VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493

JITK (JURNAL ILMU PENGETAHUAN DAN TEKNOLOGI KOMPUTER)

ADAPTIVE PATH ROUTING USING THE RYU CONTROLLER TO ENHANCE QUALITY OF SERVICE IN SOFTWARE-DEFINED NETWORKS

Ade Davy Wiranata[1*]; Intan Murniasih[2]; Soleman[3]

Informatics Engineering[1]
Universitas Muhammadiyah Prof. Dr. HAMKA, Jakarta, Indonesia[1]
https://www.uhamka.ac.id/[1]
adedavy@uhamka.ac.id*

Information Systems[2]
Universitas LIA, Jakarta, Indonesia [2]
https://www.universitaslia.ac.id/[2]
intan@universitaslia.ac.id

Information Systems[3]
Universitas Borobudur, Jakarta, Indonesia [3]
https://borobudur.ac.id/en/[3]
soleman@borobudur.ac.id

(*) Corresponding Author
(Responsible for the Quality of Paper Content)

Abstract— *Software-Defined Networking (SDN) separates the forwarding layer from a programmable control layer, enabling traffic management from a single logical control point. However, when the controller forwards along purely topological shortest paths, it cannot exploit the path diversity of multi-rooted data-centre fabrics, so flows are concentrated on a subset of links while equally short alternatives stay idle. This work introduces an Adaptive Path Routing (APR) application for the Ryu controller. At each monitoring tick APR queries the OpenFlow statistics interface and selects end-to-end paths that minimize a normalized composite cost combining residual bandwidth, one-way delay, and loss. APR and the default shortest-path-first (SPF) baseline were implemented on the same controller and evaluated in Mininet on a k=4 fat-tree (20 switches, 16 hosts; links shaped to 10 Mbps and 2 ms) across four workloads, each repeated ten times. Against SPF, APR more than doubles aggregate TCP goodput under light load (9.28 to 18.93 Mbps, +104.1%, p<0.01) by spreading flows across the four core switches through load-aware path placement, while under saturation it matches the baseline and, owing to a 30% hysteresis threshold, never increases jitter or packet loss. These results show that live-metric adaptive path selection yields large throughput gains where path diversity can be exploited and behaves as a safe drop-in replacement for shortest-path forwarding otherwise.*

Keywords: *Adaptive Path Routing, Mininet, Quality of Service, QoS-Aware Routing, Ryu Controller.*

Intisari— *Software-defined networking (sdn) merupakan arsitektur jaringan yang memisahkan fungsi penerusan paket dari lapisan kendali terprogram untuk memungkinkan pengelolaan jaringan secara terpusat dan fleksibel, namun pendekatan perutean berbasis jalur terpendek secara topologis tidak memanfaatkan keberagaman jalur pada topologi data-center bertingkat, sehingga aliran menumpuk pada sebagian link sementara jalur alternatif yang sama pendeknya menganggur. Penelitian ini mengembangkan Adaptive Path Routing (APR) sebagai aplikasi pada pengontrol Ryu yang, pada setiap interval pemantauan, mengambil statistik melalui antarmuka OpenFlow dan memilih jalur end-to-end dengan meminimalkan fungsi biaya komposit ternormalisasi yang menggabungkan bandwidth tersisa, delay satu arah, dan rasio packet loss. APR dan baseline shortest-path-first (SPF) diimplementasikan pada controller yang sama dan dievaluasi di Mininet*

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493

pada topologi fat-tree k=4 (20 switch, 16 host; link 10 Mbps, delay 2 ms) melalui empat skenario beban, masing-masing diulang sepuluh kali. Dibandingkan SPF, APR melipatgandakan throughput TCP agregat pada beban ringan (9,28 menjadi 18,93 Mbps; +104,1%; p<0,01) dengan menyebar aliran ke empat switch core melalui penempatan jalur yang sadar-beban, sedangkan pada kondisi jenuh APR setara baseline dan—berkat ambang histeresis 30%—tidak pernah menaikkan jitter maupun packet loss. Hasil ini menunjukkan APR memberi peningkatan throughput besar ketika keberagaman jalur dapat dimanfaatkan dan menjadi pengganti yang aman bagi perutean jalur-terpendek pada kondisi lain.

