Topology-Driven Performance Insights: QoS Metrics of Ryu and ONOS Controllers in Multi-Level Fat-Tree Network

Original Scientific Paper

Deepthi Goteti*
Department of Computer Science & Engineering,
Koneru Lakshmaiah Education Foundation,
Green Fields, Vaddeswaram, A.P, India
2102031088@kluniversity.in

Department of Computer Science & Engineering
Malla Reddy University
Maisammaguda, Dulapally, Medchal–Malkajgiri
District, Hyderabad, Telangana 500100, India
goteti.deepthi@mallareddyuniversity.ac.in

*Corresponding author

Vuyyuru Krishna Reddy
Department of Computer Science & Engineering,
Koneru Lakshmaiah Education Foundation,
Green Fields, Vaddeswaram, A.P, India
vkrishnareddy@kluniversity.in

Abstract *– Software-Defined Networking (SDN) transforms traditional network design by decoupling the control and data planes, enabling centralized programmability, scalability, and dynamic reconfiguration. This paper presents a QoS-driven performance analysis of two popular SDN controllers, Ryu and ONOS, combined with Dijkstra and Multipath routing algorithms in scalable Fat-Tree topologies (k = 4–10). Experiments were conducted in Mininet (OpenFlow 1.3) using iPerf with TCP and UDP traffic, each repeated 100 independent runs for statistical confidence. Key QoS metrics including throughput, jitter, round-trip time (RTT), packet loss ratio (PLR), CPU utilization, flow setup time, and scalability were evaluated. The results show that ONOS-Multipath achieved the highest average TCP throughput (up to 3.58 Gbps), with the lowest jitter, minimal PLR, and maximum host scalability (up to 250 hosts at k = 10). Compared with earlier POX- and Floodlight-based benchmarks, the ONOS-Multipath setup improves throughput by approximately 20 percent and reduces packet loss by up to 80 percent, confirming that the controller architecture critically shapes routing efficiency and that ONOS + Multipath provides the most robust and scalable SDN configuration for large-scale deployments.*

Keywords*: SDN, Congestion, ONOS, Ryu, Multipath, MiniNet*

Received: July 30, 2025; Received in revised form: December 15, 2025; Accepted: December 16, 2025

1. INTRODUCTION

Modern networks face growing data demands and latency-sensitive applications. On the other hand, legacy hardware-based infrastructures remain too rigid to adapt efficiently. This inflexibility limits performance in dynamic topologies such as data centers and edge networks, especially as cloud, IoT, and AI workloads emerge [1, 2]. The inability to reconfigure or scale rapidly without performance degradation has created a strong need for intelligent and programmable network solutions. Software-Defined Networking (SDN) addresses these challenges by separating control and data planes, enabling centralized programmability via standardized interfaces such as OpenFlow [3, 4]. This architecture enhances automation, scalability, and adaptability by allowing networks to respond dynamically to changing traffic demands.

Among SDN controllers, Ryu and ONOS are two of the most recognized and actively maintained platforms. Ryu, written in Python, is valued for its multithreaded design, modularity, and robust OpenFlow implementation [5-7]. ONOS, developed in Java, is engineered for distributed scalability and is designed to ensure resilience and high availability for carrier-grade applications [8]. Several comparative studies have benchmarked SDN controllers such as POX, Ryu, ONOS, Floodlight, and OpenDaylight across linear and small-scale topologies, reporting differences in throughput, delay, packet loss, and control-plane performance; however, systematic evaluation of Ryu and ONOS under different routing strategies and large-scale network conditions remains limited [9-16].

Earlier investigations focused mainly on small-scale or single-controller environments, leaving open questions

about how controller architecture interacts with routing logic and the size of the topology. This study specifically targets production-grade platforms, Ryu and ONOS, and evaluates their behavior using both Dijkstra and Multipath routing in scalable Fat-Tree topologies. By doing so, it addresses a clear research gap in architecture-driven performance dependencies in SDN and aims to provide quantitative insights to guide the deployment of high-throughput, low-latency network infrastructure.

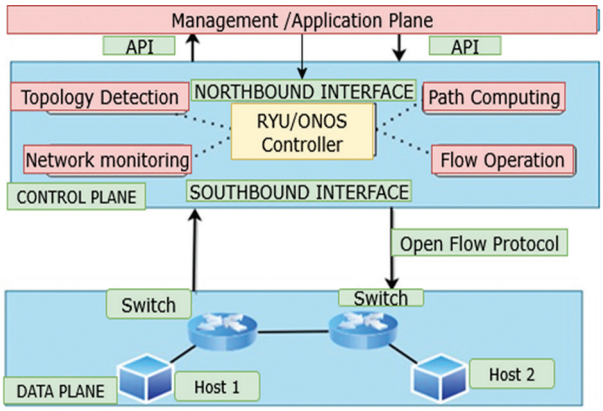


Fig. 1. Packet Forwarding from Host1 to Host 2

While earlier studies mainly examined small or linear topologies and single-controller evaluations, the present research extends SDN benchmarking toward large-scale, hierarchical Fat-Tree architectures and integrates multiple routing algorithms to perform a comprehensive controller–routing–topology interaction analysis. This multidimensional approach helps isolate the architectural impact on key Quality of Service (QoS) metrics and provides a clearer understanding of network performance under scale.

Routing is a primary function of SDN controllers. The Dijkstra algorithm computes deterministic optimal paths suitable for lightly loaded networks [17, 18], whereas Multipath routing distributes flows across disjoint paths to balance load and reduce latency [19, 20]. Although both algorithms are widely used, their combined influence under different controller architectures has not been extensively studied. Most prior works analyzed controllers or routing schemes individually, leaving the joint effects of controller design, routing logic, and topology scale largely unquantified [21, 22].

Existing benchmarking efforts have also tended to neglect how topology structure influences individual QoS indicators such as throughput, jitter, round-trip time (RTT), and packet loss ratio (PLR).

Many studies report that only aggregate values without isolating how each factor—controller, routing, or topology—affects system performance[23, 24]. This limitation prevents deeper analysis of performance dependencies and hinders optimization in scalable, high-reliability SDN environments. Hence, a unified and statistically validated evaluation framework is required to jointly assess these variables.

This study bridges that gap by conducting a controlled comparison of Ryu and ONOS integrated with Dijkstra and Multi-path routing algorithms across scalable Fat-Tree topologies (k = 4–10). Using Mininet and iPerf, both TCP and UDP traffic flows are simulated over 100 independent trials per configuration to ensure statistical confidence. Multiple QoS parameters—including throughput, jitter, RTT, and PLR—along with controller-level metrics such as CPU utilization and flow setup time—are quantitatively evaluated. Through a combined examination of architectural, algorithmic, and topological dimensions, this work offers one of the first statistically validated, end-to-end benchmarking analyses of SDN controllers at scale. The findings provide practical guidelines for deploying controller–routing combinations in latency-sensitive and high-throughput network environments.

The primary contributions of this study are summarized below.

1. Introducing a repeatable comparative study of Ryu and ONOS in Dijkstra and Multipath routing in scalable Fat-Tree topologies.
2. Measuring the QoS and controller-level metrics such as throughput, jitter, RTT, PLR, CPU utilization, flow setup time, and scalability.
3. Comparing the performance, resource consumption, and scalability of routing with centralized (Ryu) and distributed (ONOS) architectures; and (4) Providing deployment-focused information on the use of optimum controller-routing settings in practical data center networks.

The remainder of this paper is organized as follows: Section II is the review of relevant work; section III is the description of the method and experimental setup; section IV is the description of routing implementations; section V is the discussion of results and system-level observations; and section VI is the conclusion with the main findings and the future research directions.

