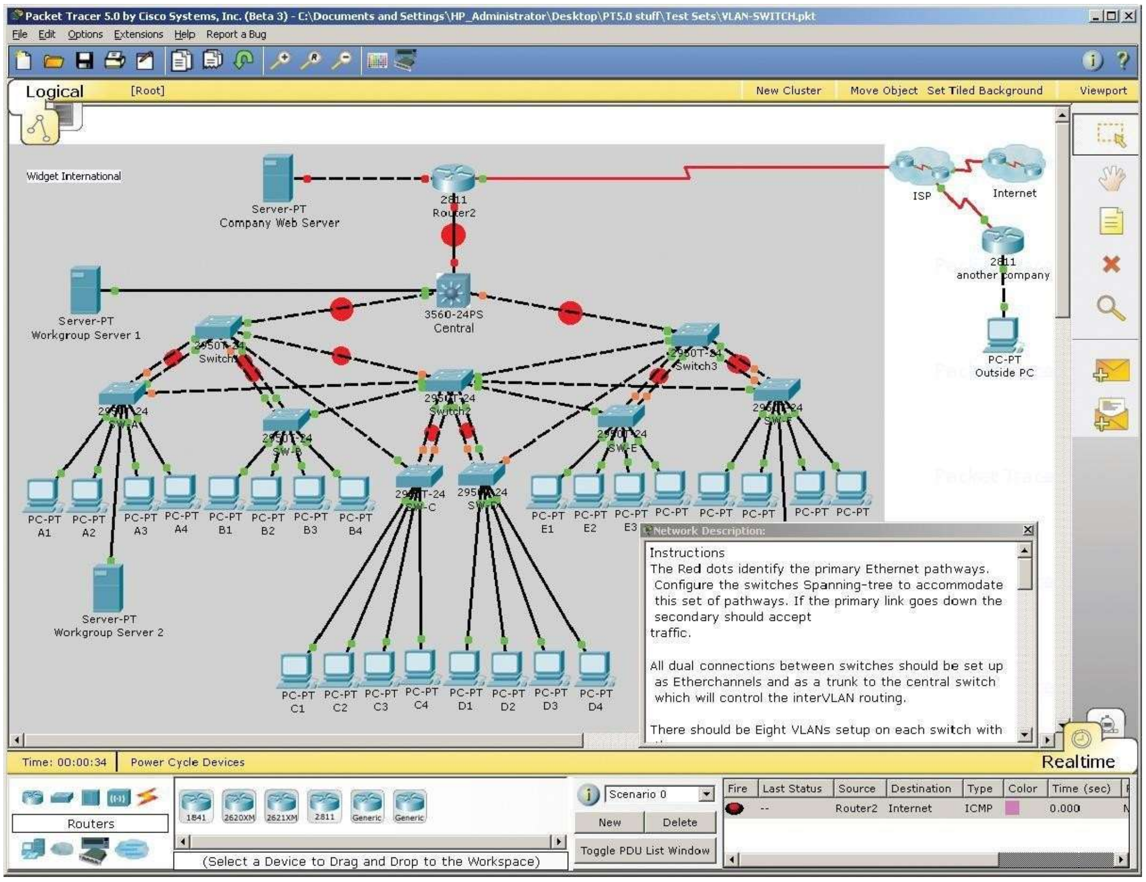
Aim: Introduction to CISCO Packet Tracer

Theory:

Cisco Packet Tracer is a robust network simulation tool integral to the Networking Academy's curriculum. It facilitates comprehensive learning through simulation, visualization, authoring, assessment, and collaboration capabilities. Packet Tracer 5.0, particularly used in CCNA Discovery and Exploration, offers powerful features for teaching and learning complex networking concepts. It allows for activity creation, customization, and collaboration, creating virtual networking environments for effective learning. With embedded activities, it provides an interactive environment for mastering CCNA-level networking concepts and protocols, extending the classroom into a virtual realm for exploration and experimentation.

Working:

Packet Tracer offers instructors various ways to demonstrate networking concepts. While not a replacement for real equipment, it allows students to practice using a model of the Cisco IOS command line interface and offers visual, drag-and-drop problem-solving with virtual networking devices. This hands-on approach is essential for learning router and switch configuration from the command line, aiding in understanding system design. Instructors can also create self-evaluated activities providing immediate feedback to students on assignment proficiency.