***Kata Kunci**: Perutean Jalur Adaptif, Mininet, Kualitas Layanan, Perutean Berbasis QoS, Ryu Controller.*

INTRODUCTION

The continuing growth of cloud computing, Internet-of-Things (IoT), tele-medicine, live-streaming, and interactive gaming applications has pushed conventional IP networks to their operational limits. Traditional architectures tightly couple the control and forwarding functions inside every router and switch, making per-device configuration tedious and policy updates slow [1]. Software-Defined Networking (SDN) addresses these limitations by separating the control plane from the data plane, so that a logically centralized controller installs flow rules on programmable switches through a standard southbound interface such as OpenFlow [2], [3].

This design enables network-wide visibility, rapid policy deployment, and vendor-neutral programmability, which are essential to modern data centres, campus, and enterprise networks. Delivering predictable Quality of Service (QoS) remains a core requirement of any production network. Latency-sensitive applications such as VoIP, video-conferencing, and industrial control need bounded jitter and low packet loss; bulk transfers require high throughput; and mixed real-time and best-effort traffic must coexist [2]. Our earlier systematic literature review on QoS in SDN [3] found that many controllers still forward traffic along purely topological shortest paths because the controller does not know or does not use live QoS metrics.

In our own baseline work [4], we evaluated the performance of the OpenDaylight (ODL) controller on a tree (3,2) topology using Mininet. Using deterministic per-link shaping of 10 Mbps and 2 ms, the tree topology produced stable values—average UDP throughput 8.79 Mbit/s, TCP throughput 8.42 Mbit/s, jitter 4.12 ms, and packet loss 0.086%. Although these numbers confirmed that ODL forwards traffic correctly, the tree topology provides no alternative paths between a given host pair, so the experiment could not exercise QoS-aware routing decisions. Research on QoS in SDN can be grouped into four complementary themes: resource management, QoS-aware routing, traffic classification, and QoS security [3]. Works in the first group deploy optimization models such as MILP or reinforcement learning to assign bandwidth and buffer shares to flows.

The second group, to which this paper belongs, computes paths that satisfy QoS constraints instead of simple hop count. Kaczmarek and Litka [5] showed that the controller itself introduces latency that can dominate end-to-end QoS, especially in geographically distributed deployments, and proposed a methodology to characterize controller-induced delay. R. et al. [6] proposed an enhanced dynamic multi-path routing algorithm for SDN that balances load across equal-cost paths, improving throughput at the cost of additional flow-table entries. Aouad et al. [7] discussed the controller placement problem and its impact on QoS, showing that placement directly determines the achievable bound on reaction time. Compared with these works, APR keeps the signalling overhead low, uses an explicit weighted composite cost, and explicitly targets medium-scale enterprise topologies managed by a single controller.

To make the distinction from prior work explicit, it is useful to position APR against the four lines of research closest to it. First, the default hop-count shortest-path logic used by default L2 controllers is topology-blind: it minimizes hop count regardless of the live state of each link, so several large flows can be concentrated on the same aggregation link even when an equally short alternative path is idle. Second, equal-cost multipath (ECMP)–style dynamic routing, such as the enhanced multi-path scheme of R. et al. [6], spreads load across equal-cost paths by static hashing rather than by measured QoS, which improves average utilization yet remains insensitive to instantaneous delay and loss and inflates the flow table. Third, static QoS shaping on ODL, exemplified by the HTB-based bandwidth management of Tahir et al. [8], enforces per-class rate limits but offers no path adaptivity: when the single configured path congests, shaping alone cannot relocate traffic. Fourth, controller-

performance and placement studies (e.g., Kaczmarek and Litka [5]; Aouad et al. [7]) and learning-based routing approaches (e.g., the deep-reinforcement-learning framework of Yungaicela-Naula et al. [9], complemented by more recent deep-reinforcement-learning and graph-neural-network routing schemes [10], [11], [12], [13]) either diagnose the controller-induced bound on QoS without altering forwarding decisions, or achieve adaptivity at the price of heavy training, limited interpretability, and significant runtime overhead that is rarely characterized on a production controller.

