Assignment 5

CS342: Computer Networks Lab

MPLS Label Forwarding Simulation

Group 50

Aeyaz Adil (230101007)
Deval Singhal (230101035)
Jayant Kumar (230101050)
Prayansh Kumar (230101081)

Contents

Chapter 1

Multiprotocol Label Switching (MPLS)

1.1 Introduction

The primary objective of this assignment is to simulate and understand the fundamental packet forwarding process of the **Multiprotocol Label Switching (MPLS)** protocol. This involves three main steps based on a fixed 4-router topology:

- Compute the **Routing Information Base (RIB)** for all routers using **Dijkstra's algorithm**.
- Define and construct the **Label Forwarding Information Base (LFIB)** entries for a specific **Forwarding Equivalence Class (FEC): traffic from R0 destined for R3**.
- Trace the path of a labeled packet through the MPLS domain using the generated LFIBs.

1.2 MPLS Overview

MPLS is a data-forwarding technique that, instead of relying on complex lookups of the destination IP address at every hop (traditional IP routing), relies on short, fixed-length labels. This shift allows for extremely fast forwarding decisions.

- **Ingress Router:** When a packet enters an MPLS network, the first router (the "Ingress" router) analyzes its destination IP (and other factors) and assigns it to a Forwarding Equivalence Class (FEC). It then "pushes" (adds) a label to the packet.
- **Transit Router:** Routers inside the network ("Transit" routers) don't look at the IP address. They only look at the label. They use the incoming label to index a simple table (the LFIB) which tells them what new label to "swap" it with and which outgoing interface to send it to. This is extremely fast.
- **Egress Router:** The last router in the path ("Egress" router) "pops" (removes) the label and forwards the original IP packet to its final destination.

1.3 Fixed Network Topology

The simulation is based on the following fixed topology, represented as a graph where nodes are routers and edges are bidirectional links with associated costs.

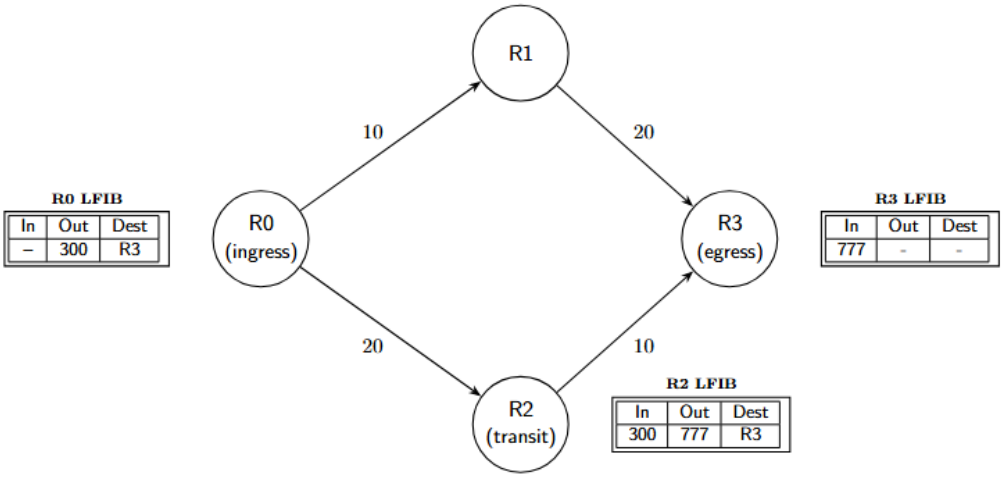

Figure 1.1: 4-Router MPLS topology with local LFIB (MPLS forwarding) tables

1.4 Shortest Path and Label Assignment

Based on the topology, the shortest-cost path for FEC R0 → R3 is calculated as:

$$R0 \rightarrow R2 \rightarrow R3 \quad \text{(Total Cost : } 20 + 10 = 30\text{)}$$

The following label assignments, simulating the result of a Label Distribution Protocol (LDP), are used to build the LFIBs:

- **R0 (Ingress):** Push Label **300** (Send to R2)
- **R2 (Transit):** In-Label 300 → Swap to Label **777** (Send to R3)
- **R3 (Egress):** In-Label 777 → **Pop** Label (Deliver Locally)

1.5 Compilation and Execution

The following steps are required to compile and run the simulation program.
Compilation: The source code (`mpls_sim.cpp`) is compiled using the g++ compiler:

```
g++ -std=c++11 -o mpls_sim mpls_sim.cpp
```

Execution: The program is run directly from the command line, which outputs both the routing tables (RIBs) and the packet forwarding trace.

```
./mpls_sim
```

The output is is saved to a file named `output.txt` as per the assignment requirements.

