Computer Networks Lab Semester Project Submission 1

Group members -

2022BITE050

2022BITE052

2022BITE055

Programming Language Used: C++

1. Implementing Physical layer functionalities.

a) Create two end devices with a dedicated connection and enable data transmission between them. Note that this is only a simulation as no real data can be shared.

```cpp
#include<iostream>
#include<vector>
using namespace std;

class Device {
    static int no;
    int id;
    vector<Device*> connections;
    public :
    Device(){
        this->id = ++no;
        cout<<"Device "<<id<<" created\n";
    }
    int getId(){
        return this->id;
    }
    void establishConnection(Device* device){
```

```
        connections.push_back(device);
    }
    void sendData(Device* reciever,string data){
        cout<<"Sending message from device "<<this->id<<" device "<<reciever-
>getId()<<endl;
        reciever->receieveData(data);
    }
    void receieveData(string data){
        cout<<"Message recieved at device : "<<this->id<<" "<<data<<endl;
    }
};
int Device :: no = 0;

void testCase1(){
    Device *device1 = new Device();
    Device *device2 = new Device();
    device1->establishConnection(device2);
    device2->establishConnection(device1);
    string data = "hello world";
    device1->sendData(device2,data);
}

int main(){
    testCase1();
}
```

Output:

```
Device 1 created
Device 2 created
Sending message from device 1 device 2
Message recieved at device : 2 hello world
```

2) Create a star topology with five end devices connected to a hub and enable communication within end devices. This should follow exact working principles of hub.

```
#include<bits/stdc++.h>
using namespace std;
```

```
class Device;

class Hub {
    static int hNo;
    int id;
    vector<Device*> devices;
    public :
    Hub() {
        this->id = hNo++;
        this->devices = {};
        cout<<"Hub "<<id<<" created\n";
    }
    int getId(){
        return this->id;
    }
    void addDevice(Device* device);
    void broadCastData(Device* sender, Device* receiver, string data);
    void broadCastAck(Device* sender, Device* receiver,string ack);
};
int Hub :: hNo = 0;

class Device {
    static int devNo;
    int id;
    Hub* hub;
    public :
    Device(){
        this->id = devNo++;
        this->hub = NULL;
        cout<<"Device "<<id<<" created\n";
    }
    int getId(){
        return this->id;
    }
    void setHub(Hub* hub){
        this->hub = hub;
    }
    void sendMessage(Device* sender,Device* reciever,string data){
        cout<<"sending message from device "<<this->id<<" to hub "<<this->hub-
>getId()<<endl;
        this->hub->broadCastData(sender,reciever,data);
    }
    void receiveMessage(Device* sender,Device* reciever,string data){
        if(reciever->getId() != this->id){
```

```
            cout<<"Device "<<this->id<<" rejected message \n";
            return;
        }
        cout<<"Device "<<this->id<<" recieved message : "<<data<<endl;
        cout<<"Sending ack from device "<<this->id<<" to device "<<sender-
>getId()<<endl;
        this->sendAck(reciever,sender);
    }
    void receiveAck(Device* sender,Device* reciever,string data){
        if(reciever->getId() != this->id){
            cout<<"Device "<<this->id<<" rejected ack \n";
            return;
        }
        cout<<"Device "<<this->id<<" recieved ack : "<<data<<endl;
    }
    void sendAck(Device* sender,Device* reciever){
        this->hub->broadCastAck(sender,reciever,"Ack");
    }
};
int Device :: devNo = 0;

void Hub:: addDevice(Device* device){
    cout<<"Hub "<<this->id<<" connected to "<<" device "<<device->getId()<<endl;
    this->devices.push_back(device);
}

void Hub::broadCastData(Device* sender, Device* receiver, string data){
    cout<<"Hub "<<this->id<<" is broadcasting \n";
    for(int i = 0; i < devices.size(); i++){
        if(devices[i] != sender){
            devices[i]->receiveMessage(sender,receiver, data);
        }
    }
}

void Hub::broadCastAck(Device* sender, Device* receiver, string data){
    cout<<"Hub "<<this->id<<" is broadcasting \n";
    for(int i = 0; i < devices.size(); i++){
        if(devices[i] != sender){
            devices[i]->receiveAck(sender,receiver, data);
        }
    }
}
```

```
void testCase2() {
    Hub* hub1 = new Hub();
    vector<Device*> devices;
    for(int i=0;i<5;i++){
        Device* device = new Device();
        devices.push_back(device);
        hub1->addDevice(device);
        device->setHub(hub1);
    }
    //dev1 sends hello to dev4
    devices[1]->sendMessage(devices[1],devices[4],"Hello");
}

int main() {
    testCase2();
}
```

