International Journal of Innovative Research and Scientific Studies, 9(9) 2026, pages: 25-40

IJIRSS
International Journal of Innovative Research & Scientific Studies

ISSN: 2617-6548

URL: www.ijirss.com

QoS-aware routing in software-defined internet of things: A comparative evaluation of Dijkstra and bellman–ford under heterogeneous traffic

Abdelali Grif[1*], Abdellah Zyane[1]

[1]*LAPSSII Laboratory, High School of Technology of Safi (EST Safi), Cadi Ayyad University-UCA, Morocco.*

Corresponding author: Abdelali Grif (*Email: a.grif.ced@uca.ac.ma*)

Abstract

The rapid growth of Internet of Things (IoT) infrastructures has created heterogeneous communication environments where multiple applications share network resources, making reliable Quality of Service (QoS) increasingly challenging, especially when routing must support diverse traffic requirements. This study evaluates the performance of two shortest-path routing algorithms—Dijkstra and Bellman-Ford—within a Software-Defined Internet of Things (SD-IoT) architecture to analyze their impact on QoS and resource utilization under heterogeneous traffic. An experimental SD-IoT testbed was implemented using the Ryu controller and Mininet emulator, where each algorithm was integrated and tested separately on a programmable topology. Three IoT applications—e-health, intelligent transportation, and industrial control—were generated simultaneously. The evaluation considered latency, packet loss, link load distribution, and CPU/RAM usage at controller and switch levels. Results show distinct behaviors: Dijkstra achieves lower latency and packet loss due to faster path computation, while Bellman-Ford introduces slightly higher delays and resource consumption because of its iterative process, yet remains stable under varying loads. Both improve link load distribution compared to default routing. Overall, routing efficiency depends on convergence behavior and traffic heterogeneity; Dijkstra suits latency-sensitive applications, whereas Bellman-Ford is more appropriate for adaptive routing scenarios, contributing to QoS-aware SD-IoT design.

Keywords: Bellman-Ford, Dijkstra, Quality of Service (QoS), Routing Algorithms, Software-Defined Internet of Things (SD-IoT), Software-Defined Networking (SDN).

DOI: 10.53894/ijirss.v9i9.11906
Funding: This study received no specific financial support.
History: Received: 8 July 2026 / **Revised:** 14 August 2026 / **Accepted:** 18 August 2026 / **Published:** 14 September 2026

Competing Interests: The authors declare that they have no competing interests.
Authors' Contributions: All authors contributed equally to the conception and design of the study. All authors have read and agreed to the published version of the manuscript.
Institutional Review Board Statement: The authors would like to clarify that generative artificial intelligence tools were used solely as language-support tools during the preparation of this manuscript. Specifically, ChatGPT was used to assist with improving the clarity, grammar, and overall quality of the English language. The AI tool was not used to generate scientific ideas, research methodology, results, or interpretations. All scientific content, analysis, conclusions, and references presented in this manuscript were developed, verified, and validated by the authors. The authors carefully reviewed and edited all AI-assisted text to ensure the accuracy, validity, and relevance of the information provided. Therefore, the authors take full responsibility for the entire content of the manuscript.
Publisher: Innovative Research Publishing

1. Introduction

The rapid growth of the Internet of Things (IoT) has profoundly transformed digital environments by connecting an increasing number of intelligent devices capable of collecting, processing, and exchanging data in real time. This expansion relies on a distributed infrastructure composed of sensors, gateways, and embedded devices that continuously interact to generate and transmit information flows originating from the physical world. Application domains are rapidly expanding: biomedical devices supporting remote monitoring in e-health systems, road sensors integrated into intelligent transportation infrastructures, instrumented production chains associated with Industry 4.0, and interconnected urban infrastructures characteristic of smart cities. A defining feature of this ecosystem remains its heterogeneity. Devices differ significantly in terms of computational capabilities, energy consumption, and communication technologies, resulting in data flows characterized by diverse temporal constraints and priority levels. In such a context, the network infrastructure no longer acts solely as a communication medium; it becomes a critical component responsible for transporting and coordinating multiple information flows while meeting the specific Quality of Service (QoS) requirements associated with each application.

Within this highly interconnected environment, Quality of Service emerges as a key factor in ensuring the reliable operation of IoT applications. The traffic generated by these systems exhibits heterogeneous requirements depending on the operational context. Certain applications—such as medical remote monitoring or industrial control—demand extremely low latency and high reliability to guarantee the timely delivery of critical information. Other services require sufficient bandwidth and minimal packet loss in order to maintain acceptable performance levels. Managing these requirements is complicated by the operational characteristics of IoT networks. Such infrastructures must cope with congestion phenomena, sudden traffic fluctuations resulting from large volumes of sensor-generated data, and often limited computational and communication resources at nodes and gateways. Additionally, these environments are inherently dynamic: network topologies and communication conditions may evolve rapidly due to device mobility, intermittent connectivity, or variations in traffic demand. Under these circumstances, conventional network architectures frequently encounter scalability and adaptability limitations, making it difficult to consistently satisfy the QoS requirements imposed by modern IoT applications.

To address these challenges, a network architecture has increasingly attracted the attention of the research community: Software-Defined Networking (SDN). The fundamental principle of SDN lies in the explicit separation between the control plane and the data plane. This architectural shift removes the traditional decision-making logic from forwarding devices and relocates it to a centralized software controller capable of orchestrating network behavior at a global level. As a result, switches become responsible for executing forwarding rules, while the controller analyzes network conditions—such as link states, queue utilization, and active traffic flows—in order to dynamically adapt routing policies. Such programmability represents a particularly promising engineering paradigm for IoT environments, where heterogeneous devices generate traffic with diverse and often competing operational requirements. Within this context emerges the concept of the Software-Defined Internet of Things (SD-IoT), an architectural model in which sensors, gateways, and communication infrastructures are coordinated through a centralized control plane capable of dynamically adjusting traffic management across the entire system. As illustrated in Figure 1 this architecture enables the SDN controller to maintain a global view of the network and dynamically orchestrate traffic flows according to the QoS requirements of heterogeneous IoT applications. This orchestration capability positions SDN as a key enabler for improving Quality of Service management in IoT networks characterized by high traffic variability and significant device heterogeneity.

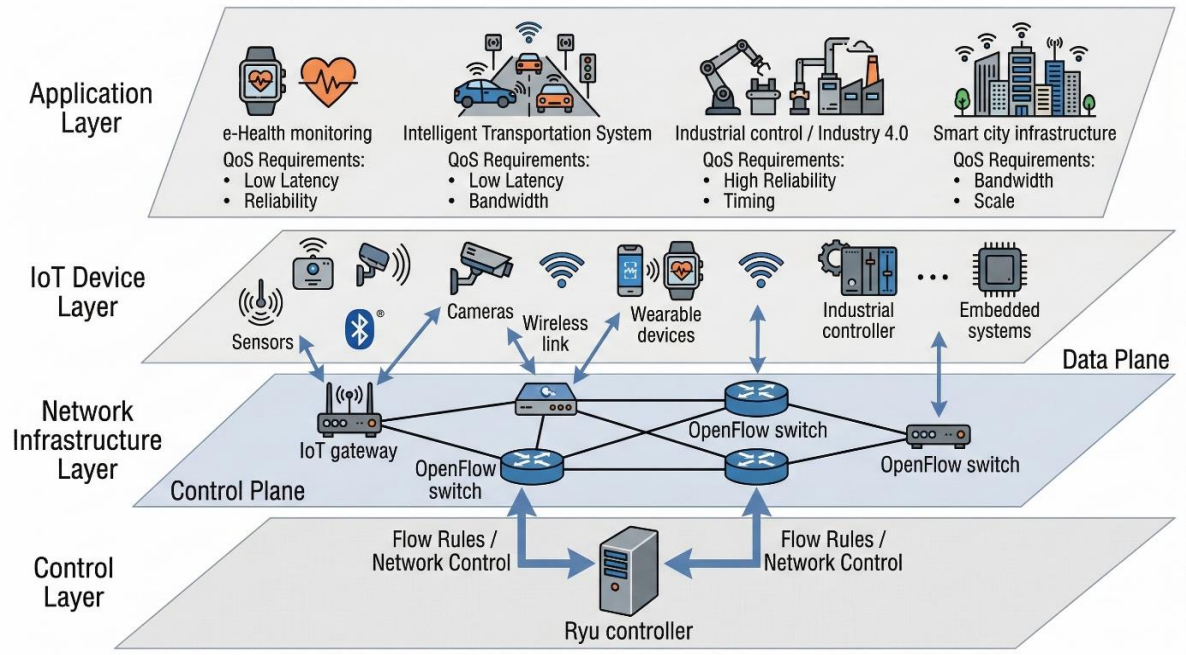

Figure 1.
Conceptual architecture of a Software-Defined Internet of Things (SD-IoT) environment illustrating heterogeneous IoT applications and centralized QoS-aware traffic management through an SDN controller.

