Enhanced Network Simulator Project Report

Group

Mohammed Mohsen Peerzada
Mohammad Muzamil Bhat
Burhaan Rasheed Zargar

Summary

This project presents a comprehensive enhancement of a basic network simulator, transforming it from a simple Layer 2 frame-forwarding system into a full-featured network simulation environment supporting Layers 2-7 of the OSI model. The enhanced simulator implements modern networking protocols including IPv4, ARP, static/dynamic routing, TCP/UDP transport protocols, and application layer services.

Project Scope and Objectives

Primary Objectives:

- Implement IPv4 addressing with CIDR notation support
- Add ARP protocol for address resolution
- Create router functionality with static and dynamic routing (RIP)
- Implement transport layer protocols (TCP with sliding window, UDP)
- Add application layer services (HTTP, DNS)
- Maintain compatibility with existing switch and hub implementations
- Support diverse network topologies

Delivered Components:

1. **Router Implementation** (`router.c`)
2. **Enhanced End Device** (`end_device.c`)
3. **Updated Network Stack** (`common.h`)
4. **Compatible Switch/Hub** (updated `switch.c` , `hub.c`)

Technical Architecture

Network Stack Implementation

```
┌─────────────────────────────────────┐
│         Application Layer           │
│        (HTTP, DNS Services)         │
├─────────────────────────────────────┤
│         Transport Layer             │
│         (TCP, UDP)                  │
├─────────────────────────────────────┤
│         Network Layer               │
│       (IPv4, ARP, Routing)          │
├─────────────────────────────────────┤
│         Data Link Layer             │
│       (Ethernet Frames)             │
├─────────────────────────────────────┤
│         Physical Layer              │
│      (File-based IPC)               │
└─────────────────────────────────────┘
```

Core Data Structures

Structure	Purpose	Key Fields
`EthernetFrame`	Layer 2 communication	src_mac, dest_mac, ethertype, payload
`IPPacket`	Layer 3 routing	src_ip, dest_ip, protocol, ttl
`ARPPacket`	Address resolution	sender_ip, sender_mac, target_ip

Structure	Purpose	Key Fields
`TCPPacket`	Reliable transport	seq_num, ack_num, flags, window_size
`UDPPacket`	Unreliable transport	src_port, dest_port, length
`RoutingEntry`	Route information	network, netmask, next_hop, metric

Device Working Principles and Commands

1. Router Implementation

Working Principle:

The router operates as a Layer 3 device, making forwarding decisions based on IP addresses. It maintains multiple data structures:

- **ARP Table**: Maps IP addresses to MAC addresses
- **Routing Table**: Contains network routes with next-hop information
- **Port Configuration**: Multiple interfaces with different IP subnets

Router Startup:

```
./router <router_id> <interface1> <ip1> <netmask1> [interface2] [ip2] [netmask2] ...

# Example:
./router router1 eth0 192.168.1.1 255.255.255.0 eth1 192.168.2.1 255.255.255.0
```

Router Operation Flow:

1. **Frame Reception**: Receives Ethernet frames on all interfaces
2. **Frame Processing**: Extracts and validates IP packets
3. **Routing Decision**: Looks up destination in routing table using longest prefix match
4. **ARP Resolution**: Resolves next-hop MAC address if unknown
5. **Packet Forwarding**: Decrements TTL and forwards to appropriate interface

Router Commands:

Command	Purpose	Example
`show arp`	Display ARP table with IP-MAC mappings	`Router> show arp`
`show routes`	Display routing table with all routes	`Router> show routes`
`show ports`	Display interface status and configuration	`Router> show ports`
`route add`	Add static route manually	`route add 192.168.3.0 255.255.255.0 192.168.2.2 eth1`
`rip enable`	Enable RIP dynamic routing protocol	`Router> rip enable`
`rip disable`	Disable RIP protocol	`Router> rip disable`
`help`	Show available commands	`Router> help`
`exit`	Shutdown router gracefully	`Router> exit`

Sample Router Output:

```
--- Routing Table ---
Network         | Netmask         | Next Hop        | Interface | Metric | Type
--------------------------------------------------------------------------------
192.168.1.0     | 255.255.255.0   | 0.0.0.0         | eth0      |      0 | Static
192.168.2.0     | 255.255.255.0   | 0.0.0.0         | eth1      |      0 | Static
192.168.3.0     | 255.255.255.0   | 192.168.2.2     | eth1      |      1 | Static
```

