Balance the Network Traffic Load in Software-Defined Networking

Aliya Arapbayeva, Alua Toktassyn, Nazerke Izbassarova, Saida Burmaganova, Zhuldyz Zhaksybayeva
Advisor Prof. Zohaib Latif
Nazarbayev University
Computer Science Department
Astana, Kazakhstan

***Abstract*—Traditional network architecture is too rigid to be scalable and efficient in traffic distribution. Software-defined networking (SDN) addresses these issues by separating the control and data planes to improve network management. However, SDN faces challenges, such as controller bottlenecks and traffic congestion. To overcome these, load-balancing algorithms are implemented. This project examines current load-balancing approaches and proposes a solution that distributes load evenly between optimal paths. The employed OpenFlow protocol allows checking real-time network distribution and testing the solution's efficiency. The study reviews existing research on various models for enhancing traffic distribution in SDN. Following a comprehensive analysis of traditional, heuristic, and AI-driven techniques, the findings reaffirm that load balancing is an efficient and scalable solution for SDN routing. Mininet is used for network emulation, Floodlight serves as the controller, and the REST API is queried to check real-time statistics. Initial results show that round-robin load balancing slightly improves network distribution performance. However, it lacks adaptability for complex topologies. Further work explores the integration of blockchain to enhance security and transparency.**

I. Introduction

Traditional networking, which depends on fixed hardware-based architectures, has long been the foundation of data communication. However, this conventional approach faces several inherent limitations, including inflexible infrastructure, high costs, and complex management. These limitations make it more challenging to adapt to the increasing demands of modern networks, which require more dynamic control, rapid scalability, and efficient traffic management.

Software-defined networking (SDN) emerged as a revolutionary solution to address the limitations of traditional networking by separating the control plane from the data plane. This change in architecture allows for centralized control, improved programmability, and increased flexibility in managing networks. Nevertheless, SDN introduces its own set of challenges, particularly related to scalability and optimal resource allocation. One significant limitation of SDN is its potential for controller bottlenecks and uneven traffic distribution, which can lead to suboptimal network performance and increased latency.

Load balancing within SDN has become an essential strategy to mitigate these issues. By employing dynamic algorithms and adaptive traffic distribution techniques, load balancing ensures network traffic is more evenly distributed across available resources. This approach minimizes congestion, enhances the efficiency of network operations, and optimizes bandwidth and processing power. Load balancing solutions in SDN address controller limitations by distributing workloads effectively, improving network reliability and scalability.

The project implements a Depth-first search (DFS) -based load-balancing solution within the Floodlight SDN controller to optimize traffic distribution. By using adaptive algorithms and Python scripts, this approach focuses on demonstrating improvements in managing traffic loads, ultimately enhancing the overall performance and resilience of SDN-based networks.

The main objectives of the project include:

- Develop a controller-based load-balancing application in an SDN environment.
- Continuously monitor link utilization to inform traffic routing decisions
- Dynamically calculate and update optimal paths based on real-time network conditions
- Implement adaptive path selection algorithms to balance the network load efficiently.
- Ensure high performance, scalability, and adaptability of the load-balancing solution
- Minimize latency and maximize throughput for improved user experience

This paper is organized into eight sections that systematically address the challenges and solutions in SDN load balancing. Section I introduces the limitations of traditional networking and the role of SDN in traffic management. Section II provides background on SDN architecture and load-balancing fundamentals. Section III reviews related works, comparing traditional and AI-driven approaches. Section IV details the project design, system architecture, and key features. Section V discusses the implementation process, including algorithms and testing. Section VI evaluates performance improvements through empirical results. Section VII examines ethical and legal considerations, and Section VIII concludes with insights and future directions, including the integration of blockchain.

II. Background

SDN is now a promising flexible and scalable network management framework. By separating the control plane from the data plane, SDN offers centralized network control, making

it easier to adapt to traffic fluctuations. Traditional load-balancing methods, such as Equal-Cost Multi-Path (ECMP) routing, were designed to distribute traffic over multiple paths of equal cost. However, these static approaches cannot often respond quickly in dynamic traffic environments [1]. Because of this, researchers have begun exploring adaptive algorithms that can handle traffic in real-time [2]. DFS and other graph-based algorithms are newer methods that have demonstrated potential in SDN load balancing. DFS, for example, helps discover alternative paths between nodes, which reduces congestion and improves overall network efficiency. Studies have shown that when DFS is implemented on SDN controllers like Floodlight, it can dynamically adjust flow rules by tracking real-time link utilization [3]. This approach is shown to enhance both latency and throughput. Moreover, the OpenFlow protocol enables SDN controllers to gather data on traffic and update routing rules based on network conditions, making it highly effective for real-time load balancing [3, 4]. Other studies have focused specifically on the Floodlight controller for load balancing, emphasizing the advantages of its REST API for monitoring. For instance, a cost-based load-balancing algorithm using Floodlight's Statistics API is developed to adjust traffic paths based on real-time link data [5]. This adaptive approach reduced packet loss and a more stable network, even under high-traffic conditions. Building on these findings, our project utilizes a DFS-based load-balancing method within Floodlight, combining the Statistics API and dynamic path selection to enhance traffic distribution and mitigate congestion.

