



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 8 : Implementation of Distance Vector Algorithm

Submitted By:

Afser Adil Olin

Roll No : AE-47

Anika Tabassum

Roll No : Rk-61

Submitted On :

April 06, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

Contents

1	Introduction	2
1.1	Objectives	2
2	Theory	2
2.1	Routing Algorithm	2
2.2	Distance Vector Routing Algorithm	2
2.3	The Distance Vector Routing Algorithm	3
2.4	Calculating the routing table	5
2.5	Calculating the forwarding table	5
2.6	Link-Cost Changes and Link Failure	5
2.6.1	Link Cost Decrease	5
2.6.2	Link Failure or Cost increase	5
2.7	Poisoned reverse	5
3	Analytical Behaviors	6
3.1	Time Complexity Analysis	6
3.2	Memory Complexity Analysis	6
4	Finding APSP	6
4.1	Graph	6
4.2	APSP	7
4.3	Forwarding table	8
5	Methodology	9
6	Implementation	9
6.1	Procedure	9
6.1.1	Extracting information at the beginning	9
6.1.2	Flooding	10
6.1.3	Bellman-Ford	10
6.2	Results	10
6.2.1	Experimental Result	11
6.3	Conclusion	12
7	Experience	12

1 Introduction

Distant vector routing protocol also called as Bellman-Ford algorithm or Ford Fulkerson algorithm used to calculate a path. A distance-vector protocol calculates the distance and direction of the vector of the next hop from the information obtained by the neighboring router. It is necessary to keep track of the topology and inform neighboring devices if any changes occur in the topology.

1.1 Objectives

- To develop an understanding of the Distance Vector Algorithm and its applications in computer networks by implementing the algorithm in a simulated network environment.
- To accurately represent the network topology by maintaining a complete and up-to-date database of network topology information.congestion, throughput, and delay.
- To quickly converge to a stable and efficient routing configuration in response to changes in the network topology.

2 Theory

2.1 Routing Algorithm

A routing algorithm is a process or set of rules that determines the best path for data packets to travel from one network node to another in a computer network. In other words, it is a method used by routers to determine the most efficient way to forward data packets between networks. A routing algorithm is a process or set of rules that determines the best path for data packets to travel from one network node to another in a computer network. In other words, it is a method used by routers to determine the most efficient way to forward data packets between networks. Routing algorithms are essential in computer networks because they help to ensure that data packets are delivered quickly and efficiently. There are many different types of routing algorithms, each with its own strengths and weaknesses. We will be discussing Distance Vector Routing (DVR) Algorithm.

2.2 Distance Vector Routing Algorithm

The distance vector routing algorithm is iterative, asynchronous, and distributed. It is distributed in that each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors. It is iterative in that this process continues on until no more information is exchanged between neighbors.

Some key points about the distance vector routing protocol :

1. **Network Information:** Every node in the network should have information about its neighboring node. Each node in the network is designed to share information with all the nodes in the network.
2. **Routing Pattern:**In DVR the data shared by the nodes are transmitted only to that node that is linked directly to one or more nodes in the network.
3. **Data sharing :** The nodes share the information with the neighboring node from time to time as there is a change in network topology.

One of the main benefits of DVRA that the DV algorithm can handle changes in network topology and link costs in a flexible manner. When a node detects a change in link cost or topology, it updates its routing table and propagates the changes to its neighbors. This allows the network to adapt to changes in traffic patterns or network failures.

A graph is used to formulate routing problems. Recall that a graph $G = (N, E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N . In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers.

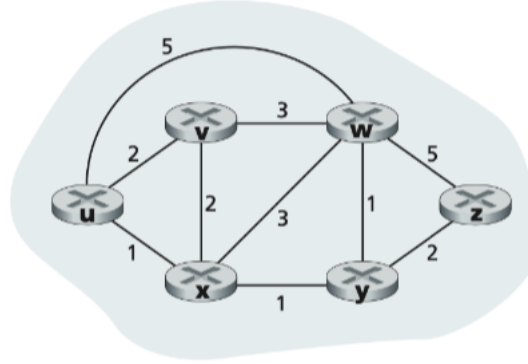


Figure 1: Initial Abstract graph model of a computer network

As shown in Figure, an edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link, the link speed, or the monetary cost associated with a link.