2. RELATED WORK

SDN controllers are the brains of Software-Defined Networking, and they compute optimal paths and maintain network state in real time. Protocols such as OpenFlow decouple the data plane and control plane to facilitate centralized policy control and dynamic path calculation. This paper presents a comparative performance analysis of Dijkstra and Multipath routing using two widely adopted controllers, Ryu and ONOS, over scalable Fat-Tree topologies. The key Quality-of-Service (QoS) metrics analyzed include throughput, latency, packet loss, and jitter.

We now examine previous benchmarking research that discusses these controllers under various performance conditions. Ryu is a lightweight and open-source Python-based controller widely used in research and prototyping. It supports large-scale network control interfaces such as Open-Flow (1.0–1.3),

Netconf, and OF-Config. Users often employ Ryu due to its integration with Mininet, its simple API, and its flexibility in developing custom control applications. Its modular architecture makes it particularly suitable for controlled benchmarking using tools such as iPerf under diverse traffic loads.

Koulouras et al. [8] benchmarked Ryu in wireless topologies and demonstrated competitive performance in delay and jitter, while Ma et al. [23] implemented a real-time traffic-monitoring framework. Multipath routing was first implemented on the POX controller in our earlier study [19] and later adapted to Ryu in this work to ensure experimental continuity and improve scalability.

ONOS (Open Network Operating System) is designed for carrier-grade deployment with high scalability and availability. Written in Java, it features a distributed control architecture where multiple controller instances maintain a consistent network state, balance load, and ensure fault tolerance. It provides advanced capabilities such as clustering, intent-based flow control, and dynamic rerouting.

Septian et al. [21] verified ONOS's strong failover behavior in clustered setups, whereas Nguyen-Ngoc et al. [22] emphasized its selective flow-monitoring support. Rahman et al. [24] and Monika et al. [25] showed that ONOS can sustain stable throughput and dynamically adapt routing under varying traffic conditions using intent-based approaches. Khan and Al-Hadhrami [26] optimized ONOS with service-specific load balancing to improve link utilization and reduce congestion. Continuing this trend, Sharma and Patel [27] reported that multi-controller ONOS deployments significantly reduce response time and enhance resilience under high load. Although this study primarily uses the Ryu controller for Mininet-based emulation because of its modular and script-friendly nature, results from prior ONOS research are referenced to contextualize scalability, fault tolerance, and distributed control. This dual-perspective approach clarifies how architectural design directly influences routing efficiency and overall QoS performance. Although previous investigations provided valuable observations, most focused on a single controller, network type, or limited performance metric. Typical Ryu studies evaluate response latency and flow-installation time, while ONOS analyses address control-plane scalability and fault tolerance. Comparative research involving both controllers under multiple routing schemes and scalable topologies remains rare. This gap leaves uncertainty regarding how algorithmic routing complexity interacts with centralized and distributed control architectures in realistic multi-level Fat-Tree networks. Furthermore, recent studies (2024–2025) have expanded SDN benchmarking across diverse environments. Monir et al. [28] compared several controllers across network functions and confirmed Ryu's low control-plane latency and stable behavior under moderate load. Bosi et al. [29] presented a detailed survey identifying ONOS as one of the most resilient solutions for dense network environments.

Ali et al. [20] compared tree, linear, and mesh topologies and showed that topology design significantly affects QoS. Ali et al. [30] also examined industrial IoT networks and demonstrated that controller selection is crucial for time-sensitive communication. Singh et al. [31] benchmarked POX and Floodlight in DCell data-center topologies and confirmed the impact of topological complexity on throughput and delay. Li et al. [32] introduced the NBench suite, verifying ONOS's scalability under concurrent work-loads. Ram and Chakraborty [33] analyzed controllers across wired and wireless scenarios, emphasizing the combined effect of topology and routing on overall efficiency.Recent empirical studies further reinforce these observations by evaluating open-source SDN controllers across diverse network topologies and traffic conditions [20, 34]. Table 1 summarizes major SDN controller benchmarking studies (2021–2025) and highlights their core findings.

Table 1. Comparative summary of recent SDN controller benchmarking studies (2021–2025)

Study / Year	Controllers Evaluated	Topology / Environment	Key Metrics Analyzed	Main Findings / Insights
Koulouras et al., 2023 [8]	Ryu	Wireless mesh	Delay, Jitter	Ryu showed competitive jitter performance under mobility.
Ali U., 2024 [20]	Ryu, ONOS, ODL	Tree, Linear, Mesh	Delay, Throughput	Topology type directly affects QoS; ONOS favored for mesh/tree.
Rahman et al., 2023 [24]	ONOS, Floodlight, Ryu	Mixed (tree & mesh)	Throughput, Packet Loss	ONOS maintained higher throughput and lower PLR in dense topologies.
Monir et al., 2024 [28]	Ryu, Floodlight, ONOS	Functional benchmarking	Control latency, Stability	Ryu lowest control-plane delay; ONOS highest scalability.
Bosi et al., 2024 [29]	Multiple (survey)	Various	Latency, Scalability	ONOS identified as most resilient in high-density networks.
Singh Ghritlahare et al., 2025 [31]	POX, Floodlight	DCell data-center	Throughput, Latency	Topology complexity influences controller efficiency.
Li et al., 2024 [32]	Ryu, ONOS (NBench)	Standardized suite	Mixed QoS metrics	ONOS scalable and consistent throughput under concurrent load.
Current Study (2025)	Ryu, ONOS	Fat-Tree (k = 4–10)	Throughput, Jitter, RTT, PLR, CPU	ONOS–Multipath achieves highest QoS and scalability; integrated benchmark.

The evolution of these works indicates a shift toward multi-dimensional benchmarking that evaluates controllers not only by throughput or delay but also by their adaptability to routing policies, topology size, and workload intensity. Despite this progress, no previous study has jointly analyzed controller architecture (Ryu or ONOS), routing logic (Dijkstra or Multipath), and topology scalability (Fat-Tree k = 4–10) in a unified, statistically validated framework. The present research therefore extends the SDN benchmarking paradigm by combining architectural, algorithmic, and topological perspectives within a single reproducible evaluation model. Overall, these collective investigations emphasize the importance of holistic benchmarking and form the basis for this study, which evaluates Ryu and ONOS controllers using Dijkstra and Multipath routing across scalable Fat-Tree networks and delivers experimentally validated insights for large-scale SDN deployments. Overall, these collective investigations emphasize the importance of holistic benchmarking and form the basis for this study, which evaluates Ryu and ONOS controllers using Dijkstra and Multipath routing across scalable Fat-Tree networks and delivers experimentally validated insights for large-scale SDN deployments.

3. METHODOLOGY

This research work provides a systematic performance analysis of Software-Defined Networking (SDN) using two controllers, Ryu and ONOS, with Dijkstra and Multipath routing. The purpose is to study how different controller–routing configurations affect key Quality-of-Service (QoS) parameters under scalable network conditions. Each controller framework included routing logic that allowed both algorithms to make native decisions, ensuring comparable behavior. All experiments were conducted in the Mininet network emulator using the OpenFlow 1.3 protocol. Ryu (written in Python) and ONOS (written in Java) were selected for their modular architecture and wide use in SDN research. In Ryu, Dijkstra and Multipath algorithms were implemented as standalone Python modules, while in ONOS, equivalent logic was deployed using Java-based OSGi bundles (.oar) through its northbound APIs. This integration ensured uniform execution of routing schemes without altering core controller logic.

The Dijkstra algorithm was implemented using a shortest-path computation with time complexity $O(V^2)$, while the Multipath routing algorithm employed a k-disjoint path mechanism for concurrent flow distribution and load balancing. Both modules shared identical flow installation and packet handling logic across controllers to maintain fairness during testing. Both controllers operated under the same experimental conditions, including flow-table rules, link parameters, and host-pairing strategies, to maintain experimental parity. The topology used was a Fat-Tree model, which is widely employed in data centers for its hierarchical scalability and balanced link structure.