III. Related Works

Traditional load balancing methods, such as Round Robin (RR) and Weighted Round Robin (WRR), are static and struggle with dynamic traffic, resulting in inefficiencies in network performance. SDNs offer centralized control, making them ideal for machine learning-driven optimization. [6] have explored reinforcement learning (RL) for adaptive load balancing, but challenges remain in capturing long-term traffic dependencies. The authors propose a novel approach that combines a Temporal Fusion Transformer (TFT) for traffic prediction with a Deep Q-Network (DQN) for dynamic load balancing. The TFT predicts future traffic, enabling the DQN to optimize real-time routing decisions. Experimental results show significant improvements in throughput, latency, and packet loss compared to RR and WRR, demonstrating the effectiveness of the proposed method. This research highlights the potential of Transformer-based models and reinforcement learning (RL) for intelligent network management in SDNs.

Expanding on the potential of intelligent load balancing, [7] provides a detailed theoretical review of routing, load balancing, and delay minimization optimization algorithms in SDN. They compare various approaches, emphasizing their applicability to large-scale networks and multi-metric routing. Their work improves SDN's efficiency and resilience by addressing issues related to scalability, real-time adaptability, and redundancy assurance. As networks evolve towards more dynamic and software-driven architectures, the optimization techniques discussed can enhance automation, improve traffic management, and reduce latency in next-generation networks. The insights from this work could drive the integration of AI-based decision-making, making SDN more autonomous and adaptable in complex environments, such as 5G, 6G, and cloud computing.

Despite the advancements in AI-driven approaches, computational efficiency remains a critical challenge. [8] address this by discussing the problem of managing load balancing without high computational cost. Traditional techniques, such as ECMP routing, are impractical in real-world networks due to the presence of different-sized flows and topologies that are unsuitable for ECMP, which specializes in equal-cost paths. Thus, this paper proposes a new yet simple algorithm: a myopic congestion-aware algorithm. The algorithm assigns a path that flows with minimal increase in network cost after assignment. It has three main features: it does not split flow, minimizes overhead, and uses dynamic routing, making it ideal for adapting to real network conditions. The algorithm remains computationally simple, as it does not utilize splitting and randomly chooses a topology section to evaluate the network cost, ensuring efficient path assignment and cost evaluation.

To further enhance dynamic load balancing, anticipating traffic patterns becomes crucial. [9] focuses on the value of predictive models for dynamic routing, particularly in predicting elephant flows—heavy high-bandwidth packet series with the same characteristics. While ECMP distributes flows over different paths, elephant flows that share the same path with other flows can overwhelm the network. The paper accentuates the need to predict elephant flows and uses historical data from Facebook's data center to test and train various predicting models. A hybrid FARIMA-RNN model is proposed, showing high accuracy in predicting elephant flows. When trained sufficiently with trace data, the model enables ECMP to preserve paths for elephant flows, preventing network overload preemptively.

In addition to these optimization techniques, [10] analyzes the integration of artificial intelligence (AI) techniques into load-balancing strategies within SDN. They categorize AI-based load-balancing methods into supervised learning, unsupervised learning, reinforcement learning, and hybrid approaches, each employing specific algorithms to enhance network performance. Supervised learning methods utilize algorithms like support vector machines and decision trees to predict traffic patterns based on labelled datasets, enabling proactive load distribution. Unsupervised learning approaches, such as clustering algorithms, identify inherent traffic patterns without prior labelling, facilitating adaptive load balancing in dynamic environments. Reinforcement learning techniques, including Q-learning and DQNs, enable SDN controllers to learn optimal load balancing policies through trial and error, improving over time. Hybrid methods combine multiple AI techniques to leverage their complementary strengths, achieving more robust and efficient load-balancing solutions. This comprehensive survey offers important insights into the ap-

plication of AI in SDN load balancing, showing the potential of machine learning algorithms to enhance network efficiency and adaptability. Although AI-based methods were not implemented, the reviewed studies informed our understanding of key factors for efficient load balancing in SDNs.

Further emphasizing the role of dynamic routing, [11] discuss how dynamic routing algorithms assist in lowering latency by modifying routing paths in real time depending on network conditions. These algorithms minimize packet delays and prevent bottlenecks by continuously monitoring network conditions and rerouting traffic from faults and congestion. This works well in SDN setups, where the centralized control plane offers a global network perspective for coordinated routing decisions. The authors introduce the AVRO (African Vulture Routing Optimization) method to optimize network conditions dynamically within an SDN framework. The three stages of the AVRO approach are initialization, which takes topology-based route optimization into account; exploration, which makes dynamic routing adjustments; and development, which improves paths for load balancing and network utilization. This method preserves security while improving performance.

To compare the effectiveness of various load balancing techniques, [12] investigate a variety of load balancing techniques on SDN architectures, including Least Connection and Shortest Delay, to determine which one most effectively lowers network load and latency. While the efficacy of these algorithms has been studied separately, their relative performance in comparable circumstances has not been investigated. The Least Connection algorithm balances server load by allocating traffic according to the number of active connections. In contrast, the Shortest Delay algorithm prioritizes paths with the lowest latency, ensuring effective data transmission. Research on SDN controllers, such as RYU, has also influenced the choice of controllers for assessing these load-balancing strategies. This study evaluates the performance of both algorithms using metrics related to CPU consumption, bandwidth, and response time, providing insights into their suitability for various network scenarios.