Packet Tracer 5.0 by Cisco Systems, Inc. (Beta 3) - C:\Documents and Settings\HP_Administrator\Desktop\PT5.0 stuff\Test Sets\VLAN-SWITCH.pkt
File Edit Options Extensions Help Report a Bug
Logical [Root] New Cluster Move Object Set Tiled Background Viewport
Widget International
Server-PT Company Web Server
2811 Router2
ISP
Internet
2811 another company
PC-PT Outside PC
Server-PT Workgroup Server 1
3560-24PS Central
2950T-24 Switch1
2950T-24 Switch2
2950T-24 Switch3
2950-24 SW-A
2950-24 SW-B
2950T-24 SW-C
2950-24 SW-D
2950T-24 SW-E
2950-24 SW-F
PC-PT A1
PC-PT A2
PC-PT A3
PC-PT A4
PC-PT B1
PC-PT B2
PC-PT B3
PC-PT B4
PC-PT C1
PC-PT C2
PC-PT C3
PC-PT C4
PC-PT D1
PC-PT D2
PC-PT D3
PC-PT D4
PC-PT E1
PC-PT E2
PC-PT E3
Server-PT Workgroup Server 2
Time: 00:00:34 Power Cycle Devices
Realtime
Routers
1841 2620XM 2621XM 2811 Generic Generic
(Select a Device to Drag and Drop to the Workspace)
Scenario 0
New Delete
Toggle PDU List Window

Network Description:

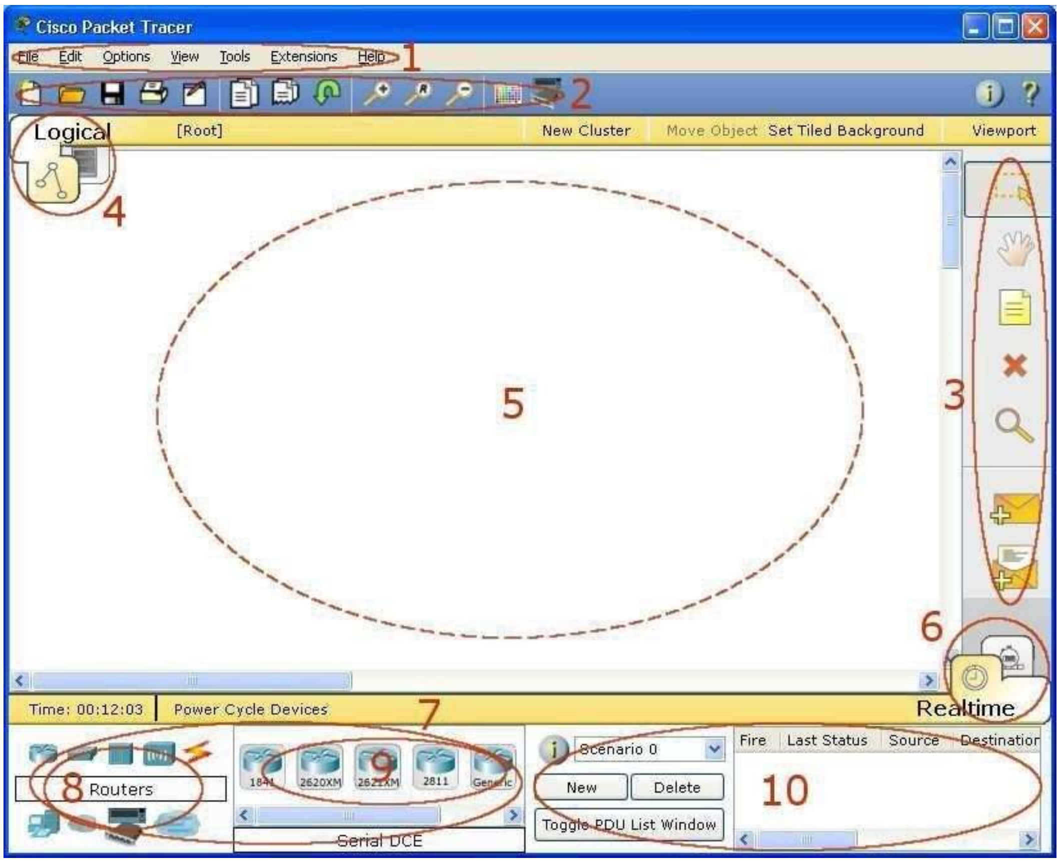
Instructions
The Red dots identify the primary Ethernet pathways. Configure the switches Spanning-tree to accommodate this set of pathways. If the primary link goes down the secondary should accept traffic.

All dual connections between switches should be set up as Etherchannels and as a trunk to the central switch which will control the interVLAN routing.

There should be Eight VLANs setup on each switch with

Fire	Last Status	Source	Destination	Type	Color	Time (sec)
●	--	Router2	Internet	ICMP	🟪	0.000

Overview:



1. Menu Bar: Provides access to basic commands like Open, Save, and Print, as well as the Activity Wizard.

2. Main Tool Bar: Offers shortcuts to File and Edit menu commands, along with buttons for Zoom, Palette, and Network Information.

3. Common Tools Bar: Provides commonly used workspace tools like Select, Move, and Delete.

4. Workspace and Navigation Bar: Allows toggling between Physical and Logical Workspaces, navigating through levels, and creating new elements.

5. Workspace: Area for creating networks, watching simulations, and viewing information.

6. Realtime/Simulation Bar: Toggles between Realtime and Simulation Modes, with buttons for device control and a clock displaying time.