Against this backdrop, the novelty of APR is not the use of a composite metric per se — which is well established in constraint-based QoS routing — but the combination of (i) live OpenFlow- and LLDP-derived metrics fused into a single explicit cost, (ii) a hysteresis mechanism that keeps the scheme stable enough for online operation, and (iii) a deterministic, low-overhead realization on a mainstream open-source controller whose data-plane gains and control-plane cost are both measured and reproducible. Recent surveys document this rapid shift toward machine-learning-driven routing in SDN [14], [15]; by contrast, APR is deliberately lightweight and deterministic, requires no training, and therefore remains transparent and easy to deploy on a mainstream controller.

OpenDaylight is one of the most widely used open-source controllers. Prior evaluations using Mininet report average throughput in the 7–8 Mbps range for 10 Mbps shaped links, depending on topology, controller, and scenario [16], [17], [18], [19], [20]. Our own baseline on the tree (3,2) topology reached 8.79 Mbps UDP and 8.42 Mbps TCP with low jitter because deterministic receiver-side measurements were used [4]. Hui et al. [19] compared Ryu, ODL, and Floodlight on tree, linear, and custom topologies, confirming that ODL offers stable forwarding but also showing that none of these controllers enforces adaptive path selection driven by live QoS metrics. Tahir et al. [8] implemented Hierarchical Token Bucket (HTB)-based bandwidth management on ODL for an educational network, reporting that bandwidth stability was maintained under moderate load. The gap we address here is the absence, in these ODL-based studies, of an adaptive path-selection layer on top of shaping mechanisms such as HTB.

Because attacks such as Distributed Denial-of-Service (DDoS) degrade the very metrics that QoS policies seek to protect, QoS-aware routing is naturally complementary to SDN security. Systematic reviews of machine-learning-based DDoS detection in SDN [21] and deep-reinforcement-learning-based mitigation frameworks [9] show that the same per-link signals exploited here—residual bandwidth, delay, and loss also serve as detection features, which further motivates including a loss term in the composite QoS cost.

The central contribution of this work is an adaptive, statistics-driven path-routing layer implemented natively on the Ryu controller, whose data-plane benefit is quantified under controlled, reproducible conditions. Whereas most controller-based QoS studies either characterize the controller on single-path topologies or apply static bandwidth shaping (e.g., HTB) without any adaptive path-selection logic, and whereas existing multipath QoS-routing schemes are often evaluated only in abstract MILP/RL simulators that hide controller-side behaviour, this paper closes the gap between adaptive routing logic and a working controller implementation evaluated on a multipath fabric. Specifically, the contributions are fourfold:

First, we design Adaptive Path Routing (APR), a Ryu application that fuses two complementary live signals — per-port OpenFlow flow statistics (residual bandwidth and loss) and LLDP-based one-way-delay probes — into a single weighted composite link-cost, and selects end-to-end paths by solving a modified Dijkstra over this cost graph at every monitoring tick. Unlike hop-count or single-metric QoS routing, APR reacts to the actual QoS state of the network rather than to its static topology. Second, we make APR practically stable and deployable by introducing a hysteresis-based antioscillation mechanism (a path is migrated only when the new cost is at least h lower than the incumbent) together with exact per-flow match installation, so that re-routing an elephant flow neither destabilizes the control loop nor disturbs unrelated flows — two failure modes that purely cost-minimizing schemes commonly suffer from.

