TCP/IP Network Simulator

Documentation

Furqan Makhdoomi (2022BITE005)
Mohammad Haashid (2022BITE054)
Mohammad Huzaif (2022BITE047)

March 24, 2025

Contents

1 Introduction

This document provides a comprehensive overview of our TCP/IP Network Simulator, a modular simulator focusing on the Physical and Data Link Layers with extensibility for higher layers. The simulator is designed to demonstrate key networking concepts including error control, flow control, and medium access control protocols.

1.1 Project Overview

The TCP/IP Network Simulator is an educational tool that allows users to:

- Create and visualize network topologies
- Simulate data transmission between network devices
- Implement and observe various network protocols
- Analyze network performance under different conditions
- Understand error detection and correction mechanisms

2 Architecture

2.1 Layered Design

The simulator follows the TCP/IP layered architecture, with current implementation focusing on:

- **Physical Layer**: Simulates the transmission of bits over physical media
- **Data Link Layer**: Handles framing, addressing, error control, and medium access

2.2 Core Components

The simulator consists of the following core components:

- **Device**: Represents network endpoints such as computers and servers
- **Link**: Represents physical connections between devices
- **Frame**: Encapsulates data for transmission at the Data Link Layer

- **MAC Address**: Provides unique identification for network devices
- **Network**: Manages the overall network topology and simulation
- **Hub**: Implements a simple signal repeater that broadcasts to all connected devices
- **Bridge/Switch**: Implements intelligent frame forwarding based on MAC addresses

These components interact to create a complete network simulation environment, with each component handling specific aspects of the network's operation.

3 Physical Layer Implementation

3.1 Network Devices

The simulator implements various network devices:

- **End Devices**: Computers, servers, and other endpoints
- **Hubs**: Simple signal repeaters that broadcast to all connected devices
- **Links**: Physical connections between devices

3.2 Transmission Medium

Links in the simulator represent the physical transmission medium with the following properties:

- Transmission delay simulation
- Medium state (busy/free) tracking
- Collision detection capabilities
- Error injection for realistic simulation

4 Data Link Layer Implementation

4.1 Framing

Data is encapsulated in frames with the following structure:

- Source MAC address
- Destination MAC address
- Sequence number
- Frame type (DATA, ACK, NAK)
- Data payload
- Checksum for error detection

4.2 Addressing

The simulator implements MAC addressing with the following features:

- Unique MAC address generation for each device
- Address comparison and matching
- Broadcast address support (FF:FF:FF:FF:FF:FF)

5 Protocol Implementations

5.1 Error Control

5.1.1 Checksum

Error detection is implemented using a simple checksum algorithm:

```python
def _calculate_checksum(self):
    """Calculate a simple checksum for error detection"""
    # Use a simple sum of bytes as checksum for demonstration
    checksum = 0
    # Include header fields in checksum
    for c in str(self.source_mac) + str(self.destination_mac) + str(self.sequence_number):
        checksum = (checksum + ord(c)) % 256
    # Include data in checksum
    for c in self.data:
        checksum = (checksum + ord(c)) % 256
```

```
    return checksum
```

Listing 1: Checksum Implementation

5.1.2 Error Injection

To test error detection and correction, the simulator can introduce random bit errors:

```
def introduce_error(self):
    """Introduce a random bit error in the frame data for testing"""
    if len(self.data) > 0:
        char_pos = random.randint(0, len(self.data) - 1)
        char_list = list(self.data)
        # Flip a random bit in the selected character
        char_code = ord(char_list[char_pos])
        bit_pos = random.randint(0, 7)
        char_code ^= (1 << bit_pos)  # Flip the bit
        char_list[char_pos] = chr(char_code)
        self.data = ''.join(char_list)
        # Don't update checksum to simulate error
```

Listing 2: Error Injection

5.2 Flow Control

5.2.1 Stop-and-Wait

The simulator implements the Stop-and-Wait protocol for basic flow control:

- Sender transmits a frame and waits for acknowledgment
- Receiver sends ACK upon successful receipt
- Timeout mechanism for retransmission
- Simple but inefficient utilization of bandwidth

5.2.2 Go-Back-N

For improved efficiency, the Go-Back-N protocol is implemented:

- Sliding window mechanism allows multiple unacknowledged frames
- Configurable window size
- Cumulative acknowledgments
- Timeout-based retransmission of all unacknowledged frames

