Simulating Multi Level Hierarchical Routing

By: Dominic Ginter, Cole Atkinson, Kelly Tay

Course: CPE 400 - Computer Communication Networks Spring 2020

Due: Thursday, April 30, 2020

Introduction

With networks constantly growing in size, the routing table of the source address grows with them proportionally causing more router memory, CPU time and bandwidth to be used. This can eventually lead to networks becoming so large that it is no longer practical for a router to have entries for every other router inside the network. Instead, routing is done hierarchically where multiple clusters are divided into regions and regions divided into nodes, or routers. For this project, the team decided to simulate hierarchical networking via UAVs. In the project, the nodes used are simulating UAVs and contain unique addresses. To accomplish the simulation, the team first created an example network. Figure 1 displays a graphic that resembles the created 3-level hierarchical network. The network consists of three clusters which each contain multiple regions. These regions furthermore contain nodes which are connected to each other which work together to form one hierarchical network.

In order to properly simulate hierarchical networking, the team first had to utilize OSPF in order to simulate current routing protocols. After this, the team then had to figure out how to make a table display not only every route between each router in a network, but also a hierarchical version as well. To find every possible path, the team utilized Python's Dijkstra's shortest pathing algorithm to calculate the weight between the UAVs as well as finding the actual path itself and format it into a table, as shown in figure 4. After this, the team then had to find the hierarchical path of each UAV. To do this, the team created a set of if/else statements that sorted the UAVs based upon cluster number and region number. The code would then organize the UAVs by utilizing Python's dictionary function, and would print out corresponding UAVs in differing regions and clusters in a table manner, as displayed in figure 5.

For additional touches, the team created an option in the program that allows the user to view every single hop between a source address and a destination address, in order to fully see what exactly is going on. Additionally, figures 1 and 6 are displayable options in the menu of the program.