2.3 The Distance Vector Routing Algorithm

Distance vector routing algorithm is also called as Bellman-Ford algorithm or Ford Fulkerson algorithm as this algorithm is used to find the shortest route from one node to another node in the network.

The routing protocol is used to calculate the best route from source to destination based on the distance or hops as its primary metric to define an optimal path. The distance vector refers to the distance to the neighbor nodes, where routing defines the routes to the established node.

Each router in the network will share the distance information with the neighboring router. All the information is gathered from the neighbor routers. With each router's information, an optimal distance is calculated and stored in the routing table. This way, the process of calculating the optimal path is done using the distant vector routing protocol.

Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y , for all nodes, y , in N . Let $\mathbf{D}_x(\mathbf{y}) = [D_x(y): y \text{ in } N]$ be node x 's distance vector, which is the vector of cost estimates from x to all other nodes, y , in N . With the DV algorithm, each node x maintains the following routing information:

- For each neighbor v , the cost $c(x,v)$ from x to directly attached neighbor, v

- Node x 's distance vector, that is, $\mathbf{D}_x = [D_x(y): y \text{ in } N]$, containing x 's estimate of its cost to all destinations, y , in N
- The distance vectors of each of its neighbors, that is, $\mathbf{D}_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x

In the distributed, asynchronous algorithm, from time to time, each node sends a copy of its distance vector to each of its neighbors. When a node x receives a new distance vector from any of its neighbors w , it saves w 's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\} \text{ for each node } y \in N$$

where,

$D_x(y)$ = The least distance from x to y

$c(x, v)$ = Node x 's cost from each of its neighbour v

$D_v(y)$ = Distance to each node from initial node

\min_v = selecting shortest distance

If node x 's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Miraculously enough, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, each cost estimate $D_x(y)$ converges to $dx(y)$, the actual cost of the least-cost path from node x to node y

Listing 1: Link-State (LS) Algorithm At each node x

```

1 Initialization:
2
3 for all destinations y in N:
4     Dx(y) = c(x, y) /* if y is not a neighbor then c(x, y) = infinity */
5 for each neighbor w
6     Dw(y) = for all destinations y in N
7 for each neighbor w
8     send distance vector Dx = [Dx(y): y in N] to w
9
10 loop
11     wait (until I see a link cost change to some neighbor w or
12         until I receive a distance vector from some neighbor w)
13
14     for each y in N:
15         Dx(y) = minv{c(x, v) + Dv(y)}
16
17 if Dx(y) changed for any destination y
18     send distance vector Dx = [Dx(y): y in N] to all neighbors
19
20 forever

```

In the DV algorithm, a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance-vector update from some neighbor. But to update its own forwarding table for a given destination y , what node x really needs to know is not the shortest-path distance to y but instead the neighboring node $v^*(y)$ that is the next-hop router along the shortest path to y .

2.4 Calculating the routing table

The DV algorithm tries to update the routing table in two cases for each node x .

- When it sees a cost change in one of its directly attached nodes.
- When it receives a Distance Vector update from one of its neighbours.

Then it uses the Bellman-Ford Equation for each node y in the network N . That is, it runs the Bellman-Ford Algorithm. If the node x 's distance vector gets updated while running the Bellman-Ford Algorithm it sends the updated distance vector to all of its neighbours.

2.5 Calculating the forwarding table

To update the forwarding table of the node, we simply need to track the neighbouring node that is the next-hop router along the shortest path. This is done in the above algorithm by tracking the parent list.

2.6 Link-Cost Changes and Link Failure

For Link Cost change we can have a cost increase or cost decrease. Link Failure can be modelled as an infinite cost increase. We can show the effects of these as follows

2.6.1 Link Cost Decrease

In case of a link cost decrease the nodes directly attached will be alerted and the algorithm will automatically minimize the cost.

