Computer Networks Lab (CS-342)

Lab Assignment - 5

Drive Link :

https://drive.google.com/drive/u/0/folders/1pUtNXXabCXxz3xYWsgUlWqpVcXVm5ys5 (README, Makefile, Code, Additional output screenshots)

GROUP - 6

Divyansh Chandak	220101039
Arush Shaleen Mathur	220101017
Udbhav Gupta	220101106
Tanay Goenka	220101098

---

1. Introduction

This report covers the implementation of the Distance Vector Routing (DVR) algorithm to simulate and manage the effects of link failures in a network. The assignment focuses on two primary loop prevention mechanisms—Split Horizon and Poisoned Reverse—to address the "count-to-infinity" problem commonly encountered in DVR protocols. These mechanisms ensure that routing loops do not lead to prolonged incorrect routing information.

2. Problem Statement

In a network of interconnected routers, each router exchanges information with its immediate neighbors to compute the shortest paths to all other routers. When a link failure occurs, certain routers may enter a loop where the routing table updates indefinitely in an attempt to find a path to the now-disconnected node. This phenomenon is known as the "count-to-infinity" problem. The goal is to simulate this problem and implement mechanisms (Split Horizon and Poisoned Reverse) to prevent it.

---

3. Implementation Details

Part 1: Distance Vector Routing (DVR) Algorithm

1. **Graph Setup**
    - The network is modeled as a graph with N nodes and M edges.
    - Each edge in the graph represents a network link with an associated cost (or weight).
    - Input is a list of edges formatted as `(source, destination, cost)`, allowing each router (node) to calculate the least-cost path to every other node.

2. **DVR Algorithm**
    - Each router maintains a routing table containing information on the shortest path to every other node in the network.
    - The DVR algorithm iteratively updates these routing tables by exchanging distance vectors with neighboring routers until each router has an optimal path to every other router.
    - **Output**: The initial routing tables for each node, displaying the shortest path to every other node before any link failures.

```
PS C:\Users\hp\Desktop\networks> ./a
Initial Routing Tables:

Routing table for Router 0:
  Destination: 4 | Cost: 7 | Next Hop: 1
  Destination: 3 | Cost: 6 | Next Hop: 1
  Destination: 2 | Cost: 3 | Next Hop: 1
  Destination: 0 | Cost: 0 | Next Hop: 0
  Destination: 1 | Cost: 2 | Next Hop: 1

Routing table for Router 1:
  Destination: 4 | Cost: 5 | Next Hop: 2
  Destination: 3 | Cost: 4 | Next Hop: 2
  Destination: 2 | Cost: 1 | Next Hop: 2
  Destination: 0 | Cost: 2 | Next Hop: 0
  Destination: 1 | Cost: 0 | Next Hop: 1

Routing table for Router 2:
  Destination: 4 | Cost: 4 | Next Hop: 3
  Destination: 3 | Cost: 3 | Next Hop: 3
  Destination: 2 | Cost: 0 | Next Hop: 2
  Destination: 0 | Cost: 3 | Next Hop: 1
  Destination: 1 | Cost: 1 | Next Hop: 1

Routing table for Router 3:
  Destination: 4 | Cost: 1 | Next Hop: 4
  Destination: 3 | Cost: 0 | Next Hop: 3
  Destination: 2 | Cost: 3 | Next Hop: 2
  Destination: 0 | Cost: 6 | Next Hop: 2
  Destination: 1 | Cost: 4 | Next Hop: 2

Routing table for Router 4:
  Destination: 4 | Cost: 0 | Next Hop: 4
  Destination: 3 | Cost: 1 | Next Hop: 3
  Destination: 2 | Cost: 4 | Next Hop: 3
  Destination: 0 | Cost: 7 | Next Hop: 3
  Destination: 1 | Cost: 5 | Next Hop: 3
```

Part 2: Simulating Link Failure

1. **Link Failure Simulation**
    - To observe the effects of a network link failure, a critical edge (e.g., between nodes 5 and 4) is removed.
    - The DVR algorithm re-runs, updating the routing tables in response to the change. Without loop prevention, this can cause the count-to-infinity problem, where certain nodes increment their distance values indefinitely in an attempt to route around the failed link.
    - **Condition**: The simulation stops if any node's distance to a destination exceeds a threshold (100 in this case), indicating a loop.
2. **Observations**
    - Routing tables after the link failure may show increased distances, particularly for routes passing through the broken link.
    - Any node affected by the count-to-infinity problem will have increasingly large distances in its routing table.

```
Routing Tables after Link Failure Propagation:
Routing table for Router 1:
  Destination: 5 | Cost: 12 | Next Hop: 2
  Destination: 4 | Cost: 6 | Next Hop: 2
  Destination: 3 | Cost: 5 | Next Hop: 2
  Destination: 1 | Cost: 0 | Next Hop: 1
  Destination: 2 | Cost: 3 | Next Hop: 2

Routing table for Router 2:
  Destination: 5 | Cost: 9 | Next Hop: 3
  Destination: 4 | Cost: 3 | Next Hop: 3
  Destination: 3 | Cost: 2 | Next Hop: 3
  Destination: 1 | Cost: 3 | Next Hop: 1
  Destination: 2 | Cost: 0 | Next Hop: 2

Routing table for Router 3:
  Destination: 5 | Cost: 12 | Next Hop: 2
  Destination: 4 | Cost: 1 | Next Hop: 4
  Destination: 3 | Cost: 0 | Next Hop: 3
  Destination: 1 | Cost: 5 | Next Hop: 1
  Destination: 2 | Cost: 2 | Next Hop: 2

Routing table for Router 4:
  Destination: 5 | Cost: 13 | Next Hop: 2
  Destination: 4 | Cost: 0 | Next Hop: 4
  Destination: 3 | Cost: 1 | Next Hop: 3
  Destination: 1 | Cost: 6 | Next Hop: 3
  Destination: 2 | Cost: 3 | Next Hop: 3

Routing table for Router 5:
  Destination: 5 | Cost: 0 | Next Hop: 5
  Destination: 4 | Cost: 100 | Next Hop: Unknown
  Destination: 3 | Cost: 100 | Next Hop: Unknown
  Destination: 1 | Cost: 100 | Next Hop: Unknown
  Destination: 2 | Cost: 100 | Next Hop: Unknown
```

