COMPUTER NETWORKS

EE334

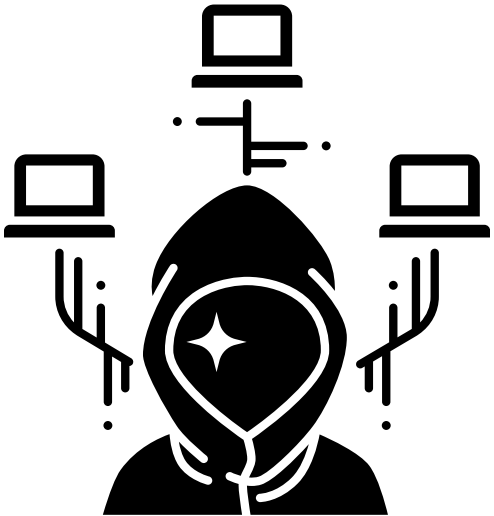
ROUTING PROTOCOL COMPARISON AND ANALYSIS

April 20, 2024

Shivam Vishwakarma
210102080
v.shivam@iitg.ac.in

This project deals with
1. Implementation and comparision of performances of different routing protocols like RIP and OSPF. Testbeds has been set up to simulate the environment to evaluate the behavior and performance of these routing protocols.

2. Analyze the strengths, weaknesses, and trade-offs of each routing protocol in terms of convergence time, scalability, load balancing, and resilience to network changes. This will help us understand the suitability of these protocols for different network scenarios.



Shivam Vishwakarma

v.shivam@iitg.ac.in

Dear Arun Sir,

I am excited to present my project on "**Routing Protocol Comparison and Analysis**" for your Computer Networks course. This project is directly aligned with the key topics covered in the course curriculum, and it will allow me to demonstrate my understanding of the fundamental concepts in network routing and protocol design.

Specifically, the project addresses the following course topics:

1. Overview of routing algorithms: The project involves implementing and evaluating the performance of routing protocols like RIP, OSPF which directly relates to the Dijkstra and Bellman-Ford algorithms covered in the course.

2. Routing protocols and operation: By implementing these routing protocols, I will gain a deeper understanding of how they function, routing algorithms, and convergence mechanisms, as discussed in the course.

3. Routing protocol analysis and comparison: The project's emphasis on analyzing the strengths, weaknesses, and trade-offs of the different routing protocols relates to the course content on evaluating the suitability of routing algorithms for various network scenarios.

4. Network simulation and testbed: The development of a simulation or testbed environment to assess the routing protocols' performance directly supports the learning objectives around network architecture, protocol design, and performance evaluation.

I believe this project is a perfect fit for the course and will contribute to my overall learning and development. I welcome the opportunity to discuss this project further with you and receive your valuable feedback.

Thank you for your consideration.

Sincerely,
Shivam

1 Introduction

This project aims to analyze and evaluate the performance of two widely adopted routing protocols, namely the Routing Information Protocol (RIP) and the Open Shortest Path First (OSPF) protocol, through simulation-based techniques. By implementing simplified versions of these protocols, the project investigates their core routing algorithms, specifically the Bellman-Ford algorithm for RIP and Dijkstra's algorithm for OSPF. The evaluation focuses on key performance metrics such as convergence time, scalability, load balancing capabilities, and resilience to network changes. The insights gained from this analysis are intended to inform the selection and deployment of the most appropriate routing protocol based on the specific requirements and characteristics of a given network infrastructure.

2 Protocol Implementations

The implementation of various routing protocols, namely RIP, and OSPF, were coded in Python programming language.

2.1 Routing Information Protocol (RIP)

The Routing Information Protocol (RIP) is a distance-vector routing protocol that employs the hop count as the metric for path selection. It is designed for small, relatively homogeneous networks. RIP routers maintain routing tables that store route information about the shortest path to each destination network. These tables are periodically broadcasted to and updated by directly connected neighbors in order to maintain accurate routing information.

2.2 Open Shortest Path First (OSPF)

The Open Shortest Path First (OSPF) protocol is a link-state routing protocol that constructs a topology map of the entire network. Using this map, each OSPF router calculates a loop-free shortest path tree with itself as the root, using Dijkstra's algorithm. OSPF routers flood link-state advertisements (LSAs) to all other routers in the same area to advertise information about attached interfaces and metrics. It is well-suited for large, complex networks with redundant connections.

3 Parameters of Interest

There are lot of parameters that affect the performance of routing protocols but for this project to evaluate the protocols some of them have been chosen namely convergence time, scalability, load balancing, and resilience to network changes.