7. Network Component Box: Contains device and connection options for the workspace.

8. Device-Type Selection Box: Offers device and connection types available in Packet Tracer 5.1.

9. Device-Specific Selection Box: Allows choosing specific devices and connections for the network.

10. User Created Packet Window: Manages packets during simulation scenarios.

Key Features:

1. Workspaces: Packet Tracer offers logical and physical workspaces for building network topologies. The logical workspace allows placement and connection of virtual network devices, while the physical workspace provides a realistic representation of network devices in a physical environment.

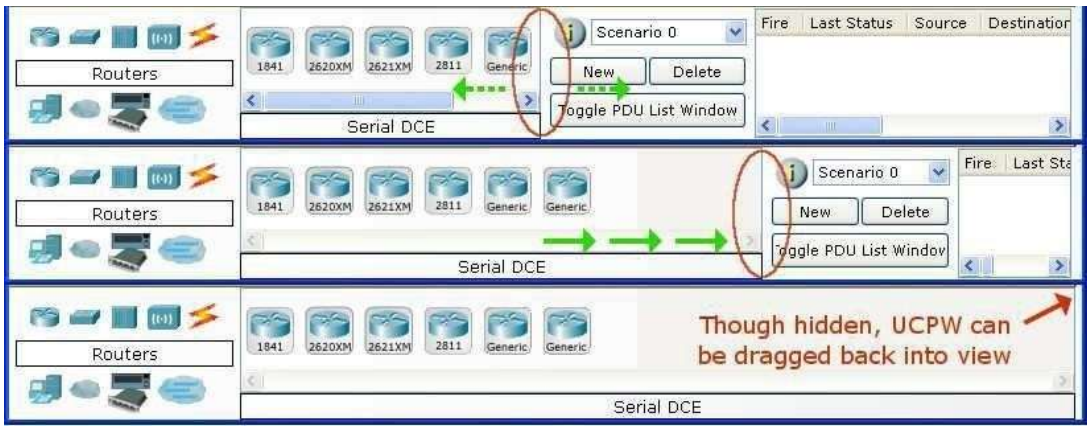
2. Modes: Packet Tracer includes real-time and simulation modes. In real-time mode, the network behaves as real devices would, offering immediate responses. Simulation mode allows users to control time intervals and observe data transfer processes, aiding in understanding network operations.

EXPERIMENT-14

Aim: Creating a basic client server topology and checking the connection.

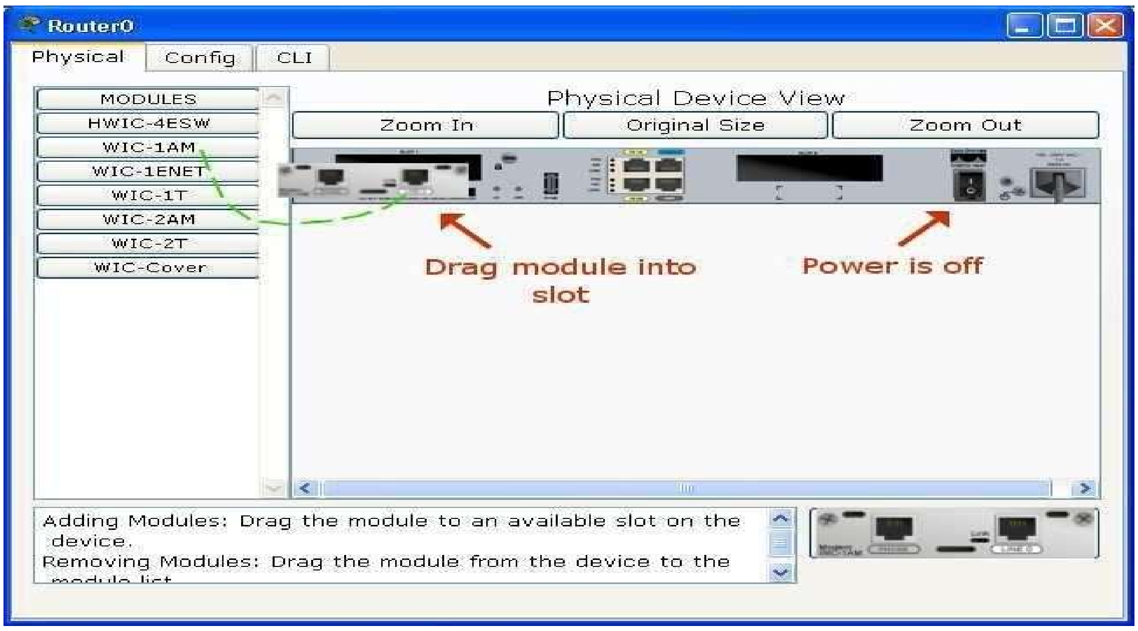
Theory:

- *Creating Devices*



a. Choose a device type from the Device Type Selection box.
b. Click on the desired device model from the Device-Specific Selection box
c. Click on a location in the workspace to put your device in that location
d. If you want to cancel your selection, press the Cancel icon for that device
e. Alternatively, you can click and drag a device from the Device-Specific Selection box onto the workspace
f. You can also click and drag a device directly from the Device-Type Selection box and a default device model will be chosen for you.

