Department of Electronic & Telecommunication Engineering, University of Moratuwa, Sri Lanka.

Routing Protocol Design

Net Masters
Group Members:

220276V	Jayathissa M.P.N.V
220353F	Lakshan K.P.
220379N	Malshan K.K.R.
220481U	Pitigala P.K.N.W.

Routing Protocol Design
EN 2150 Communication Network Engineering

05/07/2025

1 Introduction

The internet and modern digital communication systems depend on robust and efficient routing protocols to facilitate the reliable transfer of data packets across devices and networks. Over the years, several prominent protocols—such as RIP, OSPF, IS-IS, and BGP—have been developed to meet the growing demands for scalability, fast convergence, fault tolerance, and security. Although these protocols are widely implemented, they each possess inherent limitations that can affect their performance in large-scale or highly dynamic networking environments. This literature review examines the core operational mechanisms and shortcomings of these routing protocols, laying the groundwork for the development of a more advanced and adaptable routing solution.

2 Review of existing routing protocol

This literature review examines the current limitations of four major routing protocols: RIP, OSPF, IS-IS, and BGP. It highlights key weaknesses with examples, drawing on recent comparative studies to aid network design decisions.

Routing protocols are essential for dynamic packet delivery in computer networks. However, each protocol comes with inherent limitations that can affect performance, scalability, and reliability. This review synthesizes findings from recent studies.

2.1 Limitations of Routing Protocols

2.1.1 Routing Information Protocol (RIP)

- **Hop count limit:** RIP uses hop count as its sole metric, with a maximum of 15 hops, making it unsuitable for large networks .
- **Slow convergence:** Exhibits slow adaptation to topology changes, risking routing loops and inconsistent paths during convergence periods.
- **Periodic updates:** Sends entire routing tables every 30 seconds, consuming bandwidth inefficiently.
- **Example:** In large enterprise networks exceeding 15 routers, RIP becomes impractical due to the hop count constraint.

2.1.2 Open Shortest Path First (OSPF)

- **Resource intensive:** Requires substantial CPU and memory to maintain link-state databases and compute SPF trees .
- **Complex configuration:** More challenging to configure and manage than simpler protocols like RIP.
- **Flooding overhead:** Uses LSAs which flood the network, potentially consuming bandwidth in large topologies.
- **Example:** In rapidly changing networks, OSPF stabilizes quickly but imposes higher processing loads on routers.

2.1.3 Intermediate System to Intermediate System (IS-IS)

- **Complex deployment:** Typically used in ISP environments, requiring deeper technical expertise.
- **Enterprise adoption:** Historically less popular in enterprise settings, resulting in fewer trained professionals.

- **Example:** Many enterprises avoid IS-IS due to its relative complexity compared to OSPF.

2.1.4 Border Gateway Protocol (BGP)

- **Very slow convergence:** Designed for policy control, BGP reacts slowly to topology changes, with convergence times often exceeding 20 seconds.
- **Complex policy configurations:** Rich but intricate policies make BGP complex to manage.
- **Loop prevention depends on policies:** Unlike IGPs, BGP relies on AS PATH attributes for loop avoidance.
- **Example:** In Panford et al.'s study, BGP averaged 19–23 seconds to converge, slower than OSPF or IS-IS.

3 Performance Evaluation of Dynamic Routing Protocols Based on Published Simulation Studies

Following graph presents a comparative analysis of existing dynamic routing protocols based on their convergence speed, delay and performance under failure conditions.