3.1 Convergence time

Convergence time refers to how quickly a routing protocol can detect topology changes, calculate new optimal paths, and propagate updated routing information across the network. Rapid convergence is critical, as slow convergence can lead to traffic being forwarded over failed paths, resulting in packet loss, retransmissions,

and application disruptions. Protocols like OSPF employ sophisticated mechanisms like reliable flooding, designated routers, and topology databases to achieve fast convergence times, typically within seconds for common network events.

3.1.1 How it's estimated ?

By measuring the time it takes for each routing algorithm to compute the path between the randomly selected nodes, this code estimates the convergence time for each protocol in the given network topology represented by the graph G.

3.2 Scalability

As networks grow larger with more routers, links, prefixes, and administrative boundaries, routing protocols must scale efficiently. Scalability concerns include memory and CPU overhead for storing and processing routing information, update traffic volume and overhead, computation complexity of routing algorithms, and database synchronization across many devices. Link-state protocols like OSPF leverage hierarchical topologies and route summarization to enhance scalability compared to distance-vector approaches.

3.2.1 How it's estimated ?

The scalability of RIP, and OSPF routing protocols is evaluated by measuring their convergence times across increasingly larger Erdős–Rényi random graph topologies with 10, 50, 100, 500, and 10000 nodes. *Note: The Erdős–Rényi (ER) random graph model generates graphs with a given number of nodes and a specified probability of edge existence between any node pair. In the code, it creates connected ER graphs of sizes 10, 50, 100, 500, and 10000 nodes with random edge weights from 1 to 10. Using ER graphs provides controllable, random topologies across scales, enabling evaluation of routing protocol convergence times as network size increases. While not perfectly representing real networks, ER graphs offer a reasonable approach to stress-test protocol scalability through analysis of convergence times over incrementally larger random topologies.*

3.3 Load Balancing

Many routing protocols support load balancing traffic across multiple equal-cost paths to a destination, improving throughput and resilience.Load balancing can be based on packet flows, source/destination addresses, or other fields. It optimizes bandwidth utilization across redundant links and balances traffic load across resources.

3.3.1 How it's estimated ?

The core idea behind estimating the load balancing capabilities of routing protocols in the project is to simulate multiple traffic flows between randomly selected source-destination pairs, compute the paths using each protocol's routing algorithm, and analyze the distribution of traffic load across network edges. The traffic load on each edge is tracked for each protocol, and the variance of the load distribution is calculated. A lower variance indicates a more even distribution of traffic across edges, signifying better load balancing capabilities. The protocol with the minimum variance in load distribution is identified as having the best load balancing performance.

3.4 Resilience to Network Changes

Networks are dynamic, with links failing, routers crashing, policies changing, or even intentional attacks occurring. Resilient routing protocols swiftly reconverge and reroute traffic over alternate valid paths with minimal disruption when the network's topology changes. Fast reconvergence, efficient update propagation, graceful restart, non-stop forwarding, and diversity of redundant paths are key mechanisms for resilience.

3.4.1 How it's estimated ?

The resilience analysis of RIP, and OSPF routing protocols is conducted by simulating link failures in a network graph and observing the subsequent path recalculations. We iteratively removes a random link, computes new paths, and records the number of path changes for each protocol. After multiple iterations, it calculates the statistical mean and variance of these changes, providing a quantitative measure of each protocol's stability and reliability in response to network disruptions.

4 Results and Inference

Now we proceed with the result and analysis of our sumulation and sample plots.

4.1 Convergence time

- For the sample graph the average time of convergence of OSPF is less than the RIP convergence time.

