TCP/IP Model Simulation Project

Computer Networks

Submitted By	
Name:	Enrollment No:
1.Musharaf Maqbool	2022Bite053
2.Mohammad Oyais ussman	2022Bite06
3.Gazi Rayan	2022Bite049

Submission Date: March 24, 2025

Information Technology and Engineering

Contents

Abstract

This report presents a simulation project aimed at implementing and understanding the TCP/IP networking model, with a focus on the physical and data link layers. The project implements various network devices such as hubs, switches, bridges, and different network topologies like star and bus. The implementation explores fundamental networking concepts including data transmission, collision domains, broadcast domains, MAC address learning, and flow control protocols. Through this project, we demonstrate the behavior of different network components and their interactions in a simulated environment.

1 Introduction

The TCP/IP model is a conceptual framework used to understand and implement network communications. It consists of four layers: the Link Layer, the Internet Layer, the Transport Layer, and the Application Layer. This project focuses primarily on the Link Layer, which encompasses both physical and data link layer functionalities.

Understanding networking concepts is crucial for computer science students and professionals. This project provides hands-on experience with network device implementations and protocols, allowing for a deeper understanding of how data travels through networks.

2 Objectives

The main objectives of this project are:

- To implement and simulate the functionality of Layer 1 (Physical) devices such as hubs
- To implement and simulate the functionality of Layer 2 (Data Link) devices such as switches and bridges
- To implement different network topologies including star and bus
- To demonstrate the MAC address learning and forwarding mechanism in switches
- To implement and visualize flow control protocols including Stop-and-Wait and Selective Repeat
- To understand collision domains and broadcast domains in different network setups

3 TCP/IP Model Overview

The TCP/IP model is a concise version of the OSI model with four layers instead of seven. This project focuses on the Link Layer, which corresponds to the Physical and Data Link layers in the OSI model.

3.1 Physical Layer (Layer 1)

- Deals with the physical transmission of bits over a medium
- Concerned with hardware aspects like cables, connectors, voltage levels, etc.
- Devices at this layer include hubs and repeaters

3.2 Data Link Layer (Layer 2)

- Responsible for node-to-node delivery of data
- Handles framing, addressing (MAC addresses), error detection
- Devices at this layer include switches and bridges

4 Implementation Details

4.1 Project Structure

The project is organized in a modular structure with separate classes for different network components:

- `EndDevices`: Represents end devices like computers with MAC and IP addresses
- `Hub`: Implements a Layer 1 device that broadcasts all incoming data
- `Bus`: Implements a bus topology for connecting devices
- `Switch`: Implements a Layer 2 device with MAC address learning capability
- `Topology`: Contains functions to create and manage different network topologies
- `Prompt`: Manages the user interface and test case execution

4.2 Device Implementation

The `EndDevices` class forms the basis for all end devices in the network:

```cpp
class EndDevices {
private:
    string mac_address;
    string ip_address;

public:
    void setDevice(const string& mac_address, const string& ip_address)
    ;
    string getMacAddress() const;
    string getIpAddress() const;
};
```

Listing 4.1: EndDevices Class Definition

The implementation includes methods to initialize devices with MAC and IP addresses:

```cpp
void EndDevices::setDevice(const string& mac_address, const string&
    ip_address) {
    this->mac_address = mac_address;
    this->ip_address = ip_address;
}

```

```
string EndDevices::getMacAddress() const {
    return mac_address;
}

string EndDevices::getIpAddress() const {
    return ip_address;
}
```

Listing 4.2: EndDevices Implementation

4.3 Layer 1 Devices and Topologies

4.3.1 Hub Implementation

The `Hub` class implements a basic network hub that broadcasts all incoming data to all connected devices except the sender:

```
class Hub {
private:
    vector<EndDevices*> connectedDevices;
    vector<Switch*> connectedSwitches;
public:
    void connectDevice(EndDevices* device);
    void broadcastData(const string& senderMAC, const string& data);
    void displayConnectedDevices() const;
    void broadcastAck(const string& senderMAC);
    void connectToSwitch(Switch* switchDevice);
};
```

Listing 4.3: Hub Class Definition

The implementation includes:

```
void Hub::broadcastData(const string& senderMAC, const string& data) {
    cout << "\nBroadcasting data from " << senderMAC << " to all
   devices, data= " << data << endl;

    for (auto& device : connectedDevices) {
        if (device->getMacAddress() != senderMAC) {
            cout << " -> Sent to device with MAC: " << device->
   getMacAddress() << endl;
        }
    }
    for(auto & device : connectedSwitches){
       cout<<" -> sent to switch with Mac " <<device->getMacAddress()<<
   endl;
    }
}
```

Listing 4.4: Hub Implementation

4.3.2 Bus Implementation

The `Bus` class implements a bus topology where devices are connected to a common backbone:

```cpp
class Bus {
private:
    vector<EndDevices*> connectedDevices;

public:
    void connectDevice_bus(EndDevices* device);
    void transmitData_bus(const string& senderMAC, const string& data, const string& receiverMAC);
    void displayConnectedDevices_bus() const;
};
```