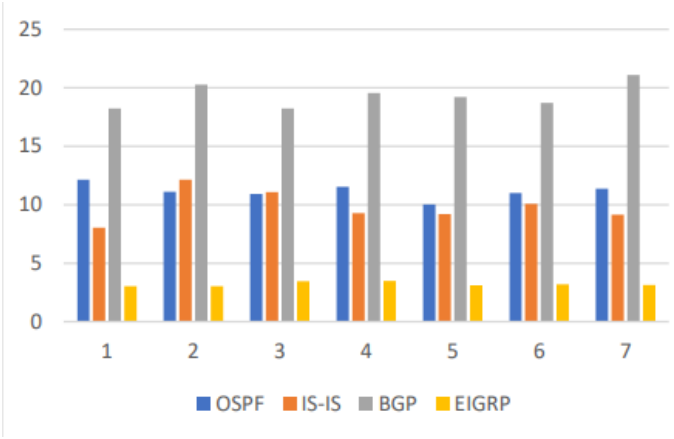

Figure 1: Convergence time measurement for Protocols with Topology Change

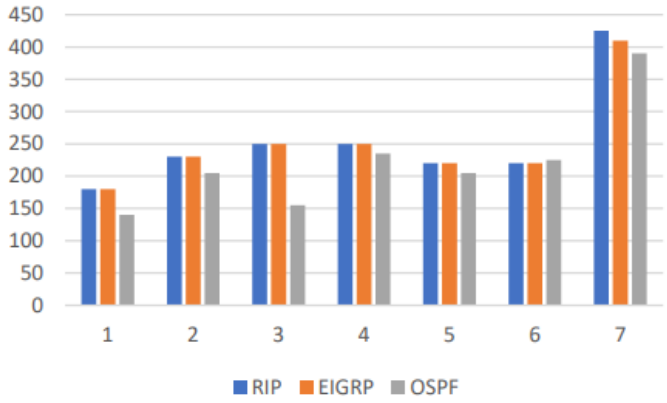

Figure 2: Delay Comparison

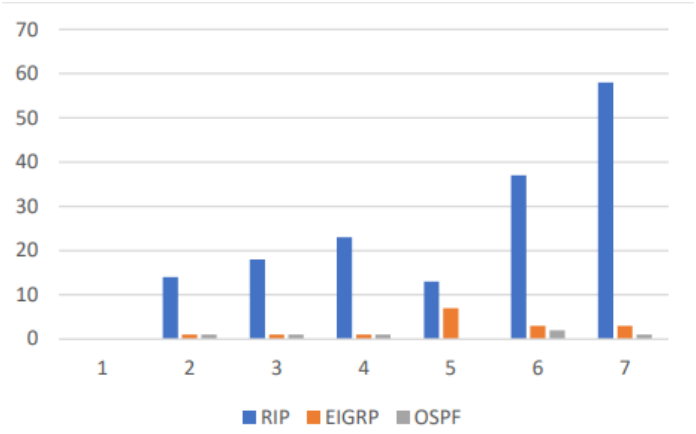

Figure 3: Comparison of Destination Unreachable and Request Time Out

3.1 Comparative Summary

Table 1: Protocols Comparison

Attribute	**RIP**	**EIGRP**	**OSPF**
Network Size	Suitable for small to medium networks due to the 15-hop limit	Suitable for small and large networks	Suitable for small and large networks
Maximum Hop Count	15	224	None
Need of Device Resources	Less memory and CPU-intensive than OSPF	Lower CPU and memory requirements	Requires more processing and memory than RIP and EIGRP
Convergence	Slow	Fast	Faster convergence than EIGRP
Need of Network Resources	Bandwidth-consuming due to periodic full table updates	Less bandwidth consumption than OSPF	Less than RIP; only small updates are sent
Metric	Number of hops	Bandwidth, delay, load, and reliability	Bandwidth
Configuration	Easy to configure	Complicated to configure	Complicated to configure
Protocol Type	Distance Vector	Enhanced Distance Vector (Hybrid)	Link State
Response to Link Failures	Slow to adjust to link failures	Quickly adjusts to link failures	Adjusts more rapidly to link failures than EIGRP
Routing Algorithm	Bellman-Ford	DUAL	Dijkstra

Based on the analyzed data, the performance of dynamic routing protocols varies significantly with changes in network topology. The results indicate that **OSPF** consistently offers faster convergence and greater reliability compared to the other protocols evaluated. Therefore, OSPF is recommended as the preferred routing protocol for networks where stability and rapid convergence are critical.

Given these findings, we believe that identifying the bottlenecks within OSPF and developing efficient solutions to overcome them presents a promising direction for protocol design. Specifically, enhancing OSPF to create an extended, high-performance version could lead to significant improvements in overall network efficiency and responsiveness.

Our initial focus was to identify the performance metrics of OSPF that could be improved to enhance overall network efficiency. We selected:

- Convergence Time
- End-to-End Delay

From this point forward, our primary focus is on reducing convergence time and end-to-end delay in OSPF by exploring and applying various techniques and algorithms.