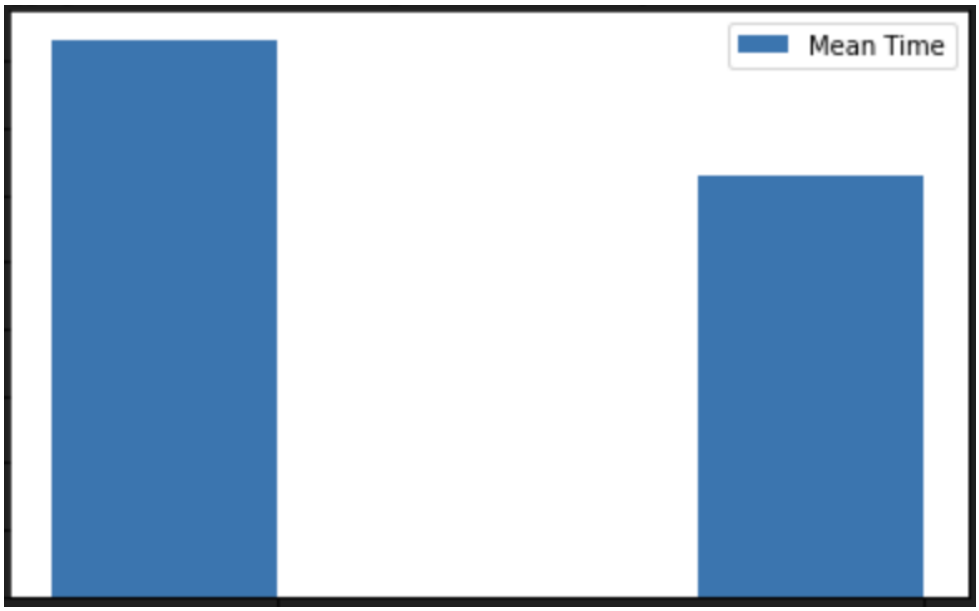


Figure 1: Convergence Time (RIP, OSPF respectively)

4.2 Scalability

- RIP exhibited increasing convergence times with the growth of the network size. This suggests that this may not scale well in large networks due to its distance-vector routing mechanism, which can lead to longer path computation times as the number of nodes increases.
- OSPF shown moderate scalability with a less steep increase in convergence times compared to RIP. It's link-state routing and hierarchical structure may contribute to better scalability in larger networks.

4.3 Load Balancing

- The load balancing efficacy is computed by the variance in traffic distribution as a load balancing metric. The variances are plotted against traffic loads to visually contrast the protocols' load distribution capabilities, with lower variances indicating superior load balancing performance.

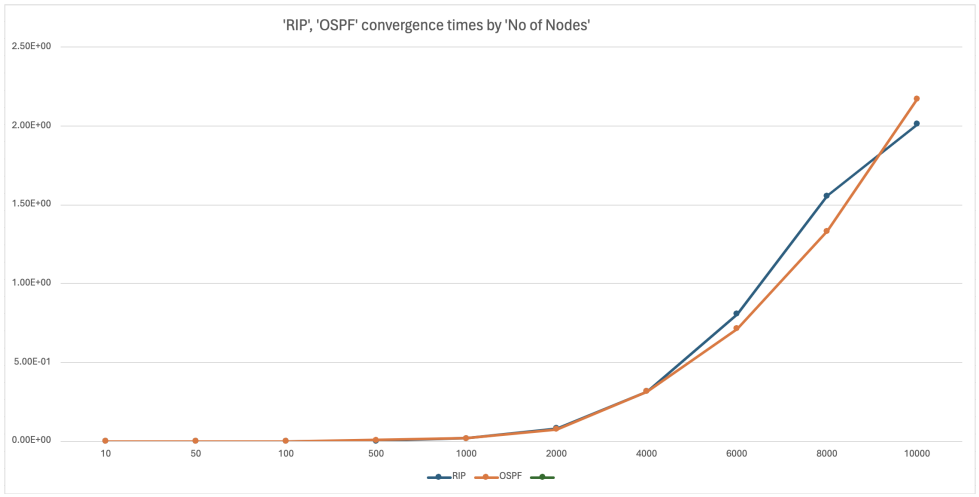


Figure 2: Scalability

- We observed that Load balancing capability of OSPF was less than RIP.Which can also be interpreted by the plot.