Listing 4.5: Bus Class Definition

The implementation includes methods for data transmission:

```cpp
void Bus::transmitData_bus(const std::string& senderMAC, const std::string& data, const std::string& receiverMAC) {
    std::cout << "\nTransmitting data from " << senderMAC << " to " << receiverMAC << "  " << data << std::endl;

    for (auto& device : connectedDevices) {
        if (device->getMacAddress() != senderMAC && device->getMacAddress() == receiverMAC) {
            std::cout << " -> message received device with MAC: " << device->getMacAddress() << std::endl;
        }
    }
}
```

Listing 4.6: Bus Implementation

4.4 Layer 2 Devices and Protocols

4.4.1 Switch Implementation

The `Switch` class implements a Layer 2 device with MAC address learning and frame forwarding capabilities:

```cpp
class Switch {
private:
    std::string mac_address;
    std::unordered_map<std::string, int> macTable;
    std::vector<std::string> connectedDevices;

public:
    static const int MAX_PORTS = 10;
    static const int WINDOW_SIZE = 3;
    static const int SEQ_NUM_SIZE = 8;

    int calculateChecksum(const std::string& frame);
    void forwardPacket(const std::string& mac_dest, int incomingPort);
    void learnMacAddress(const std::string& mac_source, int port);
    void applyAccessControl(const std::string& protocol, const std::string& senderMAC,
                              const std::string& receiverMAC, const std::string& data);
    void stopAndWait(const std::string& senderMAC, const std::string& receiverMAC, const std::string& data);
    void selectiveRepeat(const std::string& senderMAC, const std::string& receiverMAC, const std::string& data);
    void displayConnectedDevices();
    void connectDevice(const std::string& macAddress);
    void refreshMacTable();
    void connectHub(Hub* hub);
    void setDevice(const std::string & mac_address);
    std:: string getMacAddress() const;

    std:: vector<Hub*> connectedHubs;
};
```

Listing 4.7: Switch Class Definition

4.4.2 MAC Learning and Forwarding

The switch implements MAC address learning and forwarding logic:

```
void Switch::learnMacAddress(const std::string& mac_source, int port) {
    if (macTable.size() >= MAX_PORTS) {
        std::cout << "Error: Maximum port limit reached. Cannot add more devices." << std::endl;
        return;
    }
    macTable[mac_source] = port;
    connectedDevices.push_back(mac_source);
}

void Switch::forwardPacket(const std::string& mac_dest, int incomingPort) {
    if (macTable.find(mac_dest) != macTable.end()) {
        std::cout << "Packet forwarded to port " << macTable[mac_dest] << " for MAC: " << mac_dest << std::endl;
    } else {
        std::cout << "Flooding packet to all ports except " << incomingPort << " for MAC: " << mac_dest << std::endl;
    }
}
```

Listing 4.8: MAC Learning and Forwarding Implementation

4.4.3 Flow Control Protocols

The project implements two flow control protocols:

Stop-and-Wait

```
void Switch::stopAndWait(const std::string& senderMAC, const std::string& receiverMAC, const std::string& data) {
    std::srand(std::time(nullptr));
    std::cout << "[Stop-and-Wait] Sending data from " << senderMAC << " to " << receiverMAC << "..." << std::endl;

```

```cpp
    for (char c : data) {
        std::string frame(1, c);
        int checksum = calculateChecksum(frame);
        std::cout << "Sending frame: " << c << " with checksum: " << checksum << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(1));

        bool frameCorrupt = (std::rand() % 2 == 0);
        if (frameCorrupt) {
            std::cout << "Frame " << c << " corrupted! Requesting retransmission..." << std::endl;
            std::this_thread::sleep_for(std::chrono::seconds(1));
            std::cout << "Resending frame: " << c << " with checksum: " << checksum << std::endl;
            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        std::cout << "ACK received for frame: " << c << std::endl;
    }
    std::cout << "[Stop-and-Wait] Transmission successful." << std::endl;
}
```

Listing 4.9: Stop-and-Wait Protocol Implementation

Selective Repeat

```cpp
void Switch::selectiveRepeat(const std::string& senderMAC, const std::string& receiverMAC, const std::string& data) {
    std::srand(std::time(nullptr));
    std::cout << "[Selective Repeat] Sending frames from " << senderMAC << " to " << receiverMAC << "..." << std::endl;

    int base = 0;
    std::vector<bool> ackReceived(data.size(), false);
    while (base < data.size()) {
        // Implementation continues with window-based transmission and selective acknowledgment
        // ...
    }
    std::cout << "[Selective Repeat] All frames acknowledged. Transmission successful." << std::endl;
}
```

Listing 4.10: Selective Repeat Protocol Implementation

4.5 Network Topologies

4.5.1 Star Topology

The project implements two variants of star topology:

Hub-based Star Topology

```
void starTopology() {
    int device, device2;
    string message;
    Hub hub;
    for (auto& device : deviceList) {
        hub.connectDevice(&device);
    }
    // User input handling and data transmission
    // ...
    hub.broadcastData(deviceList[device].getMacAddress(), message);
    hub.displayConnectedDevices();
    hub.broadcastAck(deviceList[device2].getMacAddress());
}
```