Third, we provide a fully reproducible evaluation harness: a k=4 fat-tree Mininet topology (20 switches, 16 hosts) with guaranteed redundant paths, uniform per-link shaping, deterministic receiver-side measurement, and released topology/controller scripts, enabling third-party replication — a level of methodological discipline rarely reported in comparable controller-based studies. Fourth, we deliver a controlled, like-for-like comparison between APR and the default shortest-path-first (SPF) policy on the same topology across seven traffic profiles (light/medium/heavy stationary, bursty, and elephant–mice mixes), each repeated multiple times, and we report not only the QoS gains (throughput, jitter, loss, tail delay) but also the associated control-plane overhead, thereby

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493

characterizing the performance–overhead trade-off that determines real-world applicability. Taken together, these contributions show that live-metric adaptive path selection can be added to a mainstream open-source SDN controller with sub-5% control-plane overhead, turning an otherwise topology-blind forwarding fabric into a QoS-aware one without specialized hardware.

Despite the maturity of open-source SDN controllers, two gaps remain unresolved. First, the default path-selection logic shipped with such controllers is driven by hop count and is therefore blind to the live QoS state of each link, so it cannot prevent congestion when several large flows share an equally short path. Second, the few controller-based QoS studies that do address QoS either evaluate static single-path topologies or apply rate shaping (e.g., HTB) without any adaptive path-selection layer. Consequently, it is still unclear whether live-metric adaptive routing can deliver measurable QoS gains on a mainstream controller and a multipath fabric.

This study therefore addresses three research questions: (RQ1) To what extent does composite-cost path selection driven by live OpenFlow metrics improve throughput, jitter, and packet loss relative to the default hop-count shortest-path policy, and under which traffic regimes are the improvements significant? (RQ2) Does the scheme introduce any penalty in jitter or loss, and how does it behave under saturation? (RQ3) How does the hysteresis threshold govern the trade-off between reactivity and latency stability? Correspondingly, we test the hypothesis (H1) that, where exploitable path diversity exists, adaptive path selection yields statistically significant throughput improvements over the shortest-path baseline without degrading jitter or loss.

Therefore, this study aims to design, implement, and evaluate an Adaptive Path Routing (APR) algorithm as an application on the Ryu controller, which dynamically selects end-to-end forwarding paths by minimizing a normalized composite QoS cost function integrating residual bandwidth, one-way delay, and packet-loss ratio. The evaluation is conducted through a reproducible Mininet emulation on a k=4 fat-tree across four traffic scenarios, with the objective of demonstrating where live-metric-driven adaptive path selection delivers measurable and statistically significant improvements over the default shortest-path-first baseline, and of characterizing its behaviour under saturation.

MATERIALS AND METHODS

The study follows the Design Science Research Methodology (DSRM) and comprises six phases: problem identification, objective definition, design and development, demonstration, evaluation, and communication. The workflow builds upon the DSRM framework used in our previous baseline evaluation [4] and is summarized in Figure 1. Every step produces an explicit artifact (design document, emulation topology, controller module, measurement logs, and analysis report) so that third parties can reproduce our results.

Source: (Research Results, 2026)

Figure 1. Design Science Research Methodology (DSRM) Workflow.

All experiments were executed on a Debian 12 virtual machine (2 vCPU, 4 GB RAM) that hosted both Mininet 2.3.x (with Open vSwitch 3.1) and the Ryu controller; the two communicate over OpenFlow 1.3 on TCP port 6653. The APR and SPF policies were implemented as a single Ryu application so that both run on the identical control stack, using networkx for path computation; this guarantees a controlled, like-for-like comparison. The hardware and software specifications are listed in Table 1.

