Department of Computer Science and Engineering

University of Dhaka

CSE 3111 - Computer Networks Lab

Lab Report

Design and Implementation of a

Distance Vector Routing Algorithm

Kabya Mithun Saha
Roll: 16
Muhaiminul Islam Ninad
Roll: 43

Submitted To:

Dr. Ismat Rahman
Mr. Palash Roy
Mr. Jargis Ahmed

Submission Date: June 25, 2025

Contents

1 Introduction

Distance Vector (DV) Routing is a fundamental dynamic routing protocol that operates on a distributed basis, where each router maintains and shares its knowledge of network topology with immediate neighbors. This protocol is based on the Bellman-Ford algorithm and forms the foundation for many early routing protocols, including the Routing Information Protocol (RIP).

In Distance Vector routing, each router maintains a vector of distances (costs) to every other router in the network. The protocol operates through periodic exchanges of routing information between neighboring routers, allowing the network to converge on optimal paths. The core principle follows the Bellman-Ford equation:

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\} \tag{1}$$

where $D_x(y)$ represents the cost from node x to node y, $c(x, v)$ indicates the cost from x to its neighbor v, and $D_v(y)$ denotes neighbor v's distance to destination y.

The protocol includes several key mechanisms: periodic updates where each node sends its routing table to neighbors, incremental updates triggered by topology changes, and loop prevention techniques such as Poison Reverse. Despite its simplicity, DV routing faces challenges including slow convergence and the count-to-infinity problem, making it essential to understand both its capabilities and limitations.

2 Objectives

The primary objectives of this laboratory experiment are:

1. **Understanding DV Routing Principles**: To gain comprehensive knowledge of how Distance Vector routing operates in distributed networks, including the underlying Bellman-Ford algorithm mechanics.

2. **Implementation and Simulation**: To design and implement a complete simulation of Distance Vector routing that demonstrates periodic updates, dynamic cost changes, and convergence behavior in a multi-router network environment.

3. **Loop Prevention Analysis**: To implement and evaluate the effectiveness of Poison Reverse technique in preventing routing loops, understanding its role in maintaining network stability.

4. **Performance Evaluation**: To analyze convergence characteristics, message complexity, and the impact of dynamic link cost changes on network behavior, providing insights into the protocol's practical performance.

3 Detailed Description of the Program

3.1 Program Structure and Architecture

The implementation follows an object-oriented design with two primary classes that encapsulate the core functionality:

Router Class: This class represents individual network routers and contains:

- `id`: Unique router identifier (String)
- `neighbors`: Dictionary mapping neighbor IDs to link costs
- `routing_table`: Complete routing information with destination, cost, and next hop
- `distance_vector`: Current known costs to all destinations

Network Class: This class manages the overall network simulation:

- `routers`: Dictionary containing all router instances
- `topology_file`: Configuration file specifying network topology
- `message_count`: Counter for total messages exchanged
- `logs`: Historical record of all routing changes

3.2 Data Structures

The routing table implementation uses nested dictionaries to efficiently store and access routing information:

```
routing_table = {
    'destination': {
        'cost': integer_cost,
        'next_hop': neighbor_id
    }
}
```

Listing 1: Routing Table Data Structure

The distance vector maintains a simplified view for neighbor communication:

```
distance_vector = {
    'destination': cost_value
}
```

Listing 2: Distance Vector Structure

3.3 Bellman-Ford Algorithm Implementation

The core routing update mechanism implements the Bellman-Ford algorithm through the `update_routing_table` method:

```
def update_routing_table(self, from_neighbor, received_vector):
    changes_made = False
    neighbor_cost = self.neighbors.get(from_neighbor, float('inf'))

    for dest, received_cost in received_vector.items():
        if dest == self.id:
            continue

        # Bellman-Ford update: D_x(y) = min(D_x(y), c(x,v) + D_v(y))
        new_cost = neighbor_cost + received_cost
        current_cost = self.routing_table.get(dest, {'cost': float('inf
            ')})['cost']

        if new_cost < current_cost:
            # Found a better path
            self.routing_table[dest] = {'cost': new_cost, 'next_hop':
                from_neighbor}
            self.distance_vector[dest] = new_cost
            changes_made = True

    return changes_made
```

Listing 3: Bellman-Ford Update Implementation

3.4 Poison Reverse Implementation

Poison Reverse is implemented in the `get_distance_vector_for_neighbor` method to prevent routing loops:

```
def get_distance_vector_for_neighbor(self, neighbor_id):
    vector = copy.deepcopy(self.distance_vector)

    # Apply poison reverse
    for dest, route_info in self.routing_table.items():
        if route_info['next_hop'] == neighbor_id and dest != self.id:
            vector[dest] = float('inf')

    return vector
```

Listing 4: Poison Reverse Implementation

This ensures that if router A uses router B to reach destination C, then A advertises the route to C as having infinite cost when sending updates to B.