2. Enhanced End Device Implementation

Working Principle:

End devices operate at all OSI layers, providing complete network stack functionality:

- **Network Layer**: Handles IP addressing and ARP
- **Transport Layer**: Implements TCP and UDP protocols
- **Application Layer**: Runs HTTP and DNS services

End Device Startup:

```
./end_device <device_id> <conn_device> <port_num> <ip_address> <netmask>

# Example:
./end_device device1 router1 1 192.168.1.100 255.255.255.0
```

End Device Operation Flow:

1. **Service Initialization**: Starts HTTP (port 80) and DNS (port 53) services
2. **Frame Processing**: Handles incoming Ethernet frames
3. **Protocol Demultiplexing**: Separates ARP, IP, TCP, and UDP traffic
4. **Application Processing**: Routes requests to appropriate service handlers
5. **Response Generation**: Creates and sends response packets

End Device Commands:

Category	Command	Purpose	Example
Network	`ping <dest_ip>`	Send ICMP-like ping to destination	`ping 192.168.1.1`
Network	`arp <target_ip>`	Send ARP request for IP address	`arp 192.168.1.1`
Transport	`http <dest_ip>`	Send HTTP request to web server	`http 192.168.2.100`
Application	`dns <dest_ip> <domain>`	Send DNS query to DNS server	`dns 192.168.1.1 example.com`
Information	`show arp`	Display local ARP table	`Device> show arp`
Information	`show connections`	Display active TCP connections	`Device> show connections`
Information	`show services`	Display running application services	`Device> show services`
System	`help`	Show all available commands	`Device> help`
System	`exit`	Shutdown device gracefully	`Device> exit`

Sample End Device Output:

```
 --- Active Services ---
Port | Protocol | Status
----------------------
  80 | TCP      | Active
  53 | UDP      | Active

--- TCP Connections ---
Remote IP       | Remote Port | Local Port | State
------------------------------------------------
192.168.2.100   |        8080 |         80 | ESTABLISHED
```

3. Switch Implementation

Working Principle:

Switches operate at Layer 2, making forwarding decisions based on MAC addresses:

- **MAC Learning**: Dynamically learns source MAC addresses
- **MAC Table**: Maintains MAC-to-port mappings
- **Frame Forwarding**: Forwards frames based on destination MAC

- **Flooding**: Broadcasts frames for unknown destinations

Switch Startup:

```
./switch <switch_id> <device1_id> [device2_id] ... [deviceN_id]

# Example:
./switch switch1 device1 device2 router1
```

Switch Operation Flow:

1. **Frame Reception**: Receives frames on all ports
2. **Source Learning**: Updates MAC table with source MAC and port
3. **Destination Lookup**: Searches MAC table for destination MAC
4. **Forwarding Decision**:
   - Known MAC: Forward to specific port
   - Unknown MAC: Flood to all ports except source
   - Broadcast: Flood to all ports except source

Switch Commands:

Command	Purpose	Example
`show mac`	Display MAC address table with port mappings	`Switch> show mac`
`show ports`	Display port status and connected devices	`Switch> show ports`
`help`	Show available commands	`Switch> help`
`exit`	Shutdown switch gracefully	`Switch> exit`

Sample Switch Output:

```
--- MAC Address Table ---
MAC Address       | Port | Age (sec)
------------------------------------
AA:BB:CC:DD:EE:FF  |    1 | 15
11:22:33:44:55:66  |    2 | 8
FF:FF:FF:FF:FF:FF  |    3 | 3
------------------------------------
Total entries: 3/16
```

4. Hub Implementation

Working Principle:

Hubs operate at the Physical Layer, creating a single collision domain:

- **Signal Repeating**: Retransmits all received signals
- **No Intelligence**: No MAC learning or filtering
- **Collision Domain**: All connected devices share bandwidth
- **Half-Duplex**: Only one device can transmit at a time

Hub Startup:

```
./hub <hub_id> <device1_id> [device2_id] ... [deviceN_id]

# Example:
./hub hub1 device1 device2 device3
```

Hub Operation Flow:

1. **Frame Reception**: Receives frame on any port
2. **Signal Regeneration**: Amplifies and cleans the signal
3. **Broadcast Transmission**: Forwards frame to ALL other ports
4. **Collision Detection**: Monitors for signal collisions