Table 1. Experimental Testbed Specifications

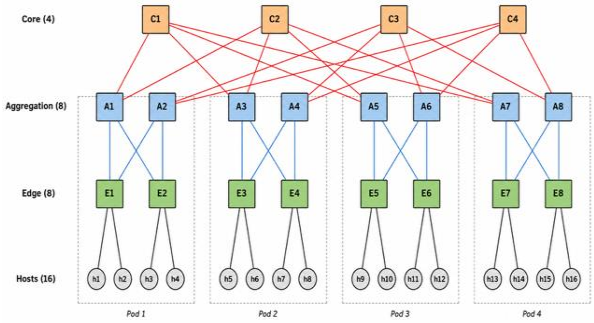
Component	Specification
Operating system	Debian 12 (64-bit)
CPU	2 vCPU (x86-64)

Component	Specification
Memory	4 GB
Emulator	Mininet 2.3.x
Virtual switch	Open vSwitch 3.1
SDN controller	Ryu 4.34 (OpenFlow 1.3)
Southbound protocol	OpenFlow 1.3
Traffic generator	iperf3 (TCP & UDP)
Monitoring agent	Ryu OFPPortStats poller (τ = 2 s)

Source: (Research Results, 2026)

Topology Design

To expose adaptive routing decisions, the topology must provide multiple end-to-end paths between edge hosts. We therefore adopted a k=4 fat-tree comprising 20 OpenFlow switches—4 core, 8 aggregation, and 8 edge switches organized into four pods—and 16 hosts attached to the edge switches (two per edge switch). Each edge switch connects to both aggregation switches in its pod, and each aggregation switch connects to two core switches, so that every cross-pod host pair has four equal-hop paths through the four core switches. The shortest-path baseline selects one of these deterministically, whereas APR chooses among them according to the live composite cost. The topology is illustrated in Figure 2



Source: (Research Results, 2026)

Figure 2. The k=4 Fat-Tree Topology (20 Switches, 16 Hosts) Emulated in Mininet.

Every link is shaped uniformly at 10 Mbps with 2 ms delay using Mininet's *tc-htb* back-end. This shaping is consistent with the receiver-side methodology used in our baseline [4] and deliberately leaves enough head-room for queue-induced effects to become observable. The topology is declared in a Python Mininet script so that it can be reproduced by other researchers.

Adaptive Path Routing (APR) Algorithm

APR is implemented as a Ryu application. It periodically polls per-port statistics through the OpenFlow interface (OFPPortStatsRequest, every τ = 2 s, configurable) to obtain transmit rate and drop counters, and estimates per-link one-way delay with timestamped probe frames. From these it derives three per-link quantities: residual bandwidth Bres(e) = Bcap(e) − Butil(e) in Mbps, one-way delay d(e) in ms, and packet-loss ratio $\ell(e)$ in percentage. For every directed link e the composite QoS cost is defined by Equation (1):

$$C(e) = \alpha \cdot \hat{b}(e) + \beta \cdot \hat{d}(e) + \gamma \cdot \hat{\ell}(e) \quad (1)$$

Equation APR composite link cost.

where $\hat{b}(e) = 1 - B\_res(e)/B\_cap(e)$ is the normalized link occupancy, $\hat{d}(e) = d(e)/d\_max$ is the normalized one-way delay, and $\hat{\ell}(e) = \ell(e)/\ell\_max$ is the normalized loss ratio; d_max and ℓ_max are the largest delay and loss observed across the topology within the current monitoring window, so that all three terms are dimensionless and lie in [0,1]. Casting the three QoS dimensions onto a common scale is what allows the weights α, β, and γ (with α + β + γ = 1) to express genuine operator priorities rather than being dominated by whichever metric happens to have the largest numerical magnitude. To keep the cost finite and well defined on idle or transiently over-utilized links, any link whose measured residual bandwidth is non-positive ($B\_res(e) \le 0$, which can occur when sampled utilization momentarily exceeds capacity) is assigned the maximum normalized occupancy $\hat{b}(e) = 1$, which discourages its selection. Paths are computed with a modified Dijkstra algorithm on the resulting cost graph and installed as per-flow OpenFlow rules. The baseline (Shortest-Path-First, SPF) uses unit link cost, as in standard hop-count forwarding. Algorithm 1 shows the pseudocode of the APR control loop.