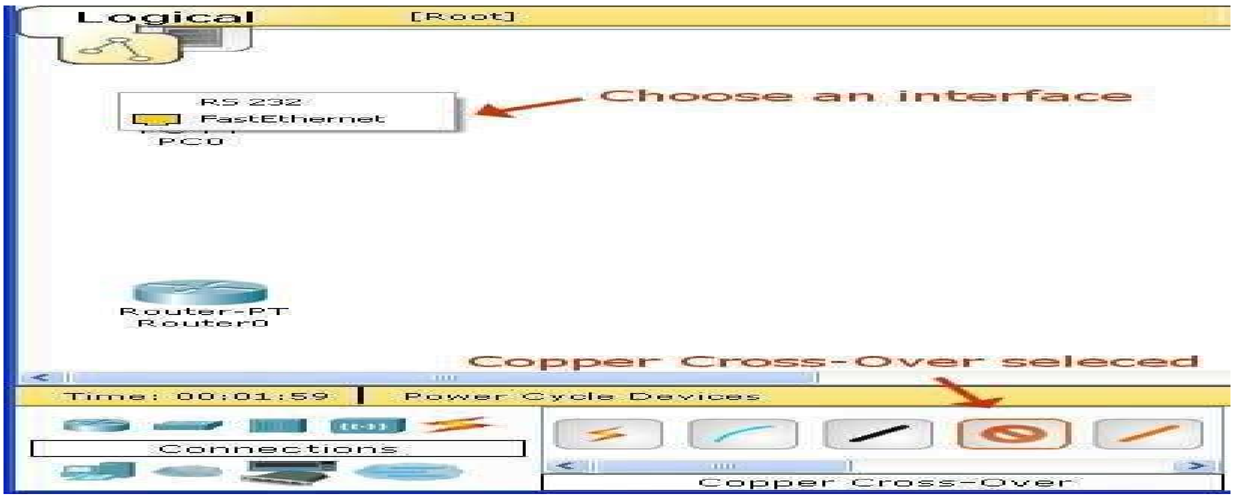
- *Adding Modules*



Adding Modules: Drag the module to an available slot on the device.
Removing Modules: Drag the module from the device to the module list

a. Click on a device to bring up its configuration window.
b. By default, you will be in the Physical Device View subpanel of the device.
c. You can browse (by clicking) through the list of modules and read their description in
a. the information box at the bottom.
d. When you have found the module, you want to add, simply drag it from the list into a compatible bay on the device picture.
e. You can remove a module by dragging it from the device back into the list.

- *Making Connections*



a. To make a connection between two devices, first click the Connections icon from the Device-Type Selection box to bring up the list of available connections.
b. Then click the appropriate cable type.
c. The mouse pointer will change into a "connection" cursor.
d. Click on the first device and choose an appropriate interface to which to connect.
e. Then click on the second device and do the same.
f. A connection cable will appear between the two devices, along with link lights showing the link status on each end (for interfaces that have link lights).

Procedure:

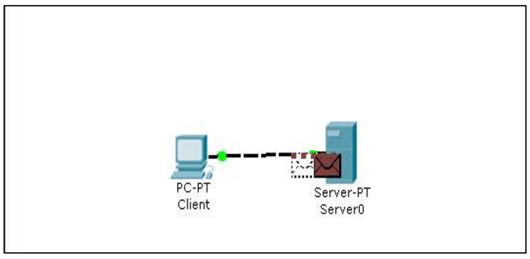
1. Select "End Devices" and choose a PC-PT to act as the client, placing it on the logical workspace.
2. Select "End Devices" again and choose a Server-PT, placing it on the logical workspace.
3. Click on "Connection," select "Copper Cross Over," and drop it onto one of the end devices. Choose "Fast Ethernet" from the options to establish a connection with green lights visible between them.
4. Click on the PC in the logical workspace and navigate to the Config tab.
5. Modify the display name to "Client."
6. Enter the IP address 192.168.0.105 in the DNS Server field.
7. Select "Fast Ethernet" from the left-hand panel.
8. Ensure the port status is checked.
9. Enter the IP address 192.168.0.110 in the IP address field; the subnet mask will be automatically filled.
10. Click on the server and access the Config tab.

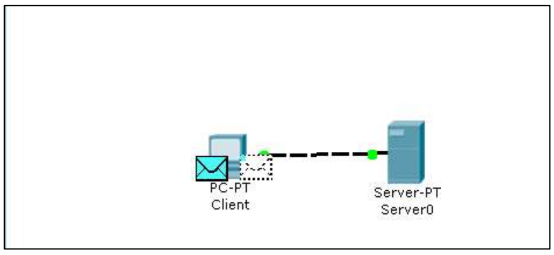
11. Enter 192.168.0.105 in the DNS Server field.
12. Select "Fast Ethernet" from the left-hand panel and fill the IP address field with 192.168.0.105.
13. Click on "DNS" from the left-hand panel and enter www.MyLab.com as the resource record name and 192.168.0.105 as the address. Press "ADD."
14. Return to the PC, click on the Desktop tab, open the web browser, and enter www.MyLab.com or the IP address 192.168.0.105. Press "Go" to access the server.