In a systematic review of load-balancing strategies, [13] classify them into conventional and AI-driven techniques. Traditional methods include static load balancing approaches like Round-Robin and Least-Connections, while AI-driven methods involve reinforcement learning and neural networks. The paper also highlights key performance evaluation metrics such as throughput, latency, packet loss, and load uniformity. For our research, this paper provides a structured classification of static vs. dynamic approaches and offers guidelines on measuring and benchmarking SDN load-balancing solutions. Additionally, it discusses multi-controller load balancing, which is relevant to large-scale SDN deployments.

Finally, [14] provides a comprehensive survey on integrating AI techniques into SDN for load balancing. The paper begins by analyzing the SDN architecture and identifying challenges related to traffic distribution and load imbalance. It then categorizes various AI-based load-balancing methods, evaluating them based on the algorithms employed, the specific problems they address, and their respective strengths and weaknesses. The survey also summarizes metrics used to assess the effectiveness of these techniques and discusses current trends and challenges in AI-based load balancing, offering insights for future research. This paper critically assesses various AI-based load-balancing methods, highlighting their advantages and limitations. It also identifies current challenges and potential areas for further investigation in AI-driven load-balancing strategies.

TABLE I
COMPARISON OF FEATURES ACROSS LITERATURE.

Ref.	RT Adapt.	Low Comp.	Scalability	Packet Loss	Throughput Enh.	Latency Red.
[6]	✓	✗	✗	✓	✓	✓
[7]	✓	✗	✓	✓	✓	✓
[8]	✓	✗	✓	✓	✓	✗
[9]	✓	✗	✗	✓	✓	✗
[10]	✓	✓	✓	✗	✓	✓
[11]	✓	✓	✓	✗	✓	✓
[12]	✓	✗	✓	✗	✓	✓
[13]	✓	✗	✓	✓	✓	✓
[14]	✓	✓	✓	✗	✓	✓
This work	✓	✓	✓	✓	✓	✓

Table I presents a detailed comparison of various SDN load-balancing approaches, evaluating their performance in key areas such as adaptability, efficiency, scalability, packet loss, throughput, and latency. While some existing methods demonstrate strengths in specific aspects, they often exhibit weaknesses in others. For example, studies [6] and [9] show limitations in scalability, whereas approaches in [8] and [9] do not effectively minimize latency. Also, studies [10], [11], [12] and [14] are characterized by significant packet loss. Many of these methods (e.g. [6], [7], [8], [9], [12] and [13]) also involve high computational complexity posing challenges for real-world implementation. The proposed approach, utilizing a DFS-based adaptive mechanism with the Floodlight controller, achieves balanced performance across all key metrics. Our approach demonstrates balanced performance across key metrics such as traffic distribution, packet loss, and latency, indicating its potential as a scalable and effective alternative to existing methods.

IV. PROJECT DESCRIPTION

This section provides a detailed description of the Python script that creates a load balancer on the Floodlight SDN controller using several algorithms. Floodlight is a Java-based and freely available controller. In Floodlight, we additionally use the OpenFlow protocol to create an environment. Also, the SDN controller communicates with the switches and routers using a communication protocol defined by OpenFlow to change and enforce network policies [15].

A. System Features

The proposed load-balancing solution introduces several features designed to enhance network performance and optimize traffic management. One key aspect is the ability to balance traffic dynamically across multiple network links. By modifying flow pathways during data transmission, the system effectively mitigates network congestion within SDN-based environments [16]. Additionally, it incorporates flow rule management, which allows the system to add or remove flow rules dynamically based on the current network state. This feature enables rapid adaptation to changing traffic patterns and network conditions [17]. While the architecture aims to be scalable and adaptable, the current implementation is tailored to a custom topology. Nevertheless, its modular design provides a foundation for extension to more diverse deployment environments in the future. [7].

B. Design

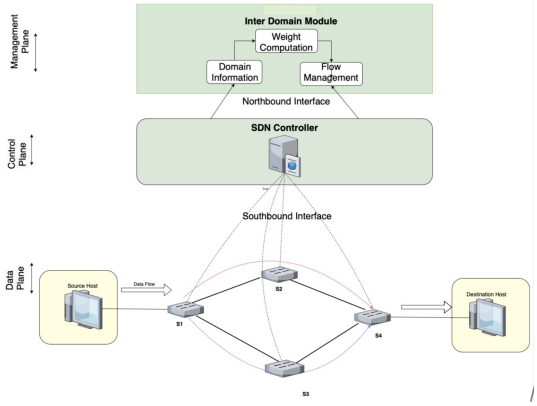


Fig. 1. The architectural model

The architectural model shown in Figure 1 reveals that a layered structure consists of 3 planes. They are the management plane, control plane, and data plane [18]. The first plane, the management plane, is responsible for weight computation and flow management across network domains. The module also interacts with the SDN controller in the control plane through the Northbound Interface. It provides centralized control and dynamic path calculation. The SDN controller then communicates with the switches in the data plane through a Southbound Interface to direct data flow between the source and destination hosts [19]. Modular design can support efficient inter-domain traffic management, optimizing load balancing across the network.

