Network Path Optimization: A Dynamic Routing Protocol Simulation

Computer Networks Course Project Report

Report by:

CS22B1080 Avanee Sarang Lakare

Contents

1 Aim

The project focuses on developing a dynamic routing protocol using the Open Shortest Path First (OSPF) routing protocol and simulates the optimization of network paths and enhances the understanding of adaptive routing in computer networks. The project aims to:

- Implement OSPF concepts, such as Link-State Advertisements (LSAs) and the Dijkstra algorithm for shortest path computation.
- Analyze the impact of network events, such as congestion and link failures, on routing decisions.
- Demonstrate the construction and updates of routing tables dynamically.
- Evaluate network performance with Quality of Service (QoS) constraints to ensure efficient data flow.
- Visualize the impact of routing decisions and network adjustments through interactive graphs.

2 Introduction

Routing protocols are critical in modern computer networks to ensure efficient and reliable data transmission. The Open Shortest Path First (OSPF) protocol is a dynamic, link-state routing protocol used widely in enterprise and service provider networks. This project simulates OSPF to enhance understanding of adaptive routing in computer networks. The simulation includes features such as route recalculation, QoS constraints, and network resilience.

Dynamic routing protocols like OSPF are crucial in modern networks to ensure efficient data flow and adapt to network changes. These protocols form the backbone of the internet, data centers, and enterprise networks, ensuring resilience and reliability. OSPF's ability to adapt to network topology changes makes it a cornerstone of routing in IP networks. Its features, such as fast convergence, scalability, and support for multiple metrics, are essential for maintaining efficient communication in dynamic environments.

In a comparative study between Routing Information Protocol (RIP), Enhanced Interior Gateway Routing Protocol (EIGRP) and Open Shortest Path First (OSPF), it was concluded that RIP gives best path for small area network but not in large network, EIGRP protocol is implemented

only in cisco routers and OSPF has the unlimited router range and takes less time to send data. So out of three protocols OSPF protocol is best suitable path for data sharing, web browsing.

Networks often encounter dynamic events, such as traffic congestion and link failures. Without a robust protocol like OSPF, routing inefficiencies can lead to delays, packet loss, and poor Quality of Service (QoS). This project addresses these challenges by simulating OSPF's adaptive routing in a network subjected to such events, ensuring minimal disruption and optimal routing.

3 System Design

1. Architecture

   The network topology for this simulation consists of nodes representing routers and weighted edges representing the links between them. The weights signify the cost of traversal based on latency, bandwidth, or other metrics. A simple mesh topology is used with five nodes (A, B, C, D, E) and edges assigned weights, latencies, and bandwidths. This allows for testing various routing scenarios effectively.

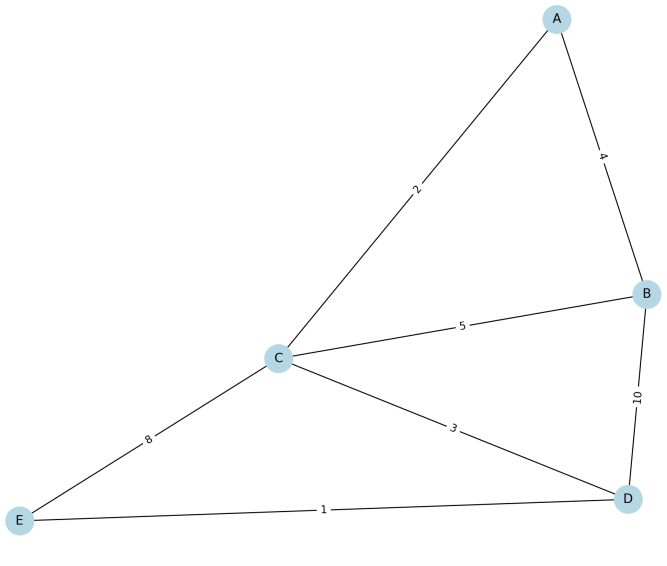

Figure 1: Initial Network Topology

2. Protocol/Concept Details

   (a) OSPF Protocol

   - Link-State Advertisements (LSA): Nodes exchange information about their directly connected links with other nodes to build a complete network view.
   - Shortest Path First (SPF) Algorithm: The Dijkstra algorithm is used to compute the shortest path from a source node to all other nodes.
   - Routing Tables: Each node builds and updates its routing table dynamically based on the network's current state.