Output:

```
Hub 0 created
Device 0 created
Hub 0 connected to  device 0
Device 1 created
Hub 0 connected to  device 1
Device 2 created
Hub 0 connected to  device 2
Device 3 created
Hub 0 connected to  device 3
Device 4 created
Hub 0 connected to  device 4
sending message from device 1 to hub 0
Hub 0 is broadcasting
Device 0 rejected message
Device 2 rejected message
Device 3 rejected message
Device 4 recieved message : Hello
Sending ack from device 4 to device 1
Hub 0 is broadcasting
Device 0 rejected ack
Device 1 recieved ack : Ack
Device 2 rejected ack
Device 3 rejected ack
```

2. Implementing Data Link layer functionalities.

1) Switch and address learning

```
class Switch{
    //MAC to port no
    unordered_map<Device*, int> macTable;
    unordered_map<int,Hub*> hubs;
    unordered_map<int,vector<Device*>> portToMac;
    int ports;
    int maxPorts;

public:
    Switch(int ports){
        this->ports = 0;
        this->maxPorts = ports;
        cout<<"Switch created\n";
    }

    void learnMAC(Device* mac){
        if(ports == maxPorts){
            cout<<"cannot add more ports"<<endl;
            return;
        }
        macTable[mac] = ports;
        this->ports += 1;
    }

    void learnPortToMac(Device* mac,int port){
        portToMac[port].push_back(mac);
    }

    void addHub(Hub* hub){
        if(ports == maxPorts){
            cout<<"cannot add more ports"<<endl;
            return;
        }
        this->hubs[this->ports] = hub;
        this->ports += 1;
    }

    void forwardFrame(Frame frame){
        if(macTable.find(frame.destMAC) != macTable.end()){
            int outPort = macTable[frame.destMAC];
```

```
            cout<<"Switch: Forwarding frame from "<<frame.srcMAC<<" to
"<<frame.destMAC<<" via Port "<<outPort<<endl;
        }else{
            cout<<"destination not found\n";
        }
    }

    void forwardFrameToHub(Device* sender,Device* reciever,string data,bool
isAck){
        for(pair<int,vector<Device*>> it : portToMac){
            int port = it.first;
            vector<Device*> devices = it.second;
            for(auto device : devices){
                if(device == reciever){
                    cout<<"Forwarding data from switch via port "<<port<<endl;
                    if(!isAck) this->hubs[port]-
>broadCastData(sender,reciever,data,false);
                    else this->hubs[port]-
>broadCastAck(sender,reciever,data,false);
                }
            }
        }
    }
};
```

2) Implement at least one error control protocol

```
bool parityCheck(Frame frame){
    int count = 0;
    for(int i=0;i<frame.data.length();i++){
        char ch = frame.data[i];
        if(ch == '1') count++;
    }
    return count % 2 == 0;
}
```

3) Implement at least one access control protocol

```
bool csma_cd(){
    srand(time(0));
    int k = 0;
    cout<<"Detecting transmission medium..."<<endl;
```

```
    while(k <= 15){
        int transmissionChance = rand() % 100;
        if(transmissionChance < 70){
            cout<<"CSMA/CD: Successful transmission.\n";
            return true;
        }else{
            cout<<"CSMA/CD: Collision detected! Retrying... k = "<<++k<<endl;
            int waitTime = rand() % int(pow(2,k)-1);
            cout<<"Waiting for "<<waitTime<<" units\n";
        }
    }
    cout<<"Aborted\n";
    return false;
}
```