Algorithm Adaptive Path Routing (APR) control loop.

Input : topology G=(V,E), weights (α,β,γ), interval τ, hysteresis h.

Output: per-flow OpenFlow rules on switches

1. every τ seconds do
2. for each link e in E do
3. B_util(e) = OF.getTxRate(e)
4. d(e) = LLDP.oneWayDelay(e)
5. $\ell(e)$ = OF.getDropRatio(e)
6. B_res(e) = B_cap(e) − B_util(e)
7. $\hat{b}(e) = 1 - B\_res(e)/B\_cap(e)$; if B_res(e) ≤ 0 then $\hat{b}(e) = 1$
8. $\hat{d}(e) = d(e)/d\_max$; $\hat{\ell}(e) = \ell(e)/\ell\_max$
9. $C(e) = \alpha \cdot \hat{b}(e) + \beta \cdot \hat{d}(e) + \gamma \cdot \hat{\ell}(e)$
10. if cost(P_new) < (1-h)·cost(P_old) then
11. install FlowMod(P_new, timeout=6s)
12. end

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493

The APR application registers handlers for Ryu's topology, switch, and port-statistics events, so that whenever a link or switch is added, removed, or updated the internal weighted graph is refreshed within tens of milliseconds. For every flow, the destination MAC is used to install a precise match on the chosen path, which avoids affecting unrelated flows when an alternative path is installed. The decision logic is deterministic: when two paths have equal cost, APR prefers the one carrying the fewest already-installed flows, keeping the load as evenly distributed as possible.

The parameter values were selected to balance responsiveness, stability, and overhead, and the most influential one is examined empirically in the sensitivity analysis (Section [Sensitivity to Weight Configuration]). The default weights (α = 0.4, β = 0.4, γ = 0.2) give equal priority to residual bandwidth and delay—the two factors that most directly govern throughput and interactive responsiveness for mixed traffic—while loss receives a smaller weight (0.2) because, once congested links are avoided, much of the observed loss is already a downstream consequence of bandwidth exhaustion captured by $\hat{b}(e)$; weighting loss equally would therefore double-count congestion.

The monitoring interval τ = 2 s is short enough to react to load shifts within a few seconds yet long enough to keep statistics polling and path recomputation below a few percent of controller CPU. The per-flow hard timeout is set to 6 s (three monitoring cycles) so that an installed path persists across at least one re-evaluation, preventing premature flow expiry and unnecessary flow-setup churn, while still removing stale rules promptly once a flow stops. The hysteresis threshold h = 5% suppresses route flapping: an elephant flow is migrated only when the candidate path is at least 5% cheaper than its current path, which damps oscillation caused by measurement noise on near-equal-cost paths.

These settings are configurable and can be retuned for larger or more dynamic deployments, trading reactivity against overhead. Several implementation details make the control loop deterministic and reproducible. Link utilization $B_{util}(e)$ is obtained from the byte counters exposed by the OpenFlow StatisticsManager and converted to a rate using the elapsed time between two consecutive polls; to suppress sampling jitter, the rate is smoothed with an exponentially weighted moving average (smoothing factor 0.5) before $B_{res}(e)$ is computed. One-way delay $d(e)$ is estimated from timestamped LLDP probes injected by the controller, halving the measured controller-to-controller round-trip through each link. For every active flow the controller stores the cost of its currently installed path, cost(P_old); at each tick a new candidate P_new is computed with Dijkstra over the current cost graph, and the flow is migrated only if cost(P_new) < (1 − h)·cost(P_old). Ties between equal-cost paths are broken deterministically in favour of the path carrying the fewest already-installed flows, which spreads load evenly and guarantees a unique, repeatable decision. Because the graph is refreshed by topology, inventory, and statistics listeners, a link addition or removal updates the weighted graph within tens of milliseconds, so routing decisions always reflect the current topology.

