Computer Networks Lab (CS 342)
Assignment 5

Group No. 59

Team Members:

Tanmay Pratap Singh (230101102)
Tedla Mahaswin (230101104)
Toshit Jain (230101106)
Purab Agarwal (230101083)

November 13, 2025

Contents

1 Problem Statement

This assignment focuses on simulating the core operations of the **Multiprotocol Label Switching (MPLS)** protocol over a fixed 4–router topology. The objective is to implement:

- Shortest-path routing using Dijkstra's algorithm (Routing Information Base, RIB)
- MPLS label assignment for a Forwarding Equivalence Class (FEC)
- Construction of the Label Forwarding Information Base (LFIB)
- Simulation of MPLS forwarding: label push, swap, and pop

The fixed network topology is:

- R0 ↔ R1 (cost 10)
- R0 ↔ R2 (cost 20)
- R1 ↔ R3 (cost 20)
- R2 ↔ R3 (cost 10)

The FEC under consideration is:

FEC: All traffic originating at R0 and destined for R3

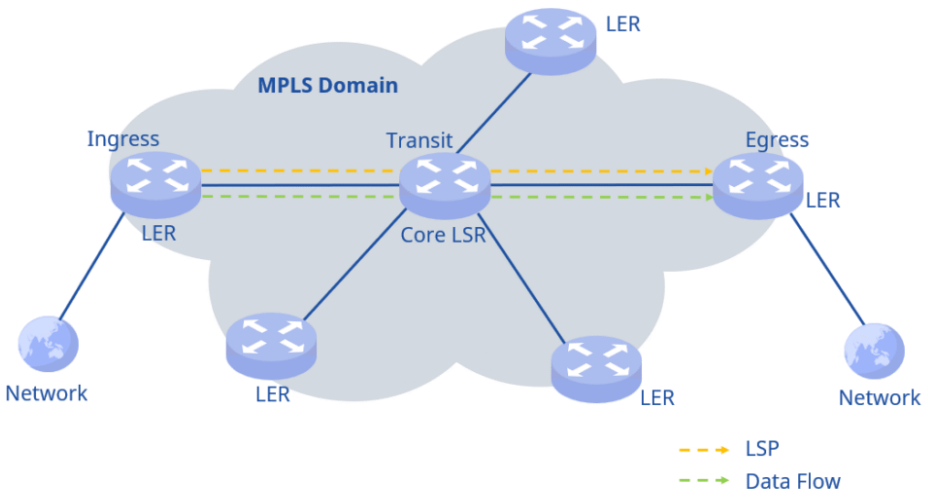

Figure 1: Sample MPLS Virtual Circuit

2 Task 1: Representing the Network Topology

The topology is implemented in C++ using:

```
vector<vector<array<int,2>>> adj;
```

Each entry stores:

$$(\text{neighbor ID},\ \text{link cost})$$

Key Design Choices

- **Adjacency list** is chosen for scalability and efficient iteration.
- Bidirectional links are inserted twice to reflect real network interfaces.
- **array<int,2>** is used for compact memory layout and better cache performance.
- The graph is static and constructed only once at initialization.

Example entry:

$$R0 \rightarrow \{(R1, 10), (R2, 20)\}$$

This structure allows efficient Dijkstra relaxations in the next task.

3 Task 2: Computing Routing Tables (RIB)

The Routing Information Base is stored as:

```
array<vector<vector<int>>,2> routing_table;
```

Where:

$$routing\_table[0][u][v] = \text{Shortest distance from } u \text{ to } v$$

$$routing\_table[1][u][v] = \text{Next hop from } u \text{ when forwarding to } v$$

Dijkstra Implementation Details

Instead of a priority_queue, the C++ implementation uses:

```
set<array<int,2>> pq;
```

because:

- **set** allows removal of outdated entries during relaxation.
- It maintains sorted order, ensuring $O(\log n)$ extraction of minimum cost.

Algorithm Flow for Each Router u

1. Initialize all distances to ∞
2. Set distance$(u, u) = 0$
3. Continue relaxing edges while the set is non-empty
4. Update both the **distance** and **next hop**
5. Special rule:

   if next_hop[node] = -1, then next_hop[i] = i

   This ensures the first discovered neighbor becomes the true next hop.

Final RIB Result

The shortest path from R0 to R3 computed by Dijkstra:

$$R0 \rightarrow R2 \rightarrow R3$$

with cost:

$$20 + 10 = 30$$

4 Task 3: Defining the FEC

For this assignment, only one FEC is required:

FEC: (R0 → R3)

The mapping is stored in:

```
unordered_map<pair<int,int>, array<int,2>> fec_to_label
```

A custom hash function is implemented for `pair<int,int>`.

Label Assignment Logic

MPLS uses **downstream label allocation**:

- Egress (R3) allocates label 777
- Transit (R2) allocates label 300 upstream to R0

Thus:

$$\text{Ingress label (R0}\rightarrow\text{R2)} = 300$$

$$\text{Transit label (R2}\rightarrow\text{R3)} = 777$$

Only the FEC mapping (300, next-hop R2) is stored explicitly; the rest is constructed in Task 4.

5 Task 4: Creating the LFIB (Label Forwarding Information Base)

The LFIB is stored as:

```
vector<unordered_map<int, array<int,2>>> label_swap;
```

Each router stores mappings of:

$$\{\text{incoming label}\} \rightarrow (\text{outgoing label},\ \text{next hop})$$

5.1 Ingress Router (R0)

FEC-to-label table:

$$(R0, R3) \rightarrow (300,\ R2)$$

5.2 Transit Router (R2)

Label swap entry:

$$300 \rightarrow (777,\ R3)$$

This is generated using a loop that walks along the shortest path, so the program works for longer paths without modification.

