**National University of Sciences and Technology (NUST)**



**School of Electrical Engineering and Computer Science (SEECS)**

**Computer Networks**

**Scenario 3**

|  |  |
| --- | --- |
| **Group Members** | **CMS ID** |
| Ayesha Siddiqa | 407198 |
| Navaal Iqbal | 409977 |
| Amna Ahmed | 408099 |

**Submitted to: Dr. Huma Ghafoor**

**COMPUTER NETWORKS END SEMESTER PROJECT (BESE-13B)**

**SCENARIO 3:**

Consider 50 nodes at random positions. Get them connected in such a way that each node must have 5 interfaces and all these 5 interfaces are connected to 5 other nodes. Only the source and destination nodes have 3 interfaces. The source is at least 20 nodes away from destination. Assign each link a random value between 1 to 15. Calculate the total number of paths from source to destination. Among those paths, select the optimal path by considering delay as a metric. Also, calculate the number of hops for each path. After finding the optimal path, now transmit five packets of equal size from source to destination, calculate the average delay, and the PDR with respect to variable density of nodes in the network. Highlight total number of iterations and the algorithm you use to find the optimal path. Repeat the experiment 20 times with different random values to calculate the average and the optimal path.

Repeat scenario 1 and consider link cost as bandwidth. Select any random value for links between 50 to 250Mbps.

**TOOLS:**

1. **Python:**

- Python stands out as a high-level, versatile programming language. Its design principles prioritize code readability through the use of significant indentation. Python is dynamically-typed and includes garbage collection. It supports various programming paradigms, including structured (particularly procedural), object-oriented, and functional programming.

2. **NetworkX:**

- NetworkX serves as a Python package dedicated to creating, manipulating, and exploring complex network structures. Offering built-in functions, NetworkX enables the implementation of graphs and the analysis of networking protocols. The utilization of this package, known as Networkx, involves leveraging its diverse functions for calculating simple paths, optimal paths, and other network-related tasks.

3. **Matplotlib:**

- Matplotlib functions as a comprehensive plotting library tailored for the Python programming language and its numerical mathematics extension, NumPy. Within the context of our network experiment, matplotlib plays a crucial role in data visualization. Specifically, it was employed for highlighting the shortest path. Additionally, the drawing of the network's topology utilized a Python library referred to as pyplotlib. This library facilitates the creation of network topology visualizations by seamlessly integrating with various NetworkX functions.

**Key Concepts:**

- **Hops**:

- A hop signifies the count of routers a packet traverses from its source node to its destination node. Each instance of data transfer between devices is considered a hop in our context, where the observed nodes represent these hops.

- **Links**:

- A link denotes a connection between two devices or systems enabling communication. This connection can be physical, like a cable, or wireless, such as Wi-Fi or cellular links. In our scenario, links are conceptualized as imaginary lines of distance facilitating packet transfer between nodes.

- **Simple Paths:**

- A simple path is characterized by the absence of repeated nodes. In the context of a weighted graph, the search for the shortest path is employed, with the constraint of disallowing negative weights. Simple paths are crucial in understanding network connectivity without revisiting nodes.

- **Shortest Paths**:

- In a weighted graph, the shortest (weighted) path from vertex u to vertex v is the path with the minimum weight from u to v. Multiple paths may have equal weight, and in such cases, all of them are considered as shortest weighted paths from u to v. Our scenario utilizes a built-in function from the NetworkX library, providing access to all minimum simple paths within our network.

**Methodology:**

To determine the optimal path, we can employ either Dijkstra's algorithm or the A\* search algorithm. Both of these algorithms are specifically designed to identify the shortest path between two nodes in a graph, utilizing a cost function—in our case, bandwidth. Alternatively, we can compute all simple paths initially and then select an optimal path based on delay.

1. **Graph Creation:**

- Start by constructing a graph with 50 nodes.

- Arrange 20 nodes in a linear fashion, while the remaining 30 are placed randomly.

- Establish edges between all nodes, ensuring that each node (excluding the source and destination) has 5 interfaces.

- Assign a weight (bandwidth) to each edge, ranging from 50 to 250 Mbps, with cost calculated as 500/bandwidth.

- Take precautions to eliminate any self-loops within the graph.

2. **Iteration Process:**

- Perform 20 iterations of the experiment.

- In each iteration, randomly set bandwidth values for edges and update the graph accordingly.

- Utilize Dijkstra's algorithm to find the optimal path based on the calculated costs.

- Calculate average cost and other relevant metrics for each iteration.

3. **Visualization:**

- Plot the graph to visually represent the network topology.

- Highlight the optimal path, including link costs, using appropriate visualization techniques.

4. **Packet Transmission:**

- Simulate the transmission of 5 packets from the source to the destination along the determined optimal path.

- Record and analyze the delay and PDR associated with packet transmission.

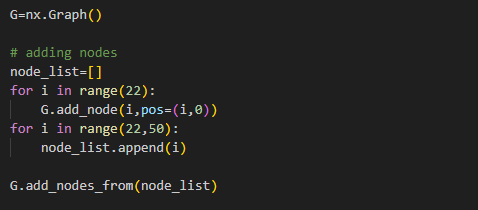
5. **Analysis and Reporting:**

- Address any remaining questions or concerns related to the experiment.

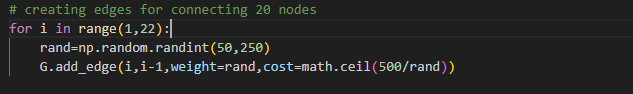
- Provide detailed answers regarding the optimal path, average cost, hop count, and network overhead.

By following this approach, we aim to systematically explore and evaluate the network's behavior under varying conditions across multiple iterations. The use of established algorithms and thorough analysis will contribute to a comprehensive understanding of the network's performance.

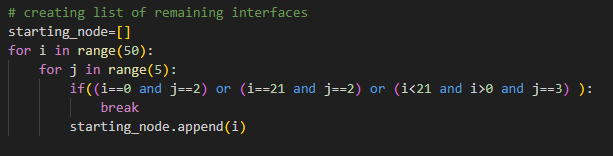
**Step 1: Adding Nodes to the Graph**



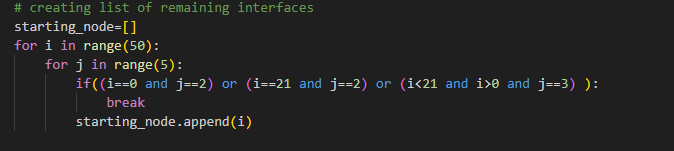
**Step 2: Creating edges for connecting 20 nodes**



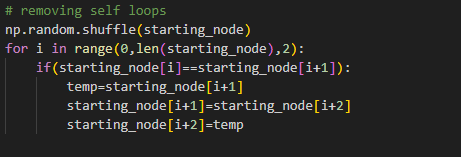
**Step 3: Creating a list of remaining interfaces**



**Step 4: Completing remaining interfaces**



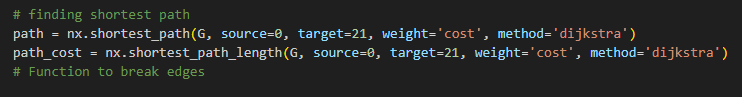
**Step 5: Removing Self-loops**



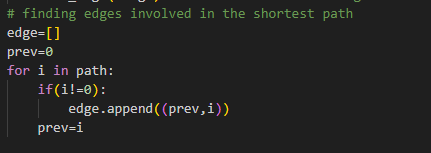
**Step 6: Plotting graph**



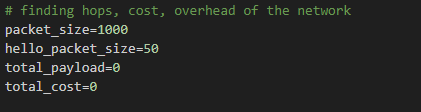
**Step 7: Finding shortest path using Dijkstra’s Algorithm**



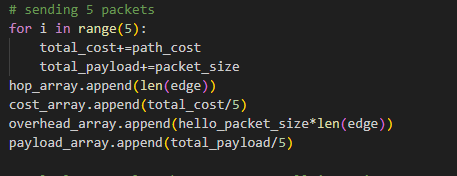
**Step 8: Finding edges involved in the shortest path**



**Step 9: Finding hops, cost, and overhead of the network**



**Step 10: Sending 5 packets**



**Source Code:**

import networkx as nx

import matplotlib.pyplot as plt

import math

import numpy as np

from networkx import all\_simple\_edge\_paths

from networkx import all\_simple\_paths

import pylab

import random

G=nx.Graph()

# adding nodes

node\_list=[]

for i in range(22):

    G.add\_node(i,pos=(i,0))

for i in range(22,50):

    node\_list.append(i)

G.add\_nodes\_from(node\_list)

# creating edges for connecting 20 nodes

for i in range(1,22):

    rand=np.random.randint(50,250)

    G.add\_edge(i,i-1,weight=rand,cost=math.ceil(500/rand))

# creating list of remaining interfaces

starting\_node=[]

for i in range(50):

    for j in range(5):

        if((i==0 and j==2) or (i==21 and j==2) or (i<21 and i>0 and j==3) ):

            break

        starting\_node.append(i)

# removing self loops

np.random.shuffle(starting\_node)

for i in range(0,len(starting\_node),2):

    if(starting\_node[i]==starting\_node[i+1]):

        temp=starting\_node[i+1]

        starting\_node[i+1]=starting\_node[i+2]

        starting\_node[i+2]=temp

# completing remaining interface

for i in range(0,len(starting\_node),2):

    rand=np.random.randint(50,250)

    G.add\_edge(starting\_node[i],starting\_node[i+1],weight=0,cost=0)

# storing positions of all nodes

dic={}

for i in range(50):

    if(i>21):

        dic[i]=(np.random.randint(0,700),np.random.randint(-15,15))

    else:

        dic[i]=(i\*10,0)

# arrays for storing data of 20 ietrations

hop\_array=[]

cost\_array=[]

overhead\_array=[]

payload\_array=[]

total\_packets = 5   # 5 packets per iteration for 20 iterations

# 20 iterations

for j in range(20):

    random.seed()

    packets\_dropped = 0

    # assigning random bandwidths to all edges

    for i in range(0,len(starting\_node),2):

        rand\_val=np.random.randint(50,250)

        u = starting\_node[i]

        v = starting\_node[i+1]

        if G.has\_edge(u, v):

            G[u][v]['weight'] = rand\_val

            G[u][v]['cost'] = math.ceil(500 / rand\_val)

    for i in range(1,22):

        rand=np.random.randint(50,250)

        G[i][i-1]['weight']=rand

        G[i][i-1]['cost']=math.ceil(500/rand)

    # Simulating link breakage for packet drop

    edges\_to\_break = list(G.edges())

    random\_edge = random.choice(edges\_to\_break)

    if np.random.rand() < 0.2:  # Example: Break a link with 10% probability

        # Choose a random edge to break

        if edges\_to\_break:

            G.remove\_edge(\*random\_edge)

            packets\_dropped += 1

    # Remove edge to restore the graph for the next iteration

    if edges\_to\_break:

       G.add\_edge(\*random\_edge)  # Restore the broken edge for the next iteration

    # configuring rendered image

    pylab.figure(1,figsize=(70,30))

    # plotting graph

    nx.draw(G,pos=dic,with\_labels=True,node\_color='#34d8eb',node\_size=400,font\_size=8)

    labels = nx.get\_edge\_attributes(G,'weight')

    nx.draw\_networkx\_edge\_labels(G,pos=dic,edge\_labels=labels,font\_size=12)

    # finding shortest path

    path = nx.shortest\_path(G, source=0, target=21, weight='cost', method='dijkstra')

    path\_cost = nx.shortest\_path\_length(G, source=0, target=21, weight='cost', method='dijkstra')

    # Function to break edges

    edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]

    num\_edges\_to\_break = random.randint(0, min(4, len(edges)))  # Randomly choose up to 4 edges to break

    edges\_to\_break = random.sample(edges, num\_edges\_to\_break)  # Randomly select edges to break

    for edge in edges\_to\_break:

        G.remove\_edge(\*edge)  # Break the edge

        # Simulate packet drop due to the broken link (you might adjust this logic if needed)

        if np.random.rand() < 0.2:

            packets\_dropped += 1  # Increment packet drop count

    # Remove edge to restore the graph for the next iteration

    for edge in edges\_to\_break:

       G.add\_edge(\*edge)  # Restore the broken edge for the next iteration

    # finding edges involved in the shortest path

    edge=[]

    prev=0

    for i in path:

        if(i!=0):

            edge.append((prev,i))

        prev=i

    # finding hops, cost, overhead of the network

    packet\_size=1000

    hello\_packet\_size=50

    total\_payload=0

    total\_cost=0

    # sending 5 packets

    for i in range(5):

        total\_cost+=path\_cost

        total\_payload+=packet\_size

    hop\_array.append(len(edge))

    cost\_array.append(total\_cost/5)

    overhead\_array.append(hello\_packet\_size\*len(edge))

    payload\_array.append(total\_payload/5)

    # Calculate total packets sent over all iterations

# Calculate Packet Delivery Ratio (PDR)

    pdr = (total\_packets - packets\_dropped) / total\_packets

    # giving the required answers

    print("Packets Dropped:",packets\_dropped)

    print("The optimal path is:")

    print(path)

    print("The average cost is: "+str(total\_cost/5))

    print("Total number of hops are: "+str(len(edge)))

    print("Total overhead of the network is: "+str(len(edge))+" hello packets.")

    print("That is for each connection established bw source and dest. we have to transmit "+str(len(edge))+" hello packets of 50 bytes in our case.")

    print("Packet Delivery Ratio (PDR):", pdr)

    # customizing the graph

    nx.draw\_networkx\_nodes(G,pos=dic,nodelist=path,node\_size=800,node\_color='red')

    nx.draw\_networkx\_nodes(G,pos=dic,nodelist=[0,21],node\_size=800,node\_color='#4dff7c')

    nx.draw\_networkx\_edges(G,pos=dic,edgelist=edge,width=5,edge\_color='yellow',arrows=True)

    nx.draw\_networkx\_edge\_labels(G,pos=dic,edge\_labels=labels,font\_size=12)

    plt.savefig("figure"+str(j)+".png")

    plt.close()

    #plt.show()

# printing average hops, cost and overhead of the system

print("Dijkstra algorithm was used to find the optimal path and there were 20 iterations for 50 nodes.")

print("The average hop count was: "+str(sum(hop\_array)/20))

print("The average cost was: "+str(sum(cost\_array)/20))

print("The average hello packets carried were: "+str(sum(overhead\_array)/(20\*50)))

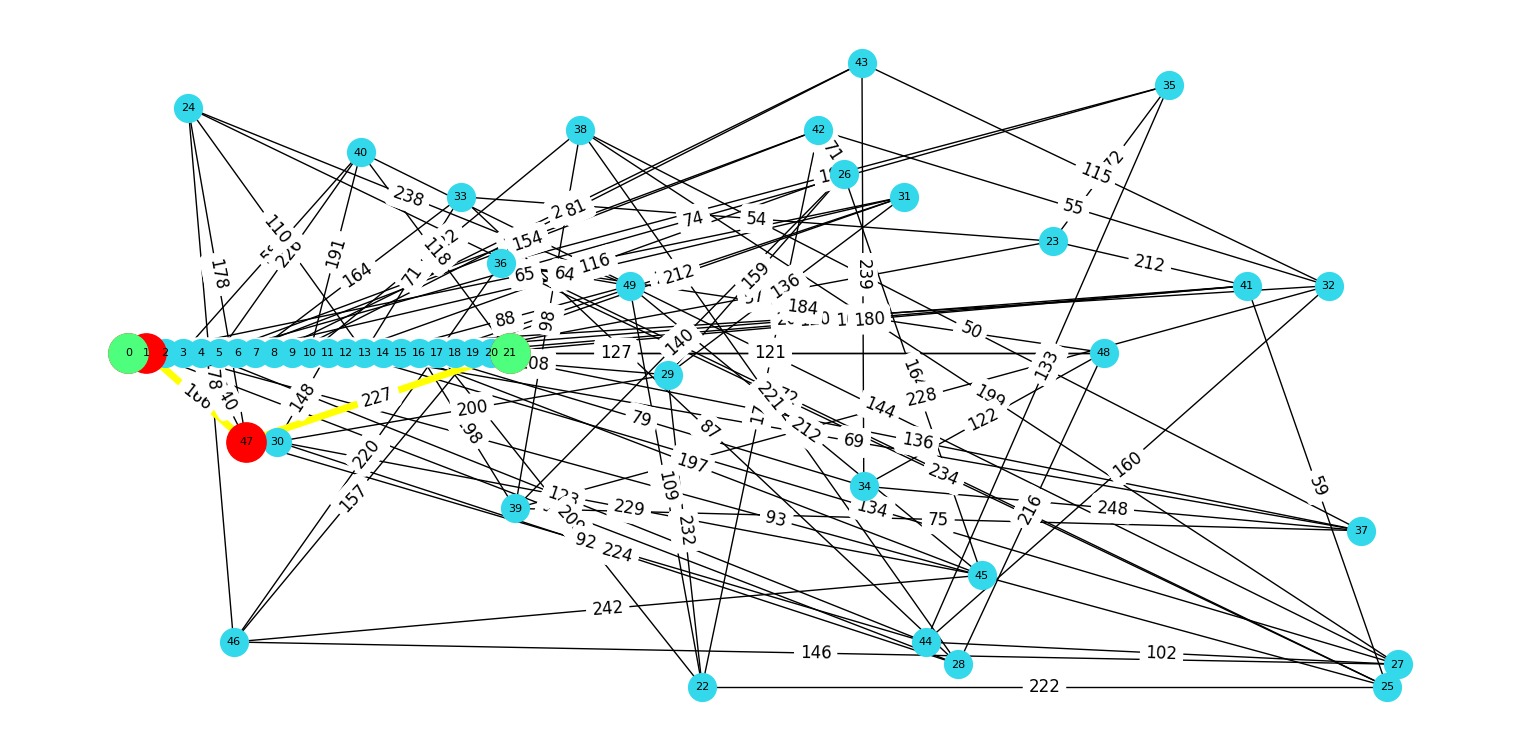
print("The average overhead(hello packets) bytes carried were: "+str(sum(overhead\_array)/20))

print("The average payload bytes carried were: "+str(sum(payload\_array)/20))

input("")

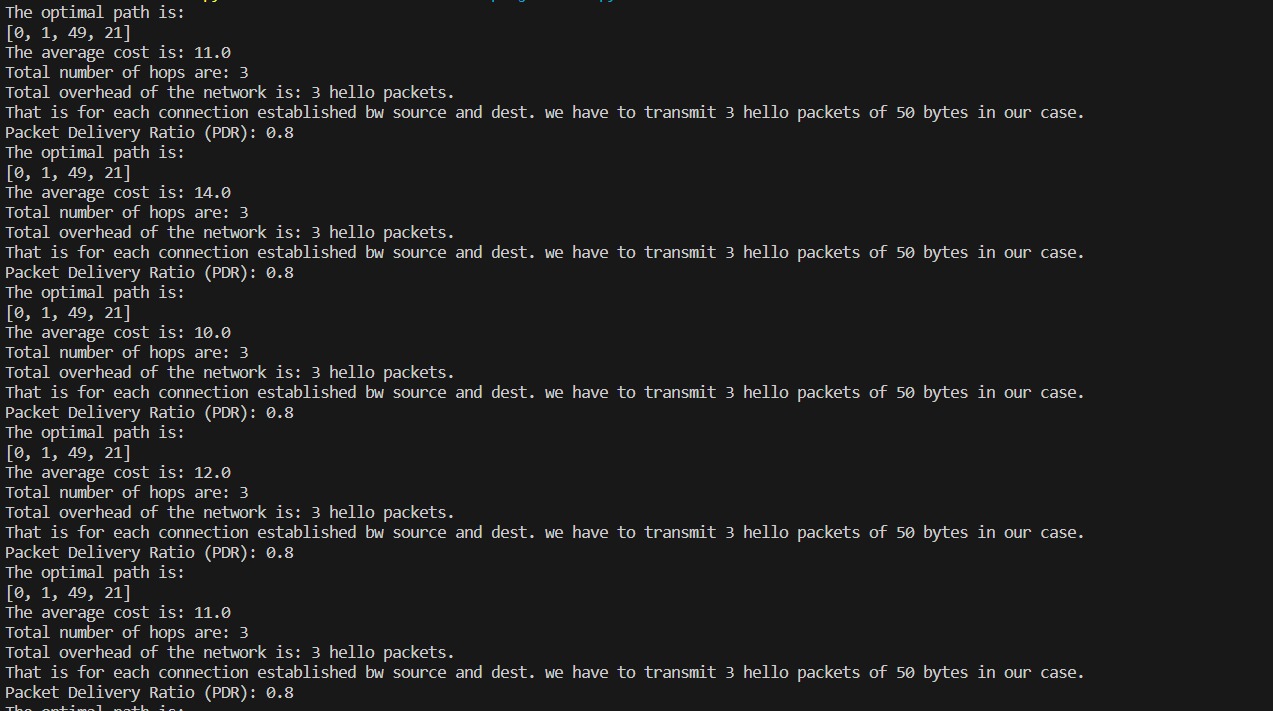
**Network Topology:**

* Topology consist of a simple graph of 50 nodes where 20 nodes are represented linearly.
* Source and destination nodes are colored green.
* Optimal path including link costs mentioned is highlighted with yellow in the topology.
* The nodes visited in the optimal path are colored red.