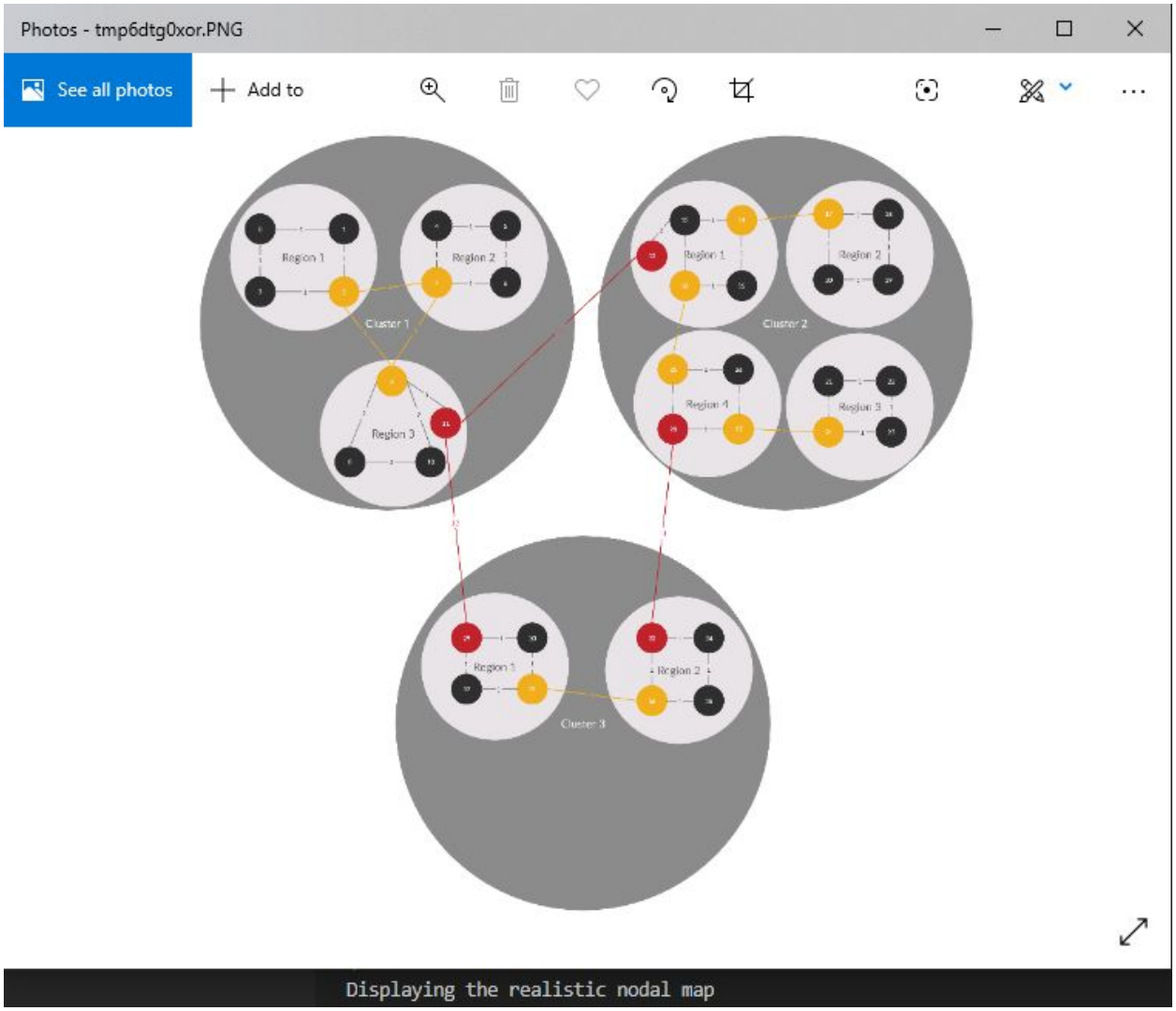

Fig.1: *Network simulated in this project (Option 4 in the program)*

Running the Program

(1) Install Python3 and Python3 Interpreter on Computer

(2) Install Pillow, Networkx, Imp and PrettyTable packages for Python

- This can be done using >pip install <package>

(3) Run the main program using >py Main.py

Code Explanation

The main function of the code has many features that can be used via a menu displayed when running the program. The menu contains 5 options that can be accessed by typing the menu number that corresponds to the feature. Figure 3 displays the menu as well as its options. The functionality of having different options that can be selected was implemented using a switch statement.

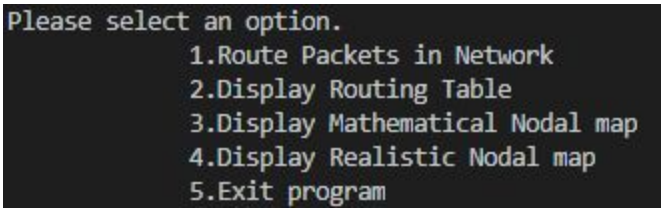

Fig. 2: *The terminal menu*

The first option of the menu allows basic packet routing. The user can input a source address and destination address to view the routing path between the two routers as well as the number of hops between these routers as shown in figure 4. Within these if else statements, the *dijkstra_path* algorithm function from the Networkx Python library was used to calculate the routing path. The *shortest_path_length* function from the same library was used to calculate the number of hops between two routers.

