Can Günyel 150200049

Introduction

The objective of Project 4 is to develop and evaluate the performance of routing algorithms, specifically focusing on Link State and Distance Vector Routing. The achieve that goal, I generate a network topology with a random number of nodes specified by the user at runtime. Once the topology is created, the program applies both Distance Vector and Link State Routing algorithms to generate forwarding tables for each node in the topology. And also calculate and visualize total cost, delay time, shortest path, runtime of algorithms and total hops for randomly chosen two nodes. In the light of the results from the experiment, we can see how the algorithms perform according to different sizes and which algorithm we can prefer for differing scales of topologies.

System Architecture

Topology Generation

The network topology is created using the *networkx* library, forming a random internet AS graph based on the user-specified number of nodes. I used random internet AS graph since it is a connected graph with similar structure to today's internet structure.

I gave my student ID as a seed to obtain a static graph structure (only edges connections not weights) to be able to track outputs. Since with larger node counts it is not easy to spot the desired nodes in the plot. But seed can be removed from code and get a random topology within each run.

Cost Assignment

The purpose of cost assignment is to introduce variability into the network topology, reflecting the different costs associated with transmitting data between nodes. These randomly assigned costs serve as the foundation for subsequent routing algorithm calculations, influencing the determination of optimal paths and overall network performance like in real-life. For each node in the graph, the function iterates through its neighbors, generating random costs for the edges if not previously assigned. By this way, it ensures to have same weight (cost) for between node x-y and y-x.

Link State Routing and Dijkstra Algorithm

Link-state routing is a network routing algorithm where each node constructs a detailed map of the entire network, including its topology and link costs. Using this information, nodes independently calculate the shortest paths to all other nodes, allowing for efficient, optimal routing decisions based on up-to-date network state information.

The *Dijkstra* funtion initializes the distances dictionary with all node distances set to infinity, except for the source node, which is set to 0. A priority queue is initialized using a heap to efficiently select nodes with minimum distances. The main loop of the algorithm extracts the node with the minimum distance from the priority queue and explores its neighbors. For each neighbor, it calculates the new distance from the source and updates if the new distance is shorter. The updated distances and nodes are added to the priority queue. The algorithm reconstructs the shortest path from the destination node to the source node using the *previous_nodes* dictionary.

The *link_state* function constructs a forwarding table for each node in the graph. It utilizes Dijkstra's algorithm to calculate the total cost and visited nodes for all possible destinations from each source node. Next hops for each destination are recorded in the forwarding table. The forwarding tables are written to a file named "link_state_forwarding_table.txt".

Distance Vector Routing and Bellman Ford Algorithm

In Distance Vector Routing each network node maintains a table that contains the distance and next-hop information to all other nodes in the network. Nodes periodically exchange and update these tables with their neighbors, facilitating the discovery of the shortest paths and enabling dynamic adaptation to changes in network topology.

The *Bellman Ford* function iteratively inform nodes for a specified number of time steps. During each iteration, it examines every node and its neighbors, updating the cost if a shorter path is found. The result includes the total cost and shortest path from a specified source to destination.

The *distance_vector* function performs Distance Vector routing using the Bellman-Ford algorithm. It iterates for the number of longest path (diameter of the graph) since it is enough for informing the whole topology. With each iteration we simulate a different time step. A forwarding table for each node with current time step is constructed and written to a file named "distance_vector_forwarding_table.txt."

Visualization

To visualize the network topology, edge weights (costs), and the calculated shortest paths for both algorithms, I employed the Matplotlib and NetworkX libraries. Initially, I converted the visited nodes into visited edges and subsequently colored them red to accentuate the identified shortest path. Additionally, I marked predetermined weight values above the corresponding edges. The combination of red-colored edges and annotated weights contributes to a visually informative representation of the graph, facilitating a better understanding of the routing algorithm outputs.

Main

In main function, first I generate a topology and then assign costs to its edges. After that, I choose two random nodes as source and destination. Certain source and destination nodes can be given in code with replacing *random.choice(list(graph.nodes())) with desired values.*

Following, I call *link_state* algorithm and calculate its runtime. In the function, I create all forwarding tables with running Dijkstra algorithm for each node. Since we know all routers neighbors by a remote device in Link State Routing all forwarding tables are created at once.