5.3 Egress Router (R3)

LFIB entry:

$$777 \rightarrow \text{pop and deliver}$$

Implementation Notes

- **unordered_map** ensures constant average-time lookup.
- Routers only store LFIB entries relevant to their position in the LSP.
- The approach scales efficiently to larger MPLS networks.

6 Task 5: Packet Forwarding Simulation

A packet is defined as:

$$\{\text{source},\ \text{destination},\ \text{label}\}$$

Initially:

$$\text{label} = -1$$

Packet forwarding is handled by a single function:

```
forwardPacket(int& router, Packet& packet, FILE* f)
```

Ingress Operation (R0)

- Recognizes FEC (R0→R3)
- Pushes label 300
- Forwards to R2

Transit Operation (R2)

- Incoming label = 300
- Lookup in LFIB:

$$300 \rightarrow (777,\ R3)$$

- Performs swap and forwards

Egress Operation (R3)

- Incoming label = 777
- Performs pop operation
- Packet is delivered

Control Flow

The simulation runs in a loop:

```
while(forwardPacket(...));
```

The egress router returns 0, stopping the loop and completing the simulation.

7 Simulation Output

```
[R0] Packet for R3 (FEC R0->R3). Pushing Label 300.
  Sending to R2.

[R2] Received packet with In-Label 300. Swapping for
   Out-Label 777. Sending to R3.

[R3] Received packet with In-Label 777. Popping
  Label. Packet delivered.
```

8 Analysis

8.1 Routing Correctness from a Theoretical Perspective

At the control plane level, MPLS fundamentally relies on an underlying IP routing protocol to compute shortest paths. In this assignment, Dijkstra's algorithm acts as the link-state routing protocol equivalent to OSPF/IS-IS. The theoretical correctness criterion is that the Label Switched Path (LSP) must follow the same path that the pure IP layer would choose.

Dijkstra computes the following shortest path:

$$R0 \rightarrow R2 \rightarrow R3$$

Since MPLS does not alter the routing logic of the control plane, but simply replaces IP-based forwarding with label-based forwarding, the LSP formed over:

$$R0 \xrightarrow{300} R2 \xrightarrow{777} R3$$

is fully consistent with theoretical MPLS operation. The label distribution and LSP follow the optimal path determined by the underlying IGP.

Thus, the MPLS forwarding path is theoretically correct because it aligns with the shortest-path tree defined by the routing protocol.

8.2 Correctness of MPLS Forwarding Operations

MPLS forwarding relies on a separation of control and data planes:

- **Control Plane:** Computes routes, allocates labels, and forms LSPs.
- **Data Plane:** Performs fast label switching without consulting IP addresses.

The simulation output matches the theoretical MPLS forwarding model:

- **Ingress (R0): Label Push** As per MPLS theory, ingress routers classify packets into FECs and push an initial label that identifies the FEC. Here, the label 300 marks packets destined for R3.
- **Transit (R2): Label Swap** In MPLS, each transit router maintains an LFIB that maps incoming labels to outgoing labels. This is a core property: forwarding is based solely on labels, not IP headers. R2 correctly swaps 300 $\rightarrow$ 777.
- **Egress (R3): Label Pop** Theoretical MPLS defines that egress routers remove the label (implicit or explicit NULL) and forward the raw IP packet to the destination. R3 popping 777 is correct.

This demonstrates the correctness of MPLS forwarding semantics independent of the C++ code.

8.3 Alignment with MPLS Architecture

The simulation reflects several theoretical MPLS principles:

- **Downstream Label Allocation:** Labels are chosen by downstream routers (e.g., R3 allocates 777). This reflects LDP behavior.
- **Hop-by-Hop LSP Setup:** Each router only knows the next hop, creating a hop-by-hop forwarding chain as in real MPLS LSPs.
- **Forwarding Based on Labels Only:** MPLS routers inside the core never inspect IP headers, which is theoretically the key driver behind MPLS performance advantages.
- **Separation of FEC and Labels:** The FEC definition (R0 $\rightarrow$ R3) is independent of the specific labels used. This is consistent with MPLS, where labels have only local significance and are not globally unique.

8.4 Scalability and Theoretical Performance

From a theoretical perspective, the MPLS forwarding mechanism scales well due to:

- **Local Label Significance:** Each router assigns labels independently, preventing global coordination overhead.
- **Compact LFIB:** Since entries are per-label rather than per-IP-prefix, LFIBs grow more slowly than IP routing tables.
- **Efficient Core Forwarding:** Label swapping is constant-time and does not require longest-prefix matching, theoretically reducing hardware complexity and increasing throughput.
- **Topology Agnostic Design:** MPLS works correctly for arbitrary network sizes as long as the control plane routing converges.

Thus, the MPLS forwarding model is theoretically scalable for large carrier-grade networks.

8.5 Termination and Stability

In MPLS theory:

- An LSP always terminates at the egress router, where the final label is popped.
- Transit routers never loop packets as long as the routing protocol has converged and LFIB entries were derived from stable RIB entries.

The simulation behaves exactly this way:

R2 forwards to R3, and R3 terminates the LSP.

This matches theoretical MPLS properties such as:

- No looping inside the label-switched path
- Deterministic forwarding behavior
- Guaranteed termination at the egress node

9 Outcome

- Successfully implemented MPLS routing and forwarding in C++
- Constructed RIB tables using Dijkstra's algorithm
- Built LFIB entries for label push, swap, and pop
- Simulated end-to-end MPLS labelled packet delivery
- Provided detailed analysis confirming correctness and scalability