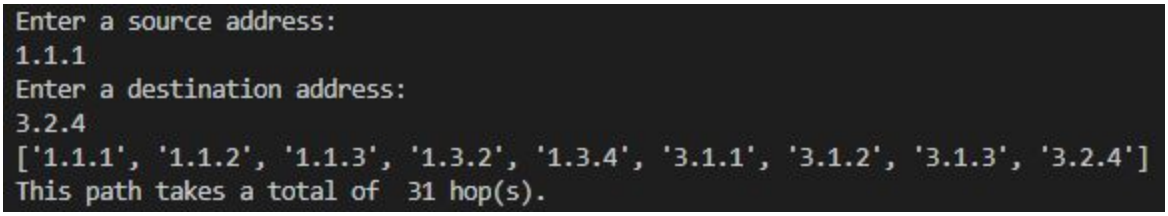

Fig. 3: *Option 1, Route Packets in Network, in action*

The second option of the menu displays the routing tables, with the option of the full table or the hierarchical table. The user can input a source address to view a routing table of all other nodes, the first hop that the node will take to get to the destination with the shortest route as well as the number of hops needed to get to the destination. After inputting the source address, the user has an option to choose between the full table, as shown in figure 5, and the hierarchical table, as shown in figure 6. The difference between the full table and the hierarchical table is that with the hierarchical table, the information can be summarized and the network efficiency is improved. This is because the hierarchical table is calculated using hierarchical routing, where routers only need to remember the nodes within their region. This means it does not have any other information regarding the nodes outside its region. Since the router does not have to remember as many nodes, it does not have to handle as much network traffic, increasing network efficiency.

```
Enter a router address:
2.1.2
Please select a routing table display option.
                1.Full Table
                2.Hierarchial Table
1
+-------------+-------+--------+
| Destination |  Line | Weight |
+-------------+-------+--------+
|    2.1.2    |  ---  |  ---   |
|    1.1.1    | 2.1.1 |   30   |
|    1.1.2    | 2.1.1 |   29   |
|    1.1.3    | 2.1.1 |   28   |
|    1.1.4    | 2.1.1 |   29   |
|    1.2.1    | 2.1.1 |   28   |
|    1.2.2    | 2.1.1 |   29   |
|    1.2.3    | 2.1.1 |   28   |
|    1.2.4    | 2.1.1 |   27   |
|    1.3.1    | 2.1.1 |   21   |
|    1.3.2    | 2.1.1 |   19   |
|    1.3.3    | 2.1.1 |   21   |
|    1.3.4    | 2.1.1 |   16   |
|    2.1.1    | 2.1.1 |   2    |
|    2.1.3    | 2.1.3 |   1    |
|    2.1.4    | 2.1.5 |   2    |
|    2.1.5    | 2.1.5 |   1    |
|    2.2.1    | 2.1.3 |   7    |
|    2.2.2    | 2.1.3 |   8    |
|    2.2.3    | 2.1.3 |   9    |
|    2.2.4    | 2.1.3 |   8    |
|    2.3.1    | 2.1.5 |   15   |
|    2.3.2    | 2.1.5 |   16   |
|    2.3.3    | 2.1.5 |   15   |
|    2.3.4    | 2.1.5 |   14   |
|    2.4.1    | 2.1.5 |   5    |
|    2.4.2    | 2.1.5 |   6    |
|    2.4.3    | 2.1.5 |   7    |
|    2.4.4    | 2.1.5 |   6    |
|    3.1.1    | 2.1.5 |   26   |
|    3.1.2    | 2.1.5 |   25   |
|    3.1.3    | 2.1.5 |   24   |
|    3.1.4    | 2.1.5 |   25   |
|    3.2.1    | 2.1.5 |   20   |
|    3.2.2    | 2.1.5 |   21   |
|    3.2.3    | 2.1.5 |   22   |
|    3.2.4    | 2.1.5 |   21   |
+-------------+-------+--------+
```