   (b) Dynamic Events

   - Congestion: Simulates weight changes on edges to reflect varying traffic conditions.
   - Link/Node Failures: Simulates the removal of links or nodes from the network to evaluate routing recalculations.

3. Tools and Technologies

   - Programming Language: Python
   - Libraries:
     - NetworkX (graph representation)
     - Matplotlib (visualization)
     - Heapq (priority queues for Dijkstra's algorithm)
   - Environment: Python 3.10+ running on any modern operating system.

4 Implementation

1. Project Code Structure

   (a) Network Initialization

   The first step in the implementation was to create a static network topology using the networkx library. This topology consists of:

   - Nodes: Representing routers in the network.
   - Edges: Representing links between the routers.

- Edge Attributes:
  - weight: Cost of traversing the link (e.g., based on latency or traffic)
  - latency: Time delay associated with the link
  - bandwidth: Maximum data transfer rate for the link

```python
# Creating a predefined network with some edges
def create_initial_network():
    G = nx.Graph()
    G.add_edge('A', 'B', weight=4, latency=10, bandwidth=100)
    G.add_edge('A', 'C', weight=2, latency=5, bandwidth=50)
    G.add_edge('B', 'C', weight=5, latency=8, bandwidth=30)
    G.add_edge('B', 'D', weight=10, latency=15, bandwidth=70)
    G.add_edge('C', 'D', weight=3, latency=7, bandwidth=40)
    G.add_edge('D', 'E', weight=1, latency=4, bandwidth=100)
    G.add_edge('C', 'E', weight=8, latency=12, bandwidth=20)
    return G
```

Figure 2: Graph Initialization

(b) Shortest Path Calculation (OSPF SPF Algorithm)

The OSPF protocol calculates the shortest path from a source node to all other nodes using Dijkstra's algorithm. A function for the same implements the algorithm:

- Maintains a distances dictionary to track the shortest distance from the source to all nodes.
- Tracks the previous nodes for each destination node in the shortest path which is useful for reconstructing the path once the shortest distance is determined using the previous nodes dictionary
- Uses a priority queue (heapq) to process nodes based on the shortest distance.
- Optionally applies a Quality of Service (QoS) constraint, ignoring links with insufficient bandwidth.

```python
# OSPF-inspired function to calculate shortest paths
def ospf_shortest_paths(G, source_node, qos_bandwidth=None):
    distances = {node: float('inf') for node in G.nodes}
    previous_nodes = {node: None for node in G.nodes}
    distances[source_node] = 0

    priority_queue = [(0, source_node)]

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_distance > distances[current_node]:
            continue

        for neighbor in G.neighbors(current_node):
            weight = G[current_node][neighbor]['weight']
            bandwidth = G[current_node][neighbor].get('bandwidth', float('inf'))

            # QoS constraint: Ignore links with insufficient bandwidth
            if qos_bandwidth and bandwidth < qos_bandwidth:
                continue

            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_node
                heapq.heappush(priority_queue, (distance, neighbor))

    shortest_paths = {}
    for node in G.nodes:
        path = []
        current = node
        while current is not None:
            path.insert(0, current)
            current = previous_nodes[current]
        if distances[node] < float('inf'):
            shortest_paths[node] = path

    return shortest_paths
```