Hub Commands:

Command	Purpose	Example
`show ports`	Display port status and connected devices	`Hub> show ports`
`help`	Show available commands	`Hub> help`
`exit`	Shutdown hub gracefully	`Hub> exit`

Sample Hub Output:

```
--- Hub Port Status ---
Port | Device ID       | Connected | Input File                 | Output File
-------------------------------------------------------------------------------
   1 | device1         | Yes       | ./tmp/device1_to_hub1_port1.bin | ./tmp/hub1_to_device1_port1.bin
   2 | device2         | Yes       | ./tmp/device2_to_hub1_port2.bin | ./tmp/hub1_to_device2_port2.bin
   3 | device3         | Yes       | ./tmp/device3_to_hub1_port3.bin | ./tmp/hub1_to_device3_port3.bin
```

Protocol Implementation and Command Workflows

ARP Protocol Workflow

```
Device A (192.168.1.100) wants to ping Device B (192.168.1.200)

1. Device A: arp 192.168.1.200
   - Creates ARP Request: Who has 192.168.1.200? Tell 192.168.1.100
   - Broadcasts frame with dest_mac = FF:FF:FF:FF:FF:FF

2. Switch/Hub: Forwards broadcast to all ports

3. Device B: Receives ARP Request
   - Checks: Is 192.168.1.200 my IP? Yes!
   - Sends ARP Reply: 192.168.1.200 is at BB:BB:BB:BB:BB:BB

4. Device A: Receives ARP Reply
   - Updates ARP table: 192.168.1.200 -> BB:BB:BB:BB:BB:BB
   - Can now send ping directly to Device B
```

Routing Protocol Workflow

```
Static Routing Configuration:

Router1> route add 192.168.3.0 255.255.255.0 192.168.2.2 eth1

1. Router validates network and netmask format
2. Adds entry to routing table:
   - Network: 192.168.3.0
   - Netmask: 255.255.255.0
   - Next Hop: 192.168.2.2
   - Interface: eth1
   - Metric: 1 (static)

RIP Dynamic Routing:

Router1> rip enable

1. Router starts RIP timer (30-second intervals)
2. Broadcasts routing table to all interfaces
3. Receives RIP updates from neighbors
4. Updates routing table with better routes
5. Applies split horizon to prevent loops
```

TCP Connection Workflow

```
HTTP Request from Device A to Device B:

Device A> http 192.168.2.100

1. Device A creates TCP connection:
   - SYN packet: seq=1000, ack=0, flags=0x02
   - Sends to 192.168.2.100:80

2. Device B receives SYN:
   - Creates connection state
   - Sends SYN-ACK: seq=2000, ack=1001, flags=0x12

3. Device A receives SYN-ACK:
   - Sends ACK: seq=1001, ack=2001, flags=0x10
   - Connection ESTABLISHED

4. Device A sends HTTP request:
   - PSH-ACK packet with "GET / HTTP/1.1" data

5. Device B HTTP service responds:
   - "HTTP/1.1 200 OK\r\nContent-Length: 13\r\n\r\nHello World!"
```

Application Service Workflow

```
DNS Query Process:

Device A> dns 192.168.1.1 example.com

1. Device A creates UDP packet:
   - Source Port: 32768 (ephemeral)
   - Dest Port: 53 (DNS)
   - Payload: "DNS_QUERY:example.com"

2. Router forwards packet to DNS server

3. DNS server (Device B) processes query:
   - Calls dns_handler() function
   - Generates response: "DNS_RESPONSE:192.168.1.100"

4. DNS server sends UDP response:
   - Source Port: 53
   - Dest Port: 32768
   - Payload: DNS response data

5. Device A receives DNS response
```

Comprehensive Command Reference

Router Commands in Detail

```
# Routing Management
Router> route add 10.0.0.0 255.0.0.0 192.168.1.2 eth0    # Add default route
Router> route add 172.16.0.0 255.255.0.0 192.168.2.1 eth1 # Add specific network

# RIP Protocol
Router> rip enable          # Start RIP protocol
Router> rip disable         # Stop RIP protocol

# Information Display
Router> show routes         # Display complete routing table
Router> show arp           # Display ARP cache
Router> show ports         # Display interface configuration

# System Commands
Router> help               # Show command help
Router> exit               # Graceful shutdown
```