3.5 Periodic Updates and Dynamic Changes

The simulation implements two types of periodic behavior:

1. **Routing Updates**: Every 5 seconds, all routers exchange distance vectors with their neighbors

2. **Cost Changes**: Every 30 seconds, a random link cost is modified to simulate network dynamics

The main simulation loop coordinates these activities:

```
def run_simulation(self, duration=120, update_interval=5,
    cost_update_interval=30):
     while current_time < duration:
         # Periodic distance vector updates
         if current_time - last_update_time >= update_interval:
             messages_sent, changes_made = self.send_distance_vectors()
             if changes_made:
                 self.print_all_routing_tables(current_time)

         # Periodic cost updates
         if current_time - last_cost_update_time >= cost_update_interval
             :
             self.update_random_link_cost()
```

Listing 5: Simulation Loop Structure

4 Network Topology Used

The simulation uses the network topology specified in the lab manual, defined in `topology.txt`:

```
A B 2
A C 5
B C 1
B D 3
C D 2
```

This configuration creates an undirected graph with the following connections:

- Router A connects to B (cost 2) and C (cost 5)
- Router B connects to A (cost 2), C (cost 1), and D (cost 3)
- Router C connects to A (cost 5), B (cost 1), and D (cost 2)
- Router D connects to B (cost 3) and C (cost 2)

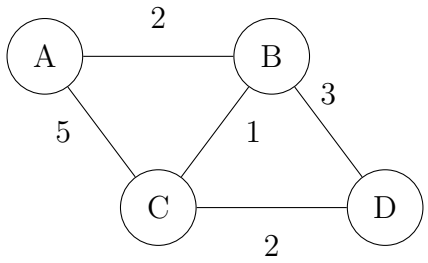

Figure 1: Network Topology Diagram

5 Screenshots and Console Output

5.1 Initial Routing Tables

Initial Routing Tables

```
============================================================
INITIAL ROUTING TABLES at Time = 0.0s
============================================================

[Time = 0.0s] Routing Table at Router A:
Dest | Cost | Next Hop
-------------------------
   A |    0 | A
   B |    2 | B
   C |    5 | C
   D |  inf | None

[Time = 0.0s] Routing Table at Router B:
Dest | Cost | Next Hop
-------------------------
   A |    2 | A
   B |    0 | B
   C |    1 | C
   D |    3 | D

[Time = 0.0s] Routing Table at Router C:
Dest | Cost | Next Hop
-------------------------
   A |    5 | A
   B |    1 | B
   C |    0 | C
   D |    2 | D

[Time = 0.0s] Routing Table at Router D:
Dest | Cost | Next Hop
-------------------------
   A |  inf | None
   B |    3 | B
   C |    2 | C
   D |    0 | D
```

5.2 Routing Tables After First Update

UPDATED ROUTING TABLES

```
============================================================
UPDATED ROUTING TABLES (Iteration 1) at Time = 5.0s
============================================================

[Time = 5.0s] Routing Table at Router A:
Dest | Cost | Next Hop
-------------------------
   A |    0 | A
   B |    2 | B
   C |    3 | B
   D |    5 | B

[Time = 5.0s] Routing Table at Router D:
Dest | Cost | Next Hop
-------------------------
   A |    5 | C
   B |    3 | B
   C |    2 | C
   D |    0 | D
```

5.3 Routing Tables After Cost Change

ROUTING TABLES AFTER COST CHANGE

```
[Time = 30.1s] Cost updated: B <-> D changed from 3 to 7

============================================================
ROUTING TABLES AFTER COST CHANGE at Time = 30.1s
============================================================

[Time = 30.1s] Routing Table at Router A:
Dest | Cost | Next Hop
-------------------------
   A |    0 | A
   B |    2 | B
   C |    3 | B
   D |    5 | C

[Time = 30.1s] Routing Table at Router B:
Dest | Cost | Next Hop
-------------------------
   A |    2 | A
   B |    0 | B
   C |    1 | C
   D |    3 | C
```