Figure 3: OSPF Algorithm

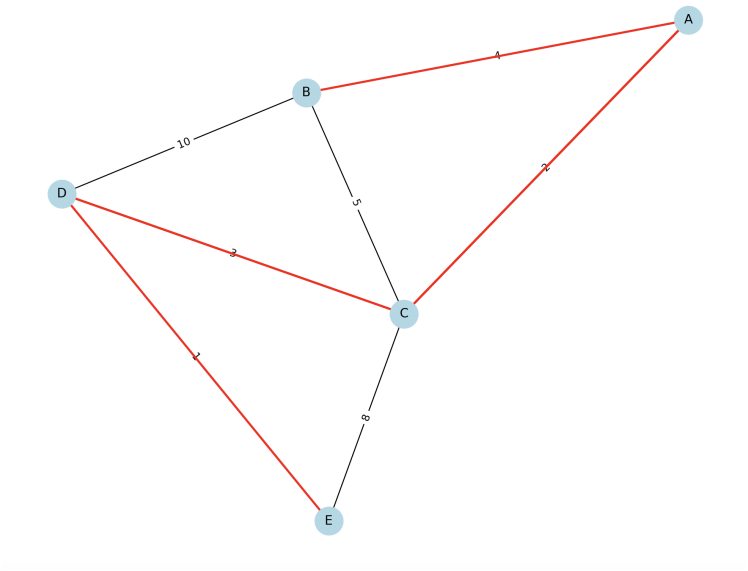

Figure 4: Highlighting shortest paths from Node A

(c) Visualization of the Network

To visualize the network, the matplotlib library was used where the function to draw the network:

- Draws the nodes and edges of the graph.
- Labels the edges with their weights (representing the cost of traversal).
- Optionally highlights the shortest paths

The usage of the matplotlib library really helps in the total understanding as you can visually see the changes happening at every iteration.

```python
# Function to display the network graph with updated weights
def draw_network(G, shortest_paths=None):
    plt.figure(figsize=(10, 8))
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=700, node_color="lightblue")
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    # Highlight the shortest paths
    if shortest_paths:
        for path in shortest_paths.values():
            edges_in_path = list(zip(path, path[1:]))
            nx.draw_networkx_edges(G, pos, edgelist=edges_in_path, edge_color='red', width=2)

    plt.title("OSPF Network Topology")
    plt.show()
```

Figure 5: Function to visualize the network

(d) Simulation of Dynamic Events

Two types of dynamic events were implemented: congestion and link/node failures.

- Congestion

  Congestion is simulated by randomly adjusting the weights of the edges. This mimics real-world scenarios where traffic conditions affect link costs.

- Link/Node Failures

  Failures remove either a node or an edge from the network graph, forcing the protocol to recalculate routes.

```python
# Simulate network congestion by changing weights
def simulate_congestion(G):
    for edge in G.edges:
        change = random.randint(-2, 3)
        G[edge[0]][edge[1]]['weight'] = max(1, G[edge[0]][edge[1]]['weight'] + change)
    print("Network weights updated due to simulated congestion.")

# Function to simulate link or node failure
def simulate_failure(G):
    failure_type = input("Simulate (node/link) failure? ").strip().lower()
    if failure_type == "node":
        node = input("Enter the node to fail: ").strip().upper()
        if G.has_node(node):
            G.remove_node(node)
            print(f"Node {node} has been removed.")
        else:
            print("Node does not exist.")
    elif failure_type == "link":
        link = input("Enter the link to fail (e.g., A-B): ").strip().upper()
        nodes = link.split('-')
        if len(nodes) == 2 and G.has_edge(nodes[0], nodes[1]):
            G.remove_edge(nodes[0], nodes[1])
            print(f"Link {link} has been removed.")
        else:
            print("Link does not exist.")
    else:
        print("Invalid option.")
```

Figure 6: Functions for simulating congestion and link/node failures

(e) Link-State Advertisement (LSA) Simulation

Each node generates an LSA, which is a list of its neighbors and the associated link attributes (weights, bandwidth, etc.). This mimics OSPF's mechanism of sharing local topology information.

```python
# Generate LSA for a node
def generate_lsa(G, node):
    if not G.has_node(node):
        print(f"Node {node} does not exist.")
        return {}

    lsa = {}
    for neighbor in G.neighbors(node):
        lsa[neighbor] = G[node][neighbor]
    print(f"LSA for Node {node}: {lsa}")
    return lsa
```

Figure 7: Link State Advertisement simulation function

2. Methodology

   (a) Build a static network topology with weighted edges.