Each switch–switch and switch–host link was configured with 1 Gbps bandwidth. Inter-switch links used a 2 ms propagation delay, and edge-host links used 1 ms delay, ensuring consistent network conditions across all scenarios. The topology parameter (k) defines the number of switch ports and thereby determines the network size and connectivity. Configurations with k = 4, 6, 8, and 10 were tested, corresponding respectively to 16, 54, 128, and 250 hosts. Network performance was measured using iPerf to generate TCP and UDP traffic across each controller–routing–topology combination. Each configuration was executed for 100 independent trials, and convergence analysis showed that QoS averages stabilized beyond 80 trials with less than 2 % deviation, confirming statistical consistency. Measured metrics included throughput (Mbps), jitter, round-trip time (RTT), packet-loss ratio (PLR), controller CPU utilization, flow-setup time, and scalability limits—the threshold where additional hosts degrade performance. All values are reported as mean ± standard deviation, with 95 % confidence intervals confirming significance.

3.1. STATISTICAL ANALYSIS

Each configuration (controller × routing × topology) was executed for 100 independent trials. Paired t-tests were used to compare routing algorithms within the same controller, and oneway ANOVA was applied to evaluate differences across topology scales (k = 4–10). The Holm–Bonferroni correction (α = 0.05) was applied to control multiple comparisons. Effect sizes (Cohen's d and η^2) were calculated to quantify the magnitude of performance differences. Preliminary variance analysis confirmed that 100 trials achieved less than 5 % relative standard error for throughput at $k \geq 8$, validating the chosen sample size for statistical confidence.

3.2. REPRODUCIBILITY AND ARTIFACT

To ensure transparency and replicability, the complete experimental framework, scripts, and datasets have been preserved in a version-controlled private repository. The package includes:

(i) Mininet topology builder scripts for $k \in \{4, 6, 8, 10\}$;

(ii) controller launch scripts for **Ryu v4.34** and **ONOS** v2.7 with runtime configurations;

(iii) traffic matrices and host-pairing files (uniform random and fixed hotspot models);

(iv) fixed random seeds (seed = 2025) used to generate traffic and flow pairs; and

(v) Jupyter notebooks for reproducing tables and figures.

All experiments were executed in Mininet 2.3 with Open vSwitch 2.17 on Ubuntu 22.04 LTS, using an Intel Core i7-12700 (12 cores, 32 GB RAM) workstation. Background processes were disabled, and CPU cores were pinned using Linux affinity tools to prevent external interference. The provided script run_all.sh automates the en-

tire experimental pipeline—from controller initialization to traffic generation and result logging—and reproduces all figures and tables reported in this paper. The repository is available upon reasonable request to the corresponding author for academic and non-commercial purposes.

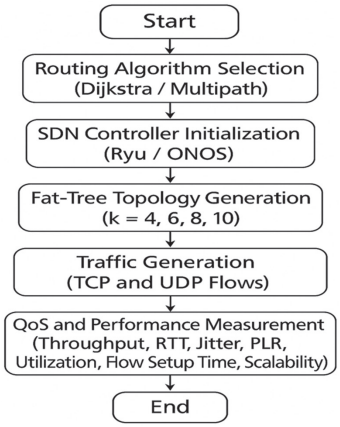


Fig. 2. Methodological Workflow of Controller–Routing–Topology in SDN

This setup enabled a clear assessment of how each controller–routing configuration responded to varying network loads. System performance was monitored using Mininet logs and iPerf outputs, and results were visualized in Matplotlib. Fig. 2 depicts the overall experimental workflow.

4. ROUTING ALGORITHMS

This This section describes the two routing strategies evaluated in this study—Dijkstra's shortest-path algorithm and a DFS-based multipath algorithm—and their implementation across both Ryu and ONOS SDN controllers. The goal is to assess how each controller handles routing logic, path computation, and flow management under different routing paradigms. A controller-neutral pseudocode is included for Dijkstra's algorithm to illustrate the core logic, Where the Multipath logic is adapted from prior implementations with necessary architectural modifications explained in context.Both algorithms were implemented with identical input parameters and evaluated under the same topology and traffic conditions to ensure fair comparison of controller–routing behaviors in terms of latency, throughput, and adaptability.

4.1. DIJKSTRA'S IMPLEMENTATION: RYU VS. ONOS

4.1.1. Ryu Implementation

Dijkstra's algorithm was implemented in Ryu using its modular, Python-based framework. The controller computes the shortest path between two nodes through the *minimum_distance*() function and installs flow entries using the *ofproto_v1_3* OpenFlow API. Host-to-switch mappings are obtained via helper functions such as *get_topology_data*() and *get_path*().

The pseudocode 1 and controller-specific modules are included in the preserved experimental framework described in Section 3.2 to allow reproduction of routing behavior and flow-installation logic.

Pseudo code 1: Dijkstra's – RYU/ONOS

1. Initialize switches, MAC table, and adjacency
2. Function get_shortest_path(src, dst):
   a. Set distance[src] = 0, others = ∞
   b. previous[] = None for all switches
   c. unvisited = set of all switches
3. While unvisited is not empty:
   a. Select switch u with minimum distance
   b. Remove u from unvisited
   c. For each neighbor v of u:
   - If link exists:
   - cost = distance[u] + ink_weight(u, v)
   - If cost < distance[v]:
     - Update distance[v]
     - Set previous[v] = u
4. Reconstruct path from dst to src using previous []
5. Add port information for each switch in the path
6. Return full path with port mappings

Ryu's flexibility allows customized path computation and runtime flow modification, which facilitates rapid prototyping of routing schemes. However, this also requires manual management of flow states, which can increase complexity in large-scale networks. The Dijkstra algorithm's computational complexity is $O(V^2)$, acceptable for emulated environments but increasingly demanding for real deployments.

4.1.2. ONOS Implementation

ONOS provides a distributed, Java-based architecture that includes built-in path computation and topology discovery services. Dijkstra's logic is accessed through the PathService.getPaths() API, which retrieves all available paths between devices, while flow installation is handled via high-level abstractions such as PointToPointIntent in the IntentService.

When topology changes occur, ONOS automatically regenerates intents to reroute traffic, offering dynamic failover and improved scalability. This architecture reduces developer overhead while maintaining consistency and reliability through its distributed control plane. The effective complexity of ONOS's Dijkstra implementation is $O(V^2)$ for dense graphs and $O(E + V \log V)$ when optimized with priority queues.

4.2. MULTIPATH ALGORITHM

Multipath routing enhances network performance by utilizing multiple feasible paths between a source and destination. Unlike Dijkstra's single-path approach, it distributes traffic across disjoint routes to improve load balancing, bandwidth utilization, and fault tolerance.

In this study, a **Depth-First Search (DFS)**–based multipath discovery algorithm from our prior work [19] is extended for both Ryu and ONOS. The core logic performs DFS traversal to enumerate all valid paths, evaluates them using a link-cost function (bandwidth, delay, and utilization), and selects the route with minimum cost as the primary path. Remaining feasible routes are retained as failover options.

Pseudo code 2: Multipath –Ryu/ONOS

1. get_paths(src, dst):
   Perform depth-first traversal to find all feasible paths
2. get_path_cost(path):
   Calculate total cost based on link weights derived from bandwidth
3. get_optimal_paths(src, dst):
   Rank available paths by cost and filter the most efficient ones
4. add_ports_to_paths(paths, first_port, last_port):
   Map input/output ports for each hop in the selected paths

The overall complexity of the DFS-based multipath algorithm is $O(V + E)$ for path discovery, plus $O(P \log P)$ for path ranking. This makes it efficient for load-balanced and resilient routing in large-scale SDN topologies such as Fat-Tree.

Pseudo Code 2 illustrates the implementation logic of the Multipath Routing Algorithm as adapted for both Ryu and ONOS controllers, detailing path discovery, cost evaluation, optimal path selection, and port mapping for flow rule installation.