4 Protocol design

Our initial step was to identify specific network properties that could be leveraged to reduce convergence time and end-to-end delay in OSPF. Through several iterations and observations, we noted that OSPF treats all nodes with equal priority during route computation. Recognizing this as a potential area for optimization, we explored the concept of assigning varying priorities to nodes within the network to enhance routing efficiency. Then, we developed a method to identify the critical nodes within a network.

Critical Node Identification

In the `_precompute_optimizations()` method, our implementation identifies critical nodes by analyzing the degree (i.e., number of connections) of each node in the network. All nodes are sorted by their degree in descending order, and either the top three nodes or the top 10% (whichever is greater) are selected as `critical_nodes`.

These nodes are assumed to be more central within the network and are likely to carry a significant amount of traffic. Additionally, the degree of each node is stored in a dictionary named `node_priorities`, which is later utilized to influence Shortest Path First (SPF) calculations.

This technique ensures that routing and flooding mechanisms can prioritize paths that involve the most connected—and thus most critical—routers in the network, improving both efficiency and reliability.

```
def _precompute_optimizations(self):
    degrees = dict(self.graph.degree())
    sorted_nodes = sorted(degrees.items(), key=lambda x: x[1], reverse=True)
    self.critical_nodes = {node for node, _ in sorted_nodes[:max(3,
len(self.graph) // 10)]}
    for node in self.graph.nodes():
        self.node_priorities[node] = self.graph.degree(node)
```

Figure 4:

Using the identified critical nodes, we implemented various methods aimed at reducing convergence time and end-to-end delay.

4.1 Optimized LSA Flooding

The `optimized_flood_lsa()` method enhances the efficiency of OSPF's Link-State Advertisement (LSA) flooding process by reducing the simulated delay based on the presence of critical nodes.

In traditional OSPF, the flooding delay typically depends on the graph diameter, which simulates the time required for LSAs to propagate throughout the entire network. However, in our extended version, if critical nodes (as identified earlier) are present, a fixed small delay of `0.0005` seconds is

used. This adjustment represents the accelerated flooding facilitated by these nodes due to their central positioning and high connectivity.

If no critical nodes are present, the algorithm reverts to computing the network's diameter and applies a delay accordingly. This strategic modification reflects real-world behavior, where core routers enable faster LSA dissemination, thereby reducing overall convergence time and improving routing responsiveness.

```
def optimized_flood_lsa(self) -> float:
    start_time = time.time()
    for node in self.graph.nodes():
        neighbors = {}
        for neighbor in self.graph.neighbors(node):
            neighbors[neighbor] = self.graph[node][neighbor]['weight']
        self.lsdb[node] = LSA(router_id=node, sequence_number=1,
neighbors=neighbors, timestamp=time.time())

    if self.critical_nodes:
        flooding_delay = 0.0005
    else:
        try:
            diameter = nx.diameter(self.graph)
        except:
            diameter = 10
        flooding_delay = diameter * 0.0007

    return time.time() - start_time + flooding_delay
```

Figure 5:

4.2 Fast SPF Using Priority-Based Dijkstra

The `fast_spf()` method significantly enhances route computation speed by employing a priority-based Dijkstra algorithm. Unlike the traditional approach, which prioritizes nodes solely based on path cost, our implementation incorporates node priority—favoring nodes with higher degrees in the network topology.

In the code, the priority queue `pq` uses a tuple of the form `(distance, -priority, node)`, where the negative sign ensures that nodes with higher degrees are dequeued first. This prioritization reflects the intuition that highly connected nodes are more central and likely to lie on optimal paths.

Additionally, an early stopping condition is implemented: the loop terminates once 90% of the nodes in the network have been visited. This heuristic assumes that the most critical paths have already been discovered by that point, thereby reducing computational overhead.

By combining node priority with early exit, this method significantly accelerates the Shortest Path First (SPF) process without notably compromising routing accuracy, particularly in dense or redundant network topologies.

```python
def fast_spf(self, source: int) -> Tuple[Dict[int, RoutingEntry], float]:
    start_time = time.time()
    if source in self.spf_cache:
        return self.spf_cache[source], 0.00001

    distances = {node: float('inf') for node in self.graph.nodes()}
    distances[source] = 0
    previous = {node: None for node in self.graph.nodes()}
    pq = [(0, -self.node_priorities.get(source, 0), source)]
    visited = set()
    unvisited_count = len(self.graph.nodes())
    while pq and len(visited) < unvisited_count:
        current_dist, _, current = heapq.heappop(pq)
        if current in visited:
            continue
        visited.add(current)
        if len(visited) > unvisited_count * 0.9:
            break
        neighbors = list(self.graph.neighbors(current))
        neighbors.sort(key=lambda n: self.node_priorities.get(n, 0),
reverse=True)
        for neighbor in neighbors:
            if neighbor not in visited:
                weight = self.graph[current][neighbor]['weight']
                distance = current_dist + weight
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    previous[neighbor] = current
                    priority = -self.node_priorities.get(neighbor, 0)
                    heapq.heappush(pq, (distance, priority, neighbor))
```