2.6.2 Link Failure or Cost increase

For a substantial increase in Cost if we communicate the increased cost to the 1st degree separated neighbours, they will compare it to their own tables and find that they know a path to the destination at a lower cost, which is erroneously through the parent from the previous state. When the directly connected node finds the message that the separated node has a lower cost path it tries to route through the separated node. But the separated node will again try to route through the parent thus creating a Routing Loop. This goes on until the link cost computed at the separated node is greater than the increased link cost. But what about a Link Failure? it will continue looping the message infinitely this is why this problem is called the Count to infinity problem. Which has to be avoided. From the above discussion we can understand the popular wisdom for DV protocols, "Good news travels fast, Bad news travels slow".

2.7 Poisoned reverse

To fix the Routing Loop we consult the forwarding table. If we find that we are sending a cost to a node that is the next hop router to the destination we simply send an infinite cost. This way there is no chance of getting an erroneous cost loop. But unfortunately this does not extend for 2nd degree neighbours and beyond. More advanced techniques like Split horizon is implemented to avoid such cases.

3 Analytical Behaviors

3.1 Time Complexity Analysis

The time complexity of the distance vector routing algorithm is $O(V^2)$, where V is the number of nodes in the network. In the initialization phase, each node needs to initialize its distance vector to every other node in the network. This requires $O(V^2)$ time.

In the update phase, each node broadcasts its distance vector to all its neighboring nodes. Each neighboring node then updates its own distance vector based on the received information. This process continues until all the nodes have updated their distance vectors. In the worst case, each node can have a maximum of $V - 1$ neighbors, which means that each node can potentially receive $V - 1$ distance vectors in each round. Thus, the update phase can take up to $O(V^2)$ time as well.

In the worst case scenario where each node sends out its entire distance vector to all its neighbors in every iteration, the total time complexity can be $O(V^3)$. However, in practice, this scenario is unlikely to occur because most networks have sparse connectivity, and the number of updates required will be much less than V^2 . Here, we are assuming that the number of edges E in the network is much less than V^2 .

3.2 Memory Complexity Analysis

The memory complexity of the distance vector routing algorithm can be represented as $O(V E)$, where V is the number of nodes and E is the number of edges in the network. This is because each node in the network needs to store its distance vector, which contains information about the cost of reaching every other node in the network. Therefore, the memory required for storing the distance vector at each node is proportional to the number of edges in the network. Since there can be at most $V - 1$ edges per node (in a fully connected network), the overall memory complexity can be expressed as $O(V E)$.

4 Finding APSP

4.1 Graph

Graph for the Distance Vector Routing(DVR) Algorithm:

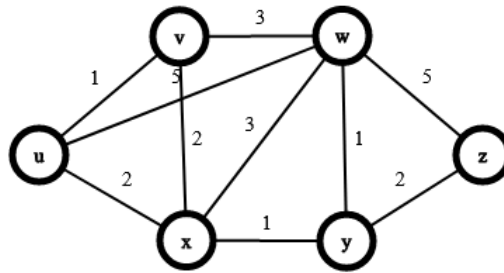


Figure 2: Initial Network Topology

4.2 APSP

These are the results for after running the algorithm on the given graph as shown above.

Dijkstra's Algorithm Results for all nodes			
Source	Destination	Shortest Path	Path Length
u	u	{u}	0
	v	{u,v}	2
	w	{u,w}	5
	x	{u,x}	1
	y	{u,x,y}	2
	z	{u,x,y,z}	4
v	u	{u,v}	2
	v	{v}	0
	w	{v,w}	3
	x	{v,x}	2
	y	{v,x,y}	3
	z	{v,x,y,z}	5
w	u	{w,x,u}	4
	v	{w,v}	3
	w	{w}	0
	x	{w,y,x}	2
	y	{w,y}	1
	z	{w,y,z}	3
x	u	{x,u}	1
	v	{x,v}	2
	w	{x,y,w}	2
	x	{x}	0
	y	{x,y}	1
	z	{x,y,z}	3
y	u	{y,x,u}	2
	v	{y,x,v}	3
	w	{y,w}	1
	x	{y,x}	1
	y	{y}	0
	z	{y,z}	2
z	u	{z,y,x,u}	4
	v	{z,y,x,v}	5
	w	{z,y,w}	3
	x	{z,y,x}	3
	y	{z,y}	2
	z	{z}	0