End Device Commands in Detail

```
# Network Layer Testing
Device> ping 192.168.1.1          # Test connectivity
Device> arp 192.168.1.1           # Resolve MAC address

# Transport Layer Testing
Device> http 192.168.2.100        # Send HTTP request
Device> dns 192.168.1.1 google.com # Query DNS server

# Information Commands
Device> show arp                   # Display ARP table
Device> show connections           # Display TCP connections
Device> show services             # Display running services

# System Commands
Device> help                      # Show all commands
Device> exit                      # Shutdown device
```

Switch/Hub Commands

```
# Switch Specific
Switch> show mac                  # Display MAC address table
Switch> show ports               # Display port status

# Hub Specific
Hub> show ports                  # Display port status

# Common Commands
Switch/Hub> help                 # Show available commands
Switch/Hub> exit                 # Shutdown device
```

Feature Implementation Details

1. Router Implementation

Features Implemented:

- **Multi-interface support**: Each router can have multiple network interfaces
- **IPv4 packet forwarding**: Routes packets based on destination IP addresses
- **Longest prefix matching**: Implements proper routing table lookup
- **TTL handling**: Decrements TTL and drops expired packets
- **ARP integration**: Resolves next-hop MAC addresses automatically

2. ARP Protocol Implementation

Features:

- **Dynamic ARP cache**: Automatically learns IP-to-MAC mappings
- **ARP request/reply handling**: Full ARP protocol implementation
- **Cache aging**: Removes stale entries automatically
- **Broadcast ARP requests**: Discovers unknown MAC addresses

3. Routing Protocols

Static Routing

- **Manual configuration**: Administrators can add/remove routes
- **Persistent routes**: Routes remain until manually removed
- **Administrative distance**: Static routes have precedence over dynamic

RIP (Routing Information Protocol)

- **Distance vector algorithm**: Bellman-Ford routing algorithm
- **Periodic updates**: 30-second update intervals
- **Split horizon**: Prevents routing loops
- **Hop count metric**: Maximum 16 hops (infinity)

RIP Update Process:

```
Every 30 seconds:
1. Broadcast routing table to all interfaces
2. Receive updates from neighboring routers
3. Update local routing table if better routes found
4. Apply split horizon to prevent loops
```

4. Transport Layer Protocols

TCP Implementation

- **Connection-oriented**: 3-way handshake establishment
- **Sliding window**: Go-Back-N flow control mechanism
- **Sequence numbers**: Reliable, ordered delivery
- **Connection states**: CLOSED, LISTEN, ESTABLISHED, FIN_WAIT

TCP State Machine:

```
CLOSED → [SYN] → ESTABLISHED
ESTABLISHED → [FIN] → FIN_WAIT → CLOSED
```

UDP Implementation

- **Connectionless**: Simple datagram service
- **Low overhead**: Minimal header processing
- **Best-effort delivery**: No reliability guarantees
- **Port multiplexing**: Multiple applications per host

5. Application Layer Services

HTTP Service (Port 80)

- **Web server functionality**: Responds to HTTP requests
- **TCP-based**: Uses reliable transport layer
- **Simple responses**: Returns "Hello World!" messages

DNS Service (Port 53)

- **Name resolution**: Maps domain names to IP addresses
- **UDP-based**: Uses unreliable but fast transport
- **Query/response model**: Handles DNS queries

6. Port Management

- **Well-known ports**: 0-1023 (HTTP=80, DNS=53)
- **Ephemeral ports**: 32768-65535 for client connections
- **Port binding**: Services bind to specific ports
- **Process isolation**: Multiple processes per device

Testing and Validation

Test Topologies Implemented

1. **Simple Router Network**: 2 devices, 1 router
2. **Multi-Router Network**: Static routing with 2 routers
3. **RIP Dynamic Network**: 3 routers with RIP protocol
4. **Switched Network**: Switch + router + devices
5. **Hub Network**: Hub-based collision domain
6. **Complex Mixed Network**: Combination of all components

Detailed Test Commands

Test Topology 1: Simple Router Network

```
# Terminal 1 - Router
./router router1 eth0 192.168.1.1 255.255.255.0 eth1 192.168.2.1 255.255.255.0

# Terminal 2 - Device A
./end_device deviceA router1 1 192.168.1.100 255.255.255.0

# Terminal 3 - Device B
./end_device deviceB router1 2 192.168.2.100 255.255.255.0

# Test Commands:
# In deviceA: ping 192.168.2.100
# In deviceA: http 192.168.2.100
# In deviceA: dns 192.168.2.100 example.com
# In router1: show arp, show routes
```