The literature dedicated to Software-Defined Internet of Things (SD-IoT) environments highlights several engineering directions aimed at improving traffic management in infrastructures characterized by strong device heterogeneity and significant traffic variability. One major research line concerns traffic classification, a mechanism that enables the identification of flows generated by IoT applications in order to dynamically adjust network policies. Machine-learning-based approaches have been proposed to analyze statistical traffic signatures and anticipate their behavior within SDN infrastructures [1]. Other studies rely on neural architectures or combine deep packet inspection with semi-supervised learning techniques to refine the categorization of application flows [2, 3]. In parallel, several works focus on load management and resource distribution, which represent essential challenges in programmable architectures where controllers must orchestrate a large number of simultaneous flows. Load balancing mechanisms have been introduced to dynamically distribute traffic across links, switches, or controllers, thereby reducing congestion and improving the overall utilization of network resources [4]. In multi-controller architectures, certain solutions incorporate dynamic switch migration techniques to balance control-plane load and enhance network resilience [4-6]. Another research direction addresses the placement of flow rules in switch tables, a challenge directly associated with the memory limitations of SDN devices. Several studies have proposed rule aggregation or optimized rule placement strategies to mitigate TCAM table saturation while preserving visibility over active flows [7]. More recent approaches employ deep learning techniques to dynamically adjust rule installation according to traffic behavior [8].

Within this landscape of solutions, particular attention has progressively focused on Quality-of-Service-aware routing, a central mechanism for orchestrating traffic flows in SD-IoT architectures. Numerous models have been proposed to select transmission paths based on metrics such as latency, available bandwidth, and link reliability. Centralized approaches leveraging the global control capabilities of SDN have been developed to optimize path selection in IoT environments [9, 10]. Other contributions introduce adaptive or heuristic routing strategies designed to accommodate network dynamics and the diversity of application requirements [11-14]. Some studies also employ nature-inspired algorithms or deep learning techniques to enhance routing performance while optimizing the energy consumption of IoT devices [15-17]. Despite these advances, existing research indicates that routing management in SD-IoT environments remains subject to complex trade-offs between path optimality, adaptability to traffic variations, and efficient utilization of network resources.

SD-IoT environments introduce a particularly demanding network dynamic, characterized by the coexistence of multiple traffic flows generated by heterogeneous applications such as medical telemetry, intelligent transportation systems, and connected industrial infrastructures. These flows exhibit distinct performance constraints—minimal latency for time-sensitive applications, high reliability for control systems, and sufficient bandwidth for data-intensive services—and evolve in contexts where network conditions may change rapidly due to traffic bursts, fluctuations in link utilization, or the reconfiguration of transmission paths. Within this framework, routing algorithms play a decisive role in the overall performance of the system, as they directly influence end-to-end latency, congestion phenomena, and the efficient utilization of network resources. Despite the large body of research dedicated to QoS optimization in SD-IoT architectures, the literature has predominantly focused on heuristic or machine-learning-based approaches, while relatively few studies have conducted in-depth analyses of the behavior of classical routing algorithms under realistic SD-IoT scenarios characterized by heterogeneous traffic and diverse QoS requirements.

In this context, several SDN controllers still rely on classical routing algorithms to compute transmission paths within the network. Among them, Dijkstra's algorithm remains widely adopted due to its computational efficiency and its ability

to rapidly determine minimum-cost paths based on a global knowledge of the network topology. This property makes it particularly well suited to SDN architectures where the controller maintains a centralized view of network state. In contrast, the Bellman–Ford algorithm follows an iterative computation process capable of accommodating variations in link costs and more readily integrating changes in network conditions, providing a certain level of robustness in dynamic environments. These two approaches rely on fundamentally different computational principles and may therefore exhibit distinct behaviors when the network operates under variable loads and heterogeneous traffic flows. Despite their frequent presence in SDN controllers and programmable networking infrastructures, comparative evaluations of their performance within SD-IoT environments—where multiple applications coexist and QoS requirements diverge—remain relatively limited in the existing literature.

Within this perspective, the present study aims to systematically analyze the behavior of two widely used routing algorithms in SDN controllers—Dijkstra and Bellman–Ford—when applied to an SD-IoT environment characterized by heterogeneous traffic flows and diverse performance constraints. The primary objective is to compare their performance in a multi-application traffic context in order to evaluate their ability to effectively manage Quality of Service in a programmable network. This analysis is guided by several research questions. First, how do these two algorithms respond to varying traffic loads in an SD-IoT architecture, particularly when the network must simultaneously handle latency-sensitive flows and others that are more tolerant to delay? Second, which of these algorithms is better suited to maintain satisfactory QoS levels depending on the types of IoT applications considered? Finally, this study examines the impact of routing strategies on the utilization of network and controller resources, particularly with respect to link load, computational resource consumption, and the overall efficiency of the network infrastructure.

To address these research questions, this paper provides several contributions aimed at improving the understanding of how routing strategies influence the performance of SD-IoT environments. First, the Dijkstra and Bellman–Ford algorithms are implemented within the Ryu SDN controller in order to enable dynamic path computation in a programmable network infrastructure. Second, an experimental scenario representative of a multi-application SD-IoT environment is designed, incorporating three types of traffic corresponding to common IoT use cases: e-health services, intelligent transportation systems, and industrial control applications, each associated with specific QoS requirements. Third, an experimental evaluation is conducted to analyze the impact of both algorithms on several network performance metrics, including latency, packet loss rate, link load distribution, and resource consumption at both the switch and controller levels in terms of CPU and memory utilization. Finally, a detailed comparative analysis is performed to identify behavioral differences between the two routing approaches and to better understand their ability to manage heterogeneous IoT traffic in a dynamic SD-IoT environment.

The remainder of this paper is organized as follows. Section 2 presents an analysis of existing studies addressing QoS improvement in SDN and SD-IoT environments. Section 3 describes the methodological framework and the experimental architecture used for the evaluation. Section 4 presents the results obtained from the conducted experiments. Section 5 provides an interpretation of these results in relation to previous studies, discusses their implications, and outlines potential directions for future research. Finally, Section 6 summarizes the main conclusions of this work.

2. Literature Review

2.1. Shortest Path Routing in Software-Defined Networks

In SDN architectures, routing decisions are transferred to the control plane, where the controller leverages a global view of the network topology to dynamically compute transmission paths. In this context, shortest-path algorithms derived from graph theory play a central role in determining optimal routes. Among them, Dijkstra's algorithm is widely adopted due to its efficiency in computing minimum-cost paths when link weights are known. Several studies have attempted to improve its execution in distributed environments. Some approaches rely on MapReduce architectures to accelerate path computation in hybrid Cloud–Edge–IoT infrastructures [18] while other distributed implementations based on Hadoop–MapReduce aim to reduce convergence time in large-scale network environments [19]. Multi-objective extensions have also been proposed in order to incorporate multiple network metrics—such as latency and bandwidth utilization—into the route selection process [20]. Furthermore, the performance of these algorithms strongly depends on the underlying control environment; several studies indicate that the choice of the SDN controller can directly influence throughput, packet delivery stability, and jitter variability [21]. Within this perspective, approaches combining SDN, Network Function Virtualization (NFV), and Federated Deep Reinforcement Learning have been proposed to dynamically adapt routing policies in IoT environments and improve network resource utilization [22]. Despite these advances, the analysis of shortest-path algorithm behavior in IoT scenarios characterized by heterogeneous traffic flows remains relatively limited.

2.2. Bellman-Ford in Dynamic and Hybrid Network Environments

In environments where network topology evolves frequently or where link conditions fluctuate, routing approaches must move beyond an implicit assumption often associated with classical shortest-path methods: the stability of link weights. It is precisely in such contexts that the Bellman–Ford algorithm retains a significant presence in the literature. Its iterative mechanism, based on the progressive propagation of distance estimates between neighboring nodes, allows it to incorporate variations in link costs and recompute routes when the network structure changes. Recent studies exploit this property in highly dynamic environments. For instance, Bellman–Ford has been applied in Low Earth Orbit (LEO) satellite networks, where orbital movements continuously modify network topology; when combined with clustering techniques, it contributes to improving routing resilience in the presence of rapidly changing connectivity conditions [23]. Other research integrates the algorithm into Hybrid Software-Defined Networking architectures, where some network segments still

operate using traditional protocols while others are managed by a centralized SDN control plane. In this context, the heuristic H-STE algorithm employs Bellman–Ford to simultaneously adjust OSPF weights and traffic distribution, enabling control over a large portion of network flows while reducing maximum link utilization [24]. These findings illustrate the ability of Bellman–Ford to operate effectively in hybrid and evolving network environments; however, its behavior remains insufficiently explored when such dynamics are combined with the traffic heterogeneity and variability characteristic of SD-IoT environments.

2.3. Comparative Studies of Routing Algorithms in SDN

Comparative studies of routing algorithms constitute another line of research within SDN architectures, aiming to evaluate their respective performance in programmable environments. Some works have compared Dijkstra, Bellman–Ford, an extended version of Dijkstra, and Floyd–Warshall within the POX controller, showing that Bellman–Ford can provide greater stability and more effective packet loss management in certain experimental scenarios [25]. Other studies have implemented several algorithms—particularly Dijkstra, Bellman–Ford, Floyd–Warshall, and A*—in the RYU controller, highlighting the superior performance of Dijkstra in terms of average throughput [26]. A complementary analysis considering convergence time, memory consumption, and scalability also indicates that Dijkstra is generally better suited for large-scale topologies, whereas Bellman–Ford remains relevant in specific contexts [27]. These studies provide useful comparative insights; however, they often rely on homogeneous traffic scenarios and a limited set of evaluation metrics, leaving relatively few analyses of their behavior in SD-IoT environments characterized by heterogeneous traffic flows and diverse QoS requirements.