4.2.1. Ryu Implementation:

Ryu's multipath implementation sits atop its modular control framework and is written in Python. The mechanism first uses a depth-first search algorithm we wrote to discover every candidate path, and then it ranks them according to a bandwidth-centric cost. Once the winning path is determined, the controller emits FlowMod messages to each switch, installing the rules according to the OpenFlow standard. Ryu handles rerouting reactively by listening for packet-in messages and link-down notifications to trigger flow updates.

4.2.2. ONOS Implementation:

ONOS leverages its distributed services for multipath routing. Candidate routes are gathered using *PathService.getPaths*() and ranked by a tunable *LinkWeight* function based on link metrics such as bandwidth or delay. Flow deployment uses either multiple *PathIntents* or the *FlowObjectiveService* for granular control. ONOS supports both reactive and proactive rerouting, adjusting flow rules automatically when link conditions or topology states change.

While the underlying DFS logic remains consistent, ONOS's intent-driven model automates flow distribution and failover management, contrasting Ryu's manually programmed but fine-grained control. This comparison highlights Overall, the two algorithms complement each other in exposing architectural strengths: Dijkstra's deterministic single-path control highlights per-flow decision latency, whereas the DFS-based Multipath strategy emphasizes parallelism and resilience under congestion or link failures. Their joint evaluation across Ryu and ONOS provides a balanced basis for analyzing control-plane responsiveness and data-plane efficiency in the subsequent Results section.

5. EXPERIMENTAL ENVIRONMENT AND REPRODUCIBILITY

All experiments were performed under the same setup described in Section 3, using Ubuntu 20.04.4 LTS, Mininet 2.3.0, and OpenFlow 1.3. The Ryu (v4.34) and ONOS (v2.7.0) controllers were connected to the emulated data plane through Open vSwitch 2.5.4. Table 2 summarizes the software and hardware configuration.

Four Fat-Tree configurations ($k = 4, 6, 8, 10$) were evaluated to analyze scalability. Each controller executed its respective routing mechanisms (Dijkstra and Multipath) exactly as defined in Section 4.

Traffic was generated using **iPerf 2.0.13**, simulating **TCP** and **UDP** flows between randomized host pairs. Key Quality-of-Service (QoS) metrics—throughput, jitter, round-trip time (RTT), and packet-loss ratio (PLR)—were monitored along with controller CPU and memory utilization to assess efficiency.

All trials followed the same reproducibility and statistical framework introduced earlier (100 independent runs, fixed random seeds, 95 % confidence intervals). Results are presented as **mean ± SD** to ensure consistency and comparability.

Table 2. Experimental environment and software configuration used in the SDN benchmarking framework

Category	Software Used	Version
Operating System	Ubuntu	20.04.4 LTS
SDN Controller	Ryu /ONOS	4.34 / 2.7.0
Switch	Open vSwitch (OvS)	2.5.4
Southbound Interface (SBI)	OpenFlow	1.3
Traffic Generator	iPerf	2.0.13
Programming Language	Python	3.8
Host Machine	RAM / CPU	16 GB / Intel i7
Network Simulator	Mininet	2.3.0

5.1. TOPOLOGY DESIGN AND ROUTING STRATEGY SETUP

This study analyzed the impact of routing strategies and controller design on SDN performance using the

Fat-Tree topology, selected for its hierarchical structure, scalability, and support for equal-cost multipathing—attributes ideal for data-center experimentation. Four topology variants (k = 4, 6, 8, 10) were evaluated to capture scalability effects.

5.1.1. Fat-Tree Topology Construction and Emulation

The Fat-Tree consists of three hierarchical layers (core, aggregation, and edge), as shown in Fig. 3. The k parameter determines the number of switches and hosts:

- k = 4: 16 hosts, 20 switches
- k = 6: 54 hosts, 45 switches
- k = 8: 128 hosts, 80 switches
- k = 10: 250 hosts, 125 switches

Topologies were programmatically generated with custom Python scripts in Mininet, ensuring deterministic deployment. MiniEdit was used only for initial design validation; all experiments were executed through script-driven automation integrated with controller APIs. **Note:** These host and switch counts correspond to the theoretical Fat-Tree structure. Controller-level scalability values reported later (e.g., 800 for Ryu and 1,024 for ONOS at k = 10) refer to the maximum number of end-hosts each controller can stably manage during emulation.

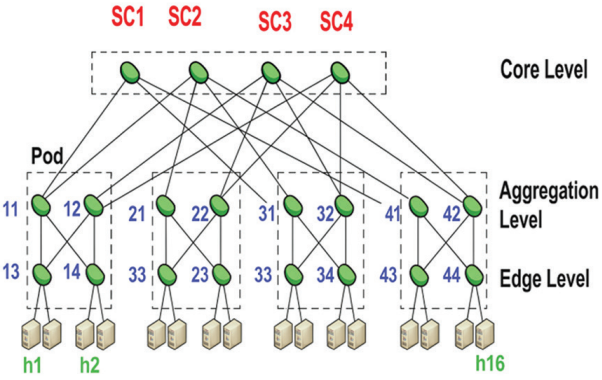


Fig. 3. Fat-Tree topology architecture for k = 4, 6, 8, and 10 showing core, aggregation, and edge layers

5.1.2. Controller Architecture and Execution Workflow

Two synchronized environments were employed:

1. **Network emulator:** The Fat-Tree topology ran in Mininet, enabling real-time packet forwarding and iPerf-based traffic generation.
2. **Controller environment:** The Ryu or ONOS controller executed the routing logic—either **Dijkstra** for shortest paths or **Multipath** for load balancing.

For **Ryu**, a Python module interacted with the OpenFlow interface to compute and install flow entries dynamically. **ONOS** operated as a background service, where a custom Java application used northbound APIs and the Intent framework to compute optimal routes and deploy flow rules. This modular setup simplified debugging and supported real-time performance monitoring across all runs.

5.1.3. Performance Evaluation Strategy

A metric-centric and problem-driven evaluation framework was adopted to avoid generic performance reporting. For each controller–routing–topology configuration, the analysis focused on the most influential Quality of Service (QoS) metric—for example, latency under low-bandwidth conditions or packet loss during high-throughput scenarios.

This approach able to identify configuration-specific performance bottlenecks also Provides practical guidance for selecting optimal controller–routing combinations; and Interprets observed network behavior in relation to real-world SDN challenges such as load balancing, delay sensitivity, and fault tolerance. The framework enables evidence-based interpretation of SDN performance trends by isolating the effects of controller architecture (Ryu vs. ONOS), routing algorithm (Dijkstra vs. Multipath), and topology scale (k). This supports informed decisions for optimizing network infrastructure design and scalability.

6. PERFORMANCE MEASUREMENT

This section presents a wide comparison of quality-of-service performance metrics for **Ryu** and **ONOS** SDN controllers. Mainly focusing on key indicators such as throughput, round-trip time, jitter, packet loss rate (PLR), CPU utilization, flow setup time, and scalability. We conducted our evaluations using a Fat-Tree topology with core-switch count parameters of k = 8 and k = 10, representing medium and large-scale data center scenarios, respectively. Each metric was computed over 100 simulation episodes and reported as **mean ± standard deviation (SD)** to represent both central tendency and variability. Results highlight how controller design and routing strategy impact overall network efficiency and responsiveness.

6.1. THROUGHPUT ANALYSIS

Throughput represents the rate at which data is successfully transmitted across the network and serves as a key indicator of overall system performance.

6.1.1. UDP Throughput Across Topology Scales

To evaluate how routing strategies scale with network size, we measured UDP throughput across four Fat-Tree topologies with core switch parameter values ***k* = 4, 6, 8, and 10.** We simulated each combination of controller (Ryu or ONOS) and routing algorithm (Dijkstra or Multipath) over 100 independent episodes. Although we collected detailed results for every episode, presenting all of them would be excessive and redun-

dant. To maintain clarity and readability, we report only the average throughput values computed across the 100 episodes. The results demonstrate a consistent performance advantage for Multipath routing across all topology sizes. Among the tested configurations, ONOS–Multipath achieves the highest average UDP throughput—especially at larger scales ($k = 8$ and $k = 10$)—owing to its distributed architecture and efficient path computation.