**Observations:**

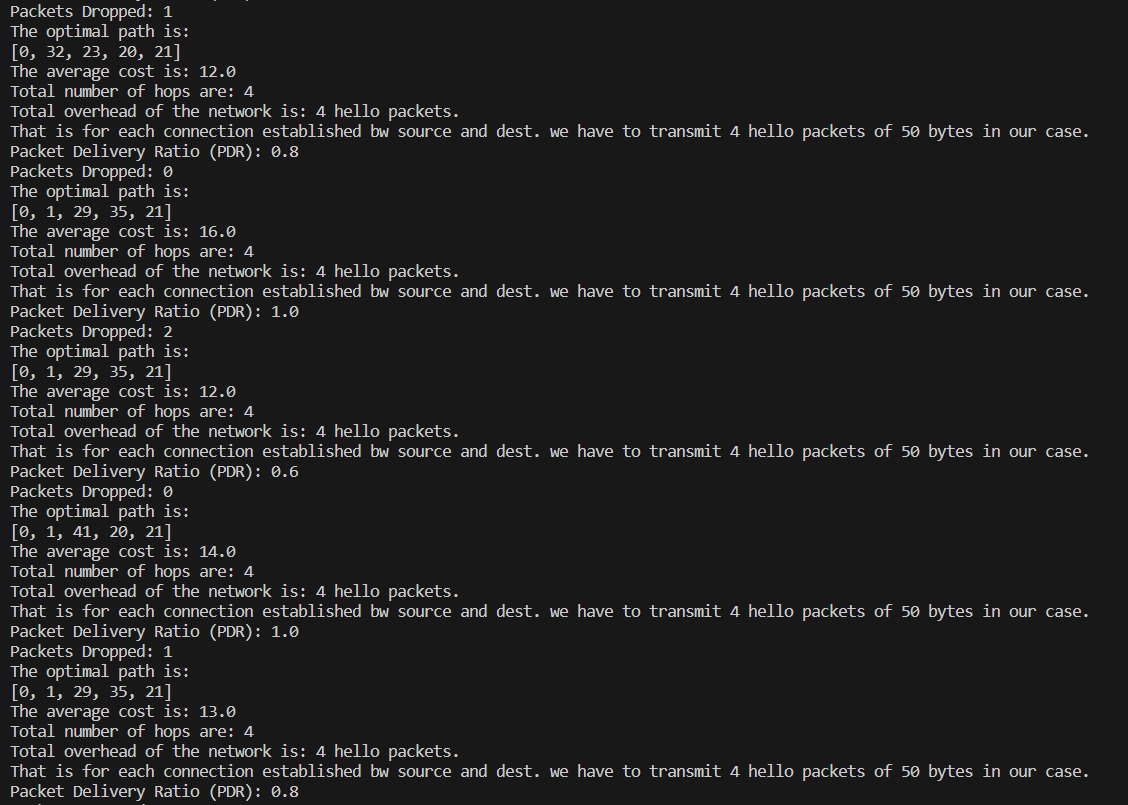
**Iteration 0 to 4:**



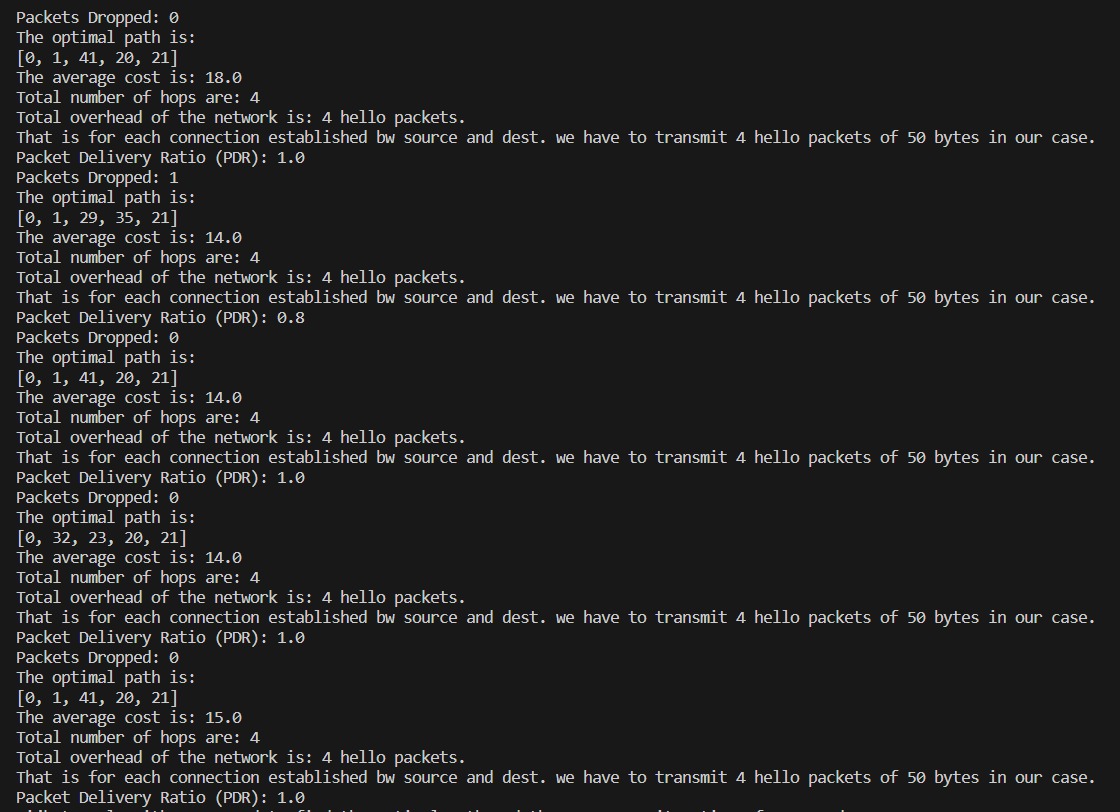
**Iteration 5 to 9:**



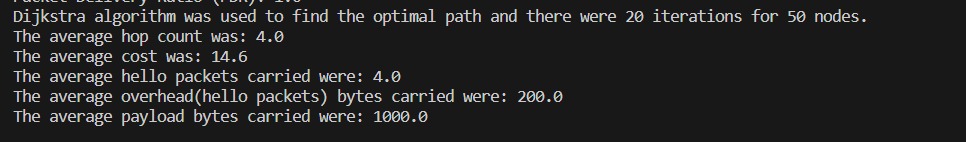
**Iteration 10 to 14:**



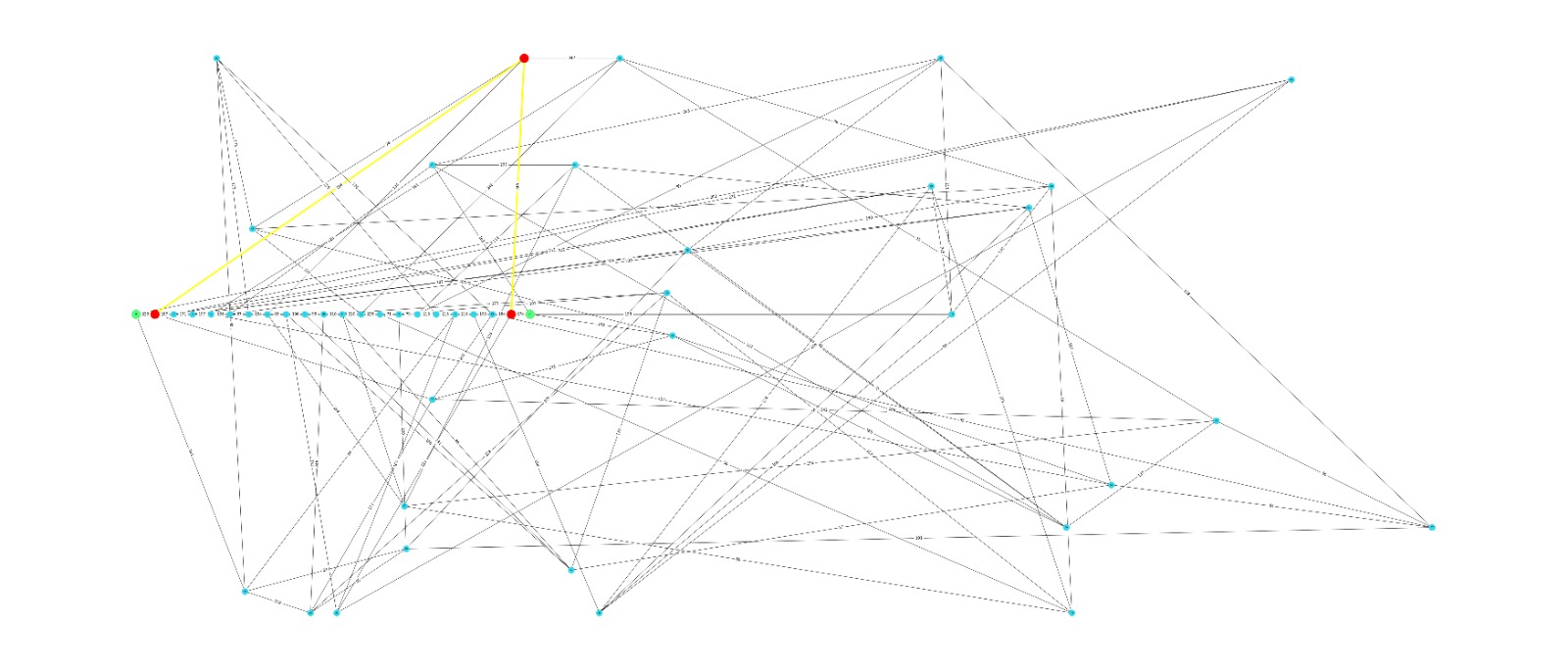
**Iteration 15 to 19:**



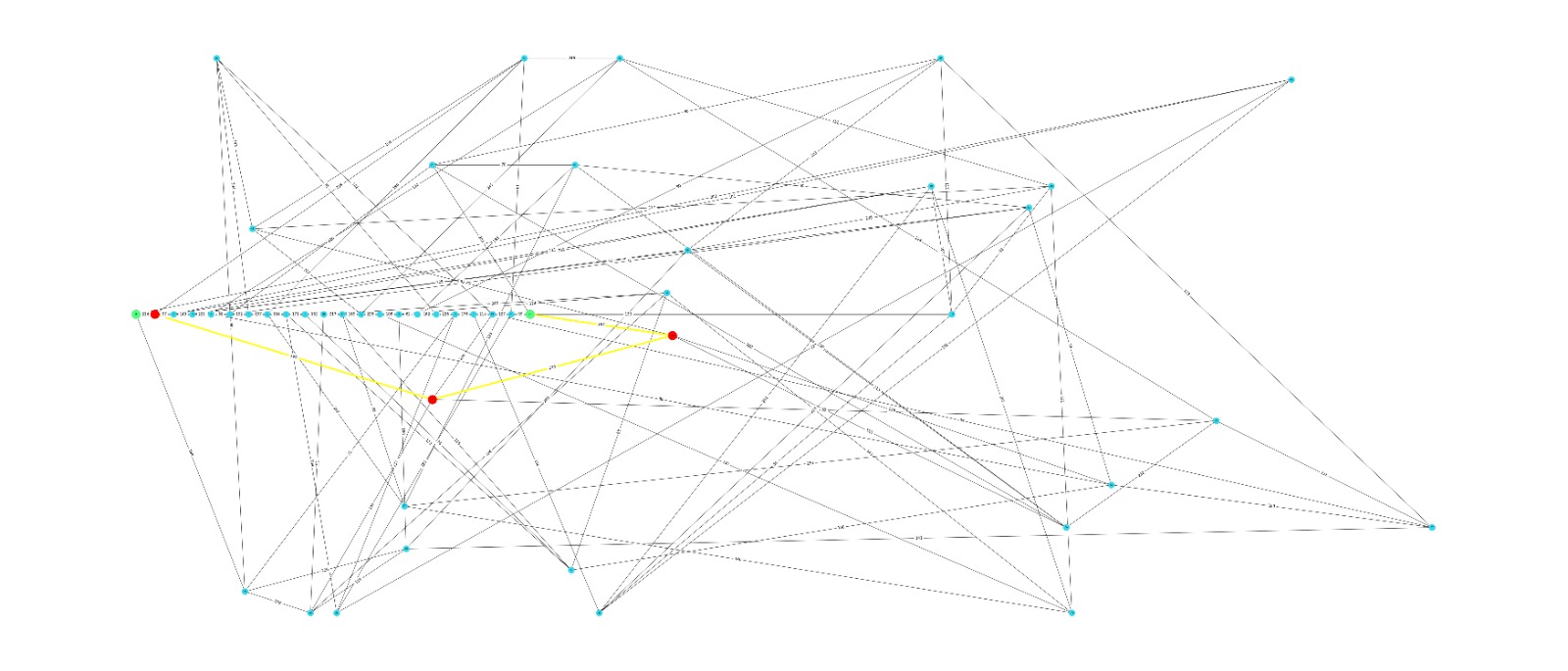
**Average hops, cost and overhead of the system**

