CS 514 Advanced Computer Networks
Final Project Report

# OSPF Simulation

Longtian Ye ly186@duke.edu
Zhihan Peng zp59@duke.edu
Zhenyuan Liang zl208@duke.edu
Weihang Guo wg98@duke.edu

## Introduction

OSPF (Open Shortest Path First), a link-state routing protocol, is known for its efficiency in routing data across large and complex networks. Our OSPF Simulation project aims to create a dynamic and interactive simulation of the OSPF routing protocol, which is integral in modern network infrastructures.

In this project, we not only focused on simulating the functional aspects of OSPF, such as path selection and network adaptation to changes, but also attempted to provide a real-time visualization of these processes. This approach provides an intuitive understanding of the protocol's internal workings and its behavior in various scenarios, including routing decisions, mechanics of updating routing tables, etc.

## Implementation Details

### 2.1 Overview features of OSPF

Our OSPF Simulation aims to emphasize several key features of the OSPF protocol, crucial for understanding its capabilities and benefits in network routing. In particular, OSPF's fast convergence ensures minimal disruption during network changes, while its ability to provide accurate and efficient routing decisions is based on the idea that each router has a complete view of the network. Loop-free paths are ensured through our implementations involving the shortest-path-first algorithm, keeping track of visited nodes, and maintaining a consistent network view.
Furthermore, our similation aim to mimic OSPF's idea of areas and partitioning. Specifically, we specify a range threshold (e.g. '> 100' in coordinate systems) to emulate the idea of OSPF into separate areas.

### 2.2. Principle Class Components

- Graph Class: Forms the backbone of our simulation, representing the network topology as a graph. It's responsible for adding and removing nodes and edges, reflecting OSPF's dynamic nature.
- Router Class: Each instance mimics an OSPF router with unique identifiers and adjacent routers, managing the Local State Database (LSDB) and processing Link State Advertisements (LSAs).
- Net Class: Acts as the central command, integrating the components and executing user commands, influencing router behaviors and network topology.

### 2.3 Exchanging LSAs

The process of exchanging LSAs is central to OSPF and is handled by our 'Graph' class.

- Adding Vertices (*add_vertex* method): The network's expansion is simulated by adding new routers as vertices in the LSDB, which is a dictionary that keeps track of each router's connections.

- Adding Edges (*add_edge* method): This method is used to simulate the establishment of links between routers, updating the LSDB with pre-defined connection costs to reflect the network's physical structure.
- Flooding LSAs (*flood_lsa* method): This is the crucial aspect to implement for OSPF, which involves the flooding of LSAs. Upon adding or updating a link, this method is automatically invoked to update the LSDBs of all relevant routers. We attempted to replicate this section to ensure all routers have the latest and consistent network topology information.

### 2.4 Path Calculation and Shortest Path Algorithm

The *find_shortest_path* method within the 'Graph' class is pivotal for OSPF's shortest path calculation. We followed the use of Dijkstra's Algorithm to iteratively update the shortest distance from source to each router and trace the optimal path.
After calculating the shortest paths, the method reconstructs the most efficient route from the source to the destination, utilizing the stored information on the shortest paths. Through this implementation, we dynamically traces and reconstructs the most efficient path, which ensures the adaptability of OSPF to network changes.

### 2.5 Updating Routing Tables

In our OSPF simulation, the process of updating routing tables is integral to maintaining accurate and efficient routing decisions. This involves each router updating its Local State Database (LSDB) upon receiving new LSAs, followed by recalculating the routing tables using Dijkstra's algorithms to adapt to network topology changes. Routers continually monitor for changes and synchronize with adjacent routers to maintain a consistent and unified network view. This dynamic updating mechanism ensures that routing paths remain optimal and reflect the latest state of the network.