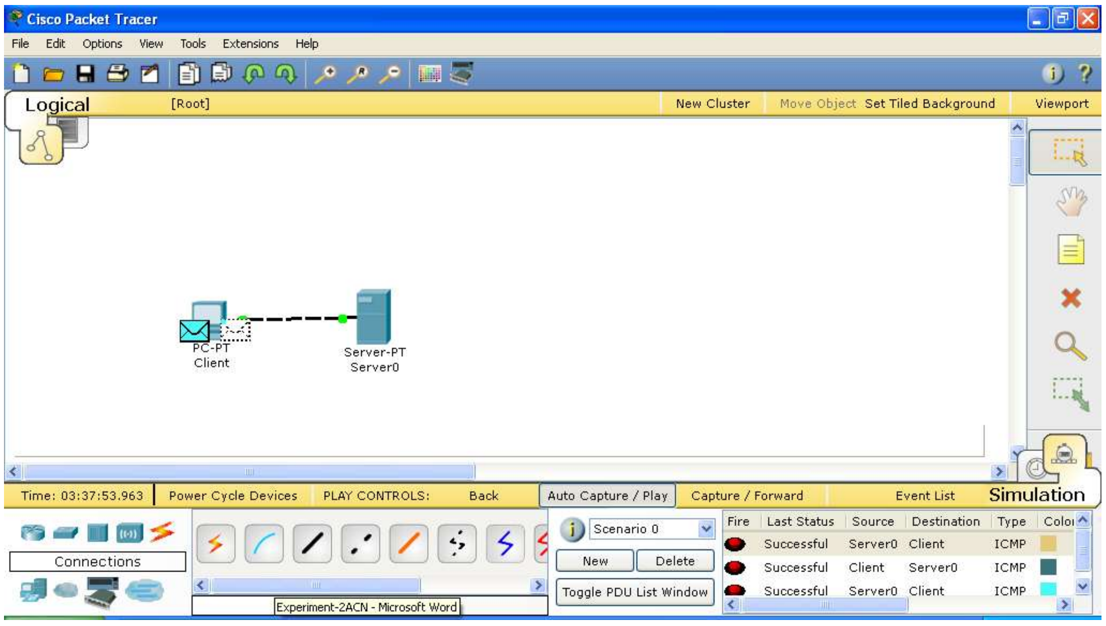
Output:

Switch to simulation mode by clicking on the simulation mode tab. Click on the "Auto Capture/Play" button and then press "Go" on the web browser to observe the packet exchange. Use the "Add Simple PDU (P)" option from the tools panel on the right-hand side to add a new packet. The event queue will be generated.

To proceed, click on "Capture/Forward." Press the "Edit filters" option in the simulation mode and select only ICMP. Further information on packets can be obtained by clicking on them.







Time: 03:37:53.963

Scenario 0

Fire	Last Status	Source	Destination	Type	Colo
●	Successful	Server0	Client	ICMP	■
●	Successful	Client	Server0	ICMP	■
●	Successful	Server0	Client	ICMP	■

EXPERIMENT-15

Aim: Study of basic network command and Network configuration commands.
Apparatus (Software): Command Prompt And Packet Tracer.

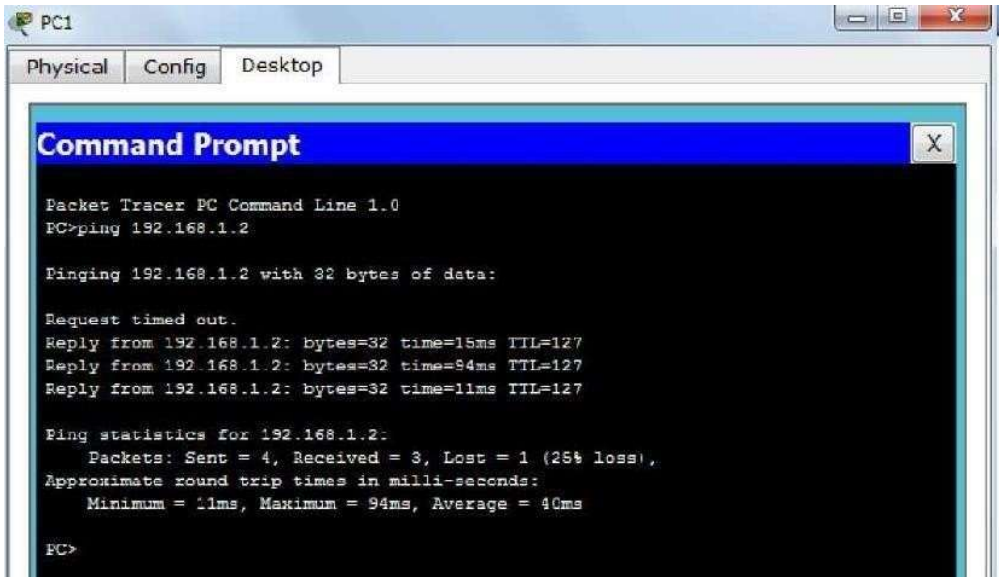
Procedure:
To do this EXPERIMENT- follows these steps:
In this EXPERIMENT- students must understand basic networking commands e.g ping,tracert etc.
All commands related to Network configuration which includes how to switch to privilege modeand normal mode and how to configure router interface and how to save this configuration to flash memory or permanent memory.
This command includes.