The system architecture comprises several key components, organized into functional modules that collaborate to achieve dynamic traffic load balancing. At the core of the architecture is the Floodlight controller, which acts as the central decision-making entity. It continuously monitors network activity and interacts with domain-specific modules to optimize traffic flow based on real-time data, utilizing the Floodlight REST API for communication.

One of the main modules is the Domain Information Module, which gathers and maintains network topology information, including switch and host connections, link utilization, and available bandwidth. Using this data, the module constructs a graph-based representation of the network, where nodes denote switches and edges represent links, each weighted by its current load or cost.

Building on this, the Path Computation Module analyzes the graph to identify potential paths between source and destination nodes. Algorithms such as Depth-First Search (DFS) are applied to evaluate all viable routes. Each path is assessed based on cumulative link cost, and the route with the minimum cost is selected for optimal traffic routing.

Once the optimal path is determined, the Flow Rule Generation Module translates this path into flow rules expressed in JSON format. These rules specify how packets should be forwarded through the network and are dynamically transmitted to the relevant switches via the Floodlight REST API. By continually updating flow rules based on real-time network conditions, the system adapts seamlessly to changing traffic patterns, thereby maintaining balanced and efficient data transmission.

C. Requirements

The development of the load-balancing solution is guided by high-level objectives and specific capabilities necessary for effective deployment.

Robustness is a critical requirement, ensuring continued operation even in cases of network congestion or link failures. Additionally, the system is optimized for efficiency by reducing latency and maximizing resource utilization across the network.

From a functional perspective, the system must compute routing paths dynamically based on real-time link utilization. It should also support the dynamic insertion and deletion of flow rules to facilitate effective load balancing. Moreover, traffic should be distributed efficiently across multiple network paths to prevent congestion and improve overall performance.

Several non-functional requirements are also considered. Performance is a crucial factor, as the system must minimize latency while making real-time decisions and updating workflow rules.

V. Progress and Implementation

This section outlines the development progress and implementation details of the proposed load-balancing solution. It describes the methodology adopted for software development, the step-by-step implementation of the system components, results and the challenges encountered during integration and testing. The iterative nature of the development process, supported by Agile practices, facilitated continuous improvement and adaptation based on testing results and feedback. The following subsections detail the engineering methodology, technical implementation, and key decisions made throughout the project.

A. Software Engineering Methodology

The load-balancing solution was developed following an iterative and incremental software engineering methodology. We selected a methodology emphasizing flexibility, effective team member communication, and incremental progress toward our project goals. Specifically, we adopted an Agile approach, which allows us to break the project into manageable sprints and respond efficiently to changes and feedback.

1. Requirements Gathering and Analysis: In the project's first phase, the team holds discussions to identify the key requirements for the load-balancing system. This process involves:

- **Functional Requirements:** Dynamic path calculation, real-time load balancing, monitoring of link utilization, and flow rule insertion and deletion.
- **Non-Functional Requirements:** Performance, scalability, and adaptability.

2. Design and Prototyping: After finalizing the requirements, we begin designing the system architecture. This design phase involves creating a network simulation using Mininet, defining the interactions between the controller (Floodlight), switches, and hosts, and selecting algorithms for path calculation and load balancing.

We develop a basic prototype to test the integration of Mininet with Floodlight and to simulate traffic flows. This prototype helps us validate our chosen design approach.

3. Iterative Development: The implementation is divided into smaller tasks, and the team completes them during iterative sprints. Each sprint aims to achieve specific objectives like implementing a specific functionality (e.g., dynamic flow rule creation, link utilization monitoring), testing it and refining the implementation based on testing feedback from the advisor. The team also regularly meets for sprint planning, progress tracking, and debugging.

4. Continuous Improvement: After every sprint, the team reviews the system's performance and usability. Based on test results and advisor feedback, the system is continuously improved. For example, the path calculation logic is fine-tuned to handle network congestion scenarios better, and flow rule insertion is optimized for faster updates.

B. Implementation Details

This section shows the key steps of implementing the project, like creating the Mininet topology to implementing path calculation, link cost calculation, and optimal path selection.

1. Floodlight Controller Setup: Floodlight is open-source, supports OpenFlow, and easily integrates with Mininet. To set it up we use a Docker container. The setup process involves the following steps:

- **Docker Container Setup:** We pull Floodlight controller image from the repository and started as a container. This simplifies the setup without manually configuring dependencies and environment settings. The container is initiated with the command in Fig. 2.

```
zhuzi@zhuzi-VirtualBox:~$ sudo docker run -p 8080:8080 -p 6653:6653 piyushk2001/
floodlight-controller
2024-11-14 08:11:25.140 INFO  [n.f.c.m.FloodlightModuleLoader] Loading modules f
rom src/main/resources/floodlightdefault.properties
2024-11-14 08:11:25.938 WARN  [n.f.r.RestApiServer] HTTPS disabled; HTTPS will n
ot be used to connect to the REST API.
2024-11-14 08:11:25.939 WARN  [n.f.r.RestApiServer] HTTP enabled; Allowing unsec
ure access to REST API on port 8080.
2024-11-14 08:11:25.939 WARN  [n.f.r.RestApiServer] CORS access control allow AL
L origins: true
```

Fig. 2. Network Traffic Distribution Before Load Balancing

In Fig. 2, port `6653` is used for OpenFlow communication, and port `8080` provides access to Floodlight's REST API.