1.6 MPLS Code Simulation

1.6.1 Task 1: Represent the Topology

This task involves initializing the adjacency matrix (`topologyMatrix`) with the fixed link costs.

```
void initializeTopology() {
    topologyMatrix.assign( n: NUM_ROUTERS,  val: vector<int>( n: NUM_ROUTERS,  value: INF));

    for (int i = 0; i < NUM_ROUTERS; ++i) {
        topologyMatrix[i][i] = 0;
    }
    // R0 <-> R1 (Cost: 10)
    topologyMatrix[0][1] = 10;
    topologyMatrix[1][0] = 10;
    // R0 <-> R2 (Cost: 20)
    topologyMatrix[0][2] = 20;
    topologyMatrix[2][0] = 20;
    // R1 <-> R3 (Cost: 20)
    topologyMatrix[1][3] = 20;
    topologyMatrix[3][1] = 20;
    // R2 <-> R3 (Cost: 10)
    topologyMatrix[2][3] = 10;
    topologyMatrix[3][2] = 10;
}
```

Figure 1.2: Initializing the router topology

1.6.2 Task 2: Compute Routing Tables (RIB)

This task implements Dijkstra's algorithm (`runDijkstra`) and the associated printing logic (`printRoutingTableRows`) to calculate and display the shortest paths for all routers.

```
void runDijkstra(int sourceRouter) {
    vector<int> dist( n: NUM_ROUTERS,  value: INF);
    vector<int> prev( n: NUM_ROUTERS,  value: -1);
    using pii = pair<int, int>;
    priority_queue<pii, vector<pii>, greater<>> pq;

    dist[sourceRouter] = 0;
    pq.push( x: { x: 0,  &: sourceRouter});

    while (!pq.empty()) {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;

        for (int v = 0; v < NUM_ROUTERS; ++v) {
            if (topologyMatrix[u][v] != INF) {
                int newDist = d + topologyMatrix[u][v];
                if (newDist < dist[v]) {
                    dist[v] = newDist;
                    prev[v] = u;
                    pq.push( x: { &: newDist,  &: v});
                }
            }
        }
    }
```

Figure 1.3: Dijkstra's algorithm

```
void printRoutingTable(int sourceRouter, const vector<int> &prev,
                       const vector<int> &dist) {
    cout << left; // Left-align text

    for (int i = 0; i < NUM_ROUTERS; ++i) {
        cout << " " << setw( n: 12) << ("R" + to_string( val: i));
        cout << "|";
        if (dist[i] == INF) {
            cout << " " << setw( n: 9) << "-";
            cout << "|";
            cout << " " << setw( n: 10) << "INF" << endl;
            continue;
        }
        if (i == sourceRouter) {
            cout << " " << setw( n: 9) << ("R" + to_string( val: i));
            cout << "|";
            cout << " " << setw( n: 10) << 0 << endl;
            continue;
        }
        int curr = i;
        while (prev[curr] != sourceRouter && prev[curr] != -1) {
            curr = prev[curr];
        }
        cout << " " << setw( n: 9) << ("R" + to_string( val: curr));
        cout << "|";
        cout << " " << setw( n: 10) << dist[i] << endl;
    }
    cout << endl;
}
```

Figure 1.4: Printing Routing Table

1.6.3 Task 3: Define a Forwarding Equivalence Class (FEC)

In this task, we define a class FEC to represent a Forwarding Equivalence Class, which groups packets that receive identical forwarding treatment within an MPLS network. The class encapsulates the source and destination router identifiers.

For this simulation, we focus on a single FEC: all traffic originating from R0 and destined for R3. This FEC will serve as the basis for label assignment and MPLS forwarding decisions in subsequent tasks.

```
class FEC {
public:
    int source;
    int destination;

    // Default constructor
    FEC() : source(0), destination(0) {}

    // Parameterized constructor
    FEC(int src, int dst) : source(src), destination(dst) {}

    /**
     * Comparison operator is required to use FEC as a key in std::map.
     * It defines how to sort FECs.
     */
    bool operator<(const FEC &other) const {
        if (source != other.source) {
            return source < other.source;
        }
        return destination < other.destination;
    }
};
```

Figure 1.5: Forwarding Equivalence Class

1.6.4 Task 4: Create Label Forwarding Tables

This task initializes the LFIBs (`ingressLFIB` and `transitLFIB`) for R0, R2, and R3 with the specified push, swap, and pop label operations.

```
void initializeLFIB() {
    // R0 (Ingress)
    // FEC (R0->R3) -> Push Label 300, Next Hop R2
    ingressLFIB[FEC( src: 0,  dst: 3)] = LabelOp( label: 300,  hop: 2);

    // R2 (Transit)
    // In-Label 300 -> Swap for Label 777, Next Hop R3
    transitLFIB[2][300] = LabelOp( label: 777,  hop: 3);

    // R3 (Egress)
    // In-Label 777 -> Pop Label
    transitLFIB[3][777] = LabelOp( label: POP_LABEL,  hop: -1);
}
```