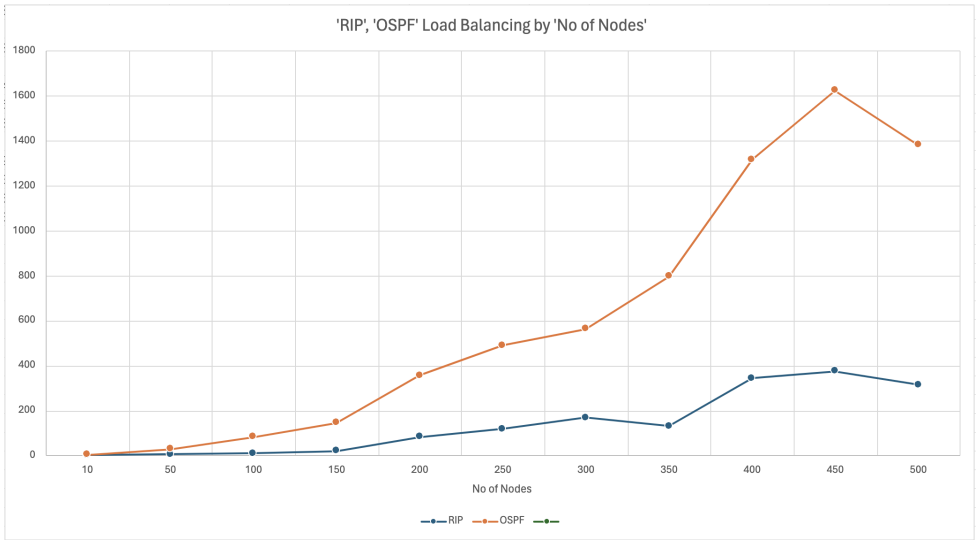


Figure 3: Load Balancing

4.4 Resilience to Network Changes

The resilience of each protocol is inferred from the mean and variance values:

- A lower mean suggests that the protocol is generally more resilient to network changes, as it requires fewer path recalculations.
- A lower variance indicates that the protocol's performance is more consistent across different network change scenarios.
- RIP is most resillient to networks changes.Then it comes OSPF.

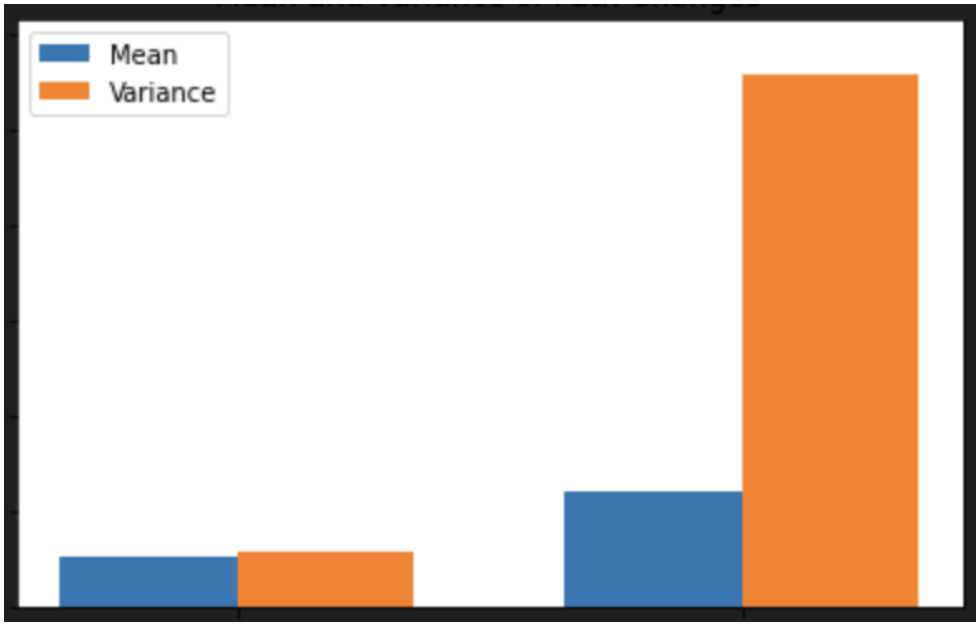


Figure 4: Resilience (RIP, OSPF respectively)

5 Conclusion

The simulation conducted provided insightful data on the performance of RIP and OSPF in a sample network environment. And after this section codes will be attached.

5.1 Convergence Time and Scalability

RIP, while simplest to implement, exhibited the slowest convergence times and poor scalability, making it less suitable for modern, expansive network infrastructures. **OSPF** showed faster convergence times and better scalability as network size increased, highlighting its suitability for larger networks.

5.2 Load Balancing and Resilience

In terms of load balancing, OSPF outperformed RIP, suggesting its ability to distribute traffic more evenly across redundant paths. When assessing resilience to network changes, RIP surprisingly showed a higher degree of resilience, suggesting its potential utility in smaller, more stable environments. In conclusion, the choice of routing protocol should be contingent upon the specific requirements of the network. For smaller networks with stable topologies, RIP could suffice. However, for larger, more dynamic networks, OSPF offers a balance between speed and efficiency, making it a more suitable choice. This simulation underscores the importance of selecting the appropriate protocol based on network size, complexity, and expected change dynamics.