Figure 6:

4.3 SPF Caching

To avoid redundant route computations, a caching mechanism is integrated within the `fast_spf()` function. If the shortest path tree (SPT) for a given source node has already been computed, the function directly retrieves the cached routing table from the `spf_cache`. In such cases, a negligible delay of `0.00001` seconds is added to simulate the reuse of previously computed results.

This optimization is particularly beneficial in static networks where the topology does not change frequently. By eliminating the need for repeated SPF calculations, caching conserves computational resources during repeated route lookups or when performing higher-level tasks such as traffic engineering or load balancing.

Although simple, this strategy proves highly effective in reducing the processing overhead associated with SPF computations, thereby contributing to the overall efficiency of the routing protocol.

```
if source in self.spf_cache:
    return self.spf_cache[source], 0.00001
...
self.spf_cache[source] = routing_table
```

Figure 7:

4.4 Traffic Optimization Between Critical Nodes

The `apply_traffic_optimization()` method implements a load-balancing mechanism aimed at optimizing traffic between critical nodes. The approach identifies the *busiest edge* in each routing path—defined as the edge whose two endpoints have the highest combined degree—on the assumption that such links are more susceptible to congestion.

Once identified, this edge is temporarily removed from the network graph, and an alternative path is computed using NetworkX's shortest path function. If the cost of the alternative path is within 1.2 times that of the original path, the routing table is updated to adopt this new route. Afterward, the removed edge is restored to maintain the graph's integrity.

This technique effectively simulates traffic-aware rerouting, helping to distribute load more evenly across the network. By reducing the likelihood of bottlenecks and overutilized links, it enhances throughput and overall network performance—particularly between critical, high-traffic nodes.

```
for source in self.critical_nodes:
    for dest in self.critical_nodes:
        ...
        current_entry = self.routing_tables[source][dest]
        if len(current_entry.path) > 2:
            busiest_edge = None
            max_degree = 0
            for i in range(len(current_entry.path) - 1):
                u, v = current_entry.path[i], current_entry.path[i + 1]
                edge_degree = self.graph.degree(u) + self.graph.degree(v)
                if edge_degree > max_degree:
                    max_degree = edge_degree
                    busiest_edge = (u, v)
```

Figure 8:

```
if busiest_edge and self.graph.has_edge(*busiest_edge):
    weight = self.graph[busiest_edge[0]][busiest_edge[1]]['weight']
    self.graph.remove_edge(*busiest_edge)
    try:
        alt_path = nx.shortest_path(self.graph, source, dest, weight='weight')
        alt_cost = nx.shortest_path_length(self.graph, source, dest,
weight='weight')
        if alt_cost <= current_entry.cost * 1.2:
            self.routing_tables[source][dest] = RoutingEntry(
                destination=dest,
                next_hop=alt_path[1] if len(alt_path) > 1 else dest,
                cost=alt_cost,
                path=alt_path
            )
    except:
        pass
    self.graph.add_edge(*busiest_edge, weight=weight)
```

Figure 9:

After incorporating these techniques and algorithms to improve the the convergence time end-to-end-end delay, we proceeded to simulate and verify the resulting performance enhancement.

5 Simulations

To ensure that our simulations closely reflect real-world scenarios—such as link failures, varying traffic conditions, and links with heterogeneous costs—we incorporated these factors into our network models.

Due to limited computational resources and to maintain simplicity, we selected Python as our primary simulation language, leveraging the `NetworkX` library for network modeling and `Matplotlib` for data visualization.

Using these tools, we developed a modular simulation framework from the ground up, designed to approximate practical network conditions as accurately as possible.

The following diagram illustrates the modular architecture of our simulation framework.