Fig. 4: *Option 2, Display Routing Table (Full Table), in action*

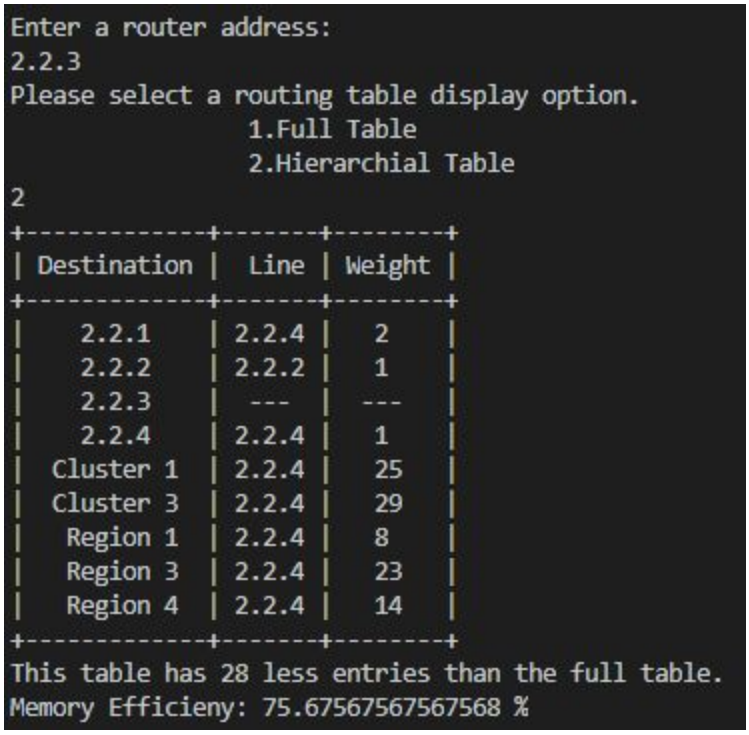

Fig. 5: *Option 2, Display Routing Table (Hierarchical Table), in action*

These two options were implemented within their own individual Python files. The PrettyTable Python library was used in order to create the table formatting used for both the full table and the hierarchical table. For the full table, a list of all the node addresses is iterated through, calculating the line and weight of each destination. The line represents the first tip that is taken after the imputed address and is calculated using the *dijkstra_path* function from the Networkx Python library. The weight represents the least amount of hops it takes to get from the inputted addresses to the destination address and is calculated using the *dijkstra_path_length* function. For the hierarchical table, a different approach had to be taken but the same functions are used for the weight and line of each destination. Since in hierarchical routing each router only has information regarding routers in its region, the only thing that matters is the cluster if the destination node is in a different cluster and the region if the destination node is in the same cluster but a different region. When iterating through the list of node addresses, it is first checked whether or not the address is in the same node. If it is, the region would get checked. If it is in the same region, the router knows that information and the node number is important. If it isn't in the same region, only the name of the region is important. If it isn't in the same cluster, only the name of the cluster is important. Based on that information, the weight and line of each destination is calculated. The memory efficiency of the hierarchical table is calculated by taking the number of elements within the hierarchical table divided by the number of elements in the full table. That number is subtracted from 100 and multiplied by 100 to get a percentage.

The third option of the menu displays the mathematical nodal map. This option is implemented using the *draw_networks* function from the Networkx Python library. Figure 6 displays the results from selecting that option.