4.3 Forwarding table

The forwarding table simulated for this graph by our algorithm was found to be

Forwarding table for all nodes			
Source	Destination	Next Hop	Path Length
u	u	none	0
	v	v	2
	w	w	5
	x	x	1
	y	x	2
	z	x	4
v	u	v	2
	v	none	0
	w	w	3
	x	x	2
	y	x	3
	z	x	5
w	u	x	4
	v	v	3
	w	none	0
	x	y	2
	y	none	1
	z	y	3
x	u	u	1
	v	v	2
	w	y	2
	x	none	0
	y	y	1
	z	y	3
y	u	x	2
	v	x	3
	w	w	1
	x	x	1
	y	none	0
	z	z	2
z	u	y	4
	v	y	5
	w	y	3
	x	y	3
	y	y	2
	z	none	0

5 Methodology

- Each node in the network maintains a distance vector that contains the cost to reach every other node in the network. This allows for a more scalable solution, as nodes do not need to maintain a full view of the entire network topology.
- DVR uses a distributed algorithm to calculate the distance vector for each node. Each node broadcasts its own distance vector to its directly connected neighbors, and updates its own distance vector based on the received distance vectors from its neighbors.
- DVR uses the Bellman-Ford algorithm to calculate the shortest path to every other node in the network. This ensures that packets are always forwarded along the least-cost path.
- DVR can adapt to changes in the network topology, but it may take some time for the routing tables to converge after a topology change.
- Once the simulations have been run, the results must be analyzed to evaluate the performance of the implementation.
- DVR is more prone to routing loops, as nodes may update their distance vector based on stale or incorrect information.
- DVR generally does not support multiple equal-cost paths, although some implementations may use heuristics to achieve load balancing and redundancy.

6 Implementation

6.1 Procedure

In our implementation we first take the information of their neighbours from a txt file named Path.txt. Then we start a threading for handling incoming messages from other routers called handle_send(). In our implementation, we updated an edge randomly and send the updated information about the graph to its neighbours after a certain time which is 30 sec. We sent the messages in different thread as sending messages is independent to receiving messages. Our implementation mainly consists of 3 part.

6.1.1 Extracting information at the beginning

In make_top() function, we extracted the information from file.txt. In this file the first column consists of IP addresses of the one node (u), the second column consists of Port number of the that node(u), the third column consists of the IP addresses of the second node(v) and the fourth column consists of Port number of the second node (v). The fifth column consists of the weight between node u and v. We made sure extracting only the edge for that server by including a if statement that take and store the result if the IP address and Port number matches of the u node.

Memory Complexity : For storing the information of edges, we need $O(V^2)$ memory .

Time Complexity : For extracting the information of an edge we need $O(1)$ time complexity.

Link State Algorithm:

Memory Complexity : For storing the information of edges, we need $O(E)$ memory .

Time Complexity : For extracting the information we need $O(E)$ time complexity.

6.1.2 Flooding

Flooding is a non-adaptive routing technique. The distance vector routing algorithm is an adaptive routing technique because the routers will dynamically adapt to the changes that occur in the network. In flooding, data packets are sent to all the outgoing links except the one it has arrived.

Distance Vector Algorithm:

Memory Complexity : It just updated the previous list, so Auxiliary space complexity is $O(1)$ given we already made a list of the adjacency list.

Time Complexity : It checks all the edges came from the message. So the time complexity will be $O(E)$.

Link State Algorithm:

Memory Complexity : It just updated the previous list, so Auxiliary space complexity is $O(1)$ given we already made a list of the adjacency list.

Time Complexity : It checks all the edges came from the message. So the time complexity will be $O(E)$.

6.1.3 Bellman-Ford

In this part, We just run the Bellman-Ford algorithm. We update the adjacency matrix of distance and their parent as we get a minimum distance from the current node(u).