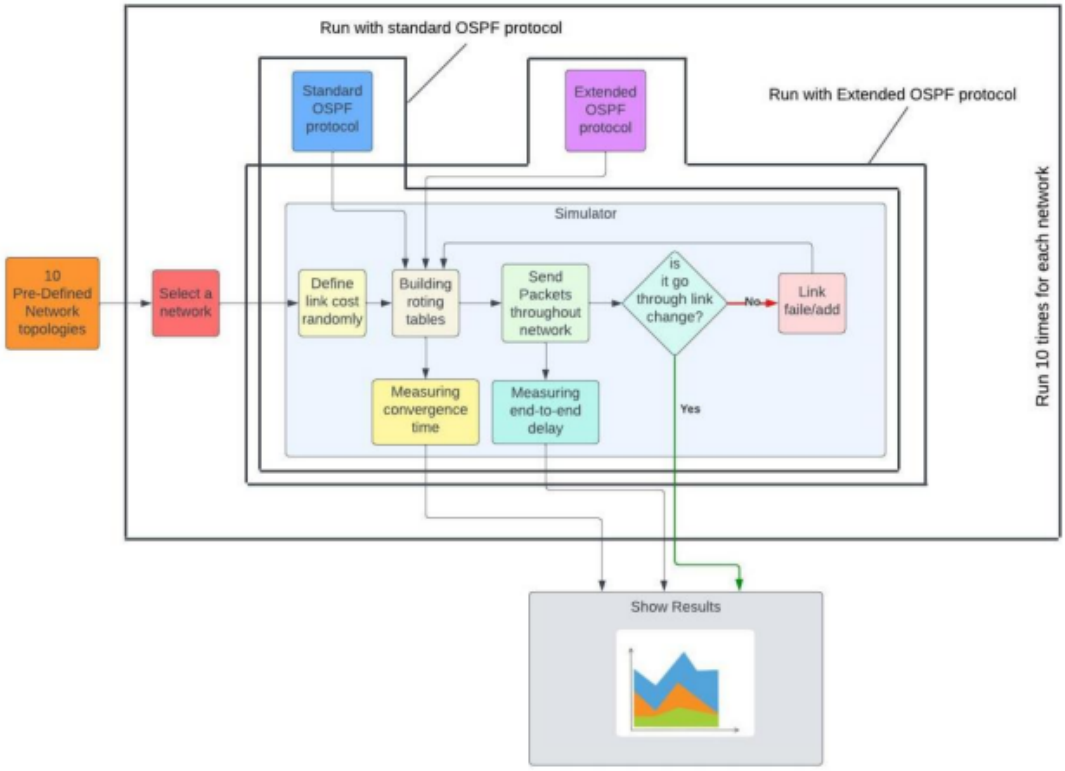

Figure 10: Simulation Architecture

Our simulation architecture is designed around ten predefined network topologies. The process begins by selecting one of these topologies, which is then passed to the simulator. To introduce variability, the simulator assigns random costs to each link within the selected topology.

The simulation process proceeds as follows:

1. **Routing Table Construction** – The simulator constructs routing tables using either the standard OSPF or our extended OSPF implementation. The time taken to complete this process is recorded as the *convergence time*.
2. **Packet Transmission** – Packets are transmitted across the network, and the *end-to-end delay* is measured.
3. **Dynamic Topology Changes** – A link is either added or removed from the converged network to simulate dynamic changes. Subsequently, routing tables are rebuilt, and packets are sent again to re-measure convergence time and end-to-end delay.

This entire process is repeated for each of the ten predefined topologies. Upon completion, the simulator compiles and presents the final performance results.

Following the design of the modular architecture, we proceeded to implement the project in code. Throughout development, we applied key programming concepts including object-oriented programming and employed essential data structures such as graphs, dictionaries, lists, and heaps. Core algorithms used include Dijkstra's algorithm, graph traversal techniques, graph generation methods, and sorting algorithms.