## Simulation and Visualization

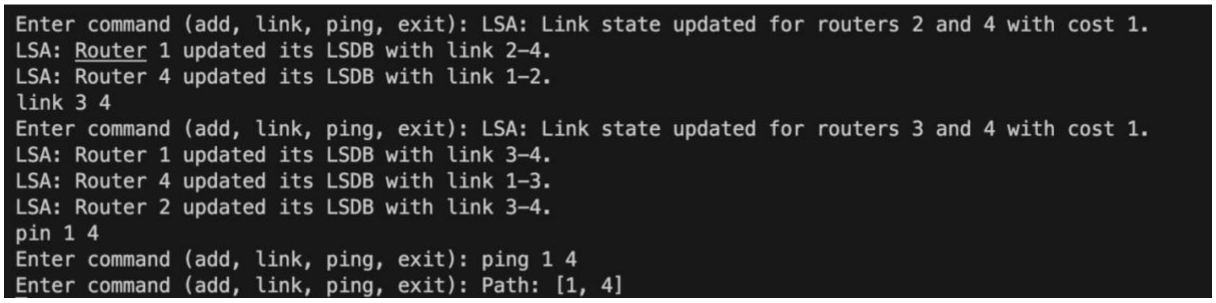
Our OSPF Simulation project leverages Pygame, a popular Python library, to create an interactive and visually appealing demonstration of the OSPF network. Pygame's graphical capabilities enable users an intuitive understanding of OSPF's operational dynamics through real-time visualization.

This visual representation is not static; it dynamically updates to reflect changes in the network topology caused by user commands or OSPF processes. By incorporating various colors and symbols, we visually differentiate between different states of routers and links (e.g., active, inactive), making the simulation both informative and engaging.

Particularly, users can interact directly with the network using a set of defined commands. These commands, processed through the Pygame interface, allow users to:

- Add Nodes: Users can expand the network by adding new routers, simulating network growth and OSPF's response to it.

- Remove Nodes or Links: This feature enables users to simulate network shrinkage or link failures, observing how OSPF adapts to such changes. We strategically included print statements to show real-time updates in the router's LSDB, so that users would be able to track the changes and understand OSPF's topology dynamically.
- Visualize Routing Paths: The simulation provides a visual representation of routing paths within the current OSPF network. Users can initiate a 'ping' command between two routers and our simulation would highlight the chosen routing path in red, distinguishing it from the other paths with default black color.



```
Enter command (add, link, ping, exit): LSA: Link state updated for routers 2 and 4 with cost 1.
LSA: Router 1 updated its LSDB with link 2-4.
LSA: Router 4 updated its LSDB with link 1-2.
link 3 4
Enter command (add, link, ping, exit): LSA: Link state updated for routers 3 and 4 with cost 1.
LSA: Router 1 updated its LSDB with link 3-4.
LSA: Router 4 updated its LSDB with link 1-3.
LSA: Router 2 updated its LSDB with link 3-4.
pin 1 4
Enter command (add, link, ping, exit): ping 1 4
Enter command (add, link, ping, exit): Path: [1, 4]
```

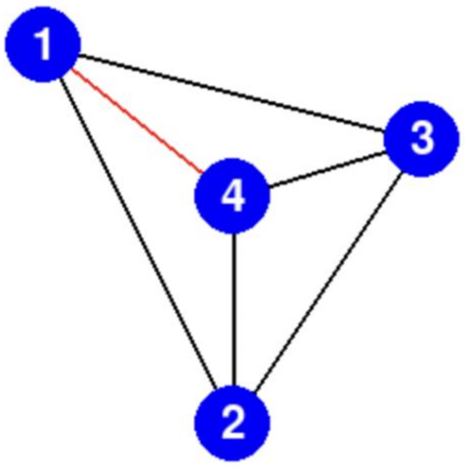


*Figure 1*

In Figure 1, when pinging from 1 to 4, the OSPF chooses the shortest path within the network, which is directly go from router 1 to router 4.