computernetwork-project-copy

April 24, 2024

[1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import time
import random
import heapq
import numpy as np
```

1 Define the network topology

[2]:
```
# Define the network topology
G = nx.Graph()
G.add_nodes_from(['A', 'B', 'C', 'D', 'E', 'F', 'G'])
G.add_edges_from([('A', 'B', {'weight': 5}), ('A', 'C', {'weight': 1}), ('B',␣
↪'D', {'weight': 2}),
                  ('B', 'E', {'weight': 3}), ('C', 'E', {'weight': 4}), ('C',␣
↪'F', {'weight': 6}),
                  ('D', 'E', {'weight': 1}), ('E', 'F', {'weight': 2}), ('E',␣
↪'G', {'weight': 7}),
                  ('F', 'G', {'weight': 1})])
```

2 Implement RIP

[3]:
```
def rip(G, source, target):
    distances = {node: float('inf') for node in G.nodes()}
    distances[source] = 0
    predecessors = {node: None for node in G.nodes()}

    while True:
        updated = False
        for u, v, data in G.edges(data=True):
            if distances[u] + data['weight'] < distances[v]:
                distances[v] = distances[u] + data['weight']
                predecessors[v] = u
                updated = True
        if not updated:
```

```
            break

    path = []
    node = target
    while node is not None:
        path.append(node)
        node = predecessors[node]
    path.reverse()
    return path
```

3 Implement OSPF

[4]:
```
def ospf(G, source, target):
    distances = {node: float('inf') for node in G.nodes()}
    distances[source] = 0
    predecessors = {node: None for node in G.nodes()}
    pq = [(0, source)]

    while pq:
        d, u = heapq.heappop(pq)
        if d > distances[u]:
            continue
        for v, data in G[u].items():
            dist = distances[u] + data['weight']
            if dist < distances[v]:
                distances[v] = dist
                predecessors[v] = u
                heapq.heappush(pq, (dist, v))

    path = []
    node = target
    while node is not None:
        path.append(node)
        node = predecessors[node]
    path.reverse()
    return path
```

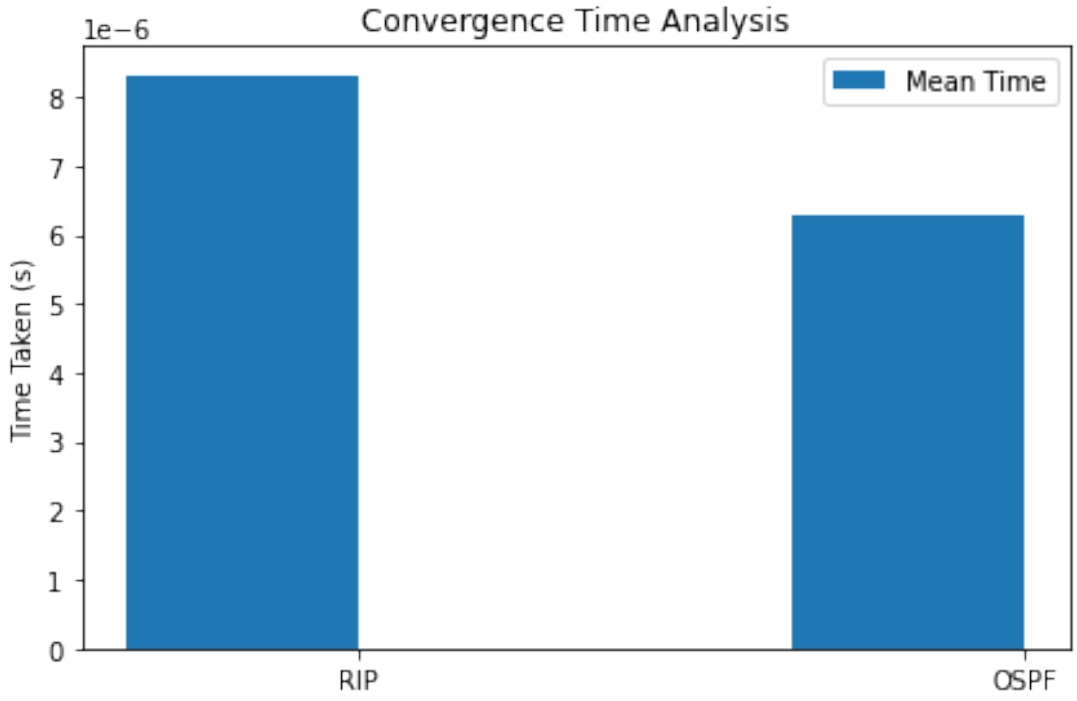
4 Test bench for convergence time

[19]:
```
protocols = ['RIP', 'OSPF']
times = {protocol: [] for protocol in protocols}

def test_convergence_time():
    print("Testing convergence time...")
    for _ in range(10):  # Run the routing protocol 10 times
```