Configuring the router commands:

- General commands to configure the network
- Privileged mode commands of a router
- Router processes and statistics
- IP commands
- Other IP commands ex: show ip route etc.

PING:

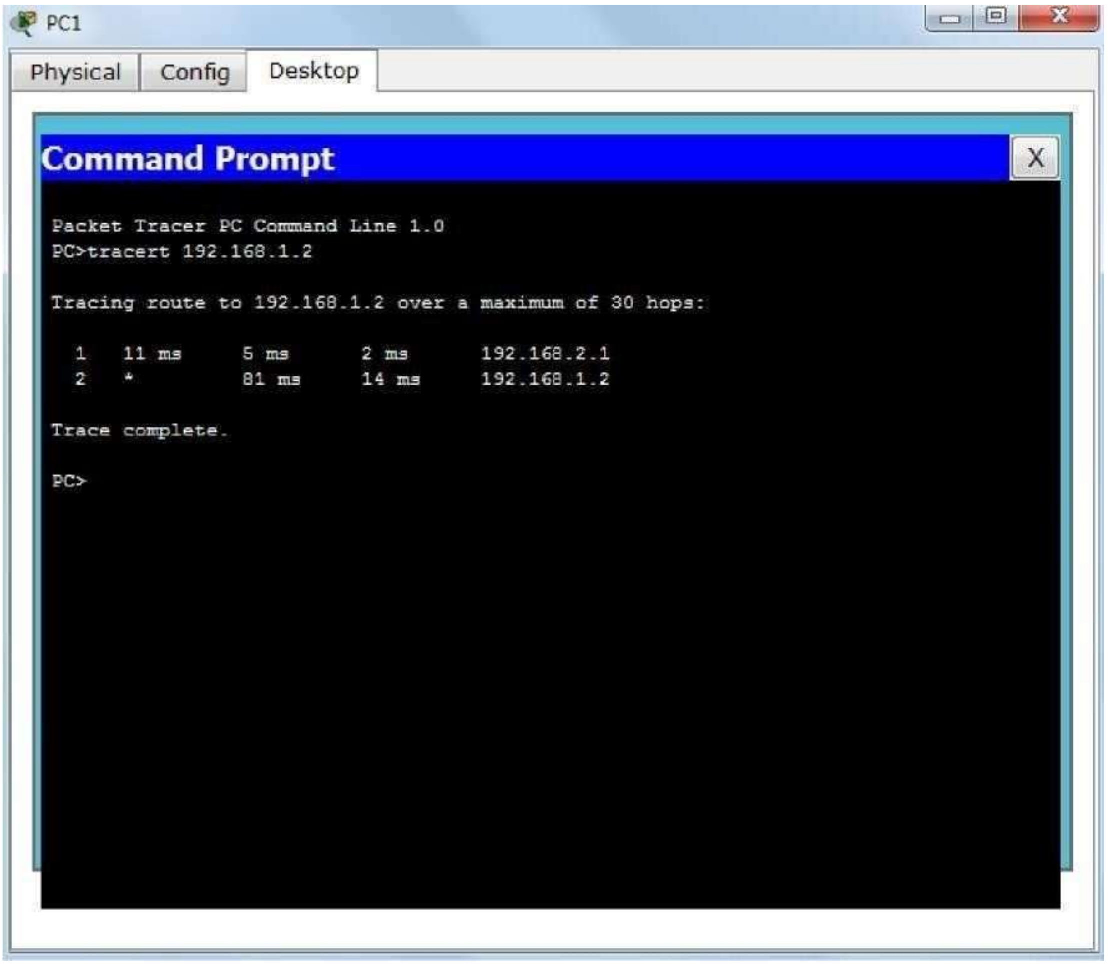
ping (8) sends an ICMP ECHO_REQUEST packet to the specified host. If the host responds, youget an ICMP packet back. Sound strange? Well, you can "ping" an IP address to see if a machineis alive. If there is no response, you know something is wrong.



```
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Request timed out.
Reply from 192.168.1.2: bytes=32 time=15ms TTL=127
Reply from 192.168.1.2: bytes=32 time=94ms TTL=127
Reply from 192.168.1.2: bytes=32 time=11ms TTL=127

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 11ms, Maximum = 94ms, Average = 40ms

PC>
```

TRACEROUTE:

Tracert is a command which can show you the path a packet of information takes from



```
Packet Tracer PC Command Line 1.0
PC>tracert 192.168.1.2

Tracing route to 192.168.1.2 over a maximum of 30 hops:

  1   11 ms     5 ms      2 ms      192.168.2.1
  2   *         81 ms     14 ms     192.168.1.2

Trace complete.

PC>
```

your computer to one you specify. It will list all the routers it passes through until it reaches its destination, or fails to and is discarded. In addition to this, it will tell you how long each 'hop' from router to router takes.