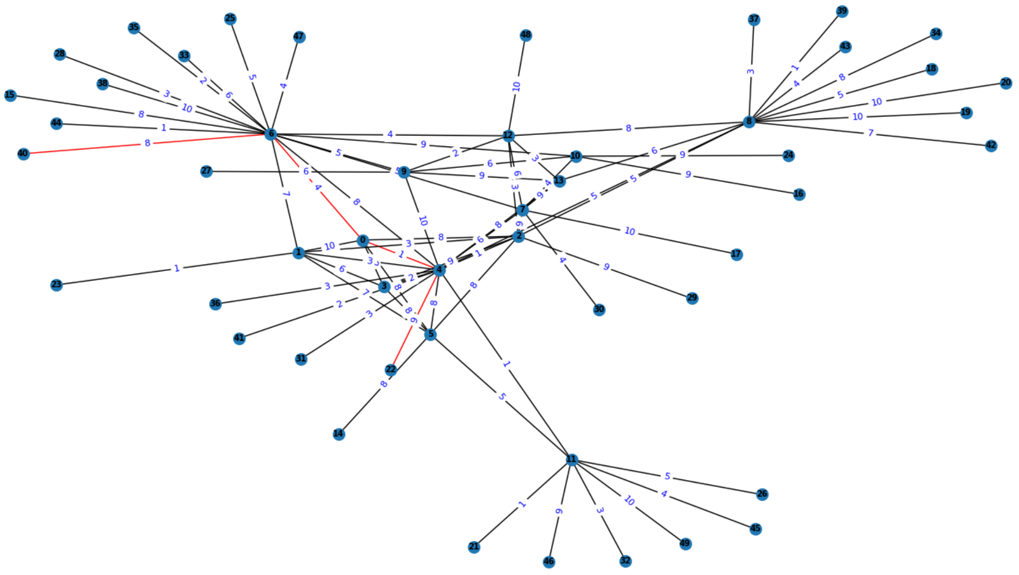
Following, I call *distance_vector* algorithm and calculate its runtime. In the function, I create all forwarding tables for a time step with running Bellman Ford algorithm for each node. Since we know only neighbor routers in Distance Vector Routing at each time step forwarding tables are updated with their neighbors' updated paths.

For visualization purposes, with the shortest path and total cost values returned from both routing algorithms to main. I print *Visited Nodes in order, Total hops, Total cost, Runtime* results and draw the graph with shortest path marked with red color. Distance Vector Routing takes more than its outputted runtime. Since I add some extra steps to properly simulate forwarding tables step by step.

Can Günyel 150200049

Results and Discussion

As my result I have both runtime for algorithm and forwarding tables for each router. But since it is not possible to show all shortest path for each node in a graph. I visualized shortest path for randomly chosen source and destination routers. However, it is calculated for all nodes. visited nodes, total hops, total cost, runtime outputs printed to console, topology and shortest path is plotted and forwarding tables are logged into text files What is required by packet transmission is not fully explained and I could not find anything similar in the slides. That's why I sent an e-mail to the assistant and the lecturer, but I did not receive a response from either of them. Therefore, packet transmission delay is roughly calculated by taking into account the total cost and the number of total hops. Sample topology, shortest path plot and console out is given below.



```
Source Node: 40
Destination Node: 22
Visited Nodes in order: [40, 6, 0, 4, 22]
Total hops: 4
Total cost: 22
Packet Transmission Delay: 9.3 milliseconds
Link State Runtime: 0.5088 seconds
Visited Nodes in order: [40, 6, 0, 4, 22]
Total hops: 4
Total cost: 22
Packet Transmission Delay: 9.3 milliseconds
Distance Vector Runtime: 1.2116 seconds
```

Router number	5	15	50	100
Runtime Link State	0.0033 s	0.0261 s	0.5088 s	3.906 s
Runtime Distance Vector	0.0006 s	0.0313 s	1.2116 s	9.2191 s

As we can see above graph, as the number of routers increases the runtime for both Link State and Distance Vector routing algorithms also increases. However, it's important to note that Distance Vector routing tends to have higher runtimes compared to Link State routing, especially as the network size grows.

Can Günyel 150200049

Forwarding tables are also logged as output but since they are too large I add them as files. In Link State Routing forwarding table created once thanks to centralized system. However, in Distance Vector Routing we have differing (updated) forwarding tables at each time step. Finally, we can see both algorithms, gave the same result which is the shortest and most costless path for test nodes.