4) Implementing at least one sliding window-based ow control protocol

```
void goBackN(vector<Frame>& frames, int windowSize) {
    srand(time(0));
    int i = 0;
    while(i < frames.size()){
        int framesToSend = min(windowSize, (int)frames.size() - i);
        cout<<"Sliding Window: Sending "<< framesToSend<<" frames...\n";
        int j = i;
        while(j < i + framesToSend){
            int errorChance = rand() % 100;
            if(errorChance < 80){
                cout<<"Frame "<<j<<" Sent: "<<frames[j].data<<endl;
                if(parityCheck(frames[j])){
                    cout<<"No error detected\n";
                }else{
                    cout<<"Error detected retransmitting...\n";
                    cout<<"Sliding Window: Resending "<<framesToSend<<"
frames...\n";
                    j = i;
                    continue;
                }
                j++;
            }else{
                cout<<"Error detected retransmitting...\n";
                cout<<"Sliding Window: Resending "<<framesToSend<<" frames...\n";
                j = i;
                continue;
            }
```

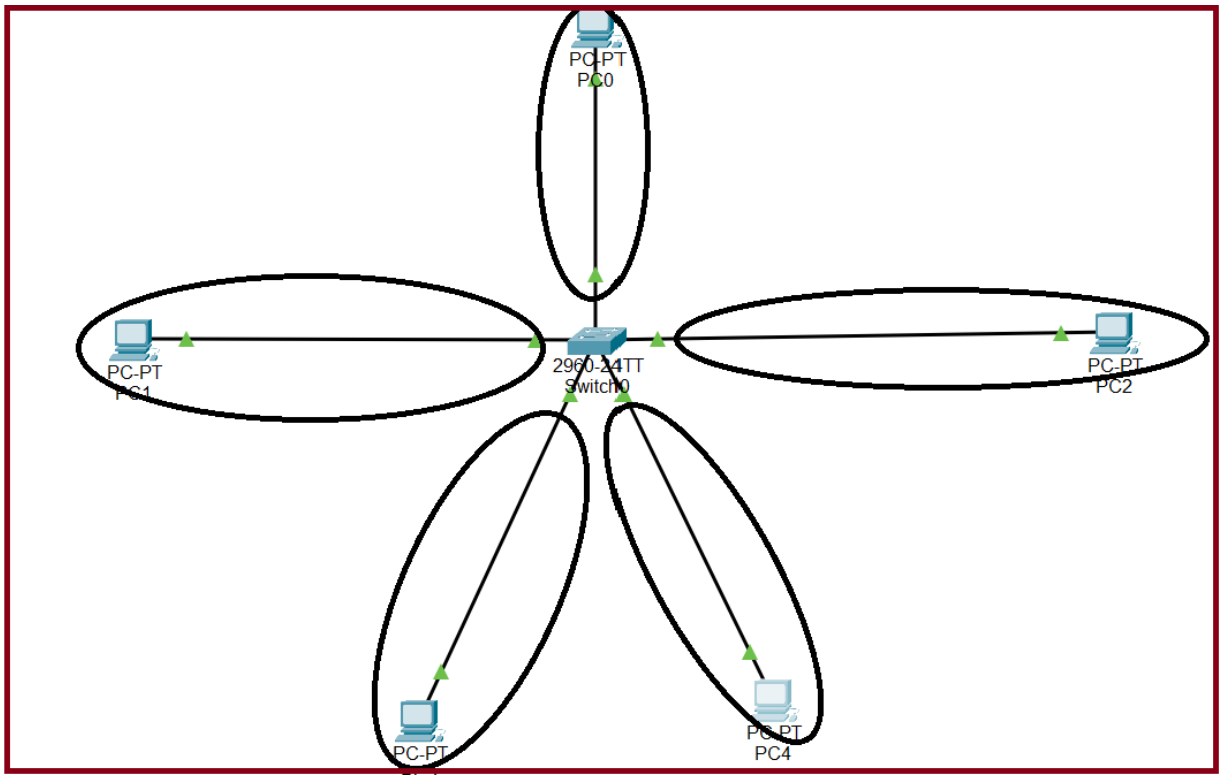
```
        }
        i += windowSize;
    }
}
```