Test Topology 6: Complex Mixed Network

```
# Terminal 1 - Router 1
./router router1 eth0 192.168.1.1 255.255.255.0 eth1 192.168.10.1 255.255.255.0

# Terminal 2 - Switch
./switch switch1 deviceA deviceB router1

# Terminal 3 - Router 2
./router router2 eth0 192.168.10.2 255.255.255.0 eth1 192.168.2.1 255.255.255.0

# Terminal 4 - Hub
./hub hub1 deviceC deviceD router2

# Terminal 5-8 - End Devices
./end_device deviceA switch1 1 192.168.1.100 255.255.255.0
./end_device deviceB switch1 2 192.168.1.200 255.255.255.0
./end_device deviceC hub1 1 192.168.2.100 255.255.255.0
./end_device deviceD hub1 2 192.168.2.200 255.255.255.0

# Configure static routes:
# In router1: route add 192.168.2.0 255.255.255.0 192.168.10.2 eth1
# In router2: route add 192.168.1.0 255.255.255.0 192.168.10.1 eth0

# Comprehensive Test Commands:
# In deviceA: ping 192.168.2.100
# In deviceB: http 192.168.2.200
# In deviceC: dns 192.168.1.100 example.com
# In all devices: show arp, show connections, show services
# In all routers: show routes, show arp, show ports
# In switch: show mac, show ports
# In hub: show ports
```

Implementation Challenges and Solutions

1. Inter-Process Communication

Challenge: Maintaining file-based IPC while adding complex protocols
Solution: Enhanced frame structures with backward compatibility

2. Protocol Stack Integration

Challenge: Layered protocol implementation without breaking existing code
Solution: Modular design with clear interface boundaries

3. Concurrency Management

Challenge: Multiple threads accessing shared data structures
Solution: Mutex-protected critical sections for all shared resources

4. Memory Management

Challenge: Preventing memory leaks in long-running processes
Solution: Careful resource allocation/deallocation and cleanup routines

Command Summary for Quick Reference

Device Startup Commands

```
# Router
./router <id> <if1> <ip1> <mask1> [if2] [ip2] [mask2] ...

# End Device
./end_device <id> <conn_device> <port> <ip> <mask>

# Switch
./switch <id> <device1> [device2] ... [deviceN]

# Hub
./hub <id> <device1> [device2] ... [deviceN]
```

Operational Commands

```
# Router Commands
show arp, show routes, show ports
route add <net> <mask> <next_hop> <if>
rip enable, rip disable

# End Device Commands
ping <ip>, arp <ip>
http <ip>, dns <ip> <domain>
show arp, show connections, show services

# Switch Commands
show mac, show ports

# Hub Commands
show ports

# Universal Commands
help, exit
```

Future Enhancement Opportunities

Short-term Enhancements

1. **OSPF Protocol**: More advanced dynamic routing
2. **VLAN Support**: Virtual LAN implementation
3. **QoS Features**: Traffic prioritization and shaping
4. **DHCP Service**: Dynamic IP address assignment

Long-term Enhancements

1. **IPv6 Support**: Next-generation IP protocol
2. **Wireless Simulation**: WiFi and cellular network models
3. **Security Protocols**: IPSec, TLS/SSL implementation
4. **Network Management**: SNMP protocol support
5. **GUI Interface**: Graphical network topology visualization

Conclusion

This enhanced network simulator successfully transforms a basic frame-forwarding system into a comprehensive networking education platform. The implementation demonstrates:

- **Complete protocol stack** from physical to application layers
- **Industry-standard protocols** (IPv4, ARP, RIP, TCP, UDP, HTTP, DNS)
- **Scalable architecture** supporting various network topologies
- **Educational value** through interactive learning and real-time feedback
- **Production-quality code** with proper error handling and thread safety
- **Comprehensive command interface** for all network operations
- **Real-world protocol behavior** with proper state machines and timers

The simulator serves as an excellent platform for networking education, allowing students to experiment with real network protocols in a controlled environment. The detailed command interface and comprehensive logging provide deep insights into network protocol operation, making it an invaluable tool for understanding how modern networks function.