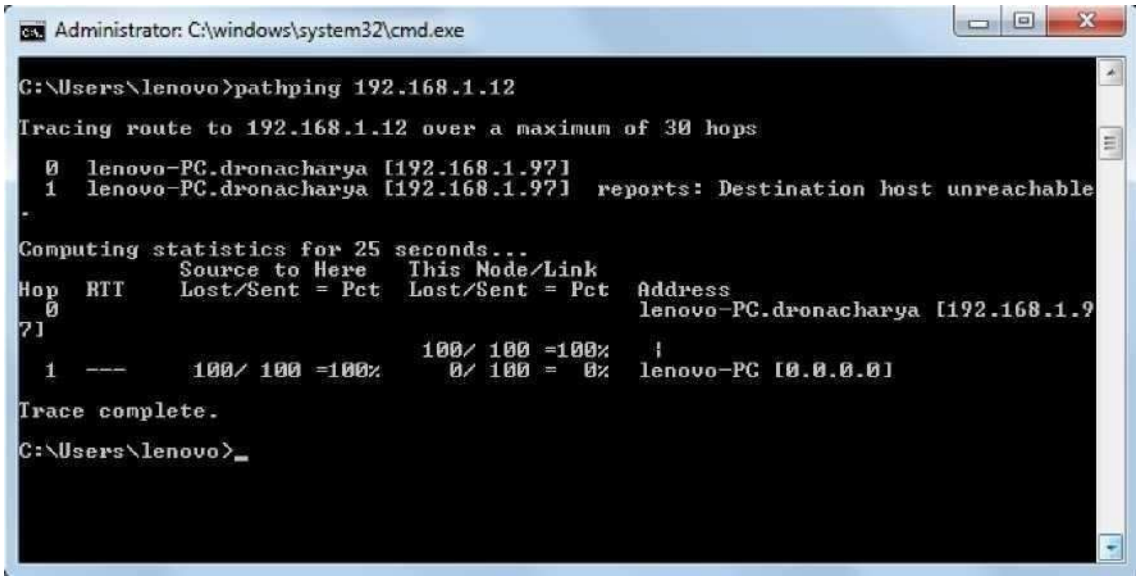
nslookup:

Displays information from Domain Name System (DNS) name servers.
NOTE :If you write the command as above it shows as default your pc's server

name firstly

pathping:

A better version of tracert that gives you statics about packet lost and latency.



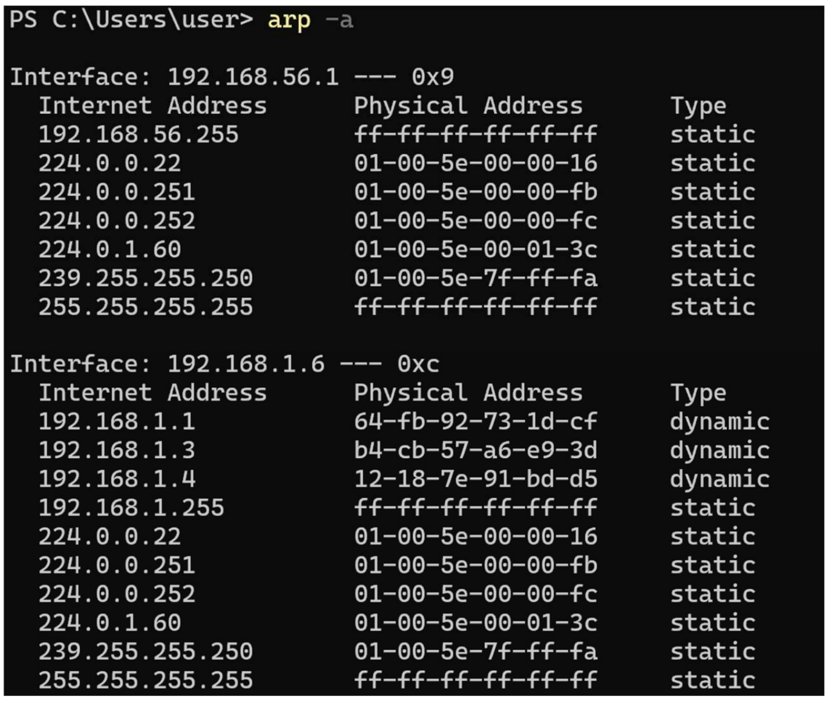
```
C:\Users\lenovo>pathping 192.168.1.12

Tracing route to 192.168.1.12 over a maximum of 30 hops

  0  lenovo-PC.dronacharya [192.168.1.97]
  1  lenovo-PC.dronacharya [192.168.1.97]  reports: Destination host unreachable
.

Computing statistics for 25 seconds...
            Source to Here   This Node/Link
Hop  RTT    Lost/Sent = Pct  Lost/Sent = Pct  Address
  0                                           lenovo-PC.dronacharya [192.168.1.9
7]
                                100/ 100 =100%   |
  1  ---     100/ 100 =100%       0/ 100 =  0%  lenovo-PC [0.0.0.0]

Trace complete.

C:\Users\lenovo>_
```