****

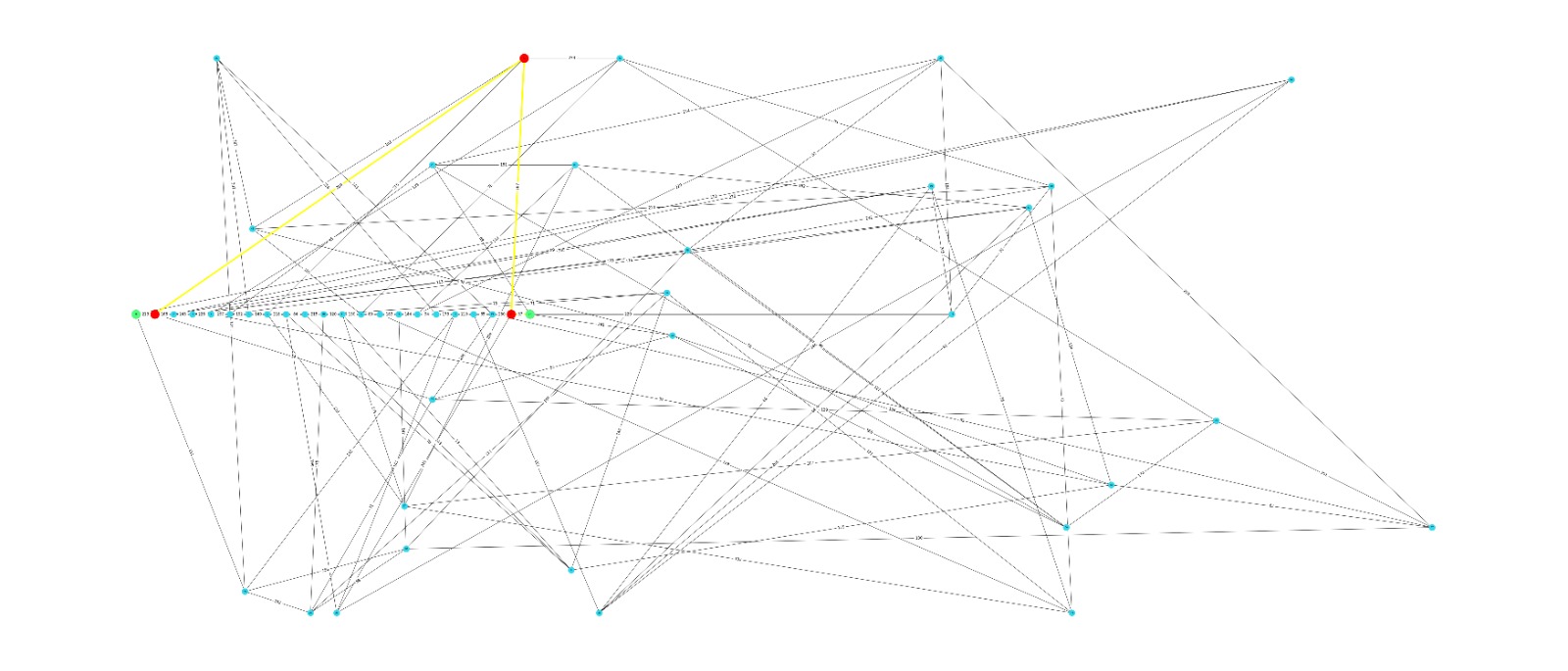
**Iteration 0 topology:**

****

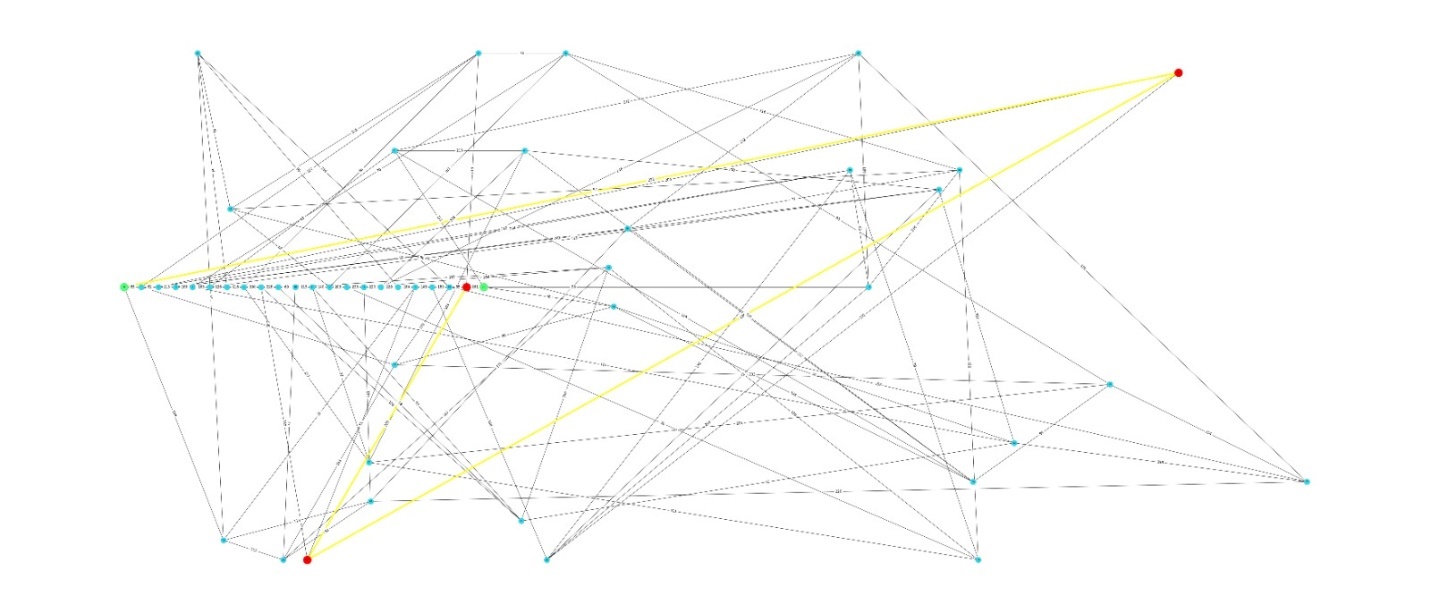
**Iteration 1 topology:**

****

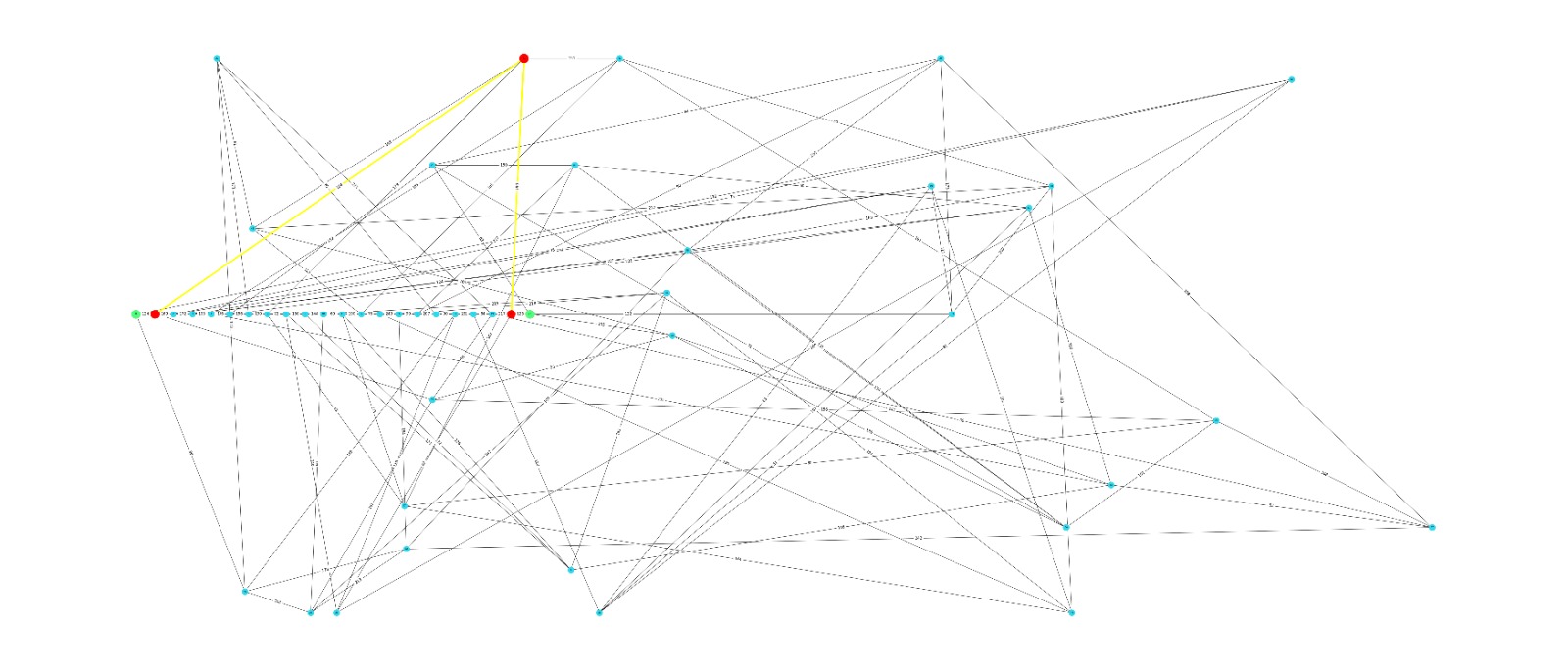
**Iteration 2 topology:**

****

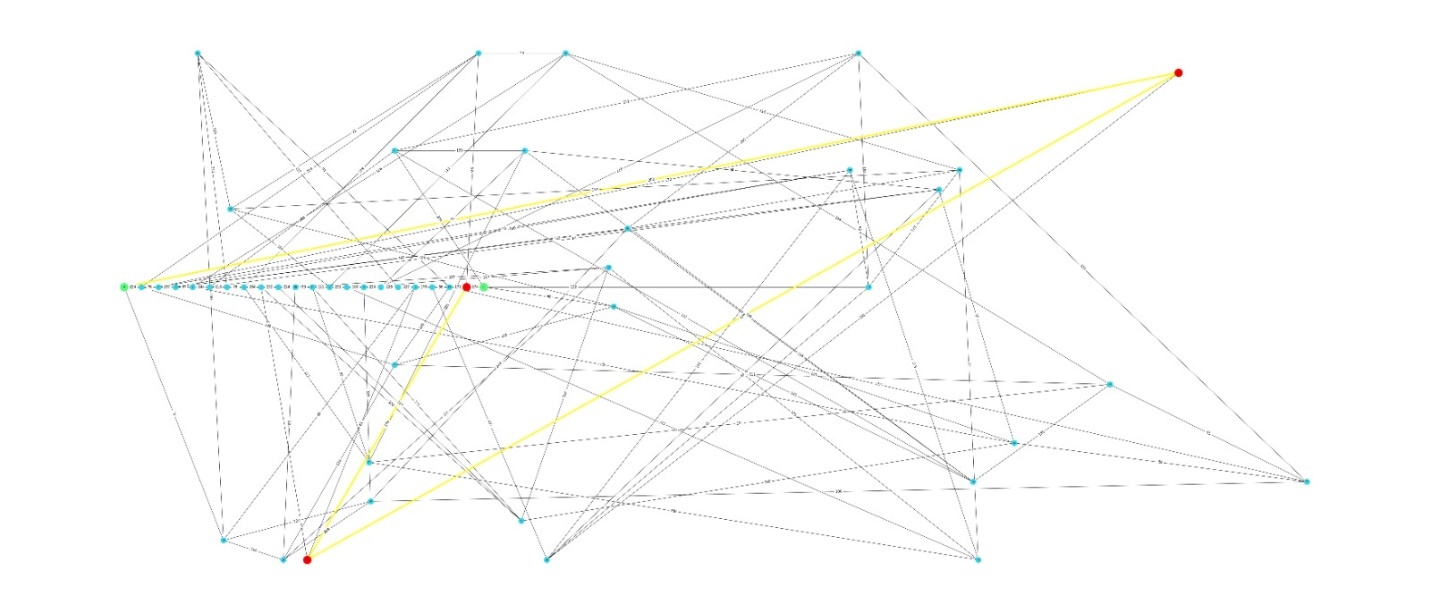
**Iteration 3 topology:**

****

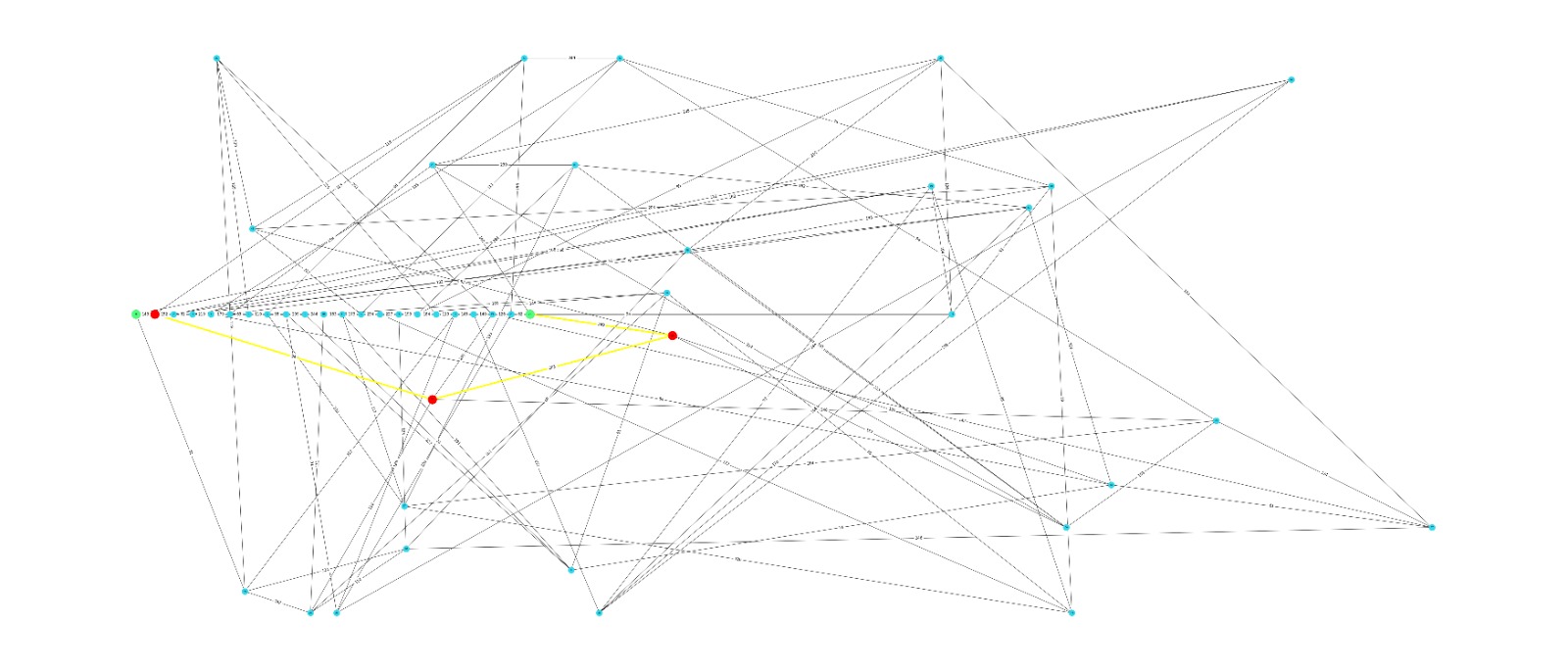
**Iteration 4 topology:**

****

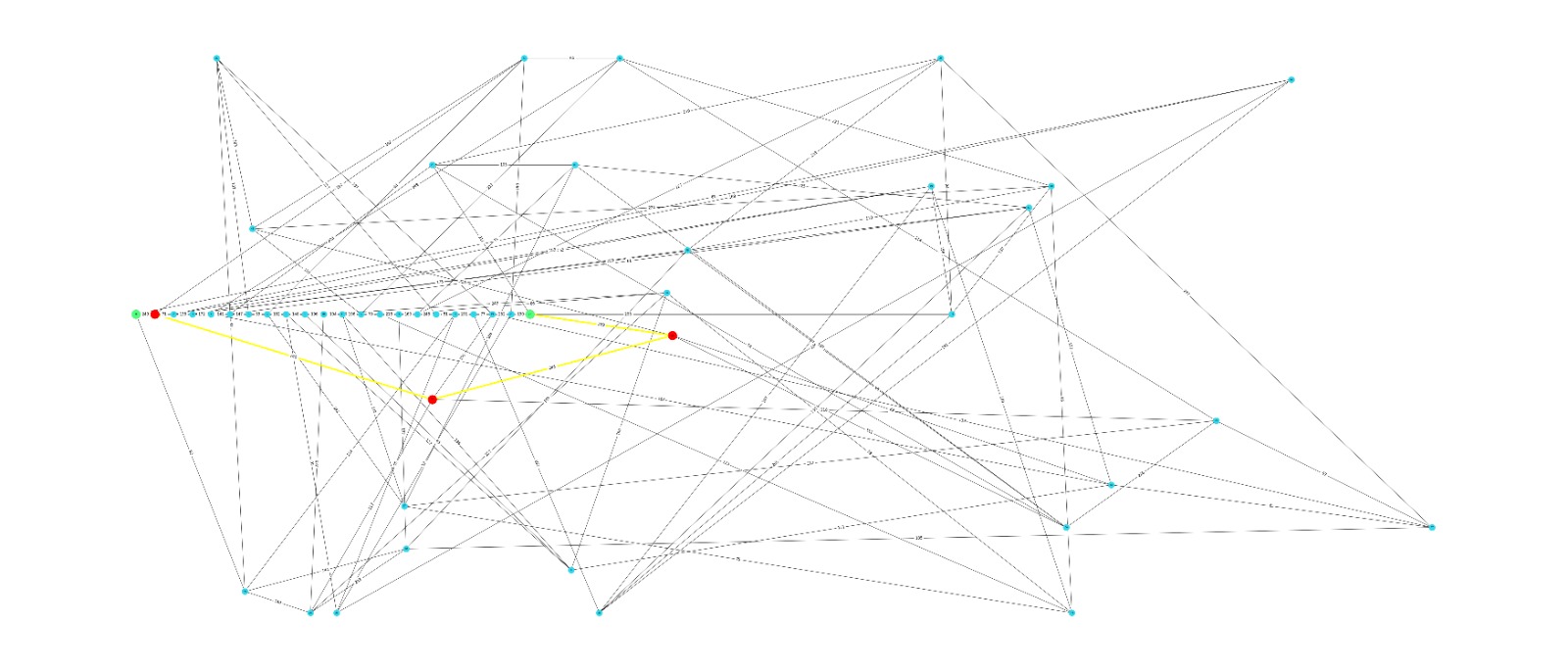
**Iteration 5 topology:**

****

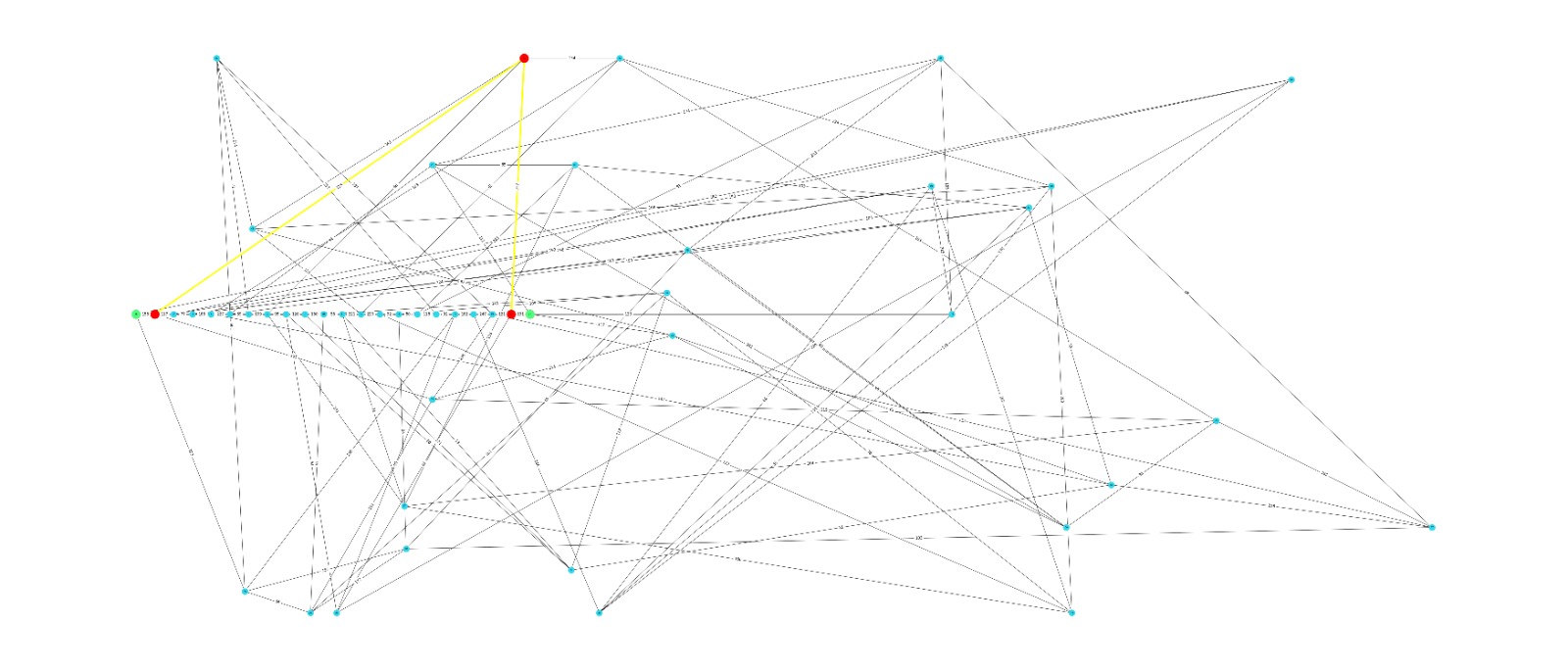
**Iteration 6 topology:**

****

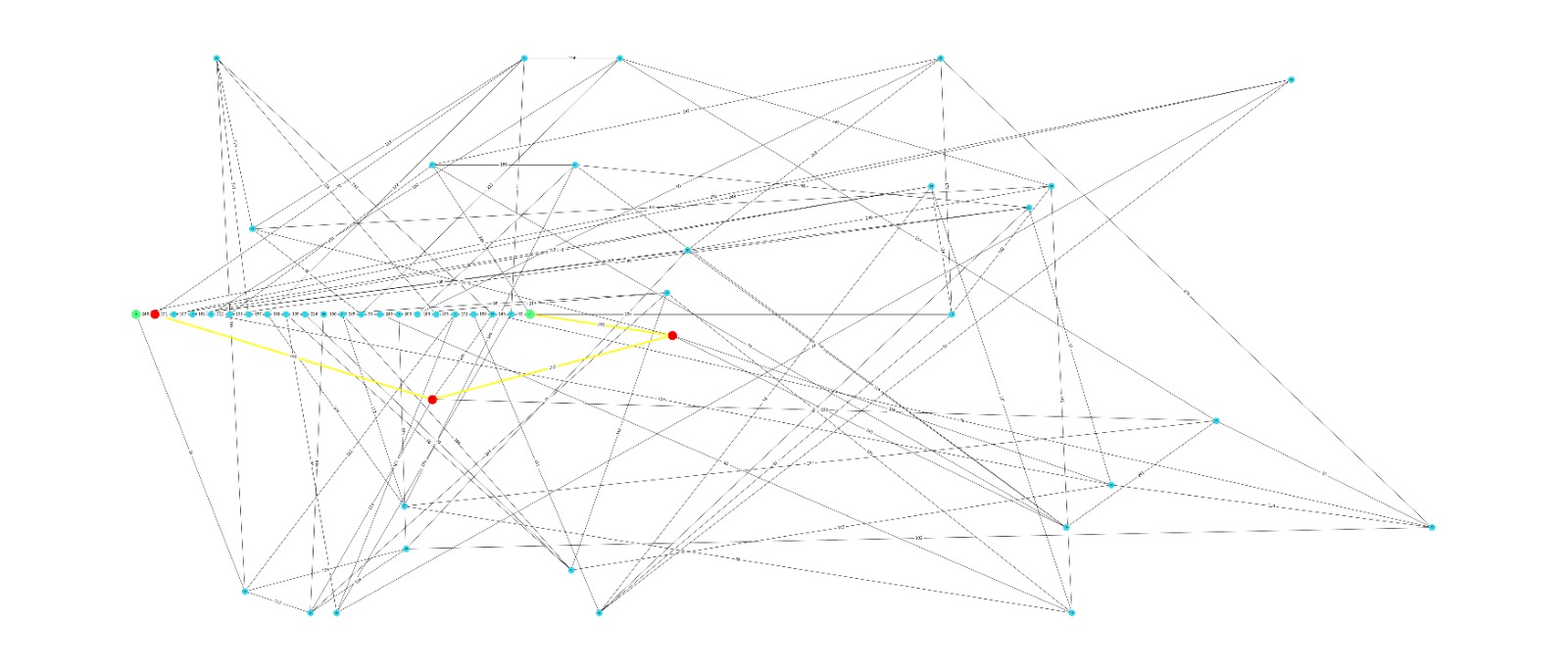
**Iteration 7 topology:**

****

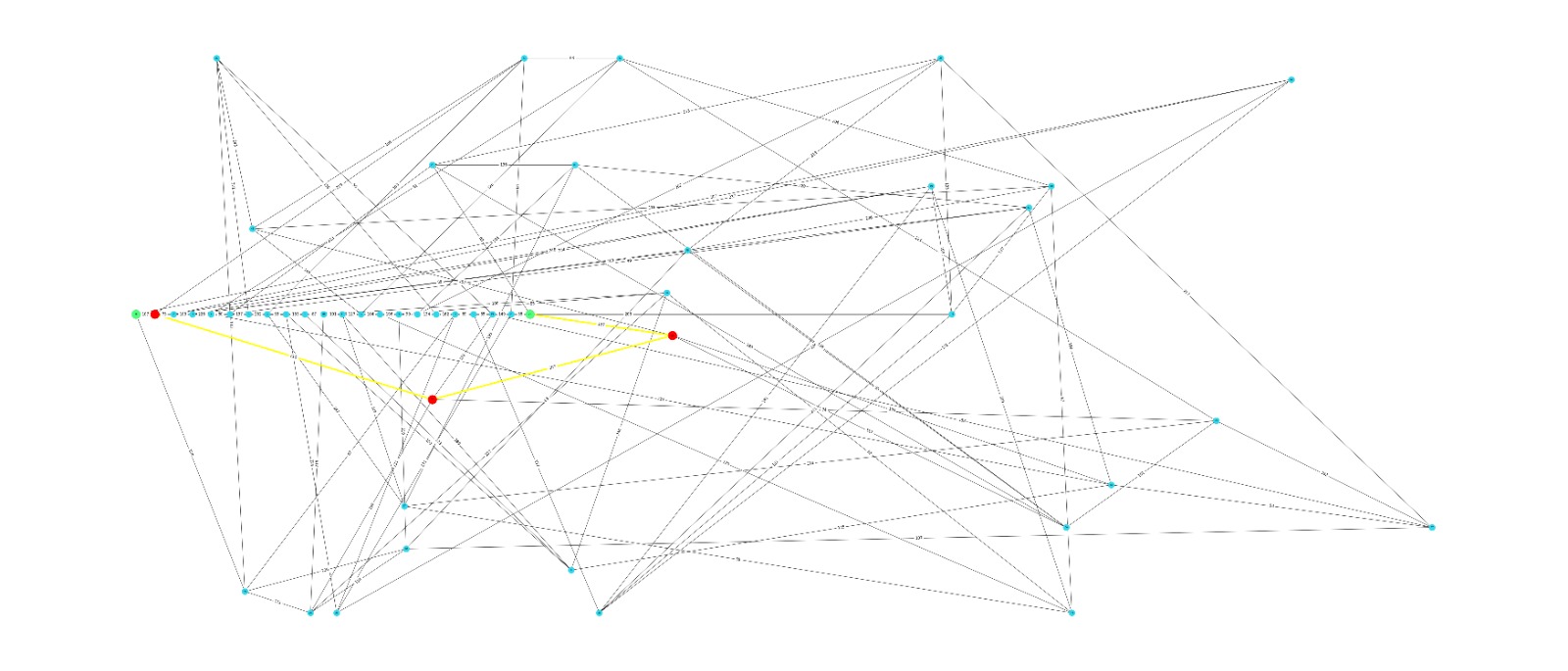