Memory Complexity : Auxiliary space Complexity $O(V^2)$

Time Complexity : By using priority queue , we get $O(VE)$

Dijkstra:

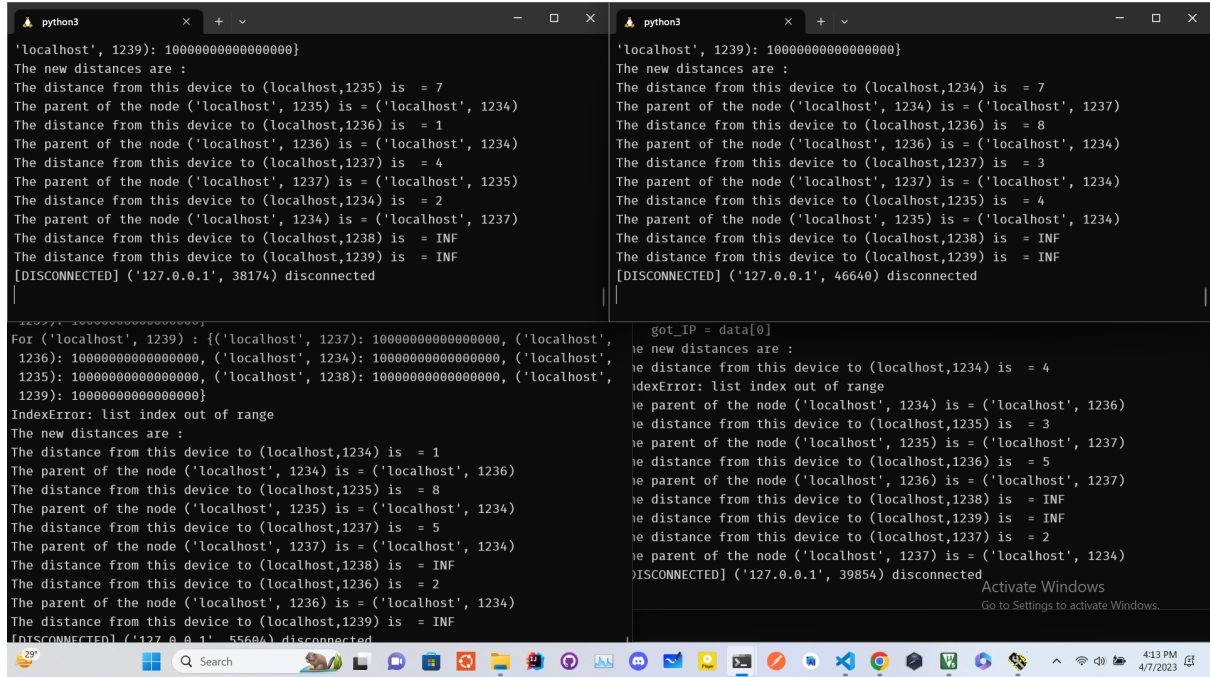
Memory Complexity : Auxiliary space Complexity $O(V^2)$

Time Complexity : By using priority queue , we get $O(E\log(V))$

6.2 Results

After updating the graph , we printed the current updated adjacency list of the graph and the distance and the parent of each node from the current node x.