- **REST API for Network Management:** We configure the controller to allow remote communication with Mininet through the REST API. This API is used to monitor link utilization and manage flow rules dynamically.

2. Mininet Topology Creation: To simulate the network environment, we create a topology with four switches and four hosts using Mininet. The specific topology is as follows:

- **Hosts and Switches:**
  - Host `h1` connected to switch `s1`
  - Host `h2` connected to switch `s2`
  - Host `h3` connected to switch `s3`
  - Host `h4` connected to switch `s4`
- **Switch-to-Switch Links:**
  - Switch `s1` connected to switch `s2`
  - Switch `s2` connected to switch `s4`
  - Switch `s1` connected to switch `s3`
  - Switch `s3` connected to switch `s4`

The topology structure is represented in Fig. 3.

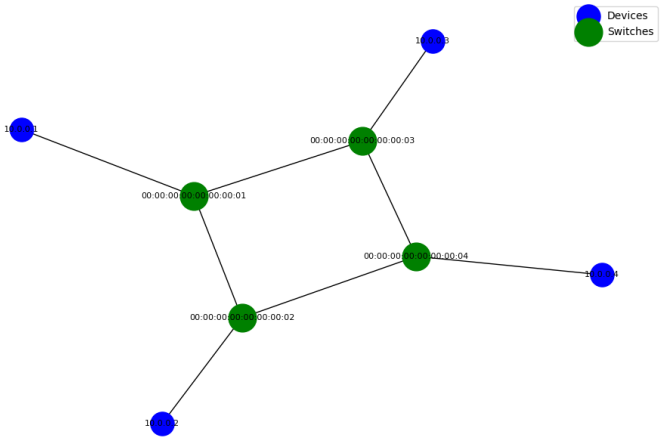


Fig. 3. The topology structure

In addition, we implement a Python script to define a custom network topology for more flexibility, as shown in Fig. 4. As previously described, this script creates a network with four switches and four hosts and specifies their connections.

After preparing the script, this topology is created in Mininet using the command shown in Fig. 5. Here, `custom` specifies using the custom topology.

```python
from mininet.topo import Topo

class MyTopo(Topo):
    def build(self):
        # Create switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        # Create hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')

        # Add links between switches and hosts
        self.addLink(h1, s1)
        self.addLink(s1, s2)
        self.addLink(s2, h2)
        self.addLink(s2, s4)
        self.addLink(s4, h4)
        self.addLink(s4, s3)
        self.addLink(s3, h3)
        self.addLink(s3, s1)

topos = {'mytopo': (lambda: MyTopo())}
```

Fig. 4. Mininet Topology Creation using Python Script

```
user@user-HP-ProDesk-600-G1-TWR:~/seniorProject$ sudo mn --custom simpleTopo.py
--topo mytopo --controller=remote,ip=127.0.0.1:6653 --switch ovsk
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (s1, s2) (s2, h2) (s2, s4) (s3, h3) (s3, s1) (s4, h4) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet-wifi> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet-wifi>
```

Fig. 5. The topology creation command in Mininet

3. Monitoring Link Utilization: The first step toward dynamic load balancing is to monitor link utilization in the network. This is accomplished by using Floodlight's Statistics API, which provides bandwidth information for each switch-port pair. The API endpoint `/wm/statistics/bandwidth/{sw}/{pt}/json` is queried periodically to fetch the current bandwidth statistics. These statistics are used to calculate the link cost in the network.

The bandwidth statistics API returns data in JSON format, including the receive and transmit bits per second for each port on each switch. This data is collected and stored for further analysis.

4. Link Cost Calculation: We determine the link cost by bringing the bandwidth utilization data for each link. So, we calculate the cost by summing the receive rate (RX) and transmit rate (TX) for a given link. The higher the bandwidth usage (RX + TX), the higher the link cost.

The link cost calculation function is implemented as follows:

Algorithm 1 Link Cost Calculation

Input: src_sw, src_pt, dst_sw, dst_pt
Output: link cost

src_rx, src_tx ← get_stats(src_sw, src_pt)
dst_rx, dst_tx ← get_stats(dst_sw, dst_pt)

cost ← src_rx + dst_tx
return cost

5. Optimal Path Selection Based on Link Cost: Since the core of the load balancing system is optimal path selection, the system selects the path with the lowest total cost. For this reason, the DFS algorithm explores all possible paths between the source and destination. For each path, the program calculates the total cost by summing up the costs of the links in that path. Then, the path with the lowest cost is selected as the optimal path.

Algorithm 2 Path Selection Based on Link Cost

Input: graph(G), src, dst
Output: optimal_path

paths ← find_all_paths(G, src, dst)
min_cost ← ∞
optimal_path ← None
for each path in paths **do**
  total_cost ← 0
  for $i = 0$ to $len(path) - 2$ **do**
    src ← path[i]
    dst ← path[i+1]
    total_cost ← total_cost + G[src][dst]['cost']
  end for
  if total_cost < min_cost **then**
    min_cost ← total_cost
    optimal_path ← path
  end if
end for
return optimal_path

6. Flow Rule Insertion and Deletion: Once the optimal path is determined, we generate flow rules in the appropriate format and use the Floodlight REST API to insert them into the switches along the path. The flow rule includes information such as:

- **Switch ID:** The switch to which the flow is applied.
- **In-port:** The port where the packet arrives.