5) Create a switch with five end devices connected to it and enable data transmission between them. Demonstrate access and each of the ow control protocols. Also, report the total number of broadcast and collision domains in the given network.

```
void testCase3() {
    Switch networkSwitch(5);
    vector<Device*> devices;
    for(int i=0;i<5;i++){
        devices.push_back(new Device());
        networkSwitch.learnMAC(devices[i]);
    }
    Device* sender = devices[1];
    Device* receiever = devices[4];
    vector<Frame> frames;
    for(int i=0;i<8;i++){
        frames.push_back({sender,receiever,"11011"});
    }
    if(csma_cd()){
        goBackN(frames, 3);
    }
}
```

Output:

```
Switch created
CSMA/CD: Collision detected! Retrying... k = 1
Waiting for 0 units
CSMA/CD: Successful transmission.
Error Control: No Error Detected
Switch: Forwarding frame from AA:BB:CC:DD:EE:01 to AA:BB:CC:DD:EE:04 via Port 4
Sliding Window: Sending 3 frames...
Error detected retransmitting...
Sliding Window: Resending 3 frames...
Error detected retransmitting...
Sliding Window: Resending 3 frames...
Frame 0 Sent: 110101
Error detected retransmitting...
Sliding Window: Resending 3 frames...
Frame 0 Sent: 110101
Frame 1 Sent: 011000
Frame 2 Sent: 101110
Sliding Window: Sending 3 frames...
Error detected retransmitting...
Sliding Window: Resending 3 frames...
Frame 3 Sent: 110101
Frame 4 Sent: 111000
Frame 5 Sent: 101110
Sliding Window: Sending 2 frames...
Frame 6 Sent: 110101
Frame 7 Sent: 011000
```