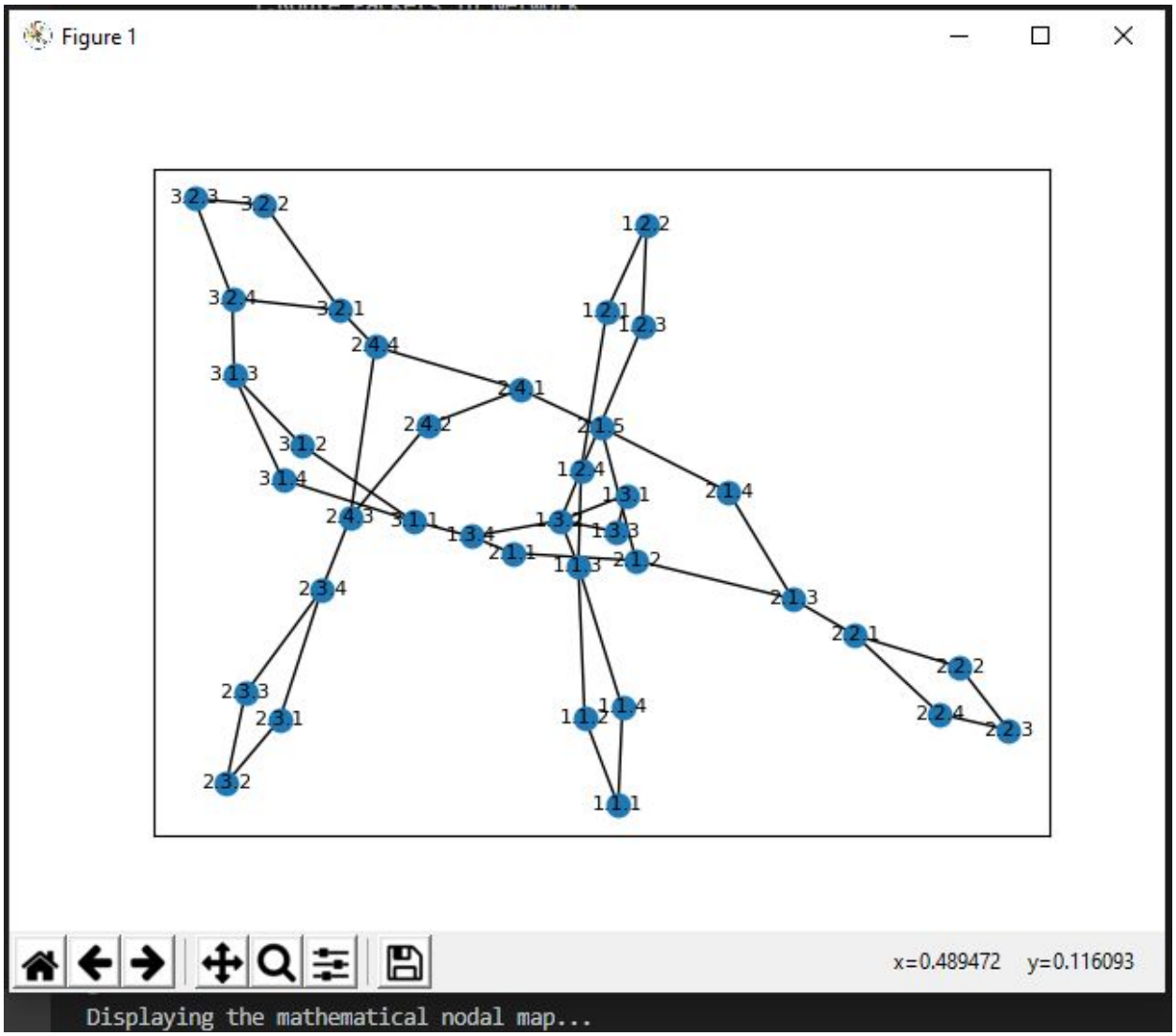

Fig. 6: *Option 3, Display Mathematical Nodal Map, in action*

The fourth option of the menu displays the realistic nodal map which can be seen in figure 1. Lastly, option five allows the user to exit the program.

Report Explaining Protocol

Link state routing is a very vital concept in the project. Link state routing states that every node in its network is to construct a map of the network that exists in, or in other words, a routing table. A routing table is essentially a graph that shows which nodes are connected to which other

nodes and how much weight is between them. Weight represents the amount of hops taken in a network to get from a source address to a destination address. Using this, each node, or UAV in the project, calculates the path it takes to get from the source UAV to, depending on the option, either every UAV in the network or the hierarchical neighboring UAVs. In the project, option 2 has two options within it that display both the hierarchical table and the full table, which is encompassed by link state routing and will be elaborated upon.

The project employs the Open Shortest Path First (OSPF) protocol to simulate routing between UAVs over a hierarchical network across different domains. This is the same protocol used for IP networks as well as mesh networks within the same domain, so it is a very malleable protocol. How this protocol works is simple: it finds the best path for a packet to travel in a network by utilizing a routing table and sends any routing changes to the router's neighbor. An OSPF routing domain can use multiple areas which are groups of routers, or regions, as used in the project. The internal routing of each region can be isolated from the rest of the routing domain which makes the routing table smaller, therefore reducing routing protocol overhead. This is utilized in the project; it is shown as having 3 clusters with multiple regions in each cluster while each region has 4 to 5 UAVs inside of it. Whenever a UAV needs to access an address of a UAV in a different cluster, OSPF protocol finds the shortest path between the two UAVs that are being tested and returns the order of the hops needed to be taken. Many issues do arise while using OSPF, however. Typically, the most common error to occur is that of a lack of establishment between neighbors. To handle this error, one must check to see if the following is true: if the init state is established, if the link is established, if there is a two-way connection established, and if a packet is corrupted. If any of the following are not established, then establishing them will fix the error of neighbor establishment.