Listing 4.11: Hub-based Star Topology Implementation

Switch-based Star Topology

```
void starTopology_switch() {
    int device, device2;
    std::string message, protocol;
    Switch networkSwitch;
    initializeEndDevices();

    // Connect devices and assign ports
    int port = 1;
    for (auto& dev : deviceList) {
        if (port > Switch::MAX_PORTS) {
            std::cout << "Max ports reached. Cannot connect more
  devices." << std::endl;
            break;
        }
        networkSwitch.learnMacAddress(dev.getMacAddress(), port++);
    }

    // User input and protocol selection
    // ...

    // Apply access control and forward packets
    networkSwitch.applyAccessControl(protocol, deviceList[device].
  getMacAddress(),
                                     deviceList[device2].getMacAddress(),
    message);

    networkSwitch.displayConnectedDevices();
    std::this_thread::sleep_for(std::chrono::seconds(5));
    networkSwitch.refreshMacTable();
}
```

Listing 4.12: Switch-based Star Topology Implementation

4.5.2 Bus Topology

The implementation of bus topology:

```
void busTopology() {
   int device, device2;
   string message;
    Bus bus;
    for (auto& device : deviceList) {
        bus.connectDevice_bus(&device);
    }

    // User input and message transmission
    // ...

    bus.transmitData_bus(deviceList[device].getMacAddress(), message,
                        deviceList[device2].getMacAddress());
    bus.displayConnectedDevices_bus();
}
```

Listing 4.13: Bus Topology Implementation

5 Test Cases and Results

5.1 Layer 1 Test Cases

Two test cases were implemented for Layer 1 devices:

1. Basic communication between two devices
2. Star topology with hub broadcasting

Example test case implementation:

```
void testcase1_layer1(){
    initializeEndDevices();
    int device, device2;
    string message;
    cout << "What is the sender device? (devices start from 0-2)" <<
  endl;
    cin >> device;
    cout << "Enter the message you want to send:" << endl;
    cin >> message;
    cout << "Enter the receiving device:" << endl;
    cin >> device2;
    cout << "Device " << deviceList[device].getMacAddress() << " is
  sending message "
          << message << " to device " << deviceList[device2].
  getMacAddress() << endl;
}
```

Listing 5.1: Layer 1 Test Case Implementation

5.2 Layer 2 Test Cases

Two test cases were implemented for Layer 2 devices:

1. Switch-based star topology with MAC address learning
2. Multi-segment network with hubs connected via a switch

Example test case implementation:

```
void testcase2_layer2() {
    // Create two hubs and a central switch
    Hub h1, h2;
    Switch centralSwitch;
    centralSwitch.setDevice("A4:B1:C2:3D:E5:F6");

    // Connect first 5 devices to h1
    for (int i = 0; i < 5; i++) {
        h1.connectDevice(&deviceList[i]);
    }

    // Connect last 5 devices to h2
    for (int i = 5; i < 10; i++) {
        h2.connectDevice(&deviceList[i]);
    }

    // Connect hubs to the central switch
    centralSwitch.connectHub(&h1);
    centralSwitch.connectHub(&h2);

    // User input and message transmission
    // ...

    // Determine which hub the sender and receiver belong to and route
    accordingly
    // ...
}
```

Listing 5.2: Layer 2 Test Case Implementation

Results showed:

- In Layer 1 devices, all traffic is broadcast to all connected devices
- In Layer 2 devices, the switch learns MAC addresses and forwards traffic selectively
- Flow control protocols demonstrate reliability mechanisms for data transmission
- Multiple collision domains are observed when using switches

6 Conclusion and Future Work

This project successfully implemented and demonstrated the functionality of various networking devices and protocols at the Link Layer of the TCP/IP model. The simulation allows for a better understanding of how hubs, switches, and different topologies affect network behavior.

Key achievements include:

- Implementation of both Layer 1 and Layer 2 devices
- MAC address learning and forwarding in switches
- Implementation of reliability protocols (Stop-and-Wait and Selective Repeat)
- Demonstration of different network topologies

Future work could include:

- Implementation of the remaining layers of the TCP/IP model
- Adding more sophisticated routing algorithms
- Implementing additional network topologies such as mesh and tree
- Adding QoS (Quality of Service) mechanisms
- Developing a graphical user interface for better visualization

7 References

1. Kurose, J. F., & Ross, K. W. (2022). Computer Networking: A Top-Down Approach (8th ed.). Pearson.

2. Tanenbaum, A. S., & Wetherall, D. J. (2021). Computer Networks (6th ed.). Pearson.

3. Forouzan, B. A. (2017). Data Communications and Networking (5th ed.). McGraw-Hill Education.

8 Appendix: Code Listings

The complete code implementation consists of the following files:

1. device.h and device.cpp: End device implementation
2. hub.h and hub.cpp: Hub implementation
3. bus.h and bus.cpp: Bus implementation
4. switch.h and switch.cpp: Switch implementation
5. topology.h and topology.cpp: Topology implementations
6. prompt.h: User interface implementation
7. main.cpp: Main program entry point
8. Makefile: Build configuration

For brevity, detailed code listings are not included in this report.