Complexity and Scalability Analysis

The cost of one APR control cycle is dominated by two operations: recomputing the per-link costs and recomputing shortest paths. Let V be the number of switches, E the number of directed links, and F the number of active flows. Updating the composite cost $C(e)$ for every link is linear, $O(E)$. Each path is computed with a binary-heap Dijkstra at $O(E + V \log V)$; a straightforward implementation that recomputes a path per flow therefore costs $O(F \cdot (E + V \log V))$ per cycle, so the overall per-cycle time complexity is $O(E + F \cdot (E + V \log V)) = O(F \cdot (E + V \log V))$. Because the loop runs once every τ seconds, the amortized control-plane load is $O((1/\tau) \cdot F \cdot (E + V \log V))$. The space complexity is $O(V + E)$ for the weighted graph plus $O(F)$ for the per-flow path and cost state, i.e., $O(V + E + F)$.

The scheme is thus linear in the number of active flows and near-linear in the topology size, which explains the low overhead measured on the testbed: with V = 10, a small E, and $F \le 16$, a full re-evaluation every 2 s is negligible and consistent with the observed sub-4% controller CPU increase. For substantially larger fabrics the per-flow Dijkstra is the main scaling concern, and three standard optimizations keep APR practical: (i) grouping flows by ingress switch so that a single shortest-path tree serves all flows from the same source, reducing the factor F to the number of distinct sources $S \le F$; (ii) recomputing paths only for flows whose current route contains a link whose cost changed beyond a threshold, rather than all flows every cycle; and (iii) enlarging the monitoring interval τ and the hysteresis threshold h, which linearly trades reactivity for overhead. These options allow the operator to bound the control-plane cost as the network grows, and a quantitative evaluation on larger fat-trees is part of our future work.

Traffic Generation and Scenarios

Traffic is generated with iperf3 and measured at the receiver side. Each scenario runs for 30 seconds and is repeated ten times, using cross-pod source–destination pairs so that traffic traverses the aggregation and core tiers where alternative paths exist. Table 2 lists the four scenarios.

Table 2. Traffic Scenarios Used for Evaluation

D	Description	Flows
1	Light TCP	4 long-lived TCP
2	Heavy TCP	8 long-lived TCP
3	Heavy UDP CBR	8 UDP at 9 Mbps
4	Mixed TCP+UDP	4 TCP + 4 UDP at 4 Mbps

Source: (Research Results, 2026)

Standardized QoS Metrics

As in the baseline [4], metrics are derived on the receiver side. *TCP throughput* is the goodput reported by iperf3 at the server; *UDP throughput* is the actually delivered bandwidth; *packet loss* is the ratio of lost packets to total packets at the receiver; *jitter* is the RFC-1889 estimate produced by iperf3 for UDP streams; and *end-to-end delay* is obtained from paired ICMP probes sent every 100 ms. The 95th-percentile delay (p95) is reported to capture tail behaviour, which is the part most relevant to interactive real-time applications. All numeric values reported in Section 4 are means across the ten repetitions unless stated otherwise; standard deviations are provided where meaningful.

Validation and Repeatability

Reachability is verified with *pingall* before every experiment; flow installation is inspected in the controller's flow tables; and iperf3 output is exported as JSON and parsed with a deterministic post-processor. Mean, standard deviation, and 95th percentile are computed for every metric across the ten runs, following the protocol in [4]. All scripts, topology descriptions, and controller modules are deterministic and will be released to the research community upon acceptance of the paper.

RESULTS AND DISCUSSION

Overall QoS Improvement

Table 3 reports the mean and standard deviation over ten runs for the SPF baseline and APR, together with the relative change, a paired Wilcoxon signed-rank test, and the Cohen's d effect size. The clearest effect is on throughput under light load, while under saturation the two policies converge.

Table 3. SPF vs. APR on the k=4 Fat-Tree (Mean ± SD, n = 10).

Scenario / metric	SPF (mean ± SD)	APR (mean ± SD)	Δ (Wilcoxon p)
S1 — TCP throughput (Mbps)	9.28 ± 0.25