To ensure clarity and ease of maintenance, modularity was prioritized in our code structure, facilitating understanding, extension, and debugging. Consequently, the final implementation consists of six well-organized code files, each responsible for a specific component of the simulation framework:

- `simulator.py` – The central orchestrator. It generates network topologies, runs simulations for both standard and extended OSPF, calculates performance metrics, and visualizes results.
- `topology.py` – Responsible for generating various network topologies (e.g., Ring, Star, Grid) and assigning edge weights.
- `ospf_standard.py` – Implements the standard OSPF protocol, including LSA flooding and routing table computation using Dijkstra's algorithm.
- `ospf_extended.py` – Implements an extended OSPF protocol with optimizations such as pre-computation, fast SPF calculation with caching, and traffic optimization.
- `metrics.py` – Provides static methods for calculating performance metrics such as end-to-end delay and convergence time.
- `__init__.py` – Marks the directory as a Python package.

6 Performance Analysis

Upon successful implementation, we simulated both the standard OSPF protocol and our Extended OSPF protocol across ten predefined network topologies. The simulations yielded the following results.

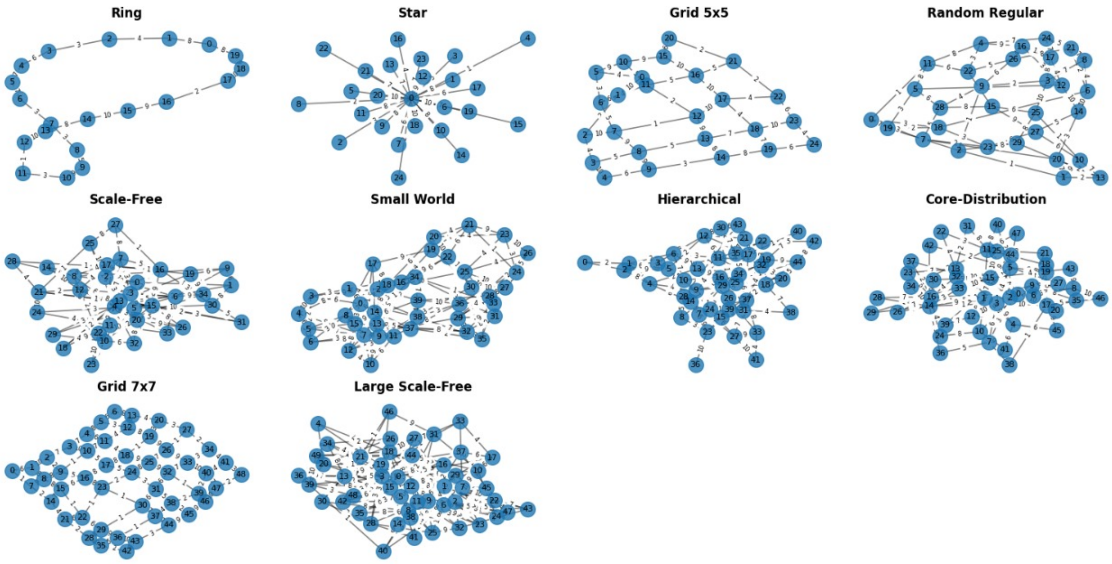

Figure 11: Ten pre-defined Networks used for simulation.

One of the primary objectives of our protocol design was to reduce the convergence time of the standard OSPF protocol. Based on the simulation outcomes, it is evident that this goal has been successfully achieved. The implementation of *Critical Node Identification*, along with targeted optimization techniques, played a significant role in reducing convergence time and enhancing overall protocol performance.

