CS425: Computer Networks
Assignment 4: Routing Protocols (DVR and LSR)

Team Members: LOHIT P TALAVAR (210564), ANJALI MALOTH (210146)

April 21, 2025

Objective

Implement Distance Vector Routing (DVR) and Link State Routing (LSR) algorithms in C++ by reading an adjacency-matrix representation of a network and producing per-node routing tables.

Assignment Repository on GitHub

Repository Structure

```
.
routing_sim.cpp     % C++ source implementing DVR and LSR
Makefile            % (optional) build rules
input1.txt          % sample adjacency matrix input
input2.txt          % sample adjacency matrix input
input3.txt          % sample adjacency matrix input
input4.txt          % sample adjacency matrix input
A4.pdf              % problem statement
README.tex          % This is LaTeX documentation
README.pdf          % This is documentation
```

Prerequisites

- C++17-compatible compiler (e.g. `g++`)
- Standard C++ library (no external dependencies)
- Unix-style shell (e.g. Bash) or equivalent

Compilation

Using Makefile

```
make
```

Manual

```
g++ -std=c++17 routing_sim.cpp -o routing_sim
```

or

```
g++ routing_sim.cpp -o routing_sim
```

Usage

```
./routing_sim <input_file>
```

Example:

```
./routing_sim inputfile.txt
```

Input Format

1. First line: integer n, number of nodes.
2. Next n lines: each contains n space-separated integers (the adjacency matrix):
   - Off-diagonal 0 indicates no link (treated as infinite cost = 9999).
   - Value 9999 explicitly represents infinite cost.
   - Diagonal entries must be 0.

Output Format

Distance Vector Routing (DVR)

```
--- Distance Vector Routing Simulation ---
--- DVR iteration k ---
Node i Routing Table:
Dest    Cost    Next Hop
...
--- DVR Final Tables ---
```

Link State Routing (LSR)

```
--- Link State Routing Simulation ---
Instructor: Adithya Vadapalli TA Incharge: Rishit and Yugul
CS425: Computer Networks A4: Routing Protocols (DVR and LSR)
Node i Routing Table:
```

```
Dest    Cost    Next Hop
...
```

Algorithm Overview

Distance Vector Routing

1. Initialize each node's cost vector with direct link costs and next-hop pointers.
2. Repeatedly exchange vectors with neighbors and update via Bellman-Ford until convergence:

   $if dist_u[v] + dist_v[d] < dist_u[d] then dist_u[d] = dist_u[v] + dist_v[d],\ nextHop_u[d] = nextHop_u[v].$

3. Terminate when no changes occur.

Link State Routing

1. Each node knows the full topology (adjacency matrix).
2. Run Dijkstra's algorithm from each node to compute shortest paths.
3. Backtrack predecessor array to determine first-hop next-hops.

Code Workflow

1. Reading the Graph

- `readGraph(const string&)` opens the input file, reads n, then the $n \times n$ matrix.
- Off-diagonal zeros ($i \neq j$) are converted to `INF` = 9999; diagonal stays 0.

2. Distance Vector Routing (`simulateDVR`)

- Initialize `dist`$[i][j]$ to direct costs, and `nextHop`$[i][j] = j$ when a link exists.
- Iteratively "exchange" tables: for each node u and neighbor v, relax

$$dist_u[d] \ = \ \min(dist_u[d],\ dist_u[v] + dist_v[d]),$$

  updating $\texttt{nextHop}_u[d]$ accordingly.
- Repeat until no update; print tables each iteration and final.

3. Link State Routing (`simulateLSR`)

- For each source node s:
  1. Initialize `dist`$[s] = 0$, others = INF; `prev`$[] = -1$.
  2. Run standard Dijkstra's algorithm using the full adjacency matrix.

3. After computing `dist[]` and `prev[]`, backtrack from each destination d to s to find the first hop.

- Print routing table for each s.

4. Main Function

- Parses command-line argument for input file.
- Calls `readGraph`, then `simulateDVR` and `simulateLSR`.