**Iteration 8 topology:**

****

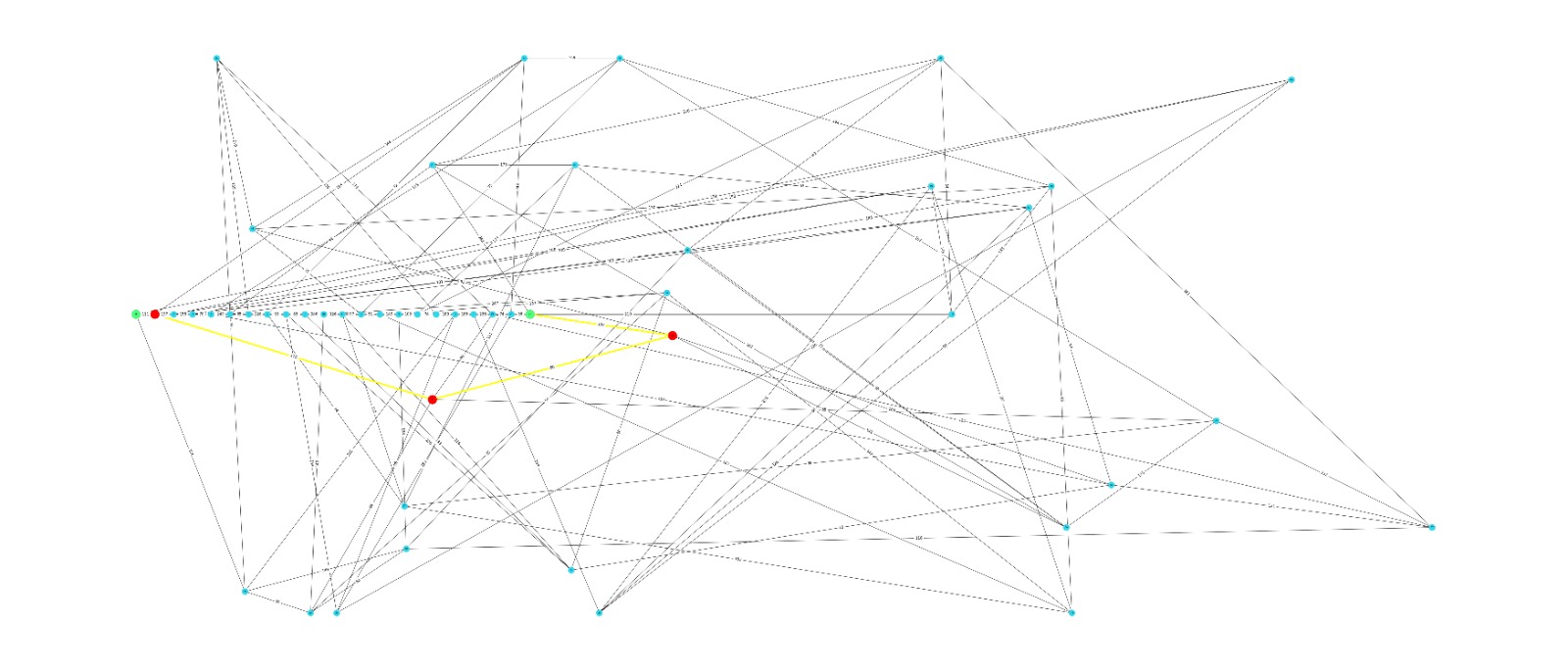
**Iteration 9 topology:**

****

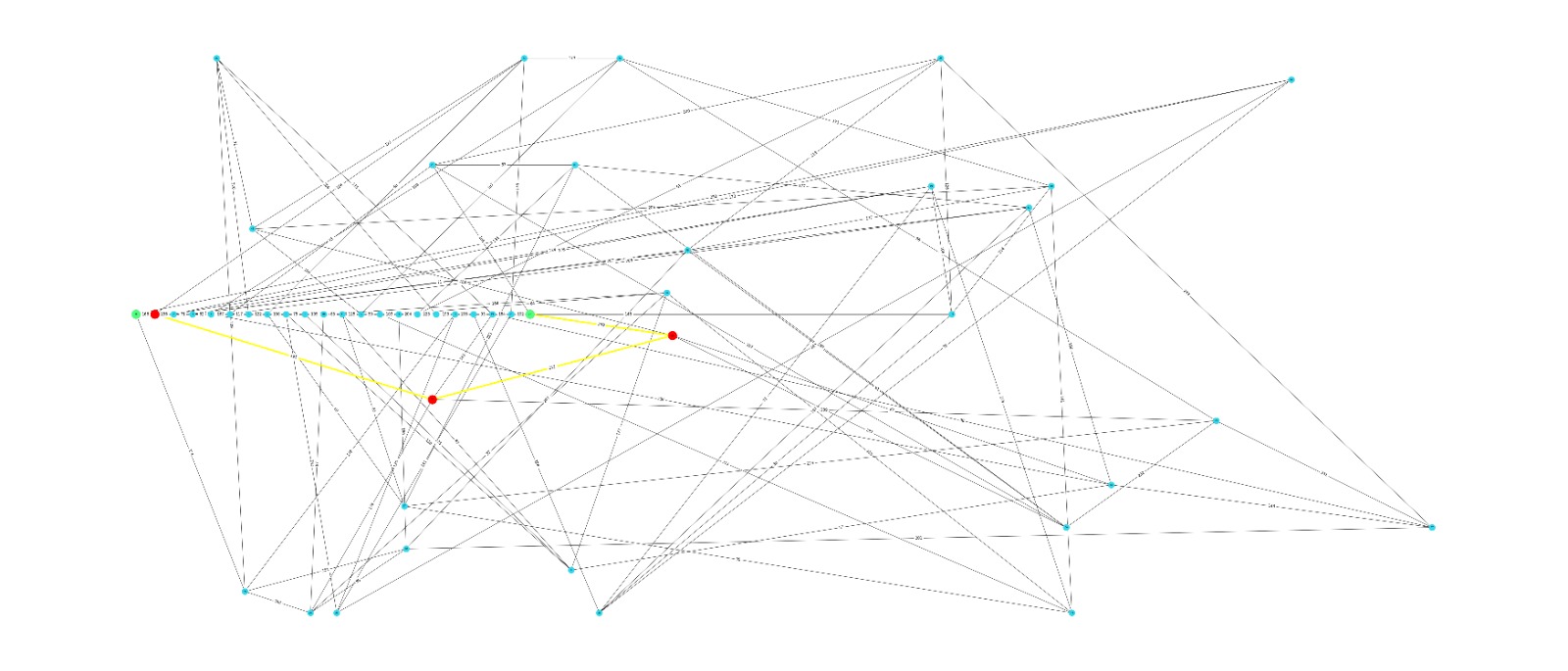
**Iteration 10 topology:**

****

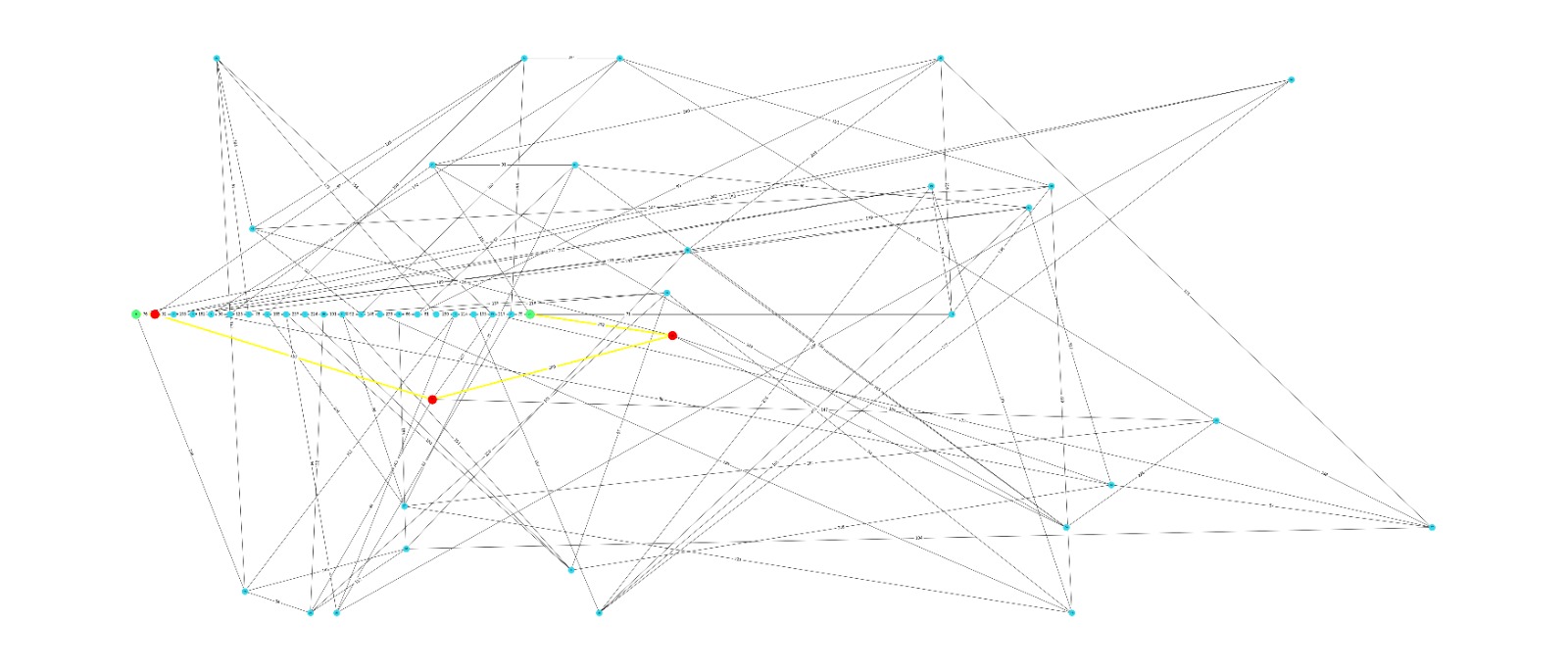
**Iteration 11 topology:**

****

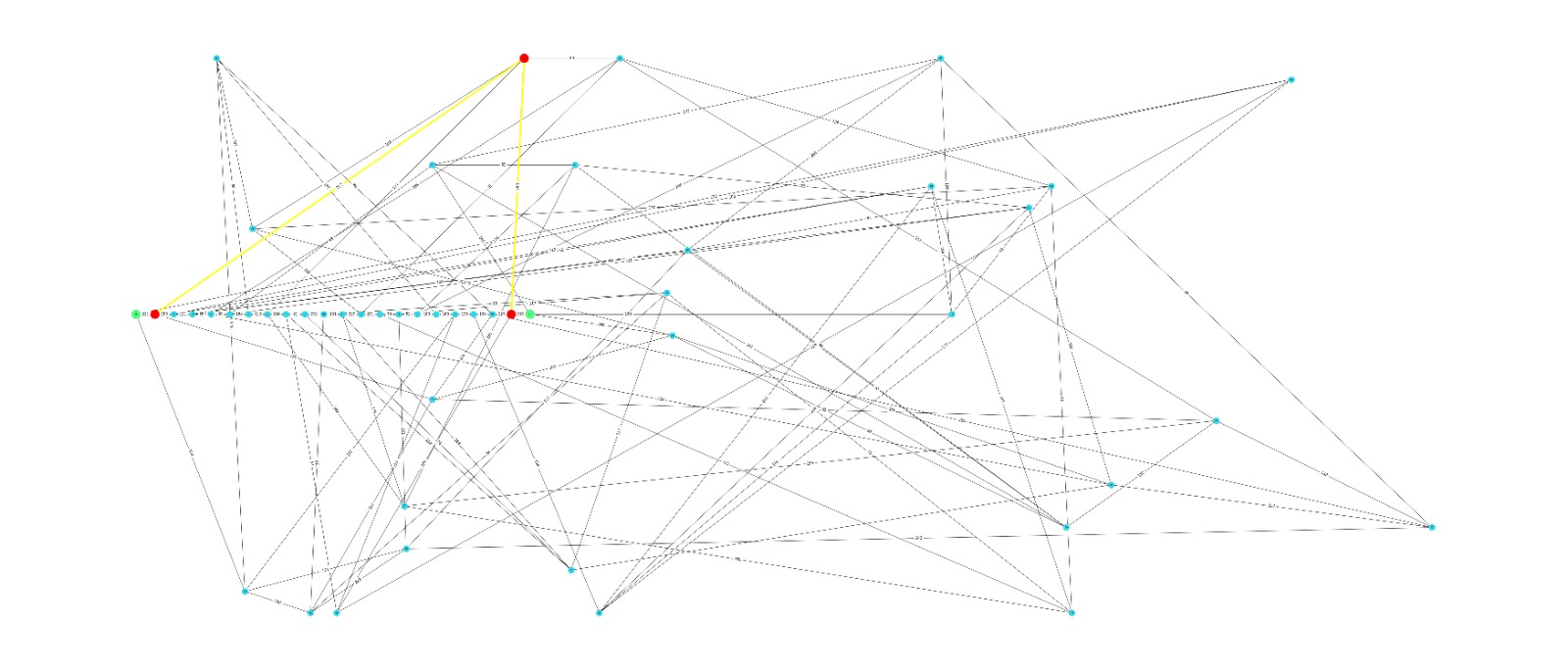
**Iteration 12 topology:**

****

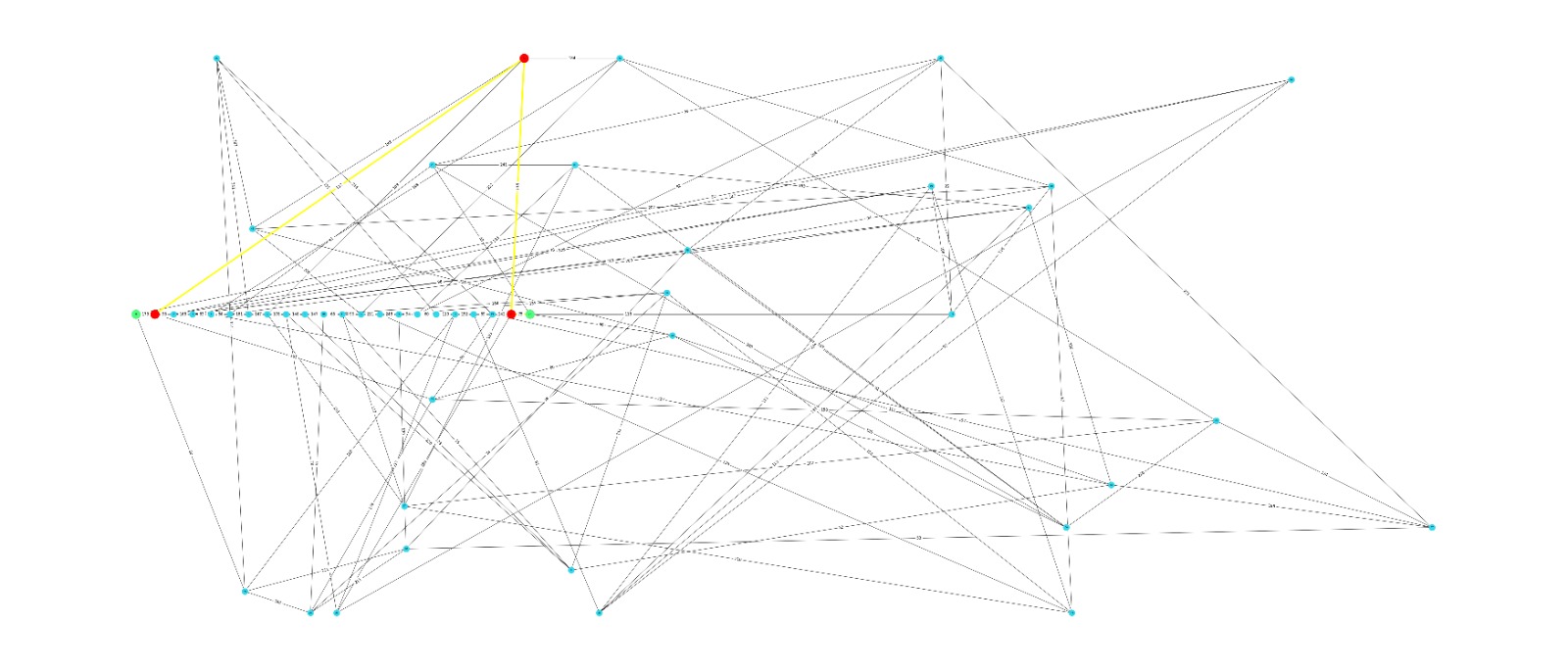
**Iteration 13 topology:**

****

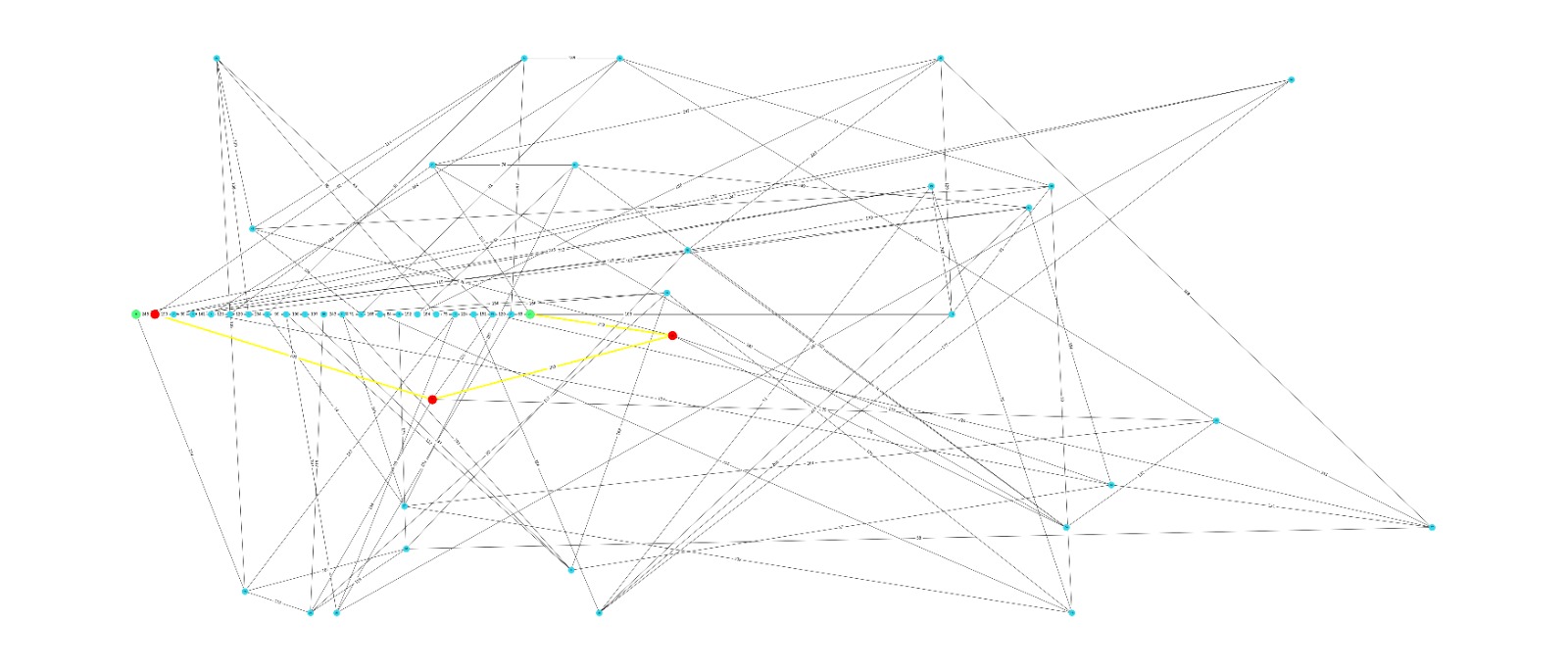
**Iteration 14 topology:**

****

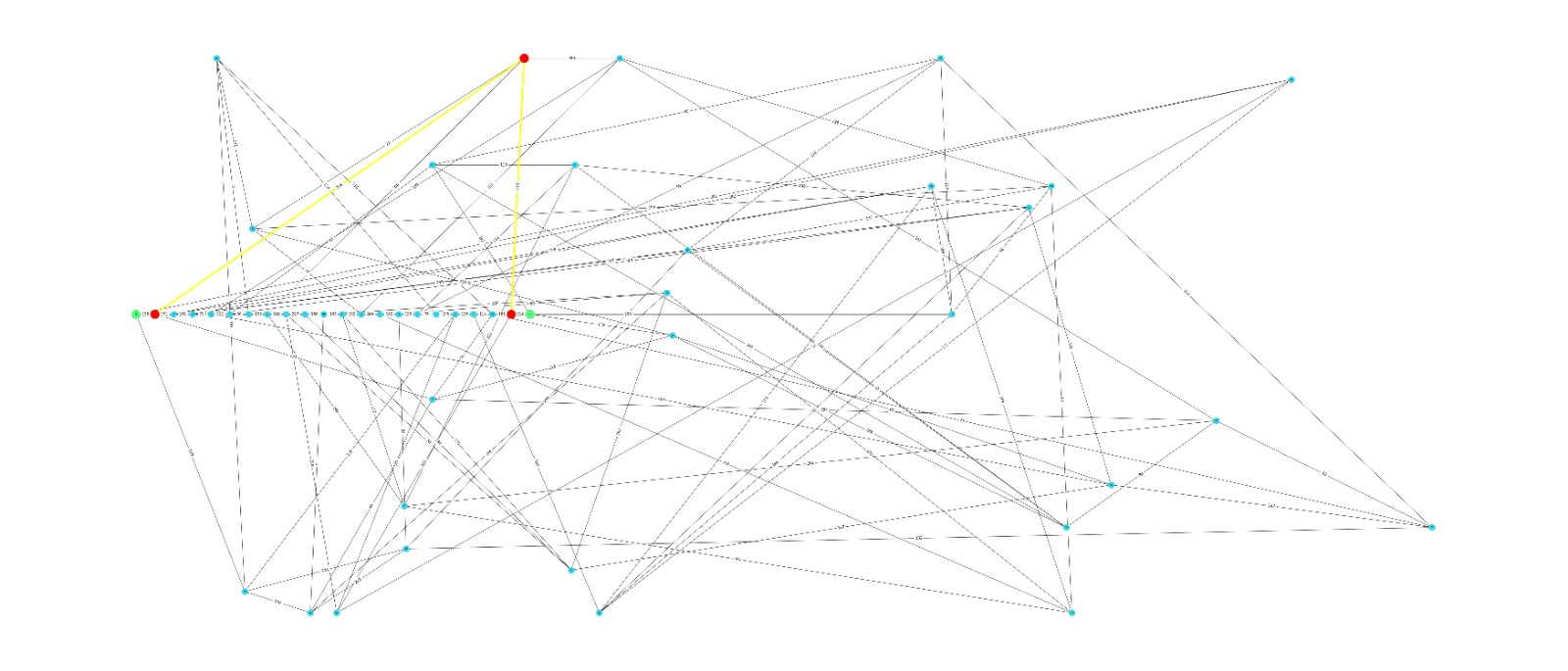
**Iteration 15 topology:**

****

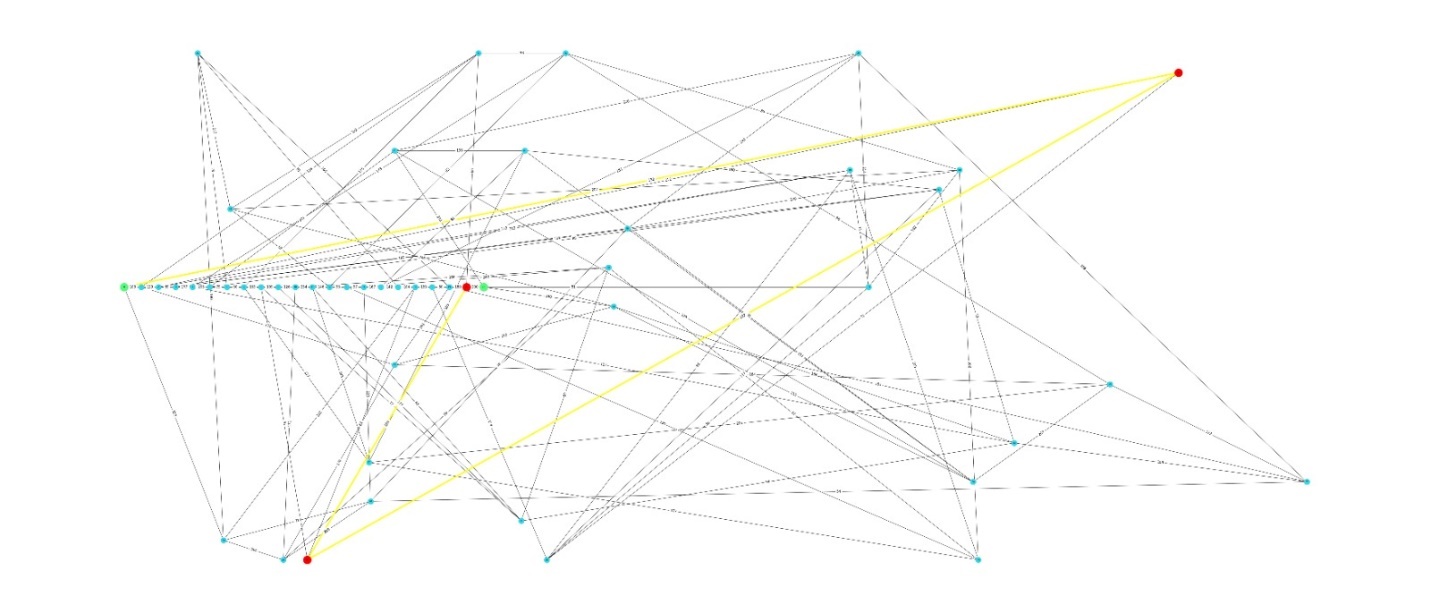
**Iteration 16 topology:**

****

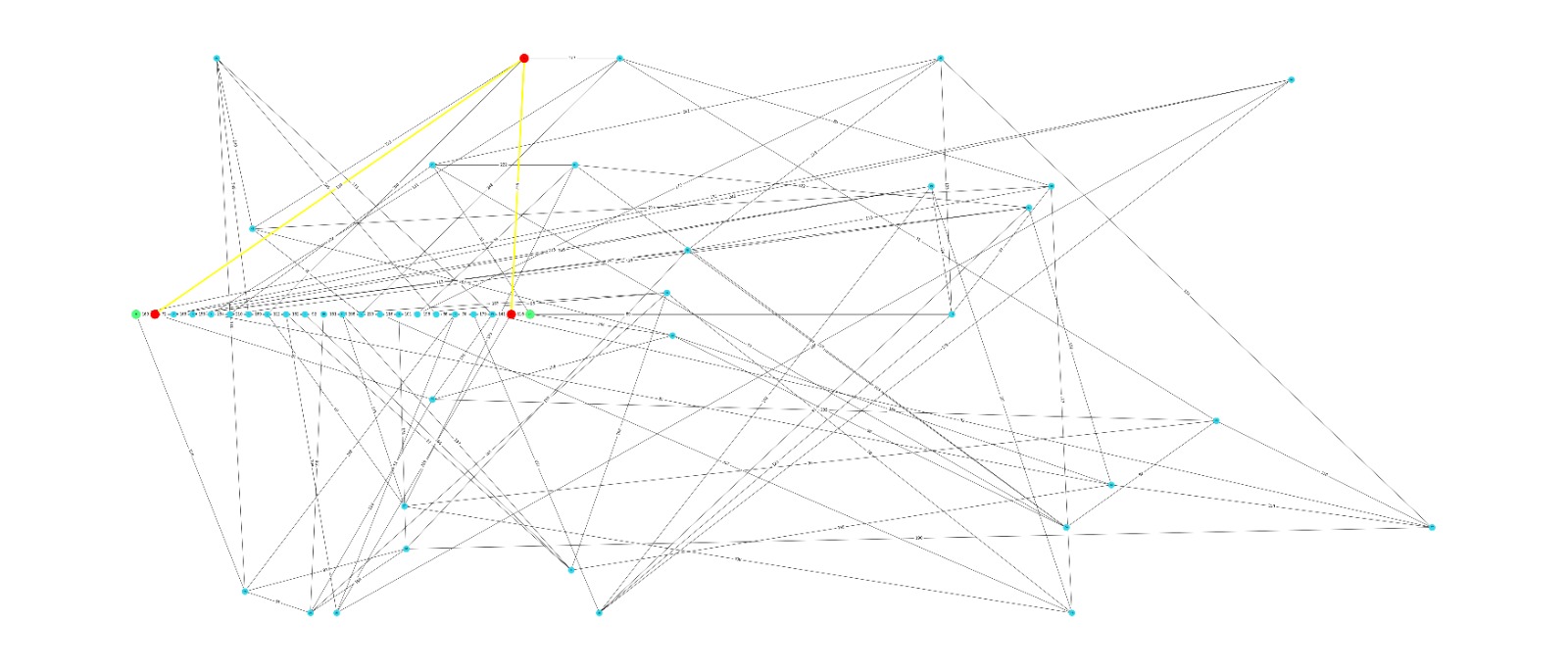
