Project Report

Routing Optimization Using Graph Algorithm

1. Introduction

With the growing importance of computer networks, optimizing routing has become a critical challenge. This project focuses on enhancing network routing using Python-based tools like NetworkX and analytical techniques to improve data flow and minimize latency. By refining routing protocols, it aims to ensure scalable, efficient, and reliable data transmission, vital for modern digital communications.

2. Networking Tools Used:

- NetworkX is a Python library used for creating, analyzing, and visualizing complex networks and graphs. It provides tools for representing nodes and edges, calculating shortest paths, centrality measures, and performing graph-related operations such as searching, clustering, and plotting.
- MANET (Mobile Ad Hoc Networks) is a type of wireless network that is self-configuring, decentralized, and consists of mobile devices (nodes) that communicate with each other without relying on fixed infrastructure. Nodes in MANETs can move dynamically, and the network topology changes frequently, making routing and communication challenging.

3. Methodology

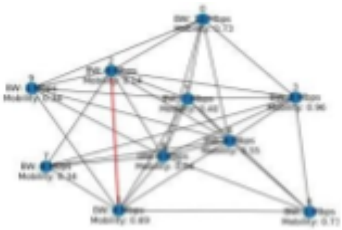
3.1. Setup (Preprocessing)

- **Import Libraries:** Utilize networkx for graph handling, pandas for data manipulation, and numpy for computations.
- **Graph Parameters:** Define total nodes, coordinate limits, and initialize an undirected graph (nx.Graph()).
- **Node Attributes:** Assign random values for bandwidth, mobility, transmission power, battery capacity, and packet loss.
- **Edges:** Establish connections based on criteria (e.g., Euclidean distance) and store graph data in a CSV file (improved.csv)

3.2. Dijkstra's Algorithm

- **Shortest Path Calculation:** From a source node, compute shortest paths to other nodes by minimizing edge weights.
- **Process:** Update distances iteratively, reconstruct the shortest path using predecessor nodes, and highlight the optimal route.

3.2.1. Results



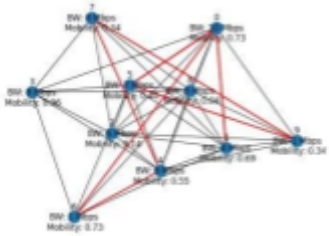
avg health of transfer 62.823870528172265

	node_a	node_b	bandwidth	mobility	transmission_power	battery_capacity	packet_loss	health
0	1	2	4	0.894825	6.449858	53.790862	0.710124	62.823871

3.3. Genetic Algorithm

- **Path Optimization:** Generate a population of 50 paths and calculate fitness based on node attributes and path efficiency.
- **Operations:**
  - Selection, crossover, and mutation ensure diversity and convergence toward optimal paths.
  - Best fitness path represents the most efficient route.

3.3.1. **Results:**



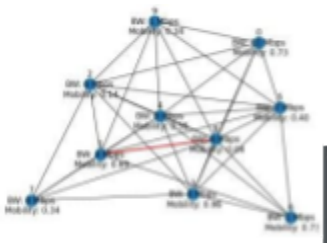
avg health of transfer 67.58585481277866

	node_a	node_b	bandwidth	mobility	transmission_power	battery_capacity	packet_loss	health
0	6	4	1	0.731111	8.281916	68.080326	0.787822	74.803009
1	4	7	6	0.647761	8.281916	55.464350	0.983749	66.194756
2	9	5	1	0.387009	10.780957	50.924202	0.741870	61.566230
3	5	0	7	0.728436	10.847772	50.924202	0.580927	67.482609
4	0	5	7	0.728436	10.847772	50.924202	0.580927	67.482609

3.4. Bellman-Ford Algorithm

- **Dynamic Weights:** Adjust edge weights based on node metrics like bandwidth and mobility.
- **Process:** Relax edges over |V| - 1 iterations to compute shortest paths while detecting negative cycles.

3.4.1. Results:



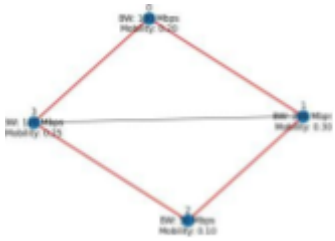
avg health of transfer 50.913689348496

	node_a	node_b	bandwidth	mobility	transmission_power	battery_capacity	packet_loss	health
0	1	0	4	0.894825	10.535894	37.58297	0.51072	50.913609

3.5. Traveling Salesman Problem (TSP) with Nearest Neighbor (NN)

- **Path Scoring:** Use node attributes (e.g., bandwidth, mobility) to evaluate "health" of routes.
- **Optimization:** Identify the cycle with the highest score, representing the most efficient network traversal.

3.5.1. Results:



Average health of transfer: 103.05

	node_a	node_b	bandwidth	mobility	transmission_power	battery_capacity
0	0	1	100	0.10	10	20
1	1	2	50	0.30	5	25
2	2	3	50	0.25	5	22
3	3	0	100	0.25	10	20

	packet_loss	health
0	0.10	129.60
1	0.15	79.55
2	0.15	76.50
3	0.10	129.65