Hierarchical routing is an essential part of the project. It is utilized to overcome the problem of routing table oversizing; it also fixes the issue of network traffic inefficiency. In this example, router A in region 1 is going to be under the microscope (see Fig. 1). In hierarchical routing, router A has knowledge about only the other routers within its region (Routers B and C). Rather than router A having information about every single other router in the network, it contains only one entry for every other region in the network. This means its hierarchical table will have information about routers B, C, D, P, N, L, and G. Table 1 displays a proper hierarchical table; as shown, it doesn't display the information about the outside nodes specifically, but it does show the next hop that is taken to get to the outside region as well as the weight of the entire journey from the source router to the destination router. This severely reduces the size of its routing table. If router A wants to send packets to router E in region 2, it simply sends it to the router in its region that is linked to region 2, which in this case is router B. If it is not linked directly, for example, if router A was seeking to send packets to router I, then the router uses OSPF to find the shortest path between the two regions and sends the packet(s) accordingly. As shown,

hierarchical routing saves not only time but a plethora of memory and proves to be an effective and widely used internet routing algorithm.

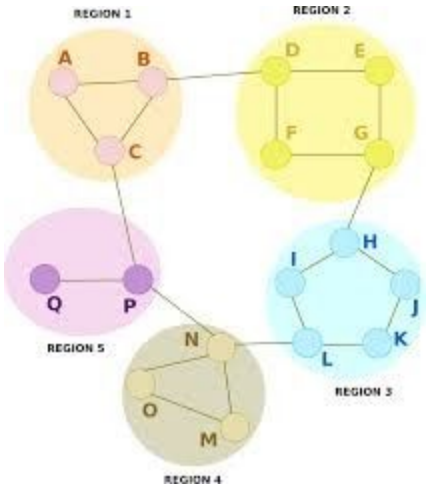

Fig. 7: *Example of a Hierarchical Routing Network [Wikimedia, website]*

In order for OSPF to work properly, Dijkstra's algorithm is key to calculate the shortest path algorithm between UAVs. The way it works is by creating a shortest path tree that stores the weight between the input UAV and destination UAV. After, all UAVs that have a direct link to the source UAV have their weight set to 0, while all UAVs outside of a direct link to the source UAV are initialized to infinite. This is so the closest UAVs will be calculated first. The algorithm then finds vertices that are not calculated and updates their weights accordingly. These weights are all measured out, and then the algorithm compares these values to each other in order to find the shortest path between the source UAV and destination UAV. Issues that arise with Dijkstra's algorithm mainly consist with memory usage and calculation time due to the amount of nodes present. For example, if a graph were to have 3.5 billion nodes, then obviously the calculation time would be far too inefficient for the user and could possibly crash the system. To handle this error, hierarchical routing is employed to save space and time. Since hierarchical routing severely cuts down on the amount of entries in a routing table, Dijkstra's algorithm can calculate the shortest path based on the hierarchical table and still find the proper weight between two UAVs in differing regions or clusters.

Gateway routers are prominent in this project. A gateway router is a node that connects to nodes outside of its region. Figure 7 shows a great example of a network. In this example, router C is considered to be a gateway router because it connects to region 5 even though it resides in region 1. In the project, a router is considered gateway when it connects to either any outside region or any outside cluster. Gateway routers also have levels. If a gateway router only connects to an outside region, it is considered level 2. If the router connects to an outside cluster, it is considered level 3. The pattern continues as described, but in this project the highest level of a

gateway router is only 3. Gateway routers are essential for hierarchical routing; this is due to the fact that hierarchical routing requires information about one router in an outside region/cluster, so the gateway router of that outside region is picked for the routing table.