Part 3: Poisoned Reverse Mechanism

1. **Mechanism Description**
    - Poisoned Reverse is a loop-prevention technique where, instead of omitting a route to the source, a router advertises the route with an infinite metric.
    - This explicit "poisoning" signals to the neighboring routers that the route is unreachable, helping to prevent loops by blocking reverse routes that could cause incorrect information propagation.
2. **Implementation**
    - After applying Poisoned Reverse in the DVR algorithm, the routers are re-simulated with the link between nodes 5 and 4 still removed.
    - Each router advertises a route back to its source neighbor with an infinite cost to indicate that it should not be considered.
3. **Results**
    - Updated routing tables demonstrate the effectiveness of Poisoned Reverse. The count-to-infinity issue is typically prevented as routers do not mistakenly route through each other to reach an unreachable node.
    - The routing tables after implementing Poisoned Reverse confirm that loop formation is significantly reduced.

Part 4: Split Horizon Technique

1. **Mechanism Description**
    - Split Horizon is another loop-prevention technique, which prevents a router from sending information about a route back in the direction from which it came.
    - By avoiding sending information back to its source, Split Horizon reduces the chance of loops by blocking invalid paths from entering the routing table.

2. **Implementation**
    - The Split Horizon technique is applied to the DVR algorithm, and the network is re-simulated with the same edge between nodes 5 and 4 removed.

- Routers are configured to not send routing information back on the link from which the information was originally learned.

3. **Results**
   - The final routing tables after applying Split Horizon show stable and accurate path information, confirming that Split Horizon successfully prevents routing loops and resolves the count-to-infinity issue.

Code:-

```
void receiveUpdate(int neighborId, int dest, int newCost) {
    // Update the stored state information of the neighbor
    neighborDistanceTables[neighborId][dest] = newCost;

    // Recalculate the distance to the destination using the updated information
    int bestCost = INFINITY_COST;
    int bestNextHop = -1;

    for (const auto& [neighbor, costToNeighbor] : neighbors) {
        int neighborDistance = neighborDistanceTables[neighbor].count(dest) ? neighborDistanceTa
        int calculatedCost = costToNeighbor + neighborDistance;
        if (calculatedCost < bestCost) {
            bestCost = calculatedCost;
            bestNextHop = neighbor;
        }
    }


    // Update distance table if a shorter path is found
    if (updateDistanceTable(dest, bestCost, bestNextHop)) {

        broadcastUpdates(dest);
    }
}
```

```
void reversePoisoning(int dest  , int neighbor ){
    int broadcastedCost  = (nextHopTable[dest] == neighbor) ? INFINITY_COST : distanceTable[dest] ;
    network->routers[neighbor].receiveUpdate(id, dest, broadcastedCost );
}
void splitHorizon(int dest  , int neighbor ){
    if(nextHopTable[dest] == neighbor) return ;
    network->routers[neighbor].receiveUpdate(id, dest,  distanceTable[dest] );

}

// Each router broadcasts its own updates to neighbors
void broadcastUpdates(int dest) {
    for (const auto& [neighbor, cost] : neighbors) {
        //normal
        // network->routers[neighbor].receiveUpdate(id, dest, distanceTable[dest] );
        //reversePoisoning
        // reversePoisoning(dest , neighbor) ;
        //splitHorizon
        splitHorizon(dest , neighbor) ;
    }
}
```

5. Results and Observations

1. **Initial DVR Output**
    - The routing tables generated by the initial DVR algorithm show the shortest paths between nodes in the graph before any link failures.
2. **Routing Tables after Link Failure**
    - After simulating the removal of an edge (e.g., between nodes 5 and 4), certain nodes demonstrate the count-to-infinity problem.
    - Some nodes have significantly increased path costs or infinitely increasing values due to incorrect route propagation.
3. **After Applying Poisoned Reverse**
    - Poisoned Reverse prevents routes from being considered when advertised as infinite.
    - Routing tables reflect a more stable network state with no evident loops or count-to-infinity issues.
4. **After Applying Split Horizon**
    - Split Horizon also effectively prevents the count-to-infinity problem by restricting certain route advertisements.

- Final routing tables confirm a loop-free network configuration with stable paths to all reachable nodes.

6. Conclusion

This assignment successfully demonstrates the implementation of the DVR algorithm and the utility of Poisoned Reverse and Split Horizon mechanisms in addressing the count-to-infinity problem. The results show that both techniques significantly reduce routing instability in the event of link failures, enhancing network robustness and convergence speed.