```
        source, target = random.sample(list(G.nodes()), 2)
        start_time = time.time()
        rip_path = rip(G, source, target)
        rip_time = time.time() - start_time
        start_time = time.time()
        ospf_path = ospf(G, source, target)
        ospf_time = time.time() - start_time
        start_time = time.time()
        times['RIP'].append(rip_time)
        times['OSPF'].append(ospf_time)

    # Calculate average times
    avg_times = {protocol: np.mean(times[protocol]) for protocol in protocols}

    # Print average times
    for protocol in protocols:
        print(f"Average {protocol} convergence time: {avg_times[protocol]:.6f}␣
↪seconds")

    # Plotting
    x = np.arange(len(protocols))  # the label locations
    width = 0.35  # the width of the bars

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, [avg_times[protocol] for protocol in␣
↪protocols], width, label='Mean Time')

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Time Taken (s)')
    ax.set_title('Convergence Time Analysis')
    ax.set_xticks(x)
    ax.set_xticklabels(protocols)
    ax.legend()

    fig.tight_layout()

    plt.show()
```

[20]:
```
test_convergence_time()
```

```
Testing convergence time…
Average RIP convergence time: 0.000008 seconds
Average OSPF convergence time: 0.000006 seconds
```



5 Test bench for scalability

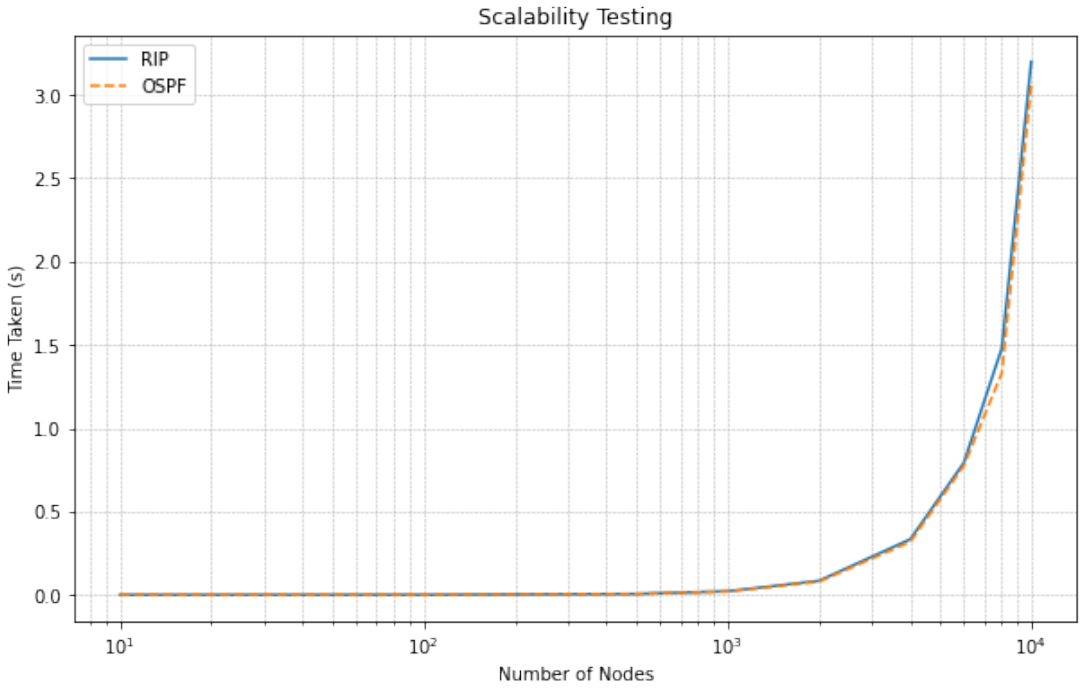
[7]:
```python
# Initialize lists to store data for plotting
protocols = ['RIP', 'OSPF']
nodes = [10, 50, 100, 500, 1000, 2000, 4000, 6000, 8000, 10000]
times = {protocol: [] for protocol in protocols}

def test_scalability():
    print("Testing scalability...")
    for n in nodes:  # Use the 'nodes' list here
        G = nx.erdos_renyi_graph(n, 0.1)
        while not nx.is_connected(G):
            G = nx.erdos_renyi_graph(n, 0.1)
        # Add 'weight' attribute to the edges
        for u, v, d in G.edges(data=True):
            d['weight'] = random.randint(1, 10)
        source, target = random.sample(list(G.nodes()), 2)
        start_time = time.time()
        rip_path = rip(G, source, target)
        rip_time = time.time() - start_time
        times['RIP'].append(rip_time)
        start_time = time.time()
```