6.2.1 Experimental Result

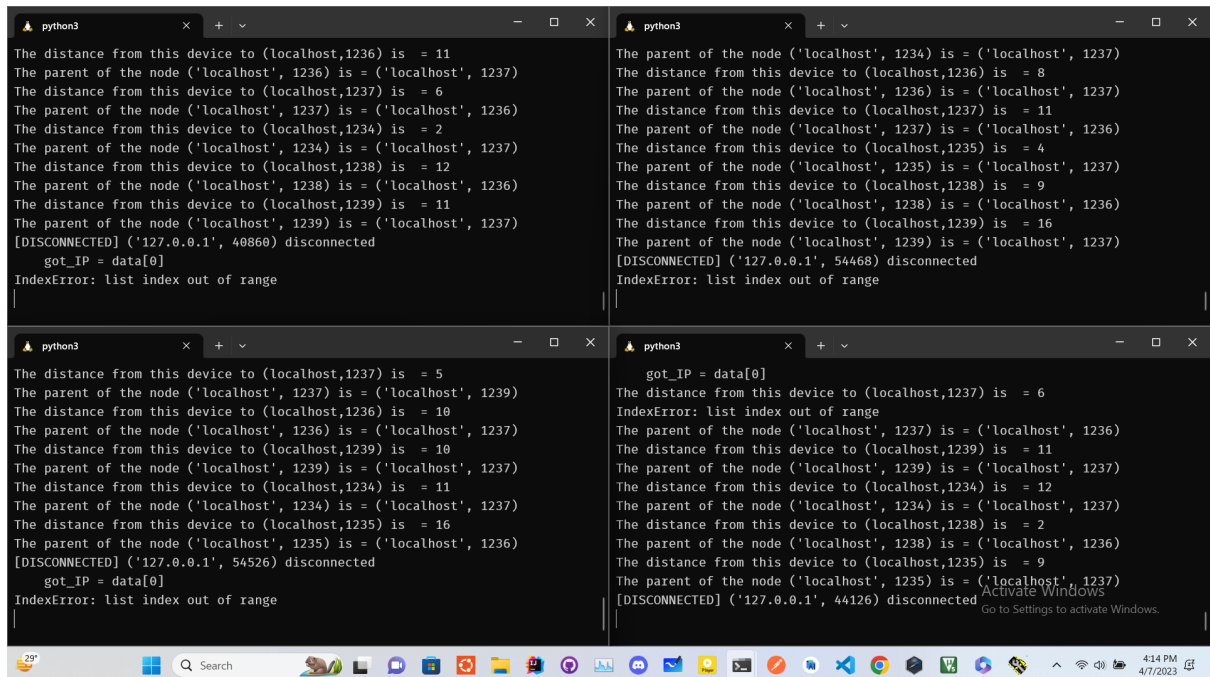


```
'localhost', 1239): 10000000000000000}
The new distances are :
The distance from this device to (localhost,1235) is = 7
The parent of the node ('localhost', 1235) is = ('localhost', 1234)
The distance from this device to (localhost,1236) is = 1
The parent of the node ('localhost', 1236) is = ('localhost', 1234)
The distance from this device to (localhost,1237) is = 4
The parent of the node ('localhost', 1237) is = ('localhost', 1235)
The distance from this device to (localhost,1234) is = 2
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1238) is = INF
The distance from this device to (localhost,1239) is = INF
[DISCONNECTED] ('127.0.0.1', 38174) disconnected

For ('localhost', 1239) : {'localhost', 1237): 10000000000000000, ('localhost', 1236): 10000000000000000, ('localhost', 1234): 10000000000000000, ('localhost', 1235): 10000000000000000, ('localhost', 1238): 10000000000000000, ('localhost', 1239): 10000000000000000}
IndexError: list index out of range
The new distances are :
The distance from this device to (localhost,1234) is = 1
The parent of the node ('localhost', 1234) is = ('localhost', 1236)
The distance from this device to (localhost,1235) is = 8
The parent of the node ('localhost', 1235) is = ('localhost', 1234)
The distance from this device to (localhost,1237) is = 5
The parent of the node ('localhost', 1237) is = ('localhost', 1234)
The distance from this device to (localhost,1238) is = INF
The distance from this device to (localhost,1236) is = 2
The parent of the node ('localhost', 1236) is = ('localhost', 1234)
The distance from this device to (localhost,1239) is = INF
[DISCONNECTED] ('127.0.0.1', 55604) disconnected

got_IP = data[0]
The new distances are :
The distance from this device to (localhost,1234) is = 4
IndexError: list index out of range
The parent of the node ('localhost', 1234) is = ('localhost', 1236)
The distance from this device to (localhost,1235) is = 3
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
The distance from this device to (localhost,1236) is = 5
The parent of the node ('localhost', 1236) is = ('localhost', 1237)
The distance from this device to (localhost,1238) is = INF
The distance from this device to (localhost,1239) is = INF
The distance from this device to (localhost,1237) is = 2
The parent of the node ('localhost', 1237) is = ('localhost', 1234)
[DISCONNECTED] ('127.0.0.1', 39854) disconnected
```