- **Out-port:** The port to which the packet should be forwarded.
- **Actions:** The action to be applied (e.g., output=2).

For example, a flow rule is inserted on switch1 to forward packets from port 1 to port 2, as shown in the following JSON:

```
{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "flow-mod-1",
  "cookie": "0",
  "priority": "32768",
  "in-port": "1",
  "active": "true",
  "actions": "output=2"
}
```

To ensure traffic flows along the selected path, we insert similar flow rules at each switch in the path. These flow rules are dynamically updated if network conditions change.

7. Dynamic Load Balancing: The dynamic aspect of the system is achieved by continuously monitoring link utilization and adjusting the path and flow rules accordingly. When a link becomes more utilized compared to others, the system recalculates the optimal path and updates the flow rules to maintain balanced traffic distribution.

VI. Evaluation

To evaluate the effectiveness of the load-balancing solution, we monitor network performance before and after applying the load-balancing algorithm.

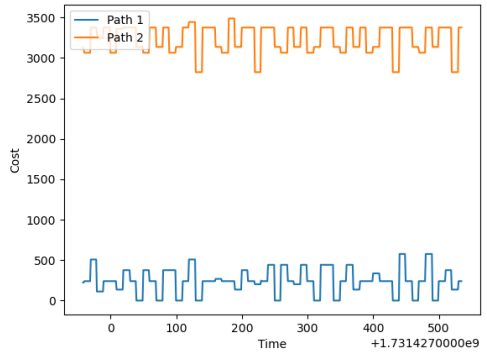


Fig. 6. Network Traffic Distribution Before Load Balancing

Fig. 6 illustrates the network traffic distribution before applying load balancing, where traffic flow is concentrated on specific paths, leading to congestion and inefficient utilization of network resources. After implementing the dynamic load balancing, as shown in Fig. 7, noticeable improvements in traffic distribution can be observed. As seen in the graph, the traffic paths begin to intersect, and the range between traffic loads across different links becomes narrower. This indicates a more balanced use of the available paths, reducing the likelihood of overloading any single link.

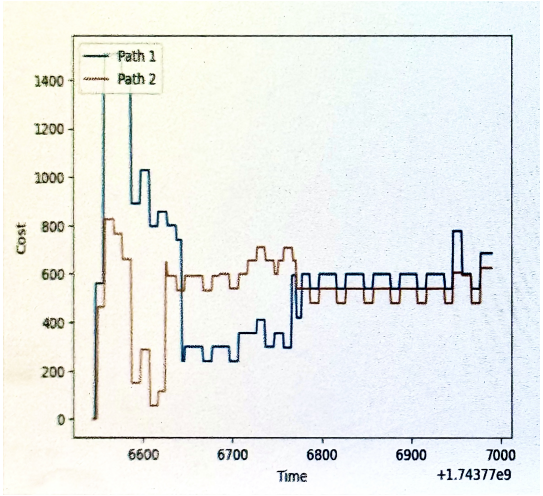
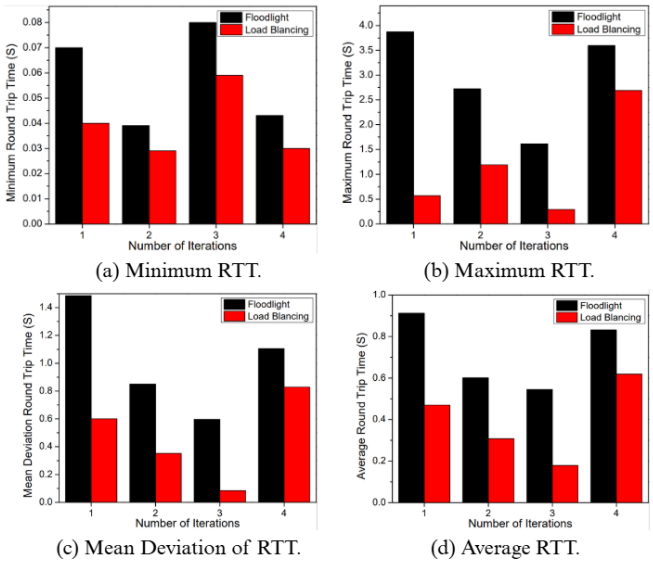


Fig. 7. Network Traffic Distribution After Load Balancing



(a) Minimum RTT. (b) Maximum RTT.

(c) Mean Deviation of RTT. (d) Average RTT.

Fig. 8. Round Trip Time (RTT) performance metrics comparing traditional Floodlight and load-balanced scenarios: (a) Minimum RTT showing baseline latency, (b) Maximum RTT demonstrating worst-case delays, (c) Mean Deviation of RTT indicating path consistency, and (d) Average RTT reflecting overall network performance. Results show significant improvement in all metrics after implementing load balancing.

As shown in Figure 8, the load balancing solution demonstrates superior performance across all RTT metrics compared to traditional Floodlight routing. These metrics, including Average, Maximum, Minimum, and Mean Deviation, were measured in four iterations (1 to 4) during data transfer between the source and destination. The most notable improvements can be observed in the maximum RTT values in Figure 8b, where latency spikes are reduced by approximately 75%.