Collision Domain = 5

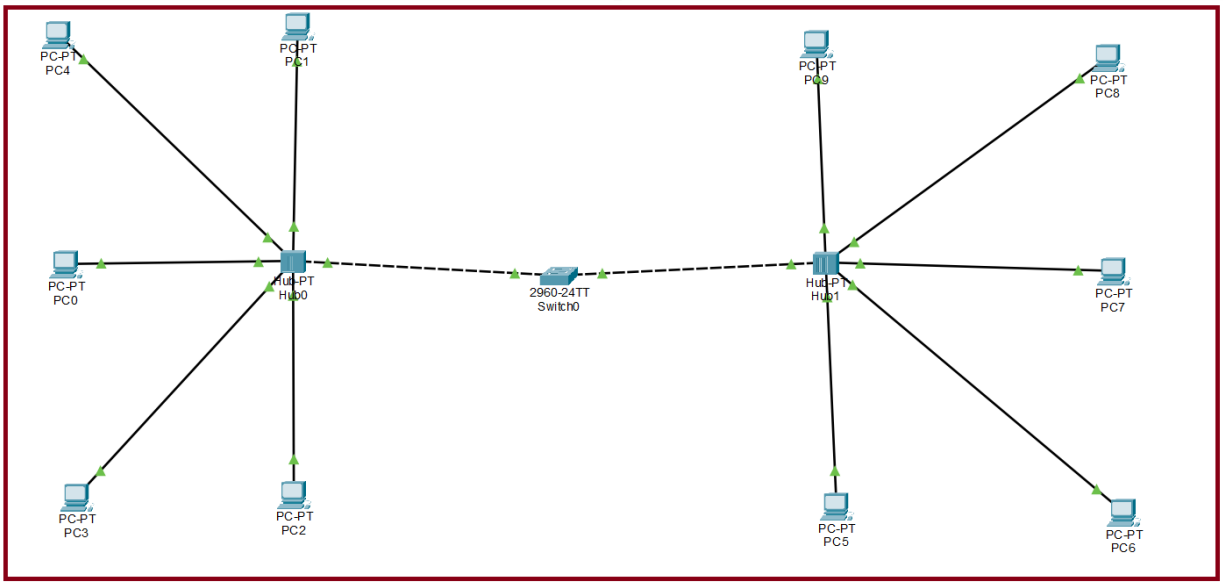
Broadcast Domain = 1

6) Create two-star topologies with fiveve end devices connected to a hub in each case and then connect two hubs using a switch. Enable communication between all 10 end devices and report the total number of broadcast and collision domains in the given network

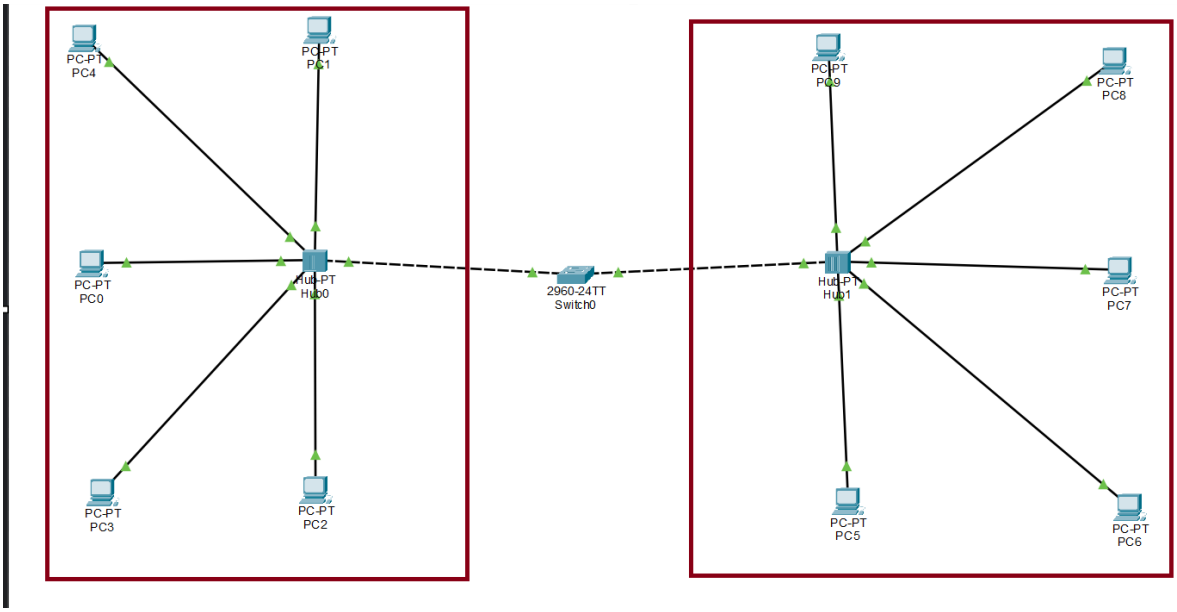
```
void testCase4() {
    Switch* networkSwitch = new Switch(2);
    Hub* hub1 = new Hub();
    Hub* hub2 = new Hub();
    networkSwitch->addHub(hub1);
    networkSwitch->addHub(hub2);
    hub1->connectSwitch(networkSwitch);
    hub2->connectSwitch(networkSwitch);
    vector<Device*> devices;
    for(int i=0;i<5;i++){
        Device* dev = new Device();
        devices.push_back(dev);
        dev->setHub(hub1);
        hub1->addDevice(dev);
```

```
        networkSwitch->learnPortToMac(dev,0);
    }
    for(int i=0;i<5;i++){
        Device* dev = new Device();
        devices.push_back(dev);
        dev->setHub(hub2);
        hub2->addDevice(dev);
        networkSwitch->learnPortToMac(dev,1);
    }
    //dev1 sends to dev7
    devices[1]->sendMessage(devices[9],"10110");
}
```

```
sending message from device 1 to hub 0
Hub 0 is broadcasting
Device 0 rejected message
Device 2 rejected message
Device 3 rejected message
Device 4 rejected message
Forwarding data from switch via port 1
Hub 1 is broadcasting
Device 5 rejected message
Device 6 rejected message
Device 7 recieved message : 10110
Sending ack from device 7 to device 1
Hub 1 is broadcasting
Device 5 rejected ack
Device 6 rejected ack
Device 8 rejected ack
Device 9 rejected ack
Forwarding data from switch via port 0
Hub 0 is broadcasting
Device 0 rejected ack
Device 1 recieved ack : Ack
Device 2 rejected ack
Device 3 rejected ack
Device 4 rejected ack
Device 8 rejected message
Device 9 rejected message
```