Out of the Box Thinking: Novel Contribution

In order to create the example network, an idea that was considered was potentially having the user decide the number of clusters, regions and nodes that the example network would have. It was concluded that this process would take too long as the user would need to individually and manually enter that information. Instead, it was decided to create and implement an example network using the Networkx Python library. This was implemented inside a separate Python file and was used all throughout this project. The team first implemented each region using the *MultiGraph* function, naming them after their prospective cluster and region. Afterwards, the nodes were added to each region using the *add_nodes_from* function. In order to add the node attributes, a nested dictionary was used. This allowed specific attributes to be obtained by just specifying 'Cluster', 'Region' or 'Node'. Lastly, the nodes were connected by edges and the values were added to each edge using the *add_edges_from* function as well as the *union* function.

The team also came up with a naming method to name the nodes so that they can be easily referenced within the program. The nodes are referenced using this format: 1.2.3., where the 1 represents the cluster, the 2 represents the region and the 3 represents the node within the region. This format was inspired by the format of IP addresses and how each number separated by the periods has a specific representation. By using this naming format and adding node attributes to it using the nested dictionary, a node such as 1.2.3 can be passed into the first index and using the second index to specify cluster, region or node, it will output that specific attribute.

Results and Analysis

To test the hierarchical routing simulation, the team generated a hierarchical table for all 37 UAVs in the example network. By doing this, we were able to see how much more efficient hierarchical routing is in comparison to full routing tables. Table 2 displays relevant information about each node in the example network. As you can see in the table, hierarchical routing reduces the number of table entries drastically. The results show that all nodes were able to reduce their routing table sizes by at least 72.97% during this simulation. Along with this, some nodes even reached a memory efficiency of 81.08%. In a realistic example, maps that display over billions of nodes in a country cannot display all nodes' paths between each other, so it is clear that these 70% to 80% improved efficiency ratings from hierarchical routing are essential. As told earlier, were a routing table to display billions of entries, not only would the time taken for the table to generate be inefficient, but searching for a specific node would also be highly inefficient. Hierarchical routing allows users to pinpoint an address purely based on location within regions and or clusters, thus heavily improving performance all around. Overall, as

networks expand, it can be concluded that the memory efficiency of a hierarchical table will expand with it. This is extremely beneficial in real-world applications as networks are growing larger every day and reducing routing table sizes allows for memory and CPU time to be used efficiently.

Node Number	Corresponding Address	Hierarchical Table Entries	Memory Efficiency	Border Gateway Router
0	1.1.1	8	78.38%	No
1	1.1.2	8	78.38%	No
2	1.1.3	8	78.38%	Level 2
3	1.1.4	8	78.38%	No
4	1.2.1	8	78.38%	No
5	1.2.2	8	78.38%	No
6	1.2.3	8	78.38%	No
7	1.2.4	8	78.38%	Level 2
8	1.3.1	8	78.38%	No
9	1.3.2	8	78.38%	Level 2
10	1.3.3	8	78.38%	No
11	1.3.4	8	78.38%	Level 3
12	2.1.1	10	72.97%	Level 3
13	2.1.2	10	72.97%	No
14	2.1.3	10	72.97%	Level 2
15	2.1.4	10	72.97%	No
16	2.1.5	10	72.97%	Level 2
17	2.2.1	9	75.68%	Level 2
18	2.2.2	9	75.68%	No
19	2.2.3	9	75.68%	No
20	2.2.4	9	75.68%	No
21	2.3.1	9	75.68%	No
22	2.3.2	9	75.68%	No
23	2.3.3	9	75.68%	No
24	2.3.4	9	75.68%	Level 2
25	2.4.1	9	75.68%	Level 2
26	2.4.2	9	75.68%	No
27	2.4.3	9	75.68%	Level 2
28	2.4.4	9	75.68%	Level 3
29	3.1.1	7	81.08%	Level 3
30	3.1.2	7	81.08%	No
31	3.1.3	7	81.08%	Level 2
32	3.1.4	7	81.08%	No
33	3.2.1	7	81.08%	Level 3
34	3.2.2	7	81.08%	No
35	3.2.3	7	81.08%	No
36	3.2.4	7	81.08%	Level 2

***Table 2**: Hierarchical Routing details for each node.*

Moreover, in option 1 of the program, users are able to simulate routing packets in a simplistic, but effective manner. This packet routing utilizes Djikstra's algorithm to find the shortest path to the destination address. Figure 8 displays the shortest path from node '1.1.1' to '3.2.4' in the example network. As shown in the figure, Dijkstra's shortest pathing algorithm discovered the

shortest path from ‘1.1.1’ to ‘3.2.4’ with a weight of 31 hops. This is quite incredible considering that there are an excess number of ways to reach the destination node, but Djikstra’s algorithm always retrieves the shortest total path in an efficient manner.