```
        ospf_path = ospf(G, source, target)
        ospf_time = time.time() - start_time
        times['OSPF'].append(ospf_time)
        start_time = time.time()
        print(f"For a network of {n} nodes:")
        print(f"RIP convergence time: {rip_time:.6f} seconds")
        print(f"OSPF convergence time: {ospf_time:.6f} seconds")

    # Plotting should be outside the loop
    plt.figure(figsize=(10, 6))
    line_styles = ['-', '--']
    for i, protocol in enumerate(protocols):
        plt.plot(nodes, times[protocol], line_styles[i], label=protocol)

    plt.xlabel('Number of Nodes')
    plt.ylabel('Time Taken (s)')
    plt.title('Scalability Testing')
    plt.legend()
    plt.grid(True, which="both", ls="--", linewidth=0.5)
    plt.xscale('log')

    plt.show()
```

[8]:
```
test_scalability()
```

```
Testing scalability…
For a network of 10 nodes:
RIP convergence time: 0.000011 seconds
OSPF convergence time: 0.000025 seconds
For a network of 50 nodes:
RIP convergence time: 0.000071 seconds
OSPF convergence time: 0.000096 seconds
For a network of 100 nodes:
RIP convergence time: 0.000227 seconds
OSPF convergence time: 0.000290 seconds
For a network of 500 nodes:
RIP convergence time: 0.005391 seconds
OSPF convergence time: 0.005615 seconds
For a network of 1000 nodes:
RIP convergence time: 0.021621 seconds
OSPF convergence time: 0.019522 seconds
For a network of 2000 nodes:
RIP convergence time: 0.083876 seconds
OSPF convergence time: 0.078499 seconds
For a network of 4000 nodes:
RIP convergence time: 0.333753 seconds
OSPF convergence time: 0.318298 seconds
```

```
For a network of 6000 nodes:
RIP convergence time: 0.792399 seconds
OSPF convergence time: 0.774510 seconds
For a network of 8000 nodes:
RIP convergence time: 1.479752 seconds
OSPF convergence time: 1.330436 seconds
For a network of 10000 nodes:
RIP convergence time: 3.199526 seconds
OSPF convergence time: 3.055354 seconds
```



6 Test bench for load balancing

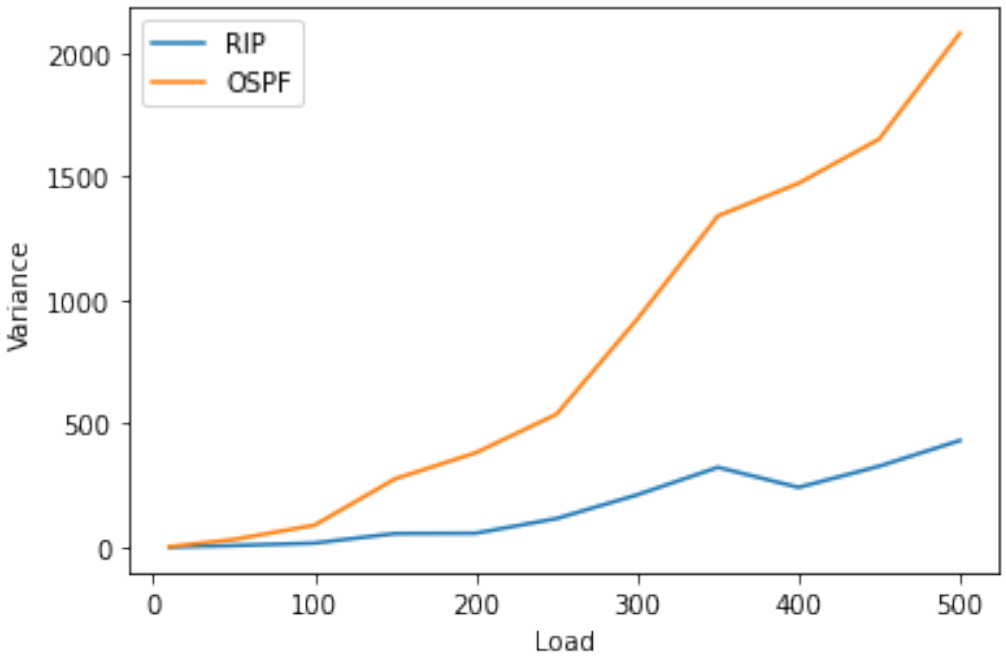
[9]:
```python
import matplotlib.pyplot as plt
import numpy as np
import random

def test_load_balancing():
    load_values = [10, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500]
    rip_variances = []
    ospf_variances = []

    for load in load_values:
        print(f"Testing load balancing for load = {load}...")
```