Broadcast domain = 1



Collision domain = 2

Source Code

```
#include<bits/stdc++.h>
using namespace std;

class Device;
class Switch;

struct Frame{
    Device* srcMAC;
    Device* destMAC;
    string data;
};

class Hub {
    static int hNo;
    int id;
    vector<Device*> devices;
    Switch* swi;
    public :
    Hub(){
        this->id = hNo++;
        this->devices = {};
        cout<<"Hub "<<id<<" created\n";
    }
    int getId(){
        return this->id;
    }
    void connectSwitch(Switch* swi){
        cout<<"Hub "<<this->id<<" connected to switch \n";
        this->swi = swi;
    }
    void addDevice(Device* device);
    void broadCastData(Device* sender, Device* receiver, string data,bool toSwi);
    void broadCastAck(Device* sender, Device* receiver,string ack,bool toSwi);
};
int Hub :: hNo = 0;

class Device {
    static int devNo;
    int id;
    Hub* hub;
    Device* macAdd;
```

```
    Switch* swi;
    unordered_map<Device*,int> connectedDevices;
    public :
    Device(){
        this->macAdd = this;
        this->id = devNo++;
        this->hub = NULL;
        cout<<"Device "<<id<<" created\n";
    }
    int getId(){
        return this->id;
    }
    Device* getMac(){
        return this->macAdd;
    }
    void setHub(Hub* hub){
        this->hub = hub;
    }
    void setSwitch(Switch* swi){
        this->swi = swi;
    }
    void sendMessage(Device* reciever,string data){
        cout<<"sending message from device "<<this->id<<" to hub "<<this->hub-
>getId()<<endl;
        this->hub->broadCastData(this,reciever,data,true);
    }
    void receiveMessage(Device* sender,Device* reciever,string data){
        if(reciever->getId() != this->id){
            cout<<"Device "<<this->id<<" rejected message \n";
            return;
        }
        cout<<"Device "<<this->id<<" recieved message : "<<data<<endl;
        cout<<"Sending ack from device "<<this->id<<" to device "<<sender-
>getId()<<endl;
        this->sendAck(reciever,sender);
    }
    void receiveAck(Device* sender,Device* reciever,string data){
        if(reciever->getId() != this->id){
            cout<<"Device "<<this->id<<" rejected ack \n";
            return;
        }
        cout<<"Device "<<this->id<<" recieved ack : "<<data<<endl;
    }
    void sendAck(Device* sender,Device* reciever){
```

```
        this->hub->broadCastAck(sender,reciever,"Ack",true);
    }
};
int Device :: devNo = 0;

void Hub:: addDevice(Device* device){
    cout<<"Hub "<<this->id<<" connected to "<<" device "<<device->getId()<<endl;
    this->devices.push_back(device);
}

bool csma_cd();

class Switch{
    //MAC to port no
    unordered_map<Device*, int> macTable;
    unordered_map<int,Hub*> hubs;
    unordered_map<int,vector<Device*>> portToMac;
    int ports;
    int maxPorts;

public:
    Switch(int ports){
        this->ports = 0;
        this->maxPorts = ports;
        cout<<"Switch created\n";
    }

    void learnMAC(Device* mac){
        if(ports == maxPorts){
            cout<<"cannot add more ports"<<endl;
            return;
        }
        macTable[mac] = ports;
        this->ports += 1;
    }

    void learnPortToMac(Device* mac,int port){
        portToMac[port].push_back(mac);
    }

    void addHub(Hub* hub){
        if(ports == maxPorts){
            cout<<"cannot add more ports"<<endl;
```

```
            return;
        }