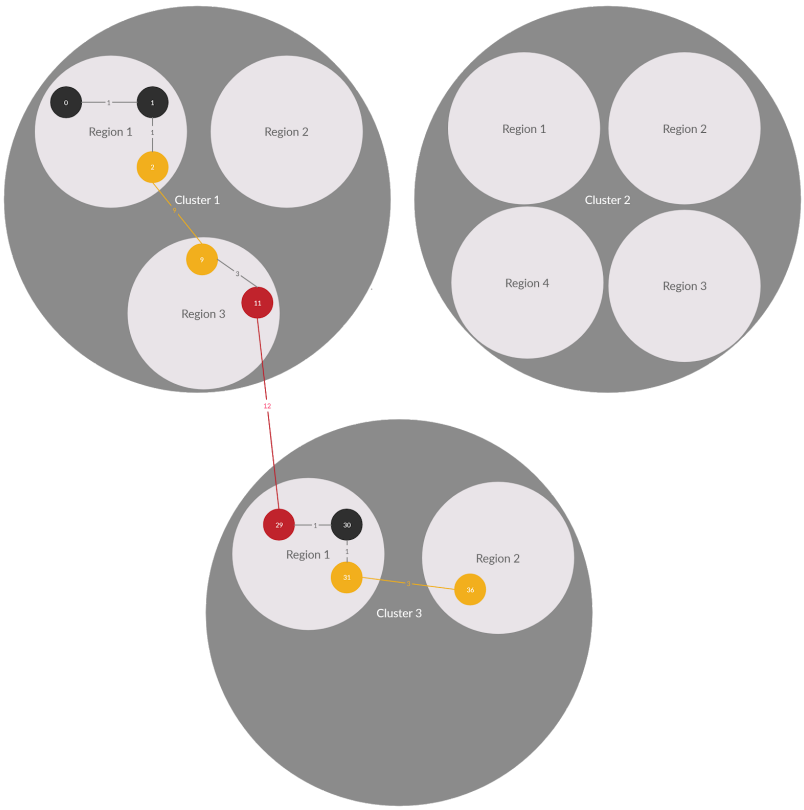

***Fig. 8**: The shortest path between ‘1.1.1’ and ‘3.2.4’*

Conclusion

In conclusion, hierarchical routing is an extremely efficient method of routing. It is utilized in an abundance of protocols and practices, as mentioned earlier with OSPF. Within hierarchical routing lies Dijkstra’s shortest pathing algorithm, which is the core of this routing method. The team utilized this base knowledge to expand and recreate a virtual network from scratch to simulate material learned within the course of CPE 400. Overall, the team gained incredible knowledge throughout the creation of this project. We learned how to work with various technologies such as Visual Studio, Git and Python. Most importantly, we all gained a better understanding of how networks function in a hierarchical manner and how real life networking protocols are used each and everyday.

Works Cited

Cisco (website) Retrieved from:
https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/12151-trouble-main.html

Geeksforgeeks (website) Retrieved from:
https://www.geeksforgeeks.org/difference-between-distance-vector-routing-and-link-state-routing/

Howstuffworks (website) Retrieved from:
https://computer.howstuffworks.com/routing-algorithm5.html

Stack Exchange (website). Retrieved from:
https://networkengineering.stackexchange.com/questions/51426/difference-between-router-and-gateway

Techtarget Network (website) Retrieved from:
https://searchnetworking.techtarget.com/definition/OSPF-Open-Shortest-Path-First

Youtube (website) Retrieved from:
https://www.youtube.com/watch?v=L0HvKXaA-BE