```
        rip_traffic = {edge: 0 for edge in G.edges()}
        ospf_traffic = {edge: 0 for edge in G.edges()}
        # Add reverse edges
        for u, v in G.edges():
            rip_traffic[(v, u)] = 0
            ospf_traffic[(v, u)] = 0

        for _ in range(load):
            source, target = random.sample(list(G.nodes()), 2)
            rip_path = rip(G, source, target)
            for i in range(len(rip_path) - 1):
                u, v = rip_path[i], rip_path[i+1]
                rip_traffic[(u, v)] += 1
                rip_traffic[(v, u)] += 1

            ospf_path = ospf(G, source, target)
            for i in range(len(ospf_path) - 1):
                u, v = ospf_path[i], ospf_path[i+1]
                ospf_traffic[(u, v)] += 1
                ospf_traffic[(v, u)] += 1


        rip_variances.append(np.var(list(rip_traffic.values())))
        ospf_variances.append(np.var(list(ospf_traffic.values())))

        print("RIP variance:", rip_variances[-1])
        print("OSPF variance:", ospf_variances[-1])

    plt.plot(load_values, rip_variances, label='RIP')
    plt.plot(load_values, ospf_variances, label='OSPF')
    plt.xlabel('Load')
    plt.ylabel('Variance')
    plt.legend()
    plt.show()
```

[10]:
```
test_load_balancing()
```

```
Testing load balancing for load = 10…
RIP variance: 0.45
OSPF variance: 1.4
Testing load balancing for load = 50…
RIP variance: 6.24
OSPF variance: 30.809999999999995
Testing load balancing for load = 100…
RIP variance: 16.439999999999998
OSPF variance: 88.84
Testing load balancing for load = 150…
```

```
RIP variance: 55.040000000000006
OSPF variance: 276.68999999999994
Testing load balancing for load = 200…
RIP variance: 56.29
OSPF variance: 382.2
Testing load balancing for load = 250…
RIP variance: 115.6
OSPF variance: 538.2
Testing load balancing for load = 300…
RIP variance: 211.81
OSPF variance: 922.9599999999998
Testing load balancing for load = 350…
RIP variance: 323.2099999999999
OSPF variance: 1341.29
Testing load balancing for load = 400…
RIP variance: 241.89000000000004
OSPF variance: 1473.8
Testing load balancing for load = 450…
RIP variance: 327.8
OSPF variance: 1652.8
Testing load balancing for load = 500…
RIP variance: 432.00999999999993
OSPF variance: 2081.44
```



7 Test bench for resilience to network changes

[11]:
```
import numpy as np
import matplotlib.pyplot as plt

def test_resilience():
    if len(G.edges()) == 0:
        print("The graph has no edges.")
        return

    rip_changes_list = []
    ospf_changes_list = []

    for _ in range(10):
        rip_paths = {}
        ospf_paths = {}

        for source in G.nodes():
            for target in G.nodes():
                if source != target:
                    rip_paths[(source, target)] = rip(G, source, target)
                    ospf_paths[(source, target)] = ospf(G, source, target)

        failed_link = random.sample(list(G.edges()), 1)[0]
        G.remove_edge(*failed_link)

        rip_changes = 0
        ospf_changes = 0

        for source, target in rip_paths:
            new_rip_path = rip(G, source, target)
            if new_rip_path != rip_paths[(source, target)]:
                rip_changes += 1

        for source, target in ospf_paths:
            new_ospf_path = ospf(G, source, target)
            if new_ospf_path != ospf_paths[(source, target)]:
                ospf_changes += 1


        rip_changes_list.append(rip_changes)
        ospf_changes_list.append(ospf_changes)

    protocols = ['RIP', 'OSPF']
    means = [np.mean(rip_changes_list), np.mean(ospf_changes_list)]
    variances = [np.var(rip_changes_list), np.var(ospf_changes_list)]
```

```
    x = np.arange(len(protocols))
    width = 0.35

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, means, width, label='Mean')
    rects2 = ax.bar(x + width/2, variances, width, label='Variance')

    ax.set_ylabel('Values')
    ax.set_title('Mean and Variance of Path Changes')
    ax.set_xticks(x)
    ax.set_xticklabels(protocols)
    ax.legend()
    # Set the limits of the y-axis
    ax.set_ylim([0, max(max(means), max(variances)) * 1.1])
    fig.tight_layout()
    plt.show()
```

[12]:
```
test_resilience()
```