Figure 1.6: Creating Label Forwarding Information Base (LFIB) for each router

1.6.5 Task 5: Simulate Packet Forwarding

This task includes the functions that simulate the packet processing at each router (R0, R2, R3) and the main function that coordinates the simulation.

```cpp
class Packet {
public:
    int source;
    int destination;
    int label;

    Packet(int src, int dst) : source(src), destination(dst), label(NO_LABEL) {}
};
```

(a) Network Packet

```cpp
class LabelOp {
public:
    int outLabel;
    int nextHop;

    // Default constructor
    LabelOp() : outLabel(0), nextHop(-1) {}

    // Parameterized constructor
    LabelOp(int label, int hop) : outLabel(label), nextHop(hop) {}
};
```

(b) Label Operation

```cpp
void processR0(Packet &packet) {
    // We now use the FEC class constructor
    FEC fec( src: packet.source,  dst: packet.destination);

    LabelOp operation = ingressLFIB.at( k: fec);
    int newLabel = operation.outLabel;
    int nextHop = operation.nextHop;

    packet.label = newLabel; // Push operation

    cout << "[R0] Packet for R3 (FEC: R0->R3). "
         << "Pushing Label " << packet.label
         << ". Sending to R" << nextHop << "." << endl;

    processR2( &: packet);
}
```

(c) Ingress Router (R0)

```cpp
void processR2(Packet &packet) {
    int inLabel = packet.label;

    LabelOp operation = transitLFIB[2].at( k: inLabel);
    int newLabel = operation.outLabel; // Swap operation
    int nextHop = operation.nextHop;

    packet.label = newLabel; // Apply the swap

    cout << "[R2] Received packet with In-Label " << inLabel
         << ". Swapping for Out-Label " << packet.label
         << ". Sending to R" << nextHop << "." << endl;

    processR3( &: packet);
}
```

(d) Transit Router (R2)

```cpp
void processR3(Packet &packet) {
    int inLabel = packet.label;

    LabelOp operation = transitLFIB[3].at( k: inLabel);
    int action = operation.outLabel;

    if (action == POP_LABEL) {
        packet.label = NO_LABEL; // Pop operation
    }

    cout << "[R3] Received packet with In-Label " << inLabel
         << ". Popping label. Packet delivered." << endl;
}
```

Figure 1.8: Egress Router (R2]3)

1.7 MPLS Simulation Output

1.7.1 Routing Information Base (RIB) Computation (Task 2)

The RIB for all four routers (R0, R1, R2, R3) was computed using Dijkstra's shortest path algorithm based on link costs. The output below shows the calculated shortest path to every other router, its next hop, and the total cost.

```
Routing Table for Router R0:
 Destination | Next Hop | Total Cost
-----------------------------------
 R0          | R0       | 0
 R1          | R1       | 10
 R2          | R2       | 20
 R3          | R1       | 30
```

(a) Router R0

```
Routing Table for Router R1:
 Destination | Next Hop | Total Cost
-----------------------------------
 R0          | R0       | 10
 R1          | R1       | 0
 R2          | R0       | 30
 R3          | R3       | 20
```

(b) Router R1

```
Routing Table for Router R2:
 Destination | Next Hop | Total Cost
-----------------------------------
 R0          | R0       | 20
 R1          | R3       | 30
 R2          | R2       | 0
 R3          | R3       | 10
```

(c) Router R2

```
Routing Table for Router R3:
 Destination | Next Hop | Total Cost
-----------------------------------
 R0          | R2       | 30
 R1          | R1       | 20
 R2          | R2       | 10
 R3          | R3       | 0
```

(d) Router R3

Figure 1.9: Routing Information Base (RIB) for all routers

1.7.2 Packet Forwarding Simulation (Task 5)

A single packet belonging to the FEC R0 → R3 was simulated through the MPLS domain. The trace below shows the label operations at each router.

```
Simulating packet from R0 to R3...
[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.
```

Figure 1.10: Console output for packet simulation

1.8 Conclusion

The simulation successfully demonstrated the core mechanics of MPLS forwarding.

- The RIBs were accurately computed using Dijkstra's algorithm, establishing the shortest path for all source-destination pairs.
- The statically defined LFIBs correctly implemented the required Label Switched Path (LSP) for the R0 $\rightarrow$ R3 FEC.
- The packet trace confirmed the label operations: **Push** at the Ingress (R0), **Swap** at the Transit (R2), and **Pop** at the Egress (R3), confirming the basic high-speed forwarding principle of MPLS.