        this->hubs[this->ports] = hub;
        this->ports += 1;
    }

    void forwardFrame(Frame frame){
        if(macTable.find(frame.destMAC) != macTable.end()){
            int outPort = macTable[frame.destMAC];
            cout<<"Switch: Forwarding frame from "<<frame.srcMAC<<" to
"<<frame.destMAC<<" via Port "<<outPort<<endl;
        }else{
            cout<<"destination not found\n";
        }
    }

    void forwardFrameToHub(Device* sender,Device* reciever,string data,bool
isAck){
        for(pair<int,vector<Device*>> it : portToMac){
            int port = it.first;
            vector<Device*> devices = it.second;
            for(auto device : devices){
                if(device == reciever){
                    if(csma_cd()){
                        cout<<"Forwarding data from switch via port
"<<port<<endl;
                        if(this->hubs[port]){
                            if(!isAck) this->hubs[port]-
>broadCastData(sender,reciever,data,false);
                            else this->hubs[port]-
>broadCastAck(sender,reciever,data,false);
                        }else{
                            cout<<"Device "<<reciever->getId()<<" recieved :
"<<data<<endl;
                            cout<<"forwarding ack via port "<<port<<endl;
                            cout<<"Device "<<sender->getId()<<" recieved :
ACK\n";
                        }
                    }
                }
            }
        }
    }
};
```

```
void Hub::broadCastData(Device* sender,Device* receiver, string data,bool toSw){
    cout<<"Hub "<<this->id<<" is broadcasting \n";
    for(int i = 0; i < devices.size(); i++){
        if(devices[i] != sender){
            devices[i]->receiveMessage(sender,receiver, data);
        }
        if(devices[i] == receiver) return;
    }
    if(this->swi && toSw){
        swi->forwardFrameToHub(sender,receiver,data,false);
    }
}

void Hub::broadCastAck(Device* sender, Device* receiver, string data,bool toSw){
    cout<<"Hub "<<this->id<<" is broadcasting \n";
    for(int i = 0; i < devices.size(); i++){
        if(devices[i] != sender){
            devices[i]->receiveAck(sender,receiver, data);
        }
        if(devices[i] == receiver) return;
    }
    if(this->swi && toSw){
        swi->forwardFrameToHub(sender,receiver,data,true);
    }
}

void addRedundantBit(Frame &frame){
    int count = 0;
    for(int i=0;i<frame.data.length();i++){
        char ch = frame.data[i];
        if(ch == '1') count++;
    }
    if(count % 2 == 0){
        frame.data = frame.data + '0';
    }else{
        frame.data = frame.data + '1';
    }
}

bool parityCheck(Frame frame){
    int count = 0;
    for(int i=0;i<frame.data.length();i++){
```

```
        char ch = frame.data[i];
        if(ch == '1') count++;
    }
    return count % 2 == 0;
}

bool csma_cd(){
    srand(time(0));
    int k = 0;
    cout<<"Detecting transmission medium..."<<endl;
    while(k <= 15){
        int transmissionChance = rand() % 100;
        if(transmissionChance < 30){
            cout<<"CSMA/CD: Successful transmission.\n";
            return true;
        }else{
            cout<<"CSMA/CD: Collision detected! Retrying... k = "<<++k<<endl;
            int waitTime = rand() % int(pow(2,k)-1);
            cout<<"Waiting for "<<waitTime<<" units\n";
        }
    }
    cout<<"Aborted\n";
    return false;
}