6 Routing Table Update Logs

The following table shows the complete routing table evolution for Router A over the simulation period:

Table 1: Router A Routing Table Updates

Time (s)	Destination	Cost	Next Hop	Change Reason
0.0	A	0	A	Initial
0.0	B	2	B	Initial
0.0	C	5	C	Initial
0.0	D	∞	None	Initial
5.0	C	3	B	Better path via B
5.0	D	5	B	New path via B
30.1	D	5	C	Cost change B-D

7 Performance Analysis

7.1 Time Complexity Analysis

The Bellman-Ford update operation has the following complexity characteristics:

- **Per-iteration complexity**: $O(V \times E)$ where V is the number of vertices and E is the number of edges
- **Space complexity**: $O(V^2)$ for storing routing tables across all routers
- **Message complexity**: $O(V \times N)$ per iteration, where N is the average number of neighbors per router

7.2 Convergence Analysis

Based on the simulation results:

Table 2: Performance Metrics

Metric	Value
Total simulation time	90.0 seconds
Total iterations	18
Total messages exchanged	432
Number of cost updates	3
Average messages per second	4.8
Average messages per iteration	24
Convergence time after cost change	2-3 iterations

7.3 Message Exchange Analysis

The simulation demonstrates that:

- Each iteration involves all routers sending distance vectors to all neighbors
- For the 4-router topology, this results in 24 messages per complete iteration
- Convergence typically occurs within 2-3 iterations after any topology change
- The protocol exhibits good scalability characteristics for small to medium networks

8 Challenges and Debugging

8.1 Implementation Challenges

Several significant challenges were encountered during the implementation:

1. **Synchronization Issues**: Initial implementation had race conditions between periodic updates and cost changes, resolved by implementing a sequential event processing approach.

2. **Poison Reverse Logic**: Ensuring correct implementation of poison reverse required careful tracking of next-hop relationships and proper vector modification before transmission.

3. **Convergence Detection**: Implementing reliable convergence detection required multiple iterations without changes to confirm stability.

4. **Infinite Cost Handling**: Managing infinity values in routing calculations required special handling in comparison operations and display formatting.

8.2 Debugging Strategies

Key debugging approaches included:

- **Comprehensive Logging**: Adding detailed logging of all routing table changes and message exchanges
- **Step-by-step Verification**: Manual verification of Bellman-Ford calculations against expected results
- **Convergence Validation**: Implementing multiple convergence checks to ensure stable routing states
- **Edge Case Testing**: Testing with various topology configurations and cost change scenarios

8.3 Resolution Approaches

The following solutions were implemented:

```
def check_convergence(self, max_iterations=5):
    for i in range(max_iterations):
        messages_sent, changes = self.send_distance_vectors()
        if not changes:
            print(f"Network converged after {i+1} iterations.")
            return True
    return False
```

Listing 6: Convergence Detection Solution

9 Conclusion

This implementation successfully demonstrates the fundamental principles of Distance Vector routing using the Bellman-Ford algorithm. The simulation reveals both the strengths and limitations of the protocol:

Successful Convergence: The implementation shows that Distance Vector routing reliably converges to optimal paths within a reasonable number of iterations. The Bellman-Ford algorithm effectively finds shortest paths, and the network adapts correctly to dynamic cost changes.

Loop Prevention Effectiveness: The Poison Reverse mechanism successfully prevents routing loops by advertising infinite costs for routes that would create circular

dependencies. This demonstrates the importance of loop prevention mechanisms in distributed routing protocols.

Dynamic Adaptation: The protocol shows good responsiveness to link cost changes, with convergence typically occurring within 2-3 iterations after topology modifications. This demonstrates the protocol's ability to maintain routing optimality in dynamic network conditions.

Foundational Importance: Despite modern advances in routing protocols, Distance Vector routing remains educationally significant as it introduces fundamental concepts used in more sophisticated protocols. Understanding DV routing provides essential background for comprehending OSPF, BGP, and other modern routing algorithms.

Practical Considerations: The simulation reveals DV routing's primary advantages of simplicity and ease of implementation, while also highlighting its limitations including slower convergence compared to link-state protocols and potential for count-to-infinity problems in certain network configurations.

The implementation successfully meets all specified objectives, providing a comprehensive understanding of Distance Vector routing behavior, convergence characteristics, and practical performance considerations. This foundation proves essential for advanced networking studies and real-world protocol design.