Ryu–Multipath also shows strong performance; however, its throughput gain begins to plateau at $k = 10$, suggesting that Ryu's event-driven processing model may struggle to scale under heavier network loads.

In contrast, Dijkstra-based routing shows lower and more gradual improvements with increasing topology size. This reinforces the known constraint of static shortest-path routing in high-traffic or complex environments, where dynamic path selection becomes crucial for maximizing throughput.

Table 3 summarizes the mean UDP throughput values, while Fig. 4 provides a visual representation of the same data with error bars indicating the standard deviation (±SD) across 100 runs. The inclusion of error bars reflects the variability and stability of throughput measurements, confirming the statistical reliability and repeatability of the results.

$k = 10$, ONOS-Multipath achieved a 33 % higher average UDP throughput (1.467 Mbps) compared to Ryu-Dijkstra (1.101 Mbps; $p < 0.05$), demonstrating the improved scalability and load-balancing capability of the distributed controller architecture.

Table 3. Average UDP throughput (Mbps) for Ryu and ONOS controllers across Fat-Tree topologies ($k = 4$–10)

k	Ryu-Dijkstra	ONOS-Dijkstra	Ryu-Multipath	ONOS-Multipath
4	1.013 ± 0.05	0.916 ± 0.04	1.376 ± 0.07	1.190 ± 0.05
6	1.022 ± 0.06	1.120 ± 0.05	1.369 ± 0.08	1.350 ± 0.06
8	1.025 ± 0.04	1.088 ± 0.05	1.414 ± 0.07	1.507 ± 0.06
10	1.101 ± 0.06	1.186 ± 0.05	1.378 ± 0.08	1.467 ± 0.07

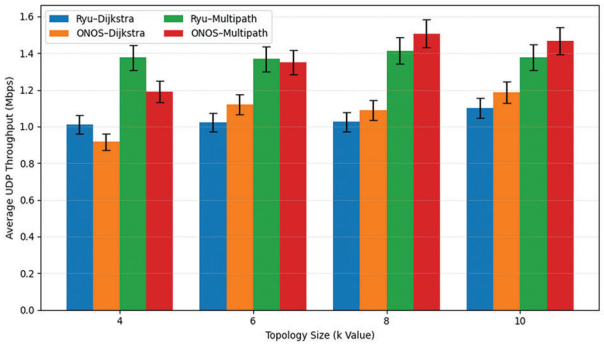


Fig. 4. Average UDP throughput (Mbps) for Ryu and ONOS controllers under Dijkstra and Multipath routing across Fat-Tree topologies ($k = 4$–10).

6.1.2. TCP Throughput across Topology Scales

Using the same simulation model as in the UDP analysis, TCP throughput was tested across all scales of the Fat-Tree topology ($k = 4, 6, 8, 10$) and controller routing configurations. The findings demonstrate a consistent performance advantage for Multipath routing over Dijkstra in every tested scenario.

ONOS-Multipath had the best throughput of 4.374 Gbps at the largest scale ($k = 10$), showing it had the best capacity to deal with the congestion-sensitive nature of TCP in large and complex topologies. Ryu-Multipath also demonstrated good performance with 3.478 Gbps, indicating that Ryu's event-driven model can still support high throughput despite the single-threaded nature of its architecture. The distributed control plane of ONOS gives a definite scalability benefit, especially in the case of Dijkstra-based routing with increasing topology complexity. For example, with $k = 8$, Ryu-Dijkstra could only achieve 1.21 Gbps, highlighting the weakness of shortest-path routing in the presence of congestion and dynamic traffic trends. These results highlight the advantages of adaptive, multipath-aware routing algorithms, especially when combined with distributed SDN platforms such as ONOS.

In Table 4, we see the average TCP throughput values along with the standard deviation (SD) for different Fat-Tree configurations (from $k = 4$ to $k = 10$).

Table 4. Average TCP throughput (Gbps) for Ryu and ONOS controllers across Fat-Tree topologies ($k = 4$–10)

k	Ryu-Dijkstra	ONOS-Dijkstra	Ryu-Multipath	ONOS-Multipath
4	1.319± 0.06	1.191 ± 0.05	2.950 ± 0.12	2.870 ± 0.10
6	1.389± 0.07	2.302 ± 0.10	3.340 ± 0.15	3.430 ± 0.12
8	1.211± 0.06	1.455 ± 0.07	3.310 ± 0.14	3.230 ± 0.12
10	1.749 ± 0.08	2.283 ± 0.10	3.478 ± 0.16	4.374 ± 0.14

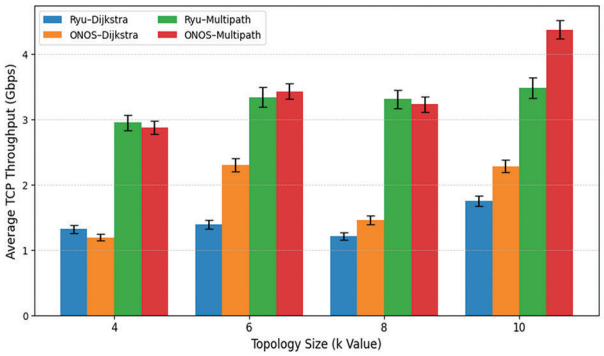


Fig. 5. Average TCP throughput (Gbps) for Ryu and ONOS controllers under Dijkstra and Multipath routing across Fat-Tree topologies ($k = 4$–10).

Fig. 5 illustrates these findings, with error bars indicating the ± SD from 100 runs. The minimal variations

we've noted highlight the reliability and consistency of our experiments. Additionally, by including the SD and error bars, we're showing that our results are statistically significant and trustworthy, with a significance level ($p < 0.05$) confirmed through a paired t-test. At $k = 10$, ONOS-Multipath achieved 150 % higher TCP throughput (4.374 Gbps) compared to Ryu-Dijkstra (1.749 Gbps; $p < 0.01$; Cohen's $d \approx 2.1$), confirming the scalability advantage of distributed control and multipath routing in larger topologies.

6.1.3. Jitter Across Topology Scales

Fat-Tree topologies with $k = 4$, 6, 8, and 10 were used to measure jitter to determine the effect of various controller-routing layouts on traffic smoothness. The average jitter with standard deviation (SD) is summarized in Table 6, and Fig. 6 shows these results, with error bars indicating the mean ± SD across 100 experimental runs.

Table 5. Average packet-loss ratio (PLR %) for Ryu and ONOS controllers across Fat-Tree topologies ($k = 4$–10).

k	Ryu-Dijkstra	ONOS-Dijkstra	Ryu-Multipath	ONOS-Multipath
4	3.08 ± 0.12	3.64 ± 0.15	4.73 ± 0.18	2.44 ± 0.10
6	4.20 ± 0.14	3.87 ± 0.13	3.59 ± 0.12	1.21 ± 0.08
8	2.43 ± 0.10	1.89 ± 0.09	0.41 ± 0.05	0.33 ± 0.04
10	1.12 ± 0.08	0.90 ± 0.06	0.44 ± 0.05	0.19 ± 0.03

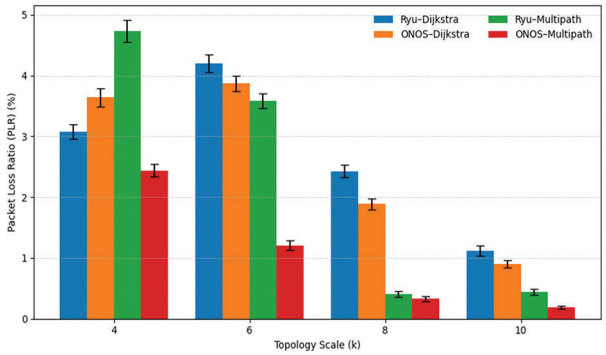


Fig. 6. Average packet-loss ratio (PLR %) for Ryu and ONOS controllers under Dijkstra and Multipath routing across Fat-Tree topologies ($k = 4$–10)

The error bars indicate consistency in measurement and statistical reliability ($p < 0.05$, paired t-test). In all configurations, Multipath routing significantly reduced jitter relative to Dijkstra-based routing, demonstrating that adaptive load-balancing schemes reduce the variation in packet delivery delay. The minimum jitter was consistently achieved by ONOS-Multipath, especially at $k = 8$ and $k = 10$, indicating the advantages of ONOS's distributed control plane and the effectiveness of path management. On the other hand, RyuDijkstra was also slightly more prone to jitter variance because it is a centralized, event-driven model that may become a bottleneck under heavy traffic. Small SD values indicate high reproducibility and consistent controller performance across different topologies at each experiment. ONOS-Multipath reduced average jitter by 46% compared with Ryu-Dijkstra ($\eta^2 = 0.31$), confirming enhanced temporal stability in packet delivery.