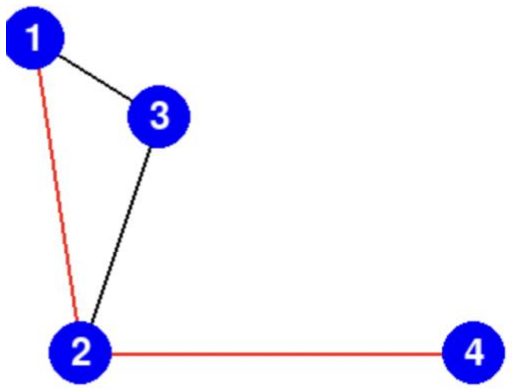
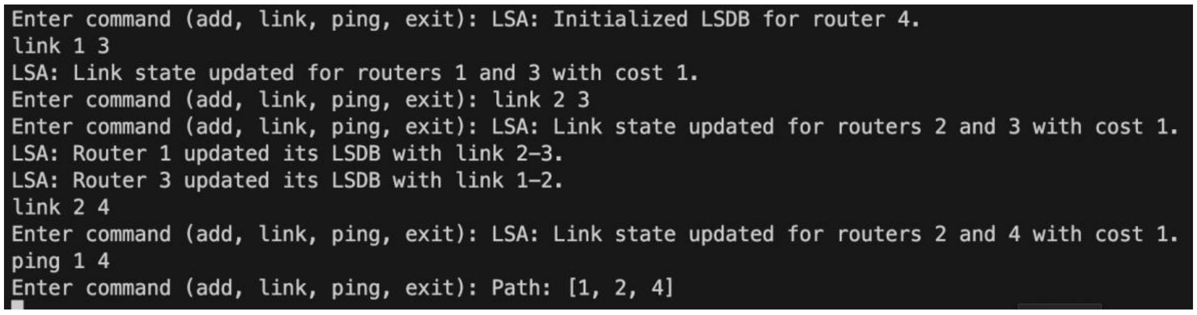


*Figure 2*



```
Enter command (add, link, ping, exit): LSA: Initialized LSDB for router 4.
link 1 3
LSA: Link state updated for routers 1 and 3 with cost 1.
Enter command (add, link, ping, exit): link 2 3
Enter command (add, link, ping, exit): LSA: Link state updated for routers 2 and 3 with cost 1.
LSA: Router 1 updated its LSDB with link 2-3.
LSA: Router 3 updated its LSDB with link 1-2.
link 2 4
Enter command (add, link, ping, exit): LSA: Link state updated for routers 2 and 4 with cost 1.
ping 1 4
Enter command (add, link, ping, exit): Path: [1, 2, 4]
```

In Figure 2, when pinging from 1 to 4, the OSPF again chooses the shorter paths, which is from 1 to 2, then to 4.

## Challenges and Solutions

In the development of our OSPF Simulation, we encountered several challenges that required revising our solutions consistently to overcome. A notable challenge was in the implementation of flooding LSAs, which initially led to unintended direct links with non-immediate neighbors and potential infinite loops. To address these, we implemented several key solutions: introducing neighbor validation to ensure LSAs are only sent to direct neighbors, creating a 'visited' set to prevent infinite loops by tracking routers that have already processed a particular LSA, and incorporating cost-based flooding control to update the Link State Database (LSDB) only with more efficient path information. These modifications helped improve the overall accuracy and stability of the network simulation, and provide a more realistic representation of LSA flooding mechanism.

Another significant challenge was integrating Pygame for an interactive and real-time visualization while managing user commands. We achieved this by implementing threading, which allowed us to run multiple processes concurrently. This ensured that the processing of user inputs, network updates, and graphical rendering could occur simultaneously without impacting the overall performance of the simulation. Additionally, the use of a priority queue for managing user commands was crucial. It helped us maintain an orderly processing of actions, guaranteeing that each change in the network was executed in sequence and accurately reflected in the visual representation.

## Conclusion

Working on the OSPF Simulation as part of our coursework proved to be a blend of hard work and valuable insights. It took us deep into OSPF, a crucial protocol in network engineering, stretching beyond just theory into solving actual problems. An essential part of this project was conducting multiple tests on our implementations, especially on the flooding algorithm. These tests were eye-openers. Initially, we might have been a bit too sure of our solutions, but the testing process highlighted the need for thorough checks, something we hadn’t fully appreciated before. Our skills in Python programming and network simulation grew alongside a deeper understanding of OSPF's mechanics. Facing and solving issues like LSA flooding and dynamic path selection showed us that understanding complex ideas is one thing, but applying them effectively is another. All in all, this OSPF Simulation project really underscored the importance of hands-on experience in grasping networking concepts, teaching us the value of testing and retesting in algorithm implementation.

## Reference

AitzazTahirCh, OSPF-Simulation, (2022), GitHub repository, https://github.com/AitzazTahirCh/OSPF-Simulation/tree/main

https://www.geeksforgeeks.org/ospf-implementation/