Figure 3: Least cost path and Parents for nodes when Server u,v,w,x is on



```
The distance from this device to (localhost,1236) is = 11
The parent of the node ('localhost', 1236) is = ('localhost', 1237)
The distance from this device to (localhost,1237) is = 6
The parent of the node ('localhost', 1237) is = ('localhost', 1236)
The distance from this device to (localhost,1234) is = 2
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1238) is = 12
The parent of the node ('localhost', 1238) is = ('localhost', 1236)
The distance from this device to (localhost,1239) is = 11
The parent of the node ('localhost', 1239) is = ('localhost', 1237)
[DISCONNECTED] ('127.0.0.1', 40860) disconnected
got_IP = data[0]
IndexError: list index out of range

The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1236) is = 8
The parent of the node ('localhost', 1236) is = ('localhost', 1237)
The distance from this device to (localhost,1237) is = 11
The parent of the node ('localhost', 1237) is = ('localhost', 1236)
The distance from this device to (localhost,1235) is = 4
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
The distance from this device to (localhost,1238) is = 9
The parent of the node ('localhost', 1238) is = ('localhost', 1236)
The distance from this device to (localhost,1239) is = 16
The parent of the node ('localhost', 1239) is = ('localhost', 1237)
[DISCONNECTED] ('127.0.0.1', 54468) disconnected
IndexError: list index out of range

The distance from this device to (localhost,1237) is = 5
The parent of the node ('localhost', 1237) is = ('localhost', 1239)
The distance from this device to (localhost,1236) is = 10
The parent of the node ('localhost', 1236) is = ('localhost', 1237)
The distance from this device to (localhost,1239) is = 10
The parent of the node ('localhost', 1239) is = ('localhost', 1237)
The distance from this device to (localhost,1234) is = 11
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1235) is = 16
The parent of the node ('localhost', 1235) is = ('localhost', 1236)
[DISCONNECTED] ('127.0.0.1', 54526) disconnected
got_IP = data[0]
IndexError: list index out of range

got_IP = data[0]
The distance from this device to (localhost,1237) is = 6
IndexError: list index out of range
The parent of the node ('localhost', 1237) is = ('localhost', 1236)
The distance from this device to (localhost,1239) is = 11
The parent of the node ('localhost', 1239) is = ('localhost', 1237)
The distance from this device to (localhost,1234) is = 12
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1238) is = 2
The parent of the node ('localhost', 1238) is = ('localhost', 1236)
The distance from this device to (localhost,1235) is = 9
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
[DISCONNECTED] ('127.0.0.1', 44126) disconnected
```

Figure 4: Least cost path and Parents for nodes when all the Servers are on

Figure 5: Least cost path and Parents for nodes again after the server y and z goes off

6.3 Conclusion

In conclusion, Distance vector routing is a type of dynamic protocol. Distant vector routing algorithm also called as Bellman-Ford algorithm or Ford Fulkerson algorithm used to calculate the shortest path in the network. The router shares the information between the neighboring node containing a direct link. The router is used to find the optimal path from source to destination. Each router in the network shares a change in topology periodically with the neighboring router

7 Experience

1. We had to collect and analyze network topology information, as this requires a deep understanding of the network's routers, connections, and associated costs.
2. We had to optimize the algorithm's performance and monitoring network stability
3. We had to evaluate the data consistency mechanism of the distributed database by intentionally introducing inconsistencies into the graph.

References

- [1] Difference between Distance vector routing and Link State routing. <https://www.geeksforgeeks.org/difference-between-distance-vector-routing-and-link-state-routing/amp/>. [Online; accessed 2023-04-06].
- [2] Snehal Jadhav. Distance vector routing algorithm. <https://www.scaler.com/topics/computer-network/distance-vector-routing-algorithm/>, oct 7 2022. [Online; accessed 2023-04-06].