Figure 8a shows the Minimum RTT values, revealing that in the traditional Floodlight scenario, latency varies between 0.01 and 0.08 s, indicating underutilized paths. The Maximum

RTT (Figure 8b) spikes to 4.0s in Iteration 1, demonstrating severe congestion on certain paths, while the Average RTT (Figure 8d) fluctuates significantly from 0.2 to 1.0 s, highlighting inefficiencies in path selection.

In contrast, the load-balanced scenario shows marked improvements in all metrics. The Maximum RTT (Figure 8b) drops sharply to 0.5s–1.5s, and the Minimum RTT (Figure 8a) tightens to 0.02s–0.04s, indicating better-balanced path utilization. The Average RTT (Figure 8d) stabilizes between 0.6s and 0.8s, while the Mean Deviation of RTT (Figure 8c) shows reduced variability, confirming more predictable dynamic rerouting. These improvements, clearly visible in Figure 8, suggest that the solution successfully maintains optimal paths and dynamically redirects traffic between two hosts, effectively reducing latency spikes and minimizing underutilization.

Further analysis suggests that an adaptive algorithm is needed for optimal load balancing, which can monitor real-time link utilization and dynamically adjust traffic paths.

Problem Encountered: One significant issue arises in ensuring seamless integration between the Floodlight controller and the Mininet simulator. During testing, the system occasionally produces garbage values for bandwidth utilization. Upon investigation, it is found that this issue occurs because cached data is not being cleared in the Floodlight controller or Mininet. As a result, stale data causes inaccurate measurements, which disrupts the dynamic path calculation logic.
Resolution: To resolve this, a process to clear cached data is implemented. Specifically, commands such as `sudo mn -c` are used to reset Mininet, which removes cached topology data. Meanwhile, `sudo docker stop CONTAINER` and `sudo docker rm CONTAINER` are employed to stop and remove Docker containers that host the Floodlight controller. Automating these steps ensures that each test begins with a clean environment.
Impact: This resolution significantly improves the accuracy of real-time link utilization data, which is critical for the effectiveness of the load-balancing algorithm.

VII. Ethical and Legal Considerations

Like any system that processes network data and interacts with communication infrastructure, it is essential to assess the ethical, legal, and societal implications of the proposed solution. This section outlines the key ethical concerns of data privacy, network security, and system accountability. It also addresses compliance with legal frameworks, such as the General Data Protection Regulation (GDPR), as well as considerations for intellectual property and the rationale behind the ethical trade-offs made during the development process. Additionally, it reflects on the long-term sustainability and social responsibility of the system, emphasizing its potential impact on users and network infrastructure.

A. Ethical Issues

One of the legal implications is data privacy. For efficient load balancing, the packets transferred to the network are recorded with their performance metrics. However, no user-defining data is stored; only quantitative statistics are collected.

Another concern that needs to be addressed is network security. The structure of the SDN makes it a target for cyber attacks. The reason is that this network is centralized to control the network. Therefore, if the central control plane is corrupted, all flow rules might be deleted or modified, further damaging the network. This creates a danger of single points of failure. Thus, SDN should be secured as much as possible by preventing all unauthorized access and increasing fault tolerance. For this, role-based access control should be implemented, along with continuous monitoring of logs and examination of potential attack vectors to the system, as well as their prevention. The Common Vulnerability Scoring System (CVSS) could be used for this. This scoring system determines the severity of the system's vulnerability [20]. Another option is to add Blockchain into our system to ensure control distribution. This ledger system will log all actions taken on the network, allowing for intrusion detection and transparency.

Finally, to ensure the fairness and reliability of the system, constant logging should be implemented. This will generate well-documented reports on each decision made for the network change.

B. Legal Compliance

According to the General Data Protection Regulation (GDPR), any data that can be used to identify individuals counts as personal data [21]. The user should permit to utilize personal data straightforwardly. We complied with the GDPR's data protection principle "by design and by default". We do not collect personal data about users; we only gather statistics about the paths' performance. The topology is entered into our system by the user, not installed in the user's technology. This way, neither IP address nor payload data is revealed. In the future, if we provide our system for installation, users will be informed about any personal data collection, storage, or sale. Also, we plan to enable IP address hashing to prevent the revealing of sensitive data, then store it only temporarily and dispose of it as soon as it becomes unnecessary.

The resources we used to achieve our goals were Mininet, Ubuntu, and Floodlight. All of them are open-source technologies, and we did not breach any terms of use while utilizing these tools.

C. Sustainability and Social Responsibility

Load balancing enhances data transmission and reduces network traffic without requiring any network hardware. It allows centralized control of network data transfer, and being aware of the whole network topology contributes to the network's sustainability. This enables users to gain a more comfortable digital experience.

VIII. Conclusion and Possible Future Work

In conclusion, we learned about the structure of the SDN, the Mininet simulator, and how to work with the Floodlight

controller. Using this knowledge, we created various topologies and a system to plot them. Our algorithm finds paths between the source and destination. Then, it computes the path cost using Floodlight Statistics API. The dynamic load balancing algorithm is used to redirect flow rules from one path to the next. The results show how changing the path for packets reduces network traffic, and comprehensive routing mechanisms can leverage network sustainability.

To further develop our system, we can optimize the load-balancing algorithm by integrating Blockchain to enhance security and transparency.