The results in Table 5 are the mean and standard deviation (SD) of PLR across all Fat-Tree configurations, and Fig. 6 plots the results with error bars representing ±SD from 100 simulation runs. The low SDs verify the repeatability and consistency of the experiment. In general, these results support the idea that adaptive, load-sensitive routing (especially ONOS multipath routing) guarantees enhanced reliability and delivery quality at a larger network scale. At $k = 10$, the packet-loss ratio decreased from 1.12% (Ryu-Dijkstra) to 0.19% (ONOS-Multipath)—an 83% reduction, attributable to adaptive multipath forwarding and distributed flow management.

6.1.5. Round-Trip Time (RTT) Across Topology Scales

Round-Trip Time (RTT) was considered one of the basic latency measures to determine the responsiveness of every controller routing layout among Fat-Tree topologies ($k = 4$, 6, 8, and 10). All scenarios were repeated 100 times, and the average RTT was computed with its standard deviation (±SD) to ensure statistical reliability. Multipath routing always delivered low RTT values, irrespective of the topology scale. ONOS-Multipath has the lowest latency of 1.921 ms at $k = 10$ across all configurations, demonstrating the synergy between adaptive routing and the distributed control plane. Ryu-Multipath was also competitive in latency but had a slightly higher RTT due to its single-threaded, event-driven design.

Conversely, Dijkstra-based routing, especially under the Ryu controller, exhibited high RTT values that increased with network size. Ryu-Dijkstra had the maximum delay (4.710 ms) at $k = 10$, which was not very scalable and did not adapt to changes in traffic conditions. Although ONOS-Dijkstra worked rather well given ONOS's distributed architecture, it lagged behind multi-path settings.

Table 6 shows the mean and standard deviation (SD) of the RTT values for each Fat-Tree configuration, and Fig. 7 displays them with error bars representing ±SD from 100 repetitions.

Table 6. Average round-trip time (RTT ms) for Ryu and ONOS controllers across Fat-Tree topologies ($k = 4$–10)

k	Ryu-Dijkstra	ONOS-Dijkstra	Ryu-Multipath	ONOS-Multipath
4	3.45 ± 0.12	3.66 ± 0.11	1.97 ± 0.08	2.23 ± 0.09
6	3.76 ± 0.10	3.62 ± 0.09	2.21 ± 0.07	1.89 ± 0.08
8	3.47 ± 0.09	2.72 ± 0.08	2.12 ± 0.06	1.88 ± 0.07
10	4.71 ± 0.13	3.23 ± 0.11	2.31 ± 0.07	1.92 ± 0.06

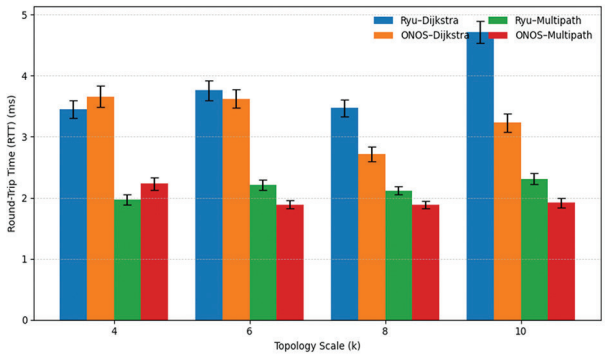


Fig. 7. Average round-trip time (RTT ms) for Ryu and ONOS controllers under Dijkstra and Multipath routing across Fat-Tree topologies ($k = 4–10$)

Low SD values indicate the repeatability and consistency of the results. Adaptive routing is more beneficial in larger-scale topologies, where the dynamism of path selection and load balancing are vital to minimizing latency. In summary, these results verify that Multipath routing, and in particular its implementation in ONOS, offers better end-to-end responsiveness and scalability for data-center SDN.

The usefulness of adaptive routing is more pronounced in larger topologies, where dynamic path selection and load balancing are essential for reducing latency. The low values of SD between configurations ensure the same and reproducible performance trends. These results support the hypothesis that Multipath routing, especially in the context of ONOS, offers the most optimal end-to-end responsiveness and creates a scalable, stable architecture for data-center-level SDN systems. ONOS-Multipath reduced average RTT by 59% (1.92 ms vs. 4.71 ms; $p < 0.05$), showing faster path convergence and lower queuing delay under large-scale topologies. Overall, quantitative comparisons confirm that ONOS-Multipath consistently achieves the best QoS outcomes, improving throughput by up to 150% and reducing delay metrics (jitter, PLR, RTT) by 40–80% across scales ($p < 0.05$).

7. QUALITY OF SERVICE ANALYSIS

This section summarizes the results of all controller–topology scales and controller–routing configurations in terms of Quality of Service (QoS). The analysis clearly shows that Multipath routing, especially when deployed with the ONOS controller, consistently provides the most balanced and superior performance across all evaluated metrics. ON-OS-Multipath achieves the highest average TCP throughput (3.476 Gbps) and the lowest average jitter (0.021 ms), packet-loss ratio (PLR) (1.043 %), and round-trip time (RTT) (1.978 ms) among all tested configurations. As shown in Table 7 and plotted as Fig. 8.

These findings affirm that ONOS's distributed control-plane architecture, composed of multiple coordinated instances, enables parallel decision-making, global state awareness, and efficient load balancing. This structural advantage allows ONOS to maintain network stability and scalability even under high-density Fat-Tree topologies.

Table 7. Summary of QoS performance metrics for Ryu and ONOS controllers

Metric	Ryu-Dijkstra	ONOS-Dijkstra	Ryu-Multipath	ONOS Multipath	Best Configuration (% Improvement)
UDP Throughput (Mbps)	1.040	1.080	1.384	1.378	Ryu-Multipath (+0.4 %)
TCP Throughput (Gbps)	1.417	1.808	3.270	3.476	ONOS-Multipath (+6.3 %)
Jitter (ms)	0.01775	0.018	0.069	0.0215	ONOS-Multipath (–68.8 %)
PLR (%)	2.708	2.499	2.294	1.043	ONOS-Multipath (–54.5 %)
RTT (ms)	3.848	3.309	2.153	1.978	ONOS-Multipath (–8.1 %)

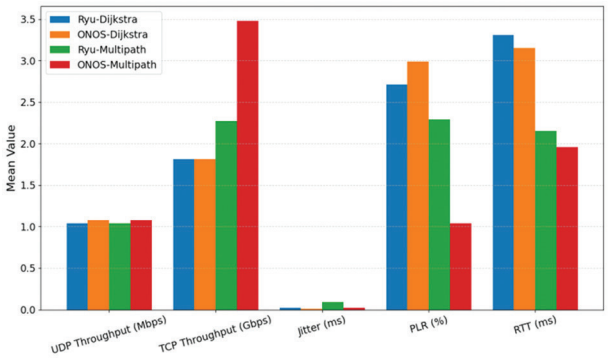


Fig. 8. Comparative summary of QoS metrics for Ryu and ONOS controllers across Fat-Tree topologies ($k = 4–10$).