5.3 Medium Access Control

5.3.1 CSMA/CD

The simulator implements Carrier Sense Multiple Access with Collision Detection:

- Carrier sensing before transmission
- Random medium busy simulation
- Collision detection during transmission
- Binary exponential backoff algorithm
- Jam signal transmission upon collision

```python
def transmit(self, frame, source):
    # Create a copy of the frame
    transmitted_frame = Frame(...)

    # Simplified CSMA/CD implementation
    attempts = 0
    max_attempts = 5

    while attempts < max_attempts:
        # Randomly make the medium busy (20% chance)
        if random.random() < 0.2:
            self.logger.info(f"Medium is busy when {source.name} tries to send frame {frame.sequence_number}")
            time.sleep(0.05)
            attempts += 1
            continue

        # Medium is free, proceed with transmission
        self.logger.info(f"{source.name} transmitting frame {frame.sequence_number} to {destination.name}")
        time.sleep(0.02)

        # Small chance of collision (10%)
        if random.random() < 0.1:
            self.logger.warning(f"Collision detected during {source.name}'s transmission")
            # Apply backoff
            backoff_time = random.uniform(0.01, 0.05) * (attempts + 1)
            self.logger.info(f"{source.name} backing off for {backoff_time:.3f}s after collision")
            time.sleep(backoff_time)
```

```python
            attempts += 1
            continue

        # Successful transmission
        destination.receive_message(transmitted_frame, source)
        return True

    # Max attempts reached
    return False
```

Listing 3: Simplified CSMA/CD Implementation

6 Simulation Features

6.1 Network Creation

The simulator provides a flexible API for creating network topologies:

- Adding and removing devices
- Creating links between devices
- Connecting and disconnecting endpoints

6.2 Message Transmission

Users can simulate data transmission with various options:

- Point-to-point communication
- Broadcasting
- Protocol selection (Stop-and-Wait, Go-Back-N)
- Error control configuration

6.3 Demonstration Modes

The simulator includes specialized demonstration modes:

- **Error Control Demo**: Shows error detection and correction
- **CSMA/CD Demo**: Demonstrates medium access control with collision handling

7 User Interface

7.1 Command Line Interface

The simulator provides an interactive command-line interface:

```
Enter command: add device pc1
2025-03-24 05:59:01,703 - p1 - INFO - Device p1 created with
    MAC ac:e9:68:a7:63:f1

Enter command: add device pc2
2025-03-24 05:59:04,621 - p2 - INFO - Device p2 created with
    MAC a2:82:e4:57:8c:a1

Enter command: add link l1 p1 p2
2025-03-24 05:59:15,609 - p1 - INFO - Connected to link l1
2025-03-24 05:59:15,609 - p2 - INFO - Connected to link l1

Enter command: send p1 01111 p2
2025-03-24 05:59:21,098 - p1 - INFO - Sending message: 01111
```

Listing 4: CLI Example

7.2 Logging

Comprehensive logging provides visibility into the simulation:

- Device-level logs
- Link-level logs
- Protocol operation logs
- Error and collision reporting

8 Theoretical Concepts

8.1 Error Control Theory

The simulator demonstrates key error control concepts:

- **Error Detection**: Using checksums to identify corrupted frames
- **ARQ (Automatic Repeat Request)**: Retransmission of corrupted or lost frames
- **Positive Acknowledgment**: Confirming successful receipt of frames

- **Negative Acknowledgment**: Requesting retransmission of specific frames

8.2 Flow Control Theory

Flow control mechanisms implemented in the simulator:

- **Sliding Window**: Allowing multiple frames in transit
- **Window Size**: Controlling the number of unacknowledged frames
- **Buffering**: Storing received frames for in-order delivery
- **Timeout**: Detecting lost frames and triggering retransmission

8.3 Medium Access Control Theory

The CSMA/CD implementation demonstrates:

- **Carrier Sensing**: Checking if the medium is busy before transmitting
- **Multiple Access**: Allowing multiple devices to share the medium
- **Collision Detection**: Identifying when multiple devices transmit simultaneously
- **Binary Exponential Backoff**: Algorithm for reducing collision probability

9 Performance Considerations

9.1 Optimization Techniques

Several optimizations are implemented:

- Efficient frame processing
- Timeout management
- Collision handling
- Backoff algorithm tuning