Network Simulator Report

Khushboo (2022BITE002)
Afsheen (2022BITE048)
Sibgat (2022BITE010)

July 7, 2025

1 Introduction

This report outlines the implementation of a full-stack Network Simulator that models the complete protocol stack—from Physical Layer to Application Layer. Each layer simulates real-world network functionalities including frame transmission, switching, routing, flow control, reliable delivery, and basic application services like Telnet and FTP.

2 Language Used

Python

3 Project Structure

```
network-simulator/

 main.py                        # Main simulation menu

 physical_layer/
    physical_layer.py          # Physical layer implementation

 data_link_layer/
    access_control.py          # CSMA/CD implementation
    bridge.py                  # Bridge simulation
    end_device.py              # End device definition
    error_control.py           # CRC error detection
    frame.py                   # Frame structure
    gbn.py                     # Go-Back-N ARQ
    switch.py                  # Switch implementation
    test_data_link.py          # Data Link Layer tests
```

```
NetworkLayer/
   host.py                    # Hosts with IP/MAC
   router.py                  # Router implementation
   serialLink.py              # Serial link logic
   testcase1.py               # Basic router test case
   testcase2.py               # Three routers with RIP test case
   __init__.py

TransportLayer/
   sliding_window.py          # Go-Back-N protocol
   transport.py               # Transport layer implementation
   test_transport_app.py      # Transport layer tests

ApplicationLayer/
   ftp_app.py                 # FTP service simulation
   telnet_app.py              # Telnet service simulation
   __init__.py
```

4 Simulator Features

1. **Physical Layer:** Simulates dedicated and star topology links.
2. **Data Link Layer:** Frame structuring, CRC error detection, ARQ protocols, CSMA/CD, switch, and bridging.
3. **Network Layer:** Implements IP addressing, ARP, and static routing using longest prefix match.
4. **Transport Layer:** Implements Go-Back-N (GBN) protocol using a sliding window mechanism.
5. **Application Layer:** Telnet, and FTP-style apps simulating client-server behavior.

5 Simulator Menu Interface

The user can interact with the simulator via a command-line menu in ‘main.py‘:

```
========== NETWORK SIMULATOR MENU ==========
1. Dedicated Link (End-to-End Connection)
2. Simulation through Hub | STAR TOPOLOGY
3. CRC Error Detection Simulation
4. Bridge Simulation
5. Go Back N Simulation
6. Switch with 5 Devices
```

```
7. Two Star Topologies with Hubs + Switch
8. Testing CSMA/CD
9. Network Test Case 1 (Basic Router)
10. Network Test Case 2 (Three Routers with RIP)
11. Transport Layer And Application Layer
12. Exit
==========================================
```

6 Transport Layer

The Transport Layer functionality is implemented in:

- `sliding_window.py`: Implements Go-Back-N (GBN) logic using a sliding window.
- `transport.py`: Integrates sender and receiver logic and connects to application layer.

Key features:

- Reliable, in-order delivery of packets.
- Cumulative acknowledgments.
- Timeout-based retransmission.
- Custom tests provided in `test_transport_app.py`.

7 Application Layer

Application-level services simulate interactive communication on top of the transport layer.

Implemented Apps

- `ftp_app.py`: The `ftp_app.py` implements a basic FTP server and client that simulates file transfer operations. It establishes separate control and data connections, using port 21 for command communication and ephemeral ports for file transfers. The implementation supports file downloads and directory listings with Go-Back-N reliability, maintaining session state throughout the transfer process.
- `telnet_app.py`: The `telnet_app.py` simulates a Telnet server-client interaction with Network Virtual Terminal (NVT) compliance. It handles user authentication and command execution through port 23, converting between local and NVT line endings while managing session state. The service provides a virtual terminal environment supporting basic commands like date and whoami with proper character echoing.

8 Network Layer Implementation

The Network Layer modules simulate IP-based routing, ARP resolution, and dynamic routing protocols as specified in the project requirements. The implementation supports both static and dynamic routing (RIP) with longest prefix matching.

Core Components

1. `host.py`:
   - Manages host devices with IP/MAC address assignment
   - Implements ARP protocol for MAC address resolution
   - Maintains ARP table cache
   - Handles basic packet sending/receiving at network layer
2. `router.py`:
   - Core router functionality with multiple network interfaces
   - Maintains routing tables (both static and dynamic)
   - Implements packet forwarding using next-hop semantics
   - Supports both directly connected and remote networks
3. `serialLink.py`:
   - Simulates point-to-point serial connections between routers
   - Handles HDLC-like framing for router-router links
   - Manages link state and bandwidth parameters

Testing Framework

- `testcase1.py`:
  - Basic router functionality test
  - Demonstrates ARP resolution within a network
  - Shows static routing between two directly connected networks
  - Validates IP packet forwarding
- `testcase2.py`:
  - Advanced test with three routers running RIP
  - Demonstrates dynamic route propagation
  - Shows route table convergence after topology changes
  - Tests longest prefix matching in complex scenarios

Key Features Implemented

1. Classless IPv4 addressing (CIDR notation)
2. ARP resolution within broadcast domains
3. Static routing configuration
4. RIP protocol implementation for dynamic routing
5. Longest prefix match routing decisions
6. Route aggregation support
7. Network topology simulation with multiple routers

The implementation follows the project specification by running all network layer functions in parallel, with detailed logging output showing the operation of each component. The test cases demonstrate both basic and advanced functionality as required in the deliverables.

Test Cases

- `testcase1.py`: Host Router ARP + packet forwarding.
- `testcase2.py`: Multi-router topology with RIP-style static routing.

9 Data Link Layer

This layer handles framing, MAC addressing, error detection, flow control, and switching.

Modules

- `frame.py`: Defines structure of a data frame.
- `switch.py`: MAC learning and packet forwarding.
- `bridge.py`: Divides broadcast domains.
- `stop_n_wait.py`: Implements basic ARQ.
- `gbn.py`: Layer-2 level Go-Back-N protocol.
- `access_control.py`: Implements CSMA/CD.
- `test_data_link.py`: Custom test case to check the working.

10 Physical Layer

Implemented in

physicallayer.py

The physical layer simulates raw bit transmission for two topologies:

- Dedicated Link: Simulates point-to-point wired transmission.
- Star Topology: Uses hub/switch to simulate broadcast/bus-based connectivity.

11 Logging and Debugging

Each layer provides console-based logs:

- ARP request/reply logs
- Routing decisions and next-hop info
- GBN window tracking and retransmissions
- Application send/receive events
- MAC learning and switch forwarding

12 Conclusion

This simulator provides an educationally rich, layered model of network behavior:

- Encapsulation from Application to Physical Layer.
- Modular codebase with reusable components.
- Debug-friendly console logs and test cases.
- Can be extended with Selective Repeat, congestion control, or DNS simulation.

13 References

- CSMA/CD – GeeksforGeeks, TutorialsPoint
- Go-Back-N – GeeksforGeeks
- ARP, IP Routing – Computer Networking: Principles, Protocols and Practice
- Socket Programming – Python Docs, StackOverflow

- Telnet/FTP – RFCs 854, 959 and Python implementations
- https://github.com/thanujann/CSMA-CD-Protocol-Simulator