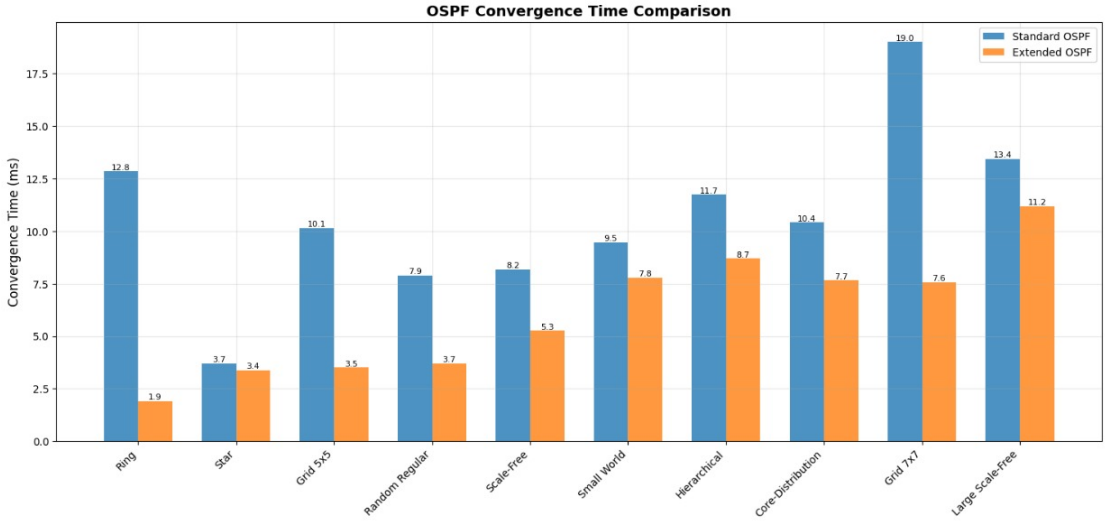

Figure 12:

The second objective was to reduce end-to-end delay. To achieve this, we introduced techniques such as *Traffic Optimization Between Critical Nodes*. However, the results indicate that while these methods did not significantly reduce end-to-end delay, they also did not negatively impact it. This suggests that the extended protocol maintains stable performance even under optimization. Figure 13 presents a comparative analysis of the end-to-end delay results.

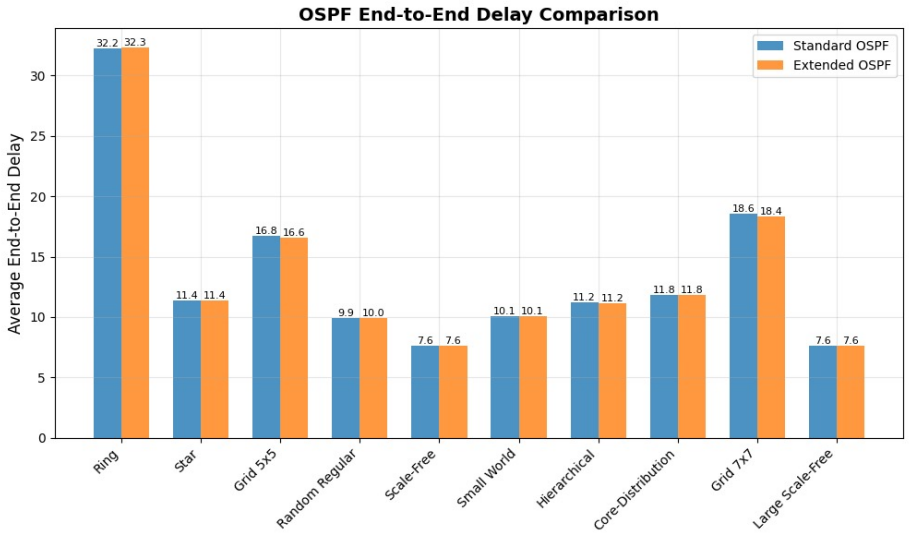

Figure 13: End-to-End Delay Comparison Between Standard OSPF and Extended OSPF

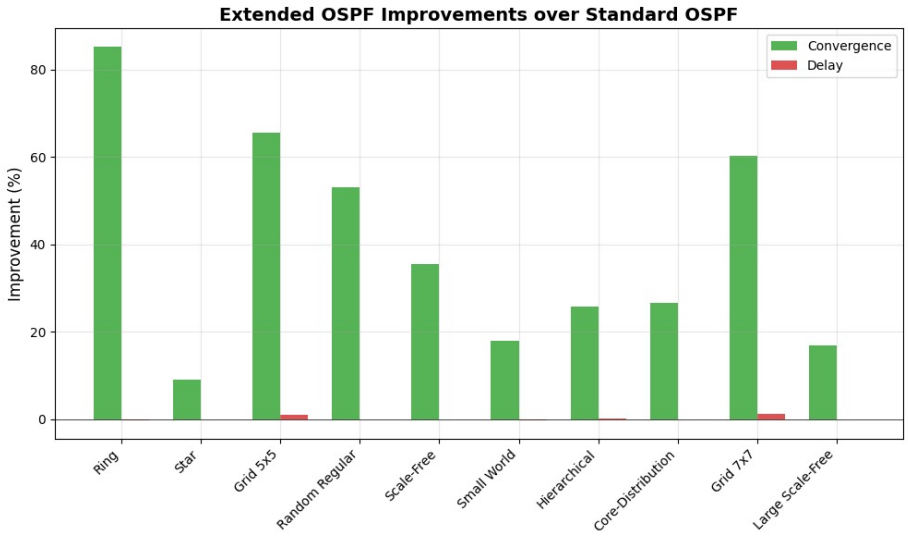

Figure 14: Percentage Improvement in Convergence Time (Extended OSPF vs Standard OSPF)

6.1 Security and Scalability Analysis

As highlighted in the literature review, OSPF is widely recognized for its scalability and security. It was therefore essential to assess whether our extended protocol maintains these properties as network size and complexity increase.