   (b) Implement OSPF concepts:

   - Generate LSAs for nodes.
   - Compute routing tables dynamically.

   (c) Simulate real-world events:

   - Change edge weights for congestion.
   - Remove nodes/edges for failures.

   (d) Visualize routing paths before and after events.

5 Testing and Results

1. Testing Strategy

   - Scenario 1: Initial routing without any dynamic events.

- Scenario 2: Routing adjustments due to congestion.
- Scenario 3: Route recalculations after link or node failure.

2. Results

   (a) Scenario 1: Initial routing

   - Path from A to E: A -> C -> D -> E.
   - Cost: 6.

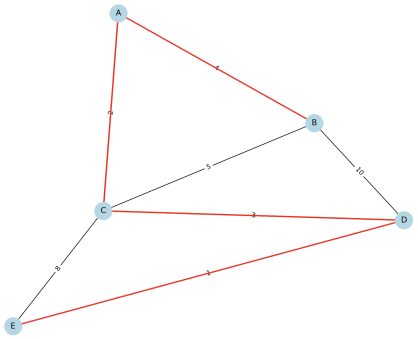

Figure 8: Cost of path from A to E

   (b) Scenario 2: Congestion

   - Path changes dynamically based on updated weights..
   - Example: Increased weight on A -> B redirects traffic via another route.

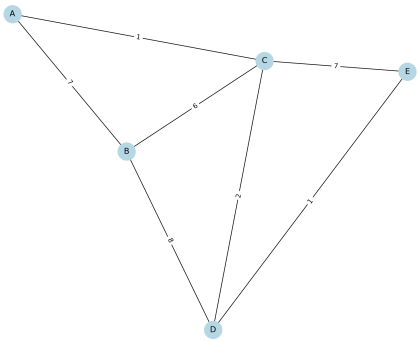

Figure 9: Increased weight on A->B due o congestion

(c) Scenario 3: Link failure

- Failure of link C -> D results in recalculated paths avoiding the failed link.

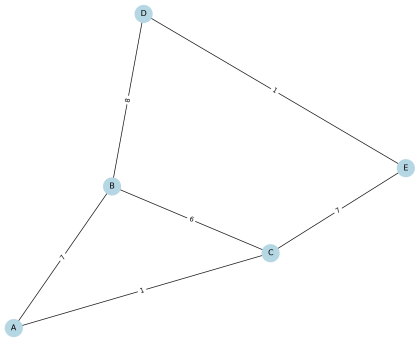

Figure 10: Graph after link failure of C->D

3. Analysis

- Congestion leads to rerouted paths, ensuring minimal disruption.
- Failures demonstrate OSPF's capability to maintain network connectivity.

6 Discussion

1. Challenges Faced

- Debugging dynamic graph updates during failure events.
- Ensuring the QoS constraint logic correctly filters unsuitable paths.
- Visualizing complex scenarios in an easy-to-understand manner.
- Dynamic graph visualization posed performance challenges with larger networks.

2. Limitations

- Limited to small networks for better visualization.
- Does not include advanced OSPF features like multi-area routing or load balancing.
- Simplified congestion model with random weight adjustments.

7 Future Enhancement

These are the proposed improvements which can be done further:

1. Hierarchical OSPF Simulation : Introduce multi-area routing.
2. Traffic Engineering : Incorporate latency, jitter and packet loss as routing metrics.
3. Real-Time Simulation : Use live data to simulate dynamic changes in a real network environment.
4. Machine Learning : redict congestion patterns and preemptively reroute traffic.

8 References

- K. Narasimha, P. Bramarambika, V. Sai Santosh, M. Vamshidhar Reddy, Sandeep Kumar, "Network Design and Implementation of Dynamic Routing Protocols Using Packet Tracer", 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)
- Atul Duvedi, Azhar Ashraf, Shikha Uniyal Gairola, "A Comparitive Study on Routing Protocols: RIP, OSPF and EIGRP", 2022 International Conference on Cyber Resilience (ICCR)