void goBackN(vector<Frame>& frames, int windowSize) {
    int i = 0;
    while(i < frames.size()){
        srand(time(0));
        int framesToSend = min(windowSize, (int)frames.size() - i);
        cout<<"Sliding Window: Sending "<< framesToSend<<" frames...\n";
        int j = i;
        while(j < i + framesToSend){
            int errorChance = rand() % 100;
            if(errorChance < 50){
                cout<<"Frame "<<j<<" Sent: "<<frames[j].data<<endl;
                // addRedundantBit(frames[j]);
                if(parityCheck(frames[j])){
                    cout<<"No error detected\n";
                }else{
                    cout<<"Error detected retransmitting...\n";
                    cout<<"Sliding Window: Resending "<<framesToSend<<"
frames...\n";
                    j = i;
```

```
                    continue;
                }
                j++;
            }else{
                cout<<"Error detected retransmitting...\n";
                cout<<"Sliding Window: Resending "<<framesToSend<<" frames...\n";
                j = i;
                continue;
            }
        }
        i += windowSize;
    }
}

void testCase3() {
    int collision_domains = 0;
    Switch* networkSwitch = new Switch(5);
    vector<Device*> devices;
    for(int i=0;i<5;i++){
        Device* dev = new Device();
        dev->setSwitch(networkSwitch);
        devices.push_back(dev);
        networkSwitch->learnMAC(devices[i]);
        networkSwitch->learnPortToMac(devices[i],i);
        collision_domains++;
    }
    cout<<"Collision domains = "<<collision_domains<<" and Broadcast domains will
be 1 always in upto DLL\n";
    Device* sender = devices[1];
    Device* receiever = devices[4];
    vector<Frame> frames;
    for(int i=0;i<8;i++){
        frames.push_back({sender,receiever,"11011"});
    }
    if(csma_cd()){
        goBackN(frames, 3);
    }
}

void testCase4() {
    int collision_domains = 0;
    Switch* networkSwitch = new Switch(2);
    Hub* hub1 = new Hub();
    Hub* hub2 = new Hub();
```

```
    // Hub* hub3 = new Hub();
    networkSwitch->addHub(hub1);
    collision_domains++;
    networkSwitch->addHub(hub2);
    collision_domains++;
    // networkSwitch->addHub(hub3);
    hub1->connectSwitch(networkSwitch);
    hub2->connectSwitch(networkSwitch);
    // hub3->connectSwitch(networkSwitch);
    vector<Device*> devices;
    for(int i=0;i<5;i++){
        Device* dev = new Device();
        devices.push_back(dev);
        dev->setHub(hub1);
        hub1->addDevice(dev);
        networkSwitch->learnPortToMac(dev,0);
    }
    for(int i=0;i<5;i++){
        Device* dev = new Device();
        devices.push_back(dev);
        dev->setHub(hub2);
        hub2->addDevice(dev);
        networkSwitch->learnPortToMac(dev,1);
    }
    cout<<"Collision domains = "<<collision_domains<<" and Broadcast domains will
be 1 always in upto DLL\n";
    // Device* dev1 = new Device();
    // devices.push_back(dev1);
    // dev1->setHub(hub3);
    // hub3->addDevice(dev1);
    // networkSwitch->learnPortToMac(dev1,2);
    //dev1 sends to dev9
    devices[1]->sendMessage(devices[9],"10110");
    // Device* dev2 = new Device();
    // devices.push_back(dev2);
    // dev2->setSwitch(networkSwitch);
    // networkSwitch->learnPortToMac(dev2,3);
    // devices[1]->sendMessage(devices[11],"10110");
}

int main() {
    // testCase3();
    testCase4();
}
```