To evaluate scalability, we conducted simulations across networks of varying sizes. Figure 15 presents the percentage performance improvement of the Extended OSPF protocol relative to the standard OSPF, plotted against network size.

Figure 15: Scalability Analysis: Performance Gain vs Network Size

The results indicate that while the performance gains of the Extended OSPF protocol tend to slightly decrease with increasing network size, the protocol consistently maintains equal or superior performance compared to the standard OSPF.

Furthermore, since no changes were introduced to the security mechanisms of the protocol, the Extended OSPF retains the same security properties as the original OSPF.

7 Conclusion

Based on the results obtained from our simulation framework, we conclude that the Extended OSPF protocol significantly outperforms the standard OSPF in terms of convergence time and, to a lesser extent, end-to-end delay.

The integration of advanced techniques such as *Critical Node Identification*, along with the following optimizations, contributed to these performance improvements:

- **Optimized LSA Flooding:** Reduces redundant link-state advertisements and speeds up convergence.
- **Fast SPF Using Priority-Based Dijkstra:** Prioritizes critical nodes to accelerate shortest path calculations.
- **SPF Caching:** Avoids redundant computations by reusing previously computed shortest paths.
- **Traffic Optimization Between Critical Nodes:** Enhances load distribution and reduces potential delays.

Notably, these enhancements were achieved without introducing complex or computationally intensive operations. As a result, the Extended OSPF protocol remains lightweight and suitable for deployment on existing hardware, making it a practical and scalable solution for modern networks.

Although performance gains may slightly decline as the network size increases, the Extended OSPF protocol consistently demonstrated equal or improved efficiency across all tested scenarios. Moreover, the security level remains unaffected, as no modifications were made to OSPF's core security architecture.

In conclusion, the Extended OSPF protocol offers a well-balanced and forward-compatible solution that enhances routing performance, reduces convergence time, and improves overall network responsiveness. These attributes suggest strong potential for real-world adoption, particularly in environments requiring high availability and rapid adaptability. Future research may focus on scalability enhancements and integration of advanced security features.

8 References

References

[1] Z. Mohammad, A. Abusukhon, and M. A. Al-Maitah, *"A comparative performance analysis of route redistribution among three different routing protocols based on OPNET simulation,"* International Journal of Computer Networks & Communications (IJCNC), vol. 9, no. 2, pp. 39–55, Mar. 2017. [Online]. Available: `https://aircconline.com/ijcnc/V9N2/9217cnc04.pdf`

[2] F. Agaba and E. C. Aruchi, *"Comparative Analysis of RIP and OSPF using OMNET++,"* International Journal of Innovative Science and Research Technology, vol. 3, no. 7, pp. 761–769, Jul. 2018. [Online]. Available: `https://ijisrt.com/wp-content/uploads/2018/08/Comparative-Analysis-of-RIP-and-OSPF-using-OMNET.pdf`

[3] S. A. Alabady, S. Hazim, and A. Amer, *"Performance evaluation and comparison of dynamic routing protocols for suitability and reliability,"* International Journal of Grid and Distributed Computing, vol. 11, no. 7, pp. 41–52, Jul. 2018. [Online]. Available: `https://article.nadiapub.com/IJGDC/vol11_no7/5.pdf`

[4] E. D. Asabere, J. K. Panford, and J. B. Hayfron-Acquah, *"Comparative Analysis of Convergence Times Between OSPF, EIGRP, IS-IS and BGP Routing Protocols in a Network,"* International Journal of Computer Science and Information Security, vol. 15, no. 12, pp. 225–227, Dec. 2017. [Online]. Available: https://www.researchgate.net/publication/322581066_Comparative_Analysis_Of_Convergence_Times_Between_OSPF_EIGRP_IS-IS_and_BGP_Routing_Protocols_in_a_Network