RARP (Reverse Address Resolution Protocol):

RARP is a network protocol used to map IP addresses to MAC addresses. It is the reverse of ARP (Address Resolution Protocol), which maps MAC addresses to IP addresses.

ARP (Address Resolution Protocol):

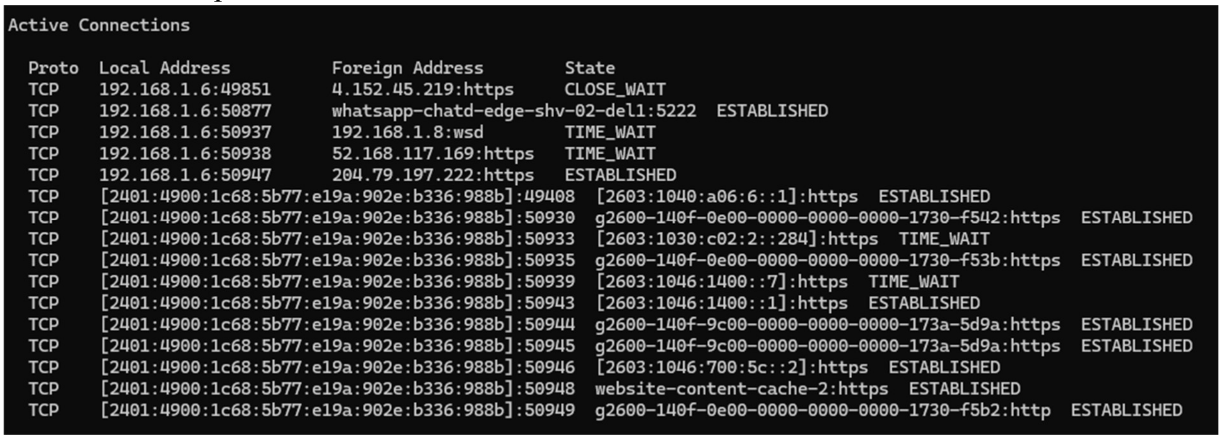
ARP is a protocol used to map a known IP address to a MAC address. It is commonly used by network devices to determine the hardware address associated with an IP address.



```
PS C:\Users\user> arp -a

Interface: 192.168.56.1 --- 0x9
  Internet Address      Physical Address      Type
  192.168.56.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.0.1.60            01-00-5e-00-01-3c     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.1.6 --- 0xc
  Internet Address      Physical Address      Type
  192.168.1.1           64-fb-92-73-1d-cf     dynamic
  192.168.1.3           b4-cb-57-a6-e9-3d     dynamic
  192.168.1.4           12-18-7e-91-bd-d5     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.0.1.60            01-00-5e-00-01-3c     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Netstat (Network Statistics):

Netstat is a command-line tool that displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. It can be used to troubleshoot network-related problems and monitor



```
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.1.6:49851      4.152.45.219:https     CLOSE_WAIT
  TCP    192.168.1.6:50877      whatsapp-chatd-edge-shv-02-del1:5222  ESTABLISHED
  TCP    192.168.1.6:50937      192.168.1.8:wsd        TIME_WAIT
  TCP    192.168.1.6:50938      52.168.117.169:https   TIME_WAIT
  TCP    192.168.1.6:50947      204.79.197.222:https   ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:49408  [2603:1040:a06:6::1]:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50930  g2600-140f-0e00-0000-0000-0000-1730-f542:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50933  [2603:1030:c02:2::284]:https  TIME_WAIT
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50935  g2600-140f-0e00-0000-0000-0000-1730-f53b:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50939  [2603:1046:1400::7]:https  TIME_WAIT
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50943  [2603:1046:1400::1]:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50944  g2600-140f-9c00-0000-0000-0000-173a-5d9a:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50945  g2600-140f-9c00-0000-0000-0000-173a-5d9a:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50946  [2603:1046:700:5c::2]:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50948  website-content-cache-2:https  ESTABLISHED
  TCP    [2401:4900:1c68:5b77:e19a:902e:b336:988b]:50949  g2600-140f-0e00-0000-0000-0000-1730-f5b2:http  ESTABLISHED
```