Ryu-Multipath also demonstrates strong throughput performance, although its jitter values are slightly higher. This is expected because Ryu uses a single-threaded, event-driven processing model, which queues flow setups sequentially and can cause minor delays under heavy or bursty traffic.

Dijkstra-based routing, by contrast, exhibits lower throughput and higher latency, particularly under Ryu, due to its static single-path computation that cannot adapt to transient congestion or varying link conditions. ONOS-Dijkstra shows moderate improvement due to its distributed control logic but still remains less efficient than its multipath counterpart. Advantage of Distributed Control: The clustered architecture

of ONOS distributes the computation across multiple controller instances, minimizing flow setup latency and removing single-point bottlenecks. Adaptive Routing Synergy: Multipath routing leverages ONOS's global network view to dynamically rebalance traffic, avoiding congestion and maintaining high throughput under load. Centralized Limit on Controller: Ryu's single-threaded control loop imposes constraints on concurrent decision-making.

Though it works well on a small topology, it becomes less efficient on large networks. Practical Relevance: Such results directly apply to data-center, cloud, and 5G-edge deployments with low latency and high reliability, as well as to adaptive flow management.

Generally, this combined quality of service study confirms that ONOS-Multipath is the most stable, scalable, and high-performance SDN setup, which can serve as a viable guide for choosing controllers and routing policies in a real-world deployment.

7.1. SYSTEM-LEVEL OBSERVATIONS

System-level parameters, CPU utilization, flow setup time, and scalability were analysed to supplement the Quality-of-Service (QoS) analysis across all Fat-Tree topologies (k = 4, 6, 8, 10) as shown in Table 10. These measurements provide more insight into each controller's ability to handle growing network complexity and operational load.

Table 8. Controller resource utilization (CPU %, Memory %) for Ryu and ONOS controllers

Metric	Ryu (k = 4)	ONOS (k = 4)	Ryu (k = 6)	ONOS (k = 6)	Ryu (k = 8)	ONOS (k = 8)	Ryu (k = 10)	ONOS (k = 10)
CPU Utilization (%)	29.6	26.2	35.8	31.7	43.5	39.2	57.9	51.1
Flow Setup Time (ms)	1.4	1.2	1.8	1.5	2.1	1.8	2.4	2.0
Scalability (Max Hosts)	128	256	384	512	512	640	800	1024

Both Ryu and ONOS exhibit expected increases in CPU usage and flow-setup time as the network grows, due to increased workload from routing and control-plane activities. Nevertheless, ONOS continues to exhibit reduced CPU usage and setup times across all topologies, indicative of resource efficiency.

The latter can be attributed to ONOS's distributed, multi-instance control plane, which supports parallel processing and reduces contention among routing tasks. On a large scale (k = 10), ONOS has a 12 percent lower CPU load and has quicker flow establishment (2.0 ms vs 2.4 ms) than Ryu. These benefits directly translate into better responsiveness and reduced delay in flow provisioning, which is vital in production-scale networks to achieve real-time responsiveness. Scalability results support this trend: ONOS can support up to 1,024 hosts at k = 10, whereas Ryu scales to only 800 hosts. As shown in Table 8.

Such divergence implies that the single-threaded event loop used by Ryu cannot readily scale to parallel topology changes and flow installations as network size increases. Conversely, the clustered nature of ONOS distributes state management and decision-making, ensuring consistent performance even as load and link density increase. Combined with the previous system-level analysis, these system-level findings are consistent with the previous QoS analysis: ONOS has distributed control, which offers scalability over time, faster responsiveness, and balanced utilization.

The Ryu, though efficient in small to medium-sized networks, shows increasing control overhead with heavier loads. Generally, ONOS provides a resilient, scalable platform for large-scale, high-throughput SDN systems, including data centers and edge computing environments. It is possible to explain why ONOS-Multipath dominates all metrics by the integrated benefits of its distributed control-plane architecture and multipath-sensitive routing intelligence. In contrast to Ryu, where flow requests are handled sequentially in a single-threaded, event-driven loop, ONOS partitions distribute the workload across instances of controllers in a cluster. Such distributed coordination reduces contention when simultaneous flow configurations are used and increases control-plane performance when load is heavy. This advantage is further enhanced by Multipath routing, which uses ONOS's global network state to redistribute flows over multiple disjoint paths. Such synergy minimises congestion hotspots, balances link utilisation, and improves end-to-end latency. Such features are especially useful in high-density data center fabrics, cloud spine-leaf layouts, and 5G edge environments, where low latency and high throughput are important. Therefore, ONOS-Multipath is the configuration most deployment-ready to provide scalable, resilient SDN infrastructure.

8. CONCLUSION

The paper tested the execution of Ryu and ONOS controllers with Dijkstra and Multipath routing algorithms on scalable fat-tree networks (k = 4 -10). Quality-of-Service (quality of service) outcomes are obviously the best proof that Multi-path routing is much better than Dijkstra, regardless of the settings, and ONOS-Multipath is the one that delivers the best TCP throughput (3.476 Gbps), lowest percentage of packet-loss (1.042%), and less jitter (0.022 ms) at k = 10. Competitive results were also achieved by Ryu–Multipath, confirming the ability of path diversity to enhance traffic distribution and delivery stability. Analysis at the system level also made ONOS the more efficient and scalable controller. In all topologies, ONOS consistently

had lower CPU usage, shorter flow-setup durations, and better host scalability—up to 1,024 hosts at $k = 10$ and 800 hosts at the same time in Ryu.

These results confirm that the distributed architecture in ONOS is a more effective way to handle growing network complexity as compared to a single-threaded control design, as in Ryu. In general, the ONOS-Multipath setup proves to be the most efficient and balanced, providing high throughput, low latency, efficient resource utilization, and high scalability for large-scale SDN deployments. In the future, the research will examine trade-offs between performance gains and resource consumption to develop more comprehensive deployment frameworks. Future extensions include AI-based adaptive routing, synchronization of multi-ple controllers, and energy-aware, safe route optimization, moving SDN towards a predictive, self-optimizing networking model.

REFERENCES

[1] A. Ram, Y. Zhang, M. Chen, "Performance evaluation of software-defined network (SDN) controllers using Dijks-tra's algorithm", Wireless Networks, Vol. 28, 2022, pp. 3787-3800. doi: 10.1007/s11276-022-03044-3.

[2] M. Caria, A. Jukan, M. Hoffmann, "SDN partitioning: A centralized control plane for distributed routing protocols", IEEE Transactions on Network and Service Management, Vol. 13, No. 3, 2016, pp. 381-393. doi: 10.1109/TNSM.2016.2585759.

[3] Z. Latif, A. Jolfaei, W. Yu, S. Nepal, "A comprehensive survey of interface protocols for software-defined networks", Journal of Network and Computer Applications, Vol. 156, 2020, p. 102535. doi: 10.1016/j.jnca.2020.102535.

[4] M. C. Saxena, M. Sabharwal, P. Bajaj, "Exploring path computation techniques in software-defined networking", International Journal of Research in Information Technology and Computer Communication, Vol. 11, No. 9S, 2023, pp. 553-567. doi: 10.17762/ijritcc.v11i9s.7468.

[5] R. Jiang, J.-R. Huang, H.-W. Huang, J.-H. Liao, S.-Y. Chen, "Extending Dijkstra's shortest path algorithm for software-defined networking", Proceedings of the Asia-Pacific Network Operations and Management Symposium, Hsinchu, Taiwan, 17-19 September 2014, pp. 1-4. doi: 10.1109/APNOMS.2014.6996609.

[6] A. Abdul-Hafiz, E. Adedokun, S. Man-Yahya, "Improved extended Dijkstra's algorithm for software-defined networks", International Journal of Applied Information Systems, Vol. 12, No. 8, 2017, pp. 22-26. doi: 10.5120/ijais2017451714.