2.4. Dijkstra vs Bellman-Ford in Programmable Networks

Among the existing studies, the work most closely related to our approach is presented in Tammanashastri, et al. [28] where the authors compare the Dijkstra and Bellman–Ford algorithms within a programmable SDN environment. Their results indicate that Dijkstra provides better performance in relatively stable topologies with positive link weights, whereas Bellman–Ford demonstrates a greater ability to adapt when network conditions evolve rapidly. This comparison therefore highlights the trade-off between computational efficiency and robustness to topology variations. However, the analysis remains limited to a generic SDN context and does not consider the specific constraints of SD-IoT environments, particularly the coexistence of heterogeneous application flows and differentiated Quality of Service requirements.

2.5. Research Gap and Contribution

The analysis of existing studies reveals several persistent limitations in the evaluation of routing algorithms within programmable network environments. As illustrated in Table 1, most comparative studies have been conducted in generic SDN settings, often relying on relatively simplified simulation scenarios and homogeneous traffic models. Evaluations typically consider a limited set of performance metrics—such as throughput or packet loss—without incorporating the broader range of indicators required to assess Quality of Service in IoT environments. In addition, relatively few studies explicitly address the coexistence of heterogeneous application flows, which represents a defining characteristic of SD-IoT architectures, where different services impose distinct performance requirements. Furthermore, the impact of routing algorithms on the resource utilization of the SDN controller and network devices is rarely examined. To address these gaps, the present work provides a comparative analysis of the Dijkstra and Bellman–Ford algorithms within an SD-IoT environment that integrates multiple representative traffic types—e-health, intelligent transportation, and industrial control—and evaluates their behavior through an extended set of QoS metrics, including latency, packet loss rate, link load distribution, as well as CPU and memory consumption at both the controller and switch levels. This approach makes it possible to identify the operational conditions under which each algorithm performs most effectively and to gain a deeper understanding of their behavior in realistic SD-IoT scenarios.

Table 1.
Comparative summary of routing algorithm evaluation studies in SDN and SD-IoT environments.

Study	Algorithms Compared	Environment	Traffic Model	QoS Metrics Evaluated	Controller Resource Analysis	Limitations
Ghimire and Basnet [25]	Dijkstra, Bellman-Ford, Extended Dijkstra, Floyd-Warshall	SDN (POX controller)	Homogeneous traffic	Packet loss, stability	No	Limited metrics and simplified simulation environment
Patel, et al. [26]	Dijkstra, Bellman-Ford, Floyd-Warshall, A*	SDN (RYU controller)	Generic traffic	Throughput	No	Evaluation limited to a single performance metric
Narang and Hossain [27]	Dijkstra vs Bellman-Ford	Generic network environment	Not specified	Convergence, memory usage, scalability	No	Does not consider IoT constraints or QoS diversity

Tammanashastri, et al. [28]	Dijkstra vs Bellman-Ford	SDN	Generic traffic	Basic routing performance	No	Does not consider heterogeneous IoT traffic
This work	Dijkstra vs Bellman-Ford	SD-IoT environment (Ryu controller)	Heterogeneous traffic (e-health, intelligent transportation, industrial control)	Latency, jitter, packet loss, throughput, link load	CPU/RAM consumption at controller and switches	Experimental topology limited in size

3. Research Methods

3.1. SD-IoT System Architecture

The experimental architecture is based on a Software-Defined Internet of Things (SD-IoT) environment in which control functions are separated from packet forwarding mechanisms. A structural separation. The control plane is centralized in a Ryu controller, responsible for orchestrating routing decisions and dynamically installing forwarding rules within the network. The data plane, in contrast, consists of OpenFlow switches responsible for forwarding packets according to the instructions received from the controller. The entire infrastructure is emulated using Mininet, a widely used environment for experimenting with SDN architectures, which allows the reproduction of programmable network topologies while maintaining precise control over network parameters.

The experimental topology, illustrated in Figure 1 consists of five interconnected SDN switches, denoted $s_1$to s_5, forming the backbone of the network. The switches execute only the forwarding rules installed by the controller—no local routing decisions are performed. Three traffic injection clusters are connected to switches s_1, s_2, and s_3, each simultaneously generating three types of IoT traffic corresponding to different application scenarios. These flows are then forwarded through the SDN infrastructure toward three application servers connected to switches $s_4$and s_5, representing respectively e-health, intelligent transportation, and industrial control services. This configuration enables the reproduction of a multi-traffic SD-IoT environment in which several categories of flows coexist and simultaneously compete for network resources.

The overall structure of this experimental infrastructure—including switches s_1–s_5, traffic generation clusters, and application servers—is illustrated in Figure 2. From a formal perspective, the network can be modeled as a directed graph $G = (V, E)$, where Vrepresents the set of network nodes (switches and hosts) and Edenotes the set of links connecting these nodes. Each link $(i, j) \in E$is associated with a weight $w(i, j)$, representing a transmission cost such as latency or link load. These weights are used by routing algorithms to determine optimal paths within the studied SD-IoT environment.

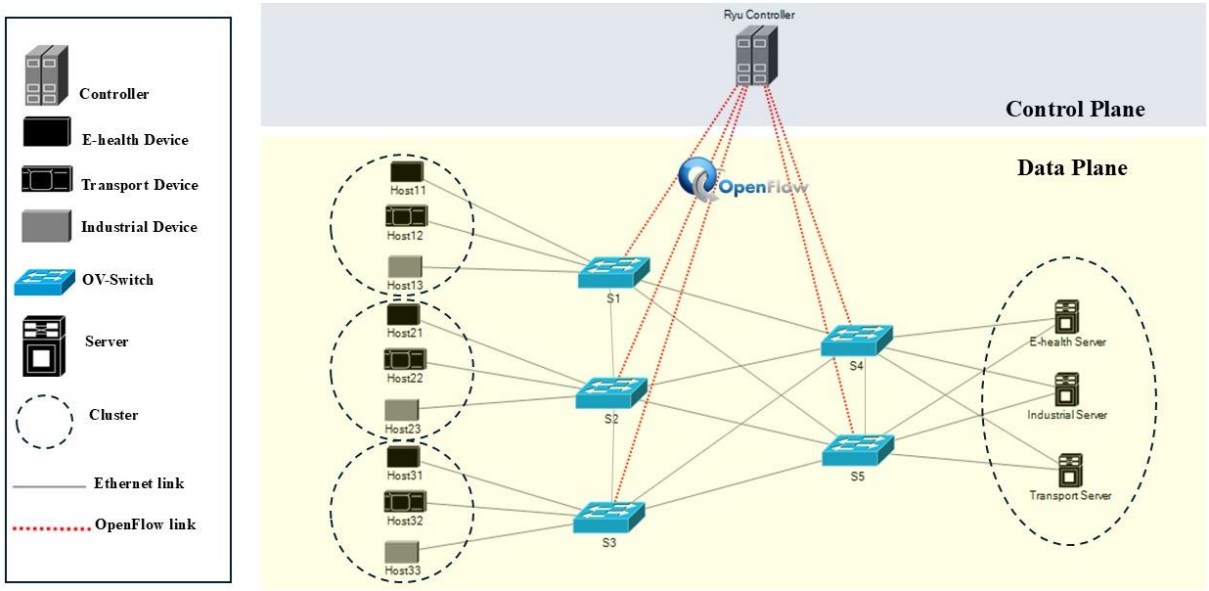

Figure 2.
Experimental SD-IoT Architecture with Ryu Controller.

3.2. Experimental Environment

The experimental evaluation was conducted in an environment based on the Mininet network emulator, which is widely used for modeling and analyzing SDN and SD-IoT architectures. This environment enables the reproduction of a programmable network infrastructure while allowing precise control over experimental parameters such as network topology, links, and application traffic flows. Centralized network management is ensured by the Ryu controller,

implemented in Python and configured to operate with the OpenFlow 1.3 protocol, enabling the dynamic installation of forwarding rules in OpenFlow switches as well as the integration of the routing algorithms under study.

The simulations were executed on the Ubuntu Linux 20.04.6 (64-bit) operating system. The Mininet environment and the Ryu controller were deployed on a VirtualBox virtual machine equipped with 8 GB of RAM and 8 CPU cores, providing sufficient computational resources for SD-IoT network emulation and the execution of routing algorithms. This virtual machine is hosted on a physical machine with 16 GB of RAM and an AMD Ryzen 7 5800U processor running at 1.9 GHz, ensuring stable execution of the experiments and reliable collection of performance metrics. The detailed description of this experimental environment is intended to ensure the scientific reproducibility of the obtained results and to facilitate the replication of experiments under similar conditions.

3.3. Routing Algorithm Implementation

The Dijkstra and Bellman–Ford routing algorithms were integrated into the Ryu controller in order to determine transmission paths within the SD-IoT infrastructure. A simple principle was adopted: only one algorithm is active at a time. Each algorithm is executed in a separate experimental scenario, enabling an independent analysis of their behavior before performing a comparative evaluation of their performance. This approach prevents any interference between routing mechanisms and ensures a fair assessment of both methods under identical network conditions.

The controller first constructs a representation of the network as a graph $G = (V, E)$using topology information collected through OpenFlow messages. The nodes of the graph correspond to SDN switches, while the edges represent the network links connecting these switches. Based on this representation, the controller's routing module executes either the Dijkstra or the Bellman–Ford algorithm in order to compute the minimum-cost path between the source and destination nodes. The Dijkstra algorithm determines the shortest path by progressively exploring the neighbors of the source node, whereas Bellman–Ford performs iterative edge relaxation, allowing it to incorporate variations in link costs within the network.

Once the optimal path has been determined, the Ryu controller generates the corresponding OpenFlow forwarding rules for each segment of the computed path. These rules are then installed in the switches using FlowMod messages, creating the necessary entries in the switches' flow tables. As a result, packets can be forwarded directly at the data plane level without involving the controller for each transmission. When network conditions evolve—such as the appearance of a new traffic flow or changes in link load—the controller can recompute paths and dynamically update routing rules, thereby ensuring continuous adaptation to traffic conditions within the studied SD-IoT environment.

3.5. SD-IoT Traffic Scenario

To evaluate the behavior of routing algorithms in an environment representative of IoT networks, several typical SD-IoT applications were simulated within the experimental infrastructure. Three traffic categories were considered: e-health, intelligent transportation, and industrial control, each characterized by distinct QoS requirements. The e-health flows correspond to connected medical applications that require rapid and reliable transmission of biomedical data; their primary requirement is very low latency in order to ensure timely responses from monitoring systems. Intelligent transportation flows represent communications between vehicles and intelligent infrastructures, characterized by dynamic traffic patterns and high sensitivity to packet loss, which may compromise the reliability of traffic information. Finally, industrial control flows correspond to communications used in automated industrial systems, where the priority is stability and regularity of transmissions to guarantee the continuous operation of control processes.

In the experimental scenario, each cluster simultaneously generates these three traffic types, thereby reproducing a multi-application environment typical of SD-IoT networks. The offered load is progressively varied in order to observe the behavior of routing algorithms under different traffic conditions. Three load levels were considered: 50 Mbps, 150 Mbps, and 300 Mbps for each injected traffic type. This configuration makes it possible to analyze network performance as the demand for resources gradually increases. Performance is then evaluated using several QoS and operational efficiency metrics, including latency, packet loss rate, link load distribution, as well as resource consumption (CPU and RAM) at both the controller and switch levels. This methodology enables a detailed observation of the impact of routing algorithms according to traffic type and load level, while highlighting the trade-offs between service quality and resource utilization in a heterogeneous SD-IoT environment.

3.6. QoS Evaluation Metrics

Performance evaluation in the SD-IoT environment relies on a set of indicators designed to simultaneously analyze the Quality of Service perceived by applications and the operational impact of routing mechanisms on the network infrastructure. Measurements are obtained from the experimental environment combining Mininet for network emulation, iPerf for traffic generation and flow analysis, as well as statistics collected by the Ryu controller. These metrics make it possible to observe how routing algorithms influence transmission dynamics when multiple IoT flows with different requirements coexist within the same infrastructure.

The evaluated indicators are as follows:

- Latency: the transmission delay between a source node and a destination node. This metric is measured using ping commands executed between Mininet hosts, providing the packet round-trip time (RTT).
- Packet Loss Rate: the proportion of packets lost during transmission. This value is estimated from the results of UDP transmissions generated with iPerf, by comparing the number of packets sent with those successfully received.

- Throughput: the effective volume of data transferred between two nodes during a given time interval. This metric is directly provided by iPerf, which also allows control of traffic parameters such as protocol type, target bandwidth, and transmission duration.
- Link Load: the utilization level of the links connecting SDN switches. This information is derived from traffic statistics collected within Mininet and from flow information reported to the controller.
  Beyond these network metrics, the study also examines the impact of routing algorithms on system resources, particularly:
- CPU Usage: the processor utilization rate at both the SDN controller and data-plane devices.
- RAM Usage: memory consumption associated with the execution of routing modules and the management of flows within the controller.

These measurements are collected using system monitoring tools and internal statistics provided by the Ryu controller. Together, these indicators make it possible to relate two fundamental dimensions of the analysis: the performance perceived by IoT applications and the computational footprint of the routing mechanisms deployed in the SD-IoT infrastructure.

3.7. Experimental Procedure

The experimental evaluation follows a structured protocol to analyze routing behavior in an SD-IoT environment under controlled conditions. The network is first deployed in Mininet with OpenFlow switches, traffic generators, and servers, allowing the Ryu controller to discover the topology. Traffic is then generated using iPerf to emulate e-health, transportation, and industrial applications in a multi-flow setting. Each scenario activates only one routing algorithm at a time: Dijkstra is used first to compute shortest paths and install FlowMod rules, followed by Bellman–Ford under identical conditions to isolate its impact. Performance metrics such as latency, packet loss, link load, CPU, and RAM usage are collected across different load levels. Each experiment is repeated multiple times with 60-second sessions to ensure stable results, enabling a reliable comparison of QoS and resource efficiency.

Table 2.
Experimental procedure used for the evaluation of routing algorithms

Step	Description	Tools / Methods
Network Initialization	Creation of the SD-IoT topology including 5 OpenFlow switches (s1–s5), three traffic generation clusters, and three application servers	Mininet
Traffic Generation	Simultaneous injection of three IoT traffic types: e-health, intelligent transportation, and industrial control	iPerf
Routing Algorithm Selection	Activation of a routing algorithm in the controller for each experimental scenario	Ryu Controller
Path Computation	Computation of the optimal path between source and destination nodes	Dijkstra or Bellman–Ford
Flow Rule Installation	Deployment of OpenFlow rules in the switches along the selected path	OpenFlow FlowMod
Traffic Transmission	Forwarding of packets through SDN switches toward application servers	SD-IoT infrastructure
QoS Metrics Collection	Measurement of network performance indicators: latency, packet loss, and link load	ping, iPerf, Mininet statistics
Resource Monitoring	Analysis of resource utilization at both controller and switch levels	CPU / RAM monitoring
Experiment Repetition	Execution of tests under different traffic load levels to ensure result reliability	Experimental iterations

4. Solutions and Results

This section presents the experimental results obtained from the SD-IoT testbed described in the previous section. The evaluation focuses on the behavior of the Dijkstra and Bellman–Ford routing algorithms under heterogeneous traffic conditions and different load levels. The analysis considers several QoS indicators, including latency, packet loss, link utilization, and resource consumption at both the controller and switch levels.

4.1. Latency Analysis

Latency corresponds to the propagation delay of a packet between a source node and a destination node within the SD-IoT infrastructure. This metric directly reflects the responsiveness of the network: the lower the transmission delay, the more effectively the communication chain satisfies the temporal constraints imposed by connected applications. In IoT environments characterized by heterogeneous traffic flows and strict timing requirements, particularly for e-health services or certain industrial control systems, the ability to control this parameter becomes essential.

The experimental results related to average latency are presented in Figure 3 and Figure 4 corresponding respectively to the scenarios executed using the Dijkstra and Bellman–Ford algorithms. The analysis reveals a systematic difference in the temporal behavior of the two routing mechanisms. Paths computed by Dijkstra generally lead to lower transmission

delays. A clear observation. The path selection process, based on the progressive exploration of minimum distances, favors the rapid establishment of stable routes, thereby reducing the time required for packets to traverse the infrastructure.

The behavior observed with Bellman-Ford differs slightly. Latency values remain close to those obtained with Dijkstra but exhibit a slightly higher average level. This variation can be associated with the intrinsic nature of the algorithm: the iterative edge-relaxation mechanism requires several update cycles before reaching a convergent state of the network graph. As a result, the computation of the optimal path unfolds over a longer temporal interval.

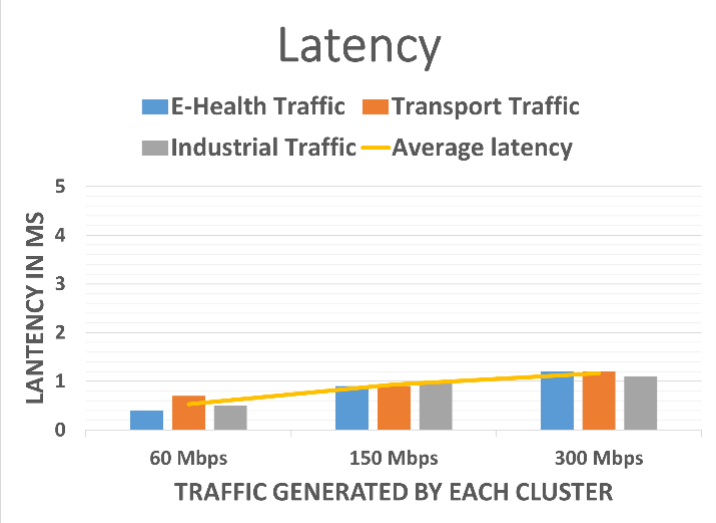

Figure 3.
Average latency using Dijkstra algorithm.