**Iteration 17 topology:**

****

**Iteration 18 topology:**

****

**Iteration 19 topology:**

****

**Table of Values:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration No. | Total No. of Hops | Shortest Path | Average Cost | Total No. of Hello Packets |
| 0 | 3 | 0,1,49,21 | 11 | 3 |
| 1 | 3 | 0,1,49,21 | 14 | 3 |
| 2 | 3 | 0,1,49,21 | 10 | 3 |
| 3 | 3 | 0,1,49,21 | 12 | 3 |
| 4 | 3 | 0,1,49,21 | 11 | 3 |
| 5 | 4 | 0,1,41,20,21 | 12 | 4 |
| 6 | 4 | 0,1,29,35,21 | 12 | 4 |
| 7 | 4 | 0,1,41,20,21 | 18 | 4 |
| 8 | 4 | 0,32,23,20,21 | 13 | 4 |
| 9 | 4 | 0,1,41,20,21 | 17 | 4 |
| 10 | 4 | 0,32,23,20,21 | 12 | 4 |
| 11 | 4 | 0,1,29,35,21 | 16 | 4 |
| 12 | 4 | 0,1,29,35,21 | 12 | 4 |
| 13 | 4 | 0,1,41,20,21 | 14 | 4 |
| 14 | 4 | 0,1,29,35,21 | 13 | 4 |
| 15 | 4 | 0,1,29,35,21 | 17 | 4 |
| 16 | 4 | 0,1,29,35,21 | 19 | 4 |
| 17 | 4 | 0,1,29,35,21 | 13 | 4 |
| 18 | 4 | 0,1,29,35,21 | 17 | 4 |
| 19 | 4 | 0,1,41,20,21 | 12 | 4 |

**Analysis of Findings:**

* Shortest/ Optimal path for every observation in 20 iteration is found.
* The shortest path in the first topology **0 to 1 to 49 to 21**
* Average cost for this observation is calculated and stated as **11.0**.
* Total number of hops between source and destination duly stated as **3**.
* Total overheads in the form of Hello Packets is **3** Packets.
* The PDR for this observation is 0.8
* The Algorithm used to find the shortest path is Dijkstra.
* Total number of iterations to find shortest path were **20**.
* Average cost for all 20 observation is calculated and stated as **11.0.**

**Conclusion:**

* We have taken special care of the fact that that at least 20 nodes separate source and destination node.
* According to the instructions the 22 nodes were placed in a linear fashion.
* All the questions such as shortest path, hop count, hello packets, over head along their averages for 20 iteration found.
* 20 iterations have been made; their results are all presented in the report.