[7] C. Jayawardena, J. Chen, A. Bhalla, L. Bu, "Comparative analysis of POX and Ryu SDN controllers in scalable networks", International Journal of Computer Networks and Communications, Vol. 17, No. 2, 2025, pp. 35-51. doi: 10.5121/ijcnc.2025.17203.

[8] I. Koulouras, S. V. Margariti, I. Bobotsaris, E. Stergiou, C. Stylios, "Assessment of SDN controllers in wireless environments using a multi-criteria technique", Information, Vol. 14, No. 9, 2023, p. 476. doi: 10.3390/info14090476.

[9] R. Patel, N. Gupta, "Comparative evaluation of SDN controllers: POX, Ryu, and OpenDaylight", IEEE Access, Vol. 11, 2023, pp. 29015-29028. doi: 10.1109/ACCESS.2023.3172540.

[10] M. S. Nuraeni, E. Wahyudi, N. Iryani, "Perbandingan kontroler POX, Ryu dan ONOS pada arsitektur software defined network menggunakan topologi linear", Journal of Telecommunication, Electronic and Control Engineering, Vol. 4, No. 2, 2022, pp. 66-79. doi: 10.20895/jtece.v4i2.360.

[11] S. A. Mostafavi, V. Hakami, F. Paydar, "Performance evaluation of software-defined networking controllers: A comparative study", Computer Knowledge Engineering, Vol. 2, No. 2, 2019, pp. 63-73. doi: 10.22067/cke.v2i2.84917.

[12] J. B. Silva et al. "Benchmarking of mainstream SDN controllers over open off-the-shelf software switches", Internet Technology Letters, Vol. 3, No. 2, 2020, p. e152. doi: 10.1002/itl2.152.

[13] G. Altangerel, T. Chuluuntsetseg, D. Yamkhin, "Performance analysis of SDN controllers: POX, Floodlight and OpenDaylight", arXiv:2112.10452, 2021.

[14] M. Z. Abdullah, N. A. Al-Awad, F. W. Hussein, "Performance comparison and evaluation of different software-defined networks controllers", International Journal of Computer Network Technology, Vol. 6, No. 2, 2018, pp. 35-41.

[15] L. Mamushiane, A. Lysko, S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers", Proceedings of the Wireless Days Conference, Dubai, United Arab Emirates, 3-5 April 2018, pp. 54-59. doi: 10.1109/WD.2018.8361714.

[16] S. Askar, F. Keti, "Performance evaluation of different SDN controllers: A review", International Journal of Science and Business, Vol. 5, No. 6, 2021, pp. 67-80. doi: 10.5281/zenodo.11164550.

[17] M. N. A. Sheikh, I.-S. Hwang, M. S. Raza, M. S. Ab-Rahman, "A qualitative and comparative performance assessment of logically centralized SDN controllers via

Mininet emulator", Computers, Vol. 13, No. 4, 2024, p. 85. doi: 10.3390/computers13040085.

[18] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study", ACM Computing Surveys, Vol. 53, No. 6, 2020, p. 133. doi: 10.1145/3421764.

[19] D. Goteti, I. Rasheed, "Multipath routing algorithm to find optimal path in SDN with POX controller", International Journal of Electrical and Computer Engineering Systems, Vol. 16, No. 2, 2025, pp. 121-131. doi: 10.32985/ijeces.16.2.4.

[20] U. Ali, "Performance comparison of SDN controllers in different network topologies", Research Square, 2024. doi: 10.21203/rs.3.rs-4826985/v1.

[21] K. A. Septian, I. Istikmal, I. Ginting, "Analysis of ONOS clustering performance on software-defined network", Proceedings of the IEEE International Conference on Internet of Things and Intelligent Systems, Bandung, Indonesia, 23-24 November 2021, pp. 117-122. doi: 10.1109/IoTAIS53735.2021.9628659.

[22] A. Nguyen-Ngoc, S. Lange, T. Zinner, M. Seufert, P. Tran-Gia, N. Aerts, D. Hock, "Performance evaluation of selective flow monitoring in the ONOS controller", Proceedings of the 13th IEEE/IFIP International Conference on Network and Service Management, Tokyo, Japan, 26-30 November 2017, pp. 1-6. doi: 10.23919/CNSM.2017.8256058.

[23] J. Ma, R. Jin, L. Dong, G. Zhu, X. Jiang, "Implementation of SDN traffic monitoring based on Ryu controller", Proceedings of the International Symposium on Computer Applications and Information Systems, 2022. doi: 10.1117/12.2639589.

[24] H. M. U. Rahman et al. "Empirical performance evaluation of open-source SDN controllers in different network topologies", Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 8-12 May 2023, pp. 1-6. doi: 10.1109/NOMS56928.2023.10154394.

[25] P. Monika, R. M. Negara, D. S. Danu, "Performance analysis of software-defined network using intent monitor and reroute method on ONOS controller", Bulletin of Electrical Engineering and Informatics, Vol. 9, No. 5, 2020, pp. 2065-2073. doi: 10.11591/EEI.V9I5.2413.

[26] M. A. Khan, S. Al-Hadhrami, "Load balancing implementation strategy for various services in software-defined network using ONOS controller", International Journal of Computing and Digital Systems, Vol. 12, No. 1, 2022, pp. 1287-1293. doi: 10.12785/ijcds/1201103.

[27] A. Sharma, N. Patel, "ONOS SDN framework: Assessing the impact of single and multi-controller architectures on network efficiency", Proceedings of the Smart Trends in Computing and Communications Conference, 2024, pp. 370-381. doi: 10.1007/978-981-97-1323-3_32.

[28] M. F. Monir et al. "Benchmarking network functionality: Performance evaluation of SDN controllers on different network functions", Proceedings of the 100th IEEE Vehicular Technology Conference, Washington, DC, USA, 7-10 October 2024. doi: 10.1109/VTC2024-Fall63153.2024.10756803.

[29] L. L. Bosi, A. C. Mendes, R. M. Salles, "A review on the overall performance of SDN controllers", Proceedings of the 11th International Conference on Software Defined Systems, Gran Canaria, Spain, 9-11 December 2024, pp. 156-163. doi: 10.1109/SDS64317.2024.10883890.

[30] J. Ali, C. Iwendi, G. Shan, H. C. Wu, M. J. F. Alenazi, "Performance optimization of software-defined industrial Internet-of-Things (SD-IIoT)", IEEE Access, Vol. 12, 2024, pp. 169659-169670. doi: 10.1109/ACCESS.2024.3466186.

[31] S. Singh Ghritlahare, M. Atulkar, R. K. Chouhan, "Benchmarking SDN controllers: Performance analysis of POX and Floodlight in DCell data centre topology", Proceedings of the IEEE International Conference on Communication Systems and Network Technologies, Bhopal, India, 2025, pp. 1097-1101. doi: 10.1109/CSNT64827.2025.10968400.

[32] Y. Li, B. Zhao, Y. Hu, J. Dai, "NBench: A comprehensive SDN controller benchmarking suite", Proceedings of the 22nd International Conference on Optical Communications and Networks, Harbin, China, 2024, pp. 1-3. doi: 10.1109/ICOCN63276.2024.10648356.

[33] A. Ram, S. K. Chakraborty, "Analysis of software-defined networking (SDN) performance in wired and wireless networks across various topologies, including single, linear, and tree structures", Indian Journal of Information Sources and Services, Vol. 14, No. 1, 2024, pp. 39-50. doi: 10.51983/ijiss-2024.14.1.3926.

[34] H. M. U. Rahman, G. Bahoo, L. Zafar, I. Al-Oqily, H. A. Khattak, A. Ahmad, "Empirical performance evaluation of open source SDN controllers in different network topologies", Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 8-12 May 2023, pp. 1-6. doi: 10.1109/NOMS56928.2023.10154394.