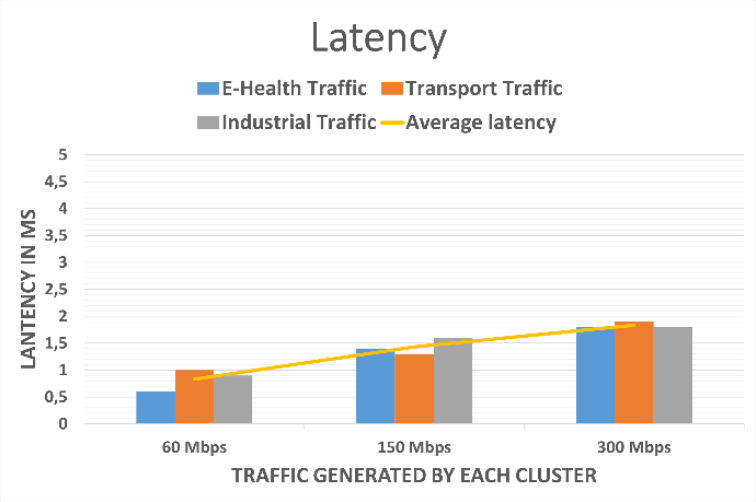

Figure 4.
Average latency using Bellman-Ford algorithm.

4.2. Packet Loss Analysis

The packet loss rate represents a direct indicator of transmission reliability in a programmable network environment. In an SD-IoT system, where multiple application flows with different requirements coexist, the ability of the routing mechanism to preserve the integrity of transmissions becomes a determining factor. Excessive packet loss may lead to performance degradation, trigger additional retransmissions, or—in the case of critical applications—disrupt service continuity. Routing robustness can therefore be assessed by observing how each algorithm maintains flow stability as network load increases.

The experimental results related to packet loss rate are presented in Figures 5 and 6 corresponding respectively to the scenarios using Dijkstra and Bellman–Ford. The analysis highlights a progressive difference in the behavior of the two approaches as traffic load increases. Routes computed by Dijkstra maintain a relatively low packet loss level, even under the highest load conditions. The shortest-path selection mechanism promotes a more stable distribution of traffic flows across the topology, thereby limiting congestion situations that could otherwise lead to packet drops.

The behavior observed with Bellman–Ford reveals a slightly different tendency. When the traffic load reaches higher levels, particularly 300 Mbps, the packet loss rate becomes more noticeable. This phenomenon can be associated with the iterative nature of the algorithm: the progressive propagation of distance information across the network graph requires a longer convergence period. During this route adjustment phase, some forwarding decisions may rely on temporarily

outdated cost information. The consequence: a temporary accumulation of packet losses before the routing paths fully stabilize.

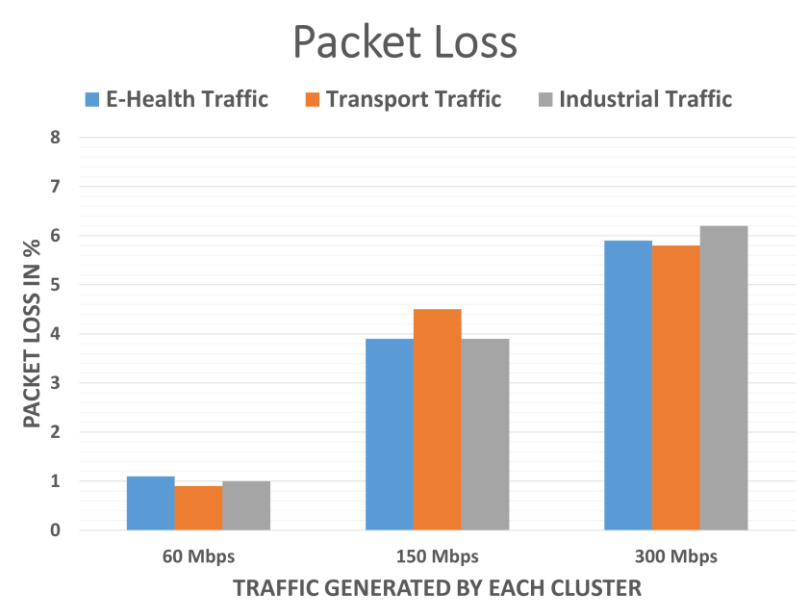

Figure 5.
Packet loss rate using Dijkstra algorithm.

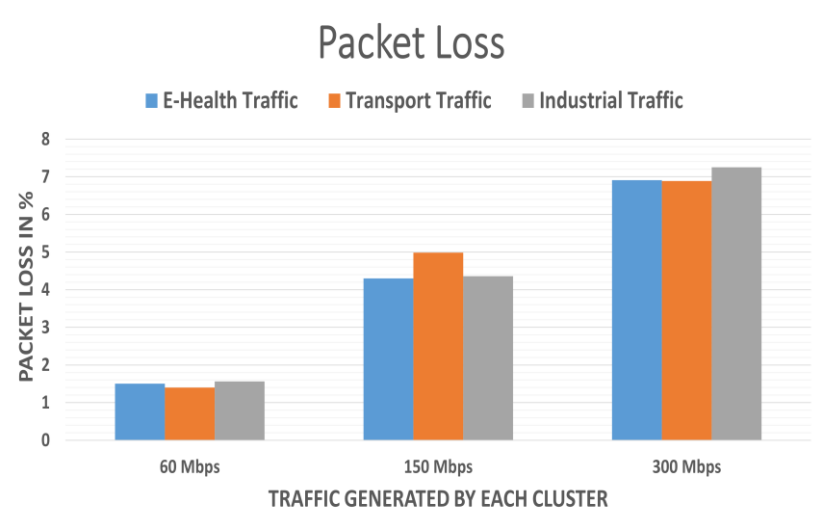

Figure 6.
Packet loss rate using Bellman-Ford algorithm.

4.3. Link Load Distribution

The distribution of load across network links constitutes a key indicator for analyzing congestion phenomena in an SD-IoT network. It represents the volume of traffic passing through each connection between the switches in the topology. When traffic concentrates on a limited number of links, certain portions of the network quickly become saturated while other resources remain underutilized. This load asymmetry may lead to increased transmission delays, higher packet loss rates, and, more broadly, a degradation of the Quality of Service perceived by applications. Conversely, a balanced distribution of flows allows a more efficient use of the overall capacity of the infrastructure.

The results related to link load distribution are illustrated in Figures 7, 8, and 9, representing respectively the initial scenario without specific optimization, followed by the configurations using the Dijkstra and Bellman–Ford algorithms. The observation of Figure 7 highlights a typical behavior of the default routing mechanism of the Ryu controller: a significant portion of the traffic is concentrated on a few central links of the topology. Some connections therefore carry a high load, while others remain lightly utilized. This unbalanced distribution increases the probability of local congestion.

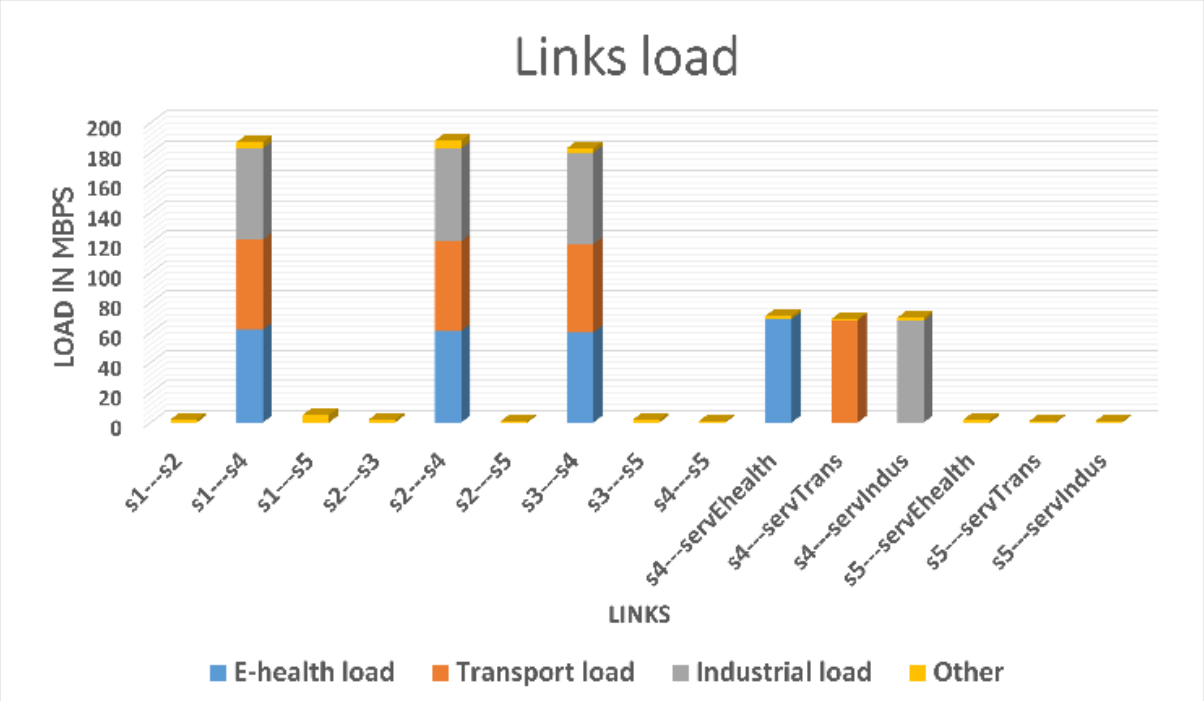

Figure. 7.
Link load distribution with default Ryu routing.

The scenarios integrating shortest-path algorithms exhibit a different dynamic. Figures 8 and 9 show that the introduction of Dijkstra or Bellman–Ford leads to a more homogeneous utilization of the available links within the topology. Traffic flows are distributed across a wider set of paths, reducing the pressure on the most heavily used connections. This effect is particularly visible in the intermediate segments of the network, where the load becomes more evenly distributed among multiple possible routes.

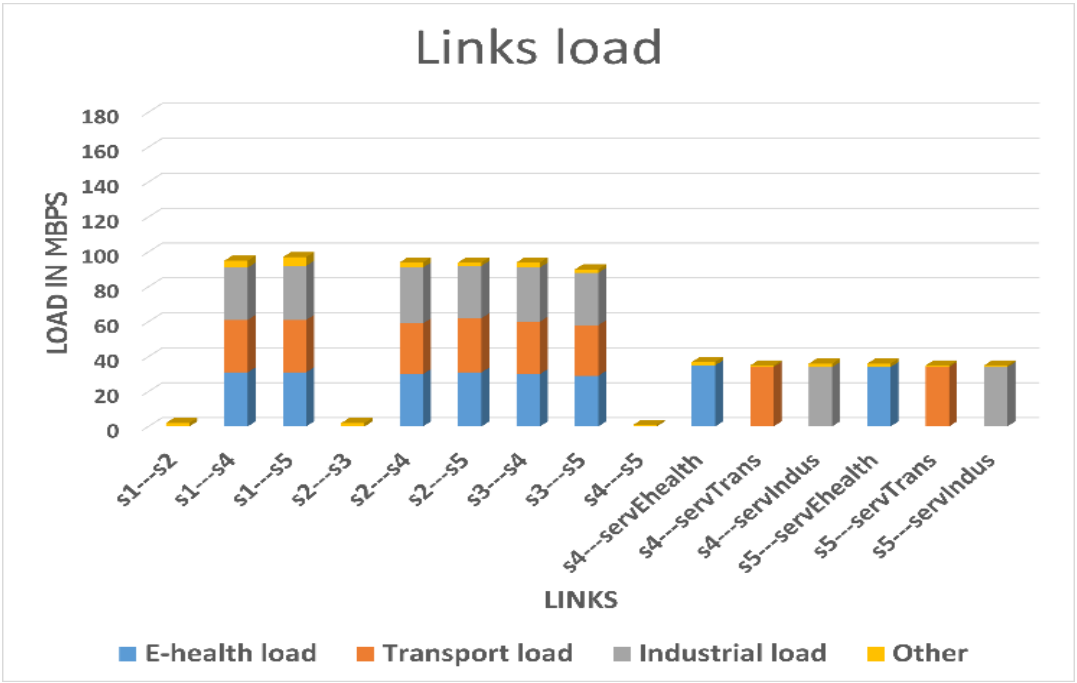

Figure 8.
Link load distribution using Dijkstra algorithm.

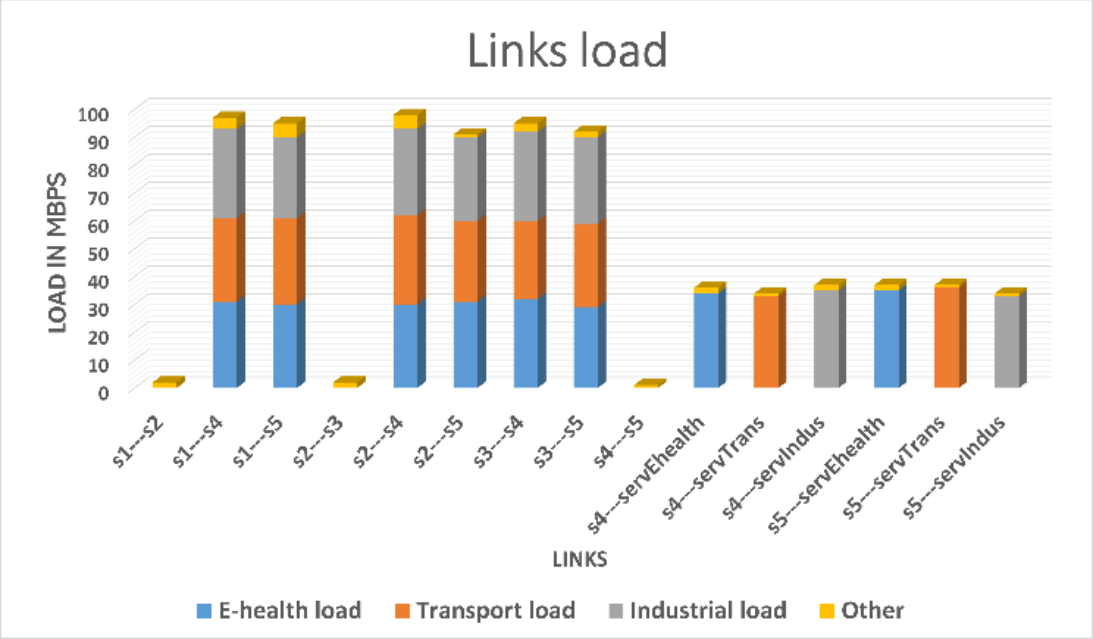

Figure 9.
Link load distribution using Bellman-Ford algorithm.

This redistribution of traffic improves the overall utilization of network resources. Fewer saturation points. Greater stability of the transmission system. In an SD-IoT environment characterized by multiple flows and variable loads, the ability of routing mechanisms to orchestrate such distribution becomes a determining factor for limiting congestion and maintaining consistent performance across the entire infrastructure.

4.4. System Resource Utilization

The analysis of resource consumption makes it possible to estimate the computational load induced by routing mechanisms within the SD-IoT infrastructure. In an SDN architecture, the centralized controller orchestrates path establishment and installs flow rules in OpenFlow switches; this centralization implies that network computation and management operations may directly influence system resource utilization. The study therefore considers two levels of observation: the switches of the data plane and the SDN controller.

Measurements related to CPU and RAM consumption of OpenFlow switches are presented in Figures 10 to 13. The results indicate that resource utilization at the switch level remains generally low for both studied algorithms. In the scenario using Dijkstra (Figures 10 and 11), processor and memory consumption remain stable even as traffic load increases, indicating that packet forwarding operations and the application of OpenFlow rules impose only a limited burden on data-plane devices.

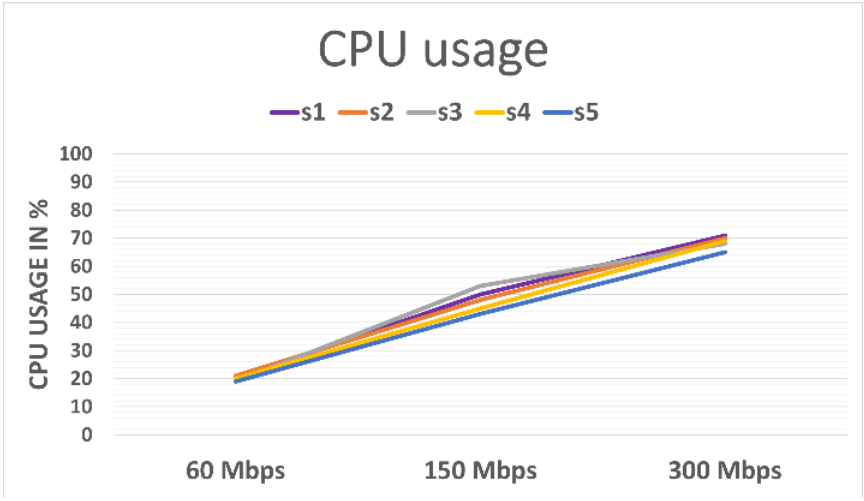

Figure 10.
Average CPU usage at switches using Dijkstra algorithm.

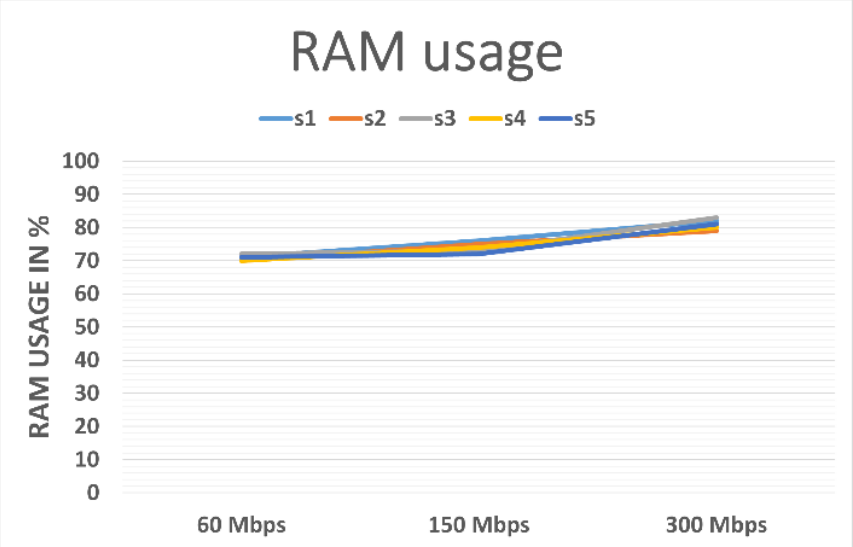

Figure 11.
Average RAM usage at switches using Dijkstra algorithm.

The behavior observed with Bellman–Ford (Figures 12 and 13) remains similar, although a slight increase appears under the highest traffic loads. This variation can be associated with the larger number of flow updates generated by the controller during route recalculation.

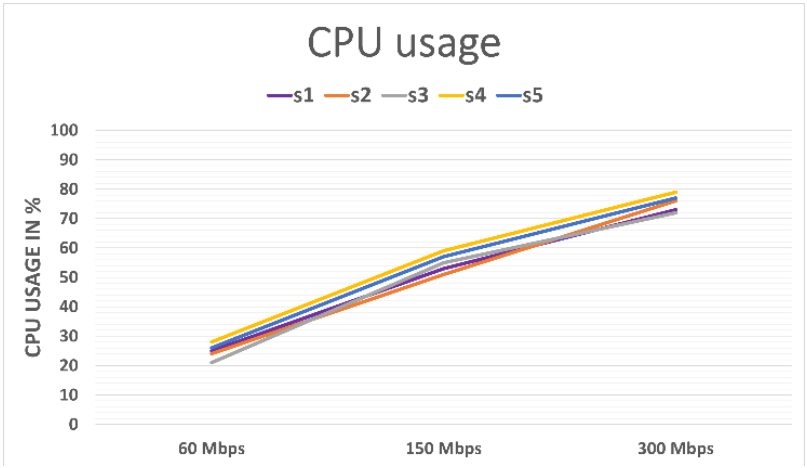

Figure 12.
Average CPU usage at switches using Bellman-Ford algorithm.

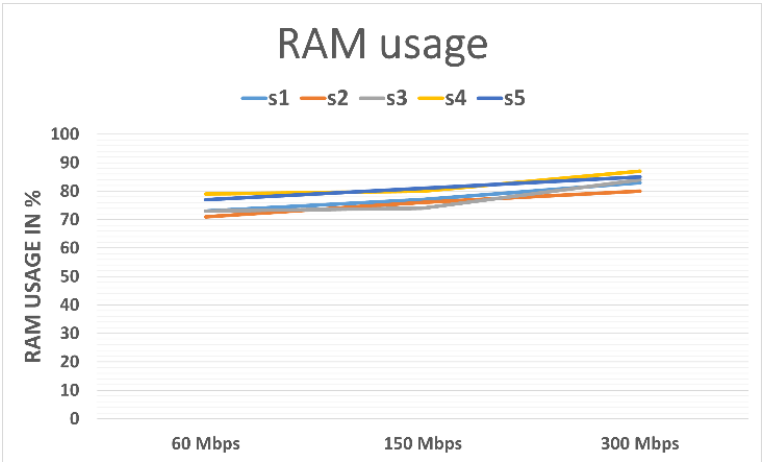

Figure 13.
Average RAM usage at switches using Bellman-Ford algorithm.

The impact on the control plane is presented separately in Table 1 which summarizes the average CPU and RAM consumption of the Ryu controller for both algorithms under different load levels. The results indicate that Bellman–Ford

imposes a higher computational load on the controller than Dijkstra, and that this difference becomes more noticeable as the traffic volume increases. This disparity can be mainly attributed to the iterative nature of the Bellman–Ford algorithm, which requires multiple edge relaxation phases before the path computation converges.

Despite this relative increase, the observed resource consumption levels remain within the limits compatible with the experimental infrastructure used in this study. However, the results suggest that as the network size or traffic intensity grows, the load placed on the controller may become a decisive factor when selecting routing mechanisms for SD-IoT environments characterized by high traffic density.

Table 3.
Average CPU and RAM usage of the Ryu controller for Dijkstra and Bellman-Ford under varying loads.

Load (Mbps)	CPU Dijkstra (%)	CPU Bellman-Ford (%)	RAM Dijkstra (MB)	RAM Bellman-Ford (MB)
50	12.4	15.8	210	245
150	18.7	24.5	230	280
300	27.9	36.4	255	310

5. Discussion

5.1. Interpretation of the Main Findings

The experimental results highlight several key insights regarding the behavior of routing mechanisms in the studied SD-IoT environment. The lower latency observed with Dijkstra can be attributed to the deterministic nature of its shortest-path computation process. The algorithm progressively constructs a set of nodes whose minimum distance from the source has been confirmed, enabling the rapid identification of stable routes within the topology. This fast convergence reduces the transient phases during which packets might otherwise be forwarded through suboptimal paths. The behavior of Bellman–Ford relies on a different principle. Distance computation is performed through an iterative edge-relaxation process in which each node updates its routing information based on that of its neighbors. This mechanism provides a certain level of adaptability to variations in topology or link cost, but it also requires a greater number of iterations before reaching a stable state. As a result, convergence extends over a longer time interval, which may explain the slightly higher latency observed in the experiments.

The analysis of link load distribution provides a second level of interpretation. When default routing is used, a significant portion of traffic tends to concentrate on specific segments of the network, increasing the likelihood of localized congestion points. The introduction of shortest-path algorithms modifies this dynamic by exploiting the network graph structure more efficiently. Traffic flows are distributed across multiple available paths, resulting in a more balanced utilization of network links and a reduction in overload situations. This redistribution of traffic also contributes to stabilizing the overall performance of the network. These observations ultimately highlight a fundamental aspect: the convergence dynamics of routing algorithms directly influence QoS performance in SD-IoT environments, affecting transmission delays, congestion management, and the computational load imposed on the SDN controller.

5.2. Comparison with Previous Studies

The results obtained in this study align with previous research focused on evaluating routing algorithms in Software-Defined Networking (SDN) environments. Several comparative studies have investigated the performance of shortest-path algorithms in programmable network infrastructures. The work presented in Patel, et al. [26] indicates that Dijkstra can achieve higher average throughput in certain SDN environments. The observations derived from our experiments partially converge with these findings. In our experiments, Dijkstra also demonstrates more stable behavior across several Quality of Service indicators, particularly latency and packet loss rate. This trend can be attributed to the rapid convergence of its path computation mechanism, which enables the algorithm to quickly establish consistent transmission routes within the network topology.

However, some differences emerge when the results are considered within the specific context of SD-IoT environments. Previous comparative studies, such as those presented in Ghimire and Basnet [25] and Narang and Hossain [27] were mostly conducted under relatively homogeneous simulation scenarios, typically characterized by a single traffic model or a limited set of evaluation metrics. In these studies, the analysis often focuses on a specific indicator, such as algorithmic convergence, throughput, or memory usage. The approach adopted in the present study extends this analytical framework by considering a multi-application scenario, where several IoT traffic types, e-health, intelligent transportation, and industrial control, coexist simultaneously and impose distinct QoS requirements. This configuration introduces network dynamics that are more representative of real operational IoT environments.

The study most closely related to our approach is presented in Tammanashastri, et al. [28] where the authors compare Dijkstra and Bellman–Ford in a programmable SDN network. Their work highlights the efficiency of Dijkstra in relatively stable topologies and emphasizes the robustness of Bellman–Ford under network variations. The results obtained in our study partially confirm these observations while providing an additional level of analysis. The evaluation conducted in an SD-IoT environment characterized by heterogeneous flows and variable traffic loads demonstrates that algorithm performance cannot be interpreted using a single metric alone. Instead, Quality of Service results from the interaction of multiple interdependent factors, including latency, packet loss, link load distribution, and controller resource consumption. This multi-metric analysis therefore contributes to a more comprehensive understanding of routing mechanism behavior within SD-IoT architectures.

To better position our contribution with respect to previous studies, Table 4 summarizes the main characteristics of existing comparative works and highlights the specific aspects addressed in the present study.

Table 4.
Comparison with Previous Studies on Routing Algorithms in SDN/SD-IoT.

Study	Routing Algorithms	Evaluation Context	Traffic Characteristics	Evaluation Scope	Key Contribution
Ghimire and Basnet [25]	Dijkstra, Bellman-Ford, Floyd–Warshall	SDN testbed (POX controller)	Homogeneous traffic	Packet loss and network stability	Comparison of shortest-path algorithms in SDN routing environments
Patel, et al. [26]	Dijkstra, Bellman-Ford, Floyd–Warshall, A*	SDN (Ryu controller)	Generic traffic	Throughput analysis	Evaluation of shortest-path routing performance using the Ryu controller
Narang and Hossain [27]	Dijkstra, Bellman-Ford	SDN simulation	Synthetic traffic	Convergence time and scalability	Algorithmic performance comparison in large-scale SDN routing scenarios
Tammanashastri, et al. [28]	Dijkstra, Bellman-Ford	Programmable SDN networks	Generic traffic	Routing performance	Analysis of routing behavior under topology variations
This study (2026)	Dijkstra, Bellman-Ford	SD-IoT environment (Mininet + Ryu)	Heterogeneous IoT traffic (e-health, transport and industrial)	Multi-metric QoS evaluation (latency, packet loss, link load, CPU/RAM)	Comprehensive QoS-oriented evaluation of routing algorithms in SD-IoT multi-application environments

5.3. Limitations of the Study

Despite the relevance of the results obtained, several limitations should be considered when interpreting the conclusions of this study:

- Experimental environment based on emulation: The experiments were conducted in an SD-IoT environment emulated using Mininet, which enables the reproduction of a programmable network with precise control over experimental parameters. However, this approach remains a software abstraction of a real infrastructure; certain characteristics observed in large-scale IoT deployments, such as hardware-induced delays, radio interference, or unpredictable traffic variations, are not fully captured within this experimental framework.
- Limited network topology size: The experimental platform relies on an infrastructure composed of five interconnected SDN switches and three traffic generation clusters. This configuration allows the reproduction of a representative multi-application environment; however, it remains more compact than operational IoT architectures. The evaluation of algorithm scalability in much larger network topologies therefore remains only partially addressed.
- Limited number of studied algorithms: The comparative analysis focuses on two classical shortest-path algorithms, Dijkstra and Bellman–Ford. These mechanisms represent well-established references in programmable networking environments; nevertheless, other more recent approaches, particularly adaptive routing methods or machine-learning-based strategies, were not included in this study. Exploring such techniques could provide additional perspectives for the dynamic optimization of routing in SD-IoT environments.

5.4. Future Research Directions

The results obtained in this study open several research directions aimed at further investigating routing mechanisms in SD-IoT environments. A first direction concerns the analysis of routing algorithm behavior in multi-controller SDN architectures, where multiple control entities cooperate to manage distributed network infrastructures. Introducing such architectures could make it possible to evaluate how the distribution of control functions influences network performance and scalability. A second research direction involves integrating machine learning mechanisms capable of dynamically adapting routing decisions according to network conditions, traffic evolution, and the QoS requirements of applications. The exploration of large-scale IoT topologies also represents an important area of investigation in order to assess the robustness and scalability of routing mechanisms in environments characterized by a high number of nodes and simultaneous traffic flows. Finally, the development of hybrid routing models, combining shortest-path algorithms with traffic prediction or adaptive optimization mechanisms, may offer more effective solutions for improving dynamic resource management and Quality of Service in SD-IoT architectures.

References

[1] A. I. Owusu and A. Nayak, "An intelligent traffic classification in SDN-IoT: A machine learning approach," presented at the IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom) (pp. 1–6), 2020.

[2] K. L. Dias, M. A. Pongelupe, W. M. Caminhas, and L. De Errico, "An innovative approach for real-time network traffic classification," *Computer Networks*, vol. 158, pp. 143-157, 2019, https://doi.org/10.1016/j.comnet.2019.04.004.

[3] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Neural network architecture based on gradient boosting for IoT traffic prediction," *Future Generation Computer Systems*, vol. 100, pp. 656-673, 2019, https://doi.org/10.1016/j.future.2019.05.060.

[4] S. Zafar, Z. Lv, N. H. Zaydi, M. Ibrar, and X. Hu, "DSMLB: Dynamic switch-migration based load balancing for software-defined IoT network," *Computer Networks*, vol. 214, p. 109145, 2022, https://doi.org/10.1016/j.comnet.2022.109145.

[5] F. Al-Tam and N. Correia, "Fractional switch migration in multi-controller software-defined networking," *Computer Networks*, vol. 157, pp. 1-10, 2019, https://doi.org/10.1016/j.comnet.2019.04.011.

[6] T. Hu, J. Lan, J. Zhang, and W. Zhao, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 452-464, 2019, https://doi.org/10.1007/s12083-018-0632-6.

[7] S. Bera, S. Misra, and A. Jamalipour, "FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530-539, 2019, https://doi.org/10.1109/JSAC.2019.2894239.

[8] T. G. Nguyen, T. V. Phan, D. T. Hoang, H. H. Nguyen, and D. T. Le, "DeepPlace: Deep reinforcement learning for adaptive flow rule placement in software-defined IoT networks," *Computer Communications*, vol. 181, pp. 156-163, 2022, https://doi.org/10.1016/j.comcom.2021.10.006.

[9] A. Al-Jawad, A. Kucminski, R. Trestian, and P. Shah, "QoS-based routing over software-defined networks," presented at the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB) (pp. 1–6). 2017.

[10] N. Saha, S. Bera, and S. Misra, "Sway: Traffic-aware QoS routing in software-defined IoT," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 390-401, 2018, https://doi.org/10.1109/TETC.2018.2847296.

[11] G. C. Deng and K.-C. Wang, "An application-aware QoS routing algorithm for software-defined networks-based IoT networking," presented at the IEEE Symposium on Computers and Communications (ISCC) (pp. 186–191), 2018.

[12] J. Park, J. Hwang, and K. Yeom, "NSAF: An approach for ensuring application-aware routing based on network QoS of applications in SDN," *Mobile Information Systems*, vol. 2019, no. 1, p. 3971598, 2019, https://doi.org/10.1155/2019/3971598.

[13] M. Ameedeen, I. E. Kamarudin, A. Zabidi, and M. Ong, "QSroute: A QoS-aware routing scheme for software-defined networking," presented at the IEEE 8th International Conference on Software Engineering and Computer Systems (ICSECS) (pp. 388–391), 2023.

[14] M. Beshley, N. Kryvinska, H. Beshley, M. Medvetskyi, and L. Barolli, "Centralized QoS routing model for delay/loss sensitive flows at the SDN-IoT infrastructure," *Computers, Materials and Continua*, vol. 69, no. 3, pp. 3727-3748, 2021, https://doi.org/10.32604/cmc.2021.018625.

[15] A. Nazari, R. Mohammadi, N. Niknami, S. S. Jazaeri, and J. Wu, "The fuzzy-IAVOA energy-aware routing algorithm for SDN-based IoT networks," *International Journal of Sensor Networks*, vol. 42, no. 3, pp. 156-169, 2023, https://doi.org/10.1504/IJSNET.2023.132543.

[16] S. K. Keshari, V. Kansal, S. Kumar, and P. Bansal, "An intelligent energy efficient optimized approach to control the traffic flow in Software-Defined IoT networks," *Sustainable Energy Technologies and Assessments*, vol. 55, p. 102952, 2023, https://doi.org/10.1016/j.seta.2022.102952.

[17] M. S. A. Muthanna *et al.*, "Deep reinforcement learning based transmission policy enforcement and multi-hop routing in QoS aware LoRa IoT networks," *Computer Communications*, vol. 183, pp. 33-50, 2022, https://doi.org/10.1016/j.comcom.2021.11.010.

[18] A. Buzachis, A. Galletta, A. Celesti, M. Fazio, and M. Villari, "An innovative MapReduce-based approach of Dijkstra's algorithm for SDN routing in hybrid cloud, edge and IoT scenarios," in *Proceedings of the European Conference on Service-Oriented Cloud Computing (pp. 185–198). Springer*, 2018.

[19] M. Fazio *et al.*, "A map-reduce approach for the Dijkstra algorithm in SDN over osmotic computing systems," *International Journal of Parallel Programming*, vol. 49, no. 3, pp. 347-375, 2021, https://doi.org/10.1007/s10766-021-00693-3.

[20] A. Choudhary, S. S. Kang, and S. Singla, "Multi-objective Dijkstra algorithm for enhancing QoS in SDN through balanced routing," presented at the International Conference on Circuit Power and Computing Technologies (ICCPCT) (pp. 1169–1175), 2024.

[21] Y. Zhang and M. Chen, "Performance evaluation of software-defined network (SDN) controllers using Dijkstra's algorithm," *Wireless Networks*, vol. 28, no. 8, pp. 3787-3800, 2022, https://doi.org/10.1007/s11276-022-03044-3.

[22] M. A. Zormati, H. Lakhlef, and S. Ouni, "Routing optimization based on distributed intelligent network softwarization for the internet of things," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (pp. 1757–1764)*, 2024.

[23] J. A. Da Silva, P. Mendes, and E. P. de Freitas, "Resilient software defined satellite networks: Combining clustering techniques with efficient routing protocols," presented at the International Conference on Software Defined Systems (SDS) (pp. 178–183), 2024.

[24] S. Mehraban and R. K. Yadav, "Routing optimization in hybrid software-defined networks: A heuristic approach," *IETE Journal of Research*, vol. 71, no. 9, pp. 2944-2962, 2025, https://doi.org/10.1080/03772063.2025.2506011.

[25] R. Ghimire and R. K. Basnet, "Shortest path routing performance evaluation over SDN environment," *Journal of Electronics and Informatics*, vol. 5, no. 4, pp. 405-422, 2024.

[26] K. Patel, J. Chaudhari, H. Mewada, H. Jayswal, R. Patel, and D. Kirange, "Shortest path forwarding in software-defined networks using RYU controller," *International Journal of Electrical and Electronics Engineering*, vol. 11, pp. 299-305, 2024.

[27] M. Narang and Z. Hossain, "Comparative analysis of shortest-path algorithms in network routing," presented at the International Conference on Computer and Communication Systems (ICCCS) (pp. 803–808), 2025.

[28] P. R. Tammanashastri, S. M. Rajagopal, and S. S. Bananjee, "Comparative analysis of Bellman-Ford and Dijkstra algorithms in software defined networking," presented at the International Conference on Data Intelligence and Cognitive Informatics (ICDICI) (pp. 1-8), 2024.