SDN introduces a centralized controller that manages all the flows. It consequently enhances network management. However, this setup introduces many risks. If the SDN is damaged, the whole network might be affected. Thus, including Blockchain, which creates nodes for additional network control, is useful for our system. Blockchain can also be used to achieve transparency because all data about chosen flows, packets transmitted, and individuals making adjustments is expected to be stored in it. These logs can then be accessed and analyzed. This way, a new option for increasing security and transparency could be introduced.

References

[1] X. Zhu et al. "An adaptive load balancing scheme for SDN-based cloud datacenters". In: *IEEE Transactions on Cloud Computing* 3.3 (July 2015), pp. 285–295.

[2] D. Kreutz et al. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

[3] J. Luo, Y. Xu, and L. Huang. "A dynamic load balancing algorithm in SDN based on real-time path optimization". In: *Journal of Network and Computer Applications* 85 (May 2017), pp. 30–39.

[4] S. H. Lee, H. J. Yoon, and J. W. Lee. "Load balancing scheme based on a link utilization model for SDN controllers". In: *IEEE Access* 4 (Oct. 2016), pp. 8194–8201.

[5] L. Zhang and Y. Xu. "Floodlight-based adaptive load balancing in software-defined networking". In: *Computer Networks* 136 (Apr. 2018), pp. 17–29.

[6] Evans Tetteh Owusu et al. *A transformer-based deep q learning approach for dynamic load balancing in software-defined networks*. 2025. arXiv: 2501.12829 [cs.NI]. URL: https://arxiv.org/abs/2501.12829.

[7] Maria Daniela Tache (Ungureanu), Ovidiu Păscuțoiu, and Eugen Borcoci. "Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization". In: *Applied Sciences* 14.14 (2024). ISSN: 2076-3417. DOI: 10.3390/app14145967. URL: https://www.mdpi.com/2076-3417/14/14/5967.

[8] Mehrnoosh Shafiee and Javad Ghaderi. "A Simple Congestion-Aware Algorithm for Load Balancing in Datacenter Networks". In: *IEEE/ACM Transactions on Networking* 25.6 (2017), pp. 3670–3682. DOI: 10.1109/TNET.2017.2751251.

[9] Jeandro de M. Bezerra et al. "Performance evaluation of elephant flow predictors in data center networking". In: *Future Generation Computer Systems* 102 (2020), pp. 952–964. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.09.031. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18330309.

[10] Ahmed Hazim Alhilali and Ahmadreza Montazerolghaem. *Artificial intelligence based load balancing in SDN: A comprehensive survey*. 2023. arXiv: 2308.02149 [cs.NI]. URL: https://arxiv.org/abs/2308.02149.

[11] Junyan Chen et al. "Dynamic routing optimization in software-defined networking based on a metaheuristic algorithm". In: *Journal of Cloud Computing* 13.1 (2024), p. 41.

[12] Heru Nurwarsito and Handy Kurniawan Ponco Widagdo. "Comparative Analysis of Load Balancing with Shortest Delay and Least Connection Methods on Software Defined Network". In: *Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology*. SIET '23. Badung, Bali, Indonesia: Association for Computing Machinery, 2023, pp. 589–598. ISBN: 9798400708503. DOI: 10.1145/3626641.3626944. URL: https://doi.org/10.1145/3626641.3626944.

[13] Mohammad Riyaz Belgaum et al. "A Systematic Review of Load Balancing Techniques in Software-Defined Networking". In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.2995849.

[14] Ahmed Hazim Alhilali and Ahmadreza Montazerolghaem. "Artificial Intelligence Based Load Balancing in SDN: A Comprehensive Survey". In: *arXiv preprint arXiv:2308.02149* (2023). URL: https://arxiv.org/abs/2308.02149.

[15] "Floodlight documentation website". In: DOI: https://floodlight.readthedocs.io/en/latest/.

[16] Yuan-Liang Lan, Kuochen Wang, and Yi-Huai Hsu. "Dynamic load-balanced path optimization in SDN-based data center networks". In: *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. 2016, pp. 1–6. DOI: 10.1109/CSNDSP.2016.7573945.

[17] Abha Kumari, Joydeep Chandra, and Ashok Sairam. "Predictive Flow Modeling in Software Defined Network". In: Oct. 2019. DOI: 10.1109/TENCON.2019.8929671.

[18] Zohaib Latif et al. "A comprehensive survey of interface protocols for software defined networks". In: *Journal of Network and Computer Applications* 156 (2020), p. 102563. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102563. URL: https://www.sciencedirect.com/science/article/pii/S1084804520300370.

[19] Zohaib Latif et al. "DOLPHIN: Dynamically Optimized and Load Balanced Path for Inter-Domain SDN Com-

munication". In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 331–346. DOI: 10.1109/TNSM.2020.3045725.

[20] FIRST.org. *CVSS v4.0 FAQ*. Accessed: 2024-11-18. 2024. URL: https://www.first.org/cvss/v4.0/faq.

[21] Richie Koch. *What is considered personal data under the EU GDPR?* https://gdpr.eu/eu-gdpr-personal-data/. Accessed: 2024-11-15. GDPR EU, 2024. URL: https://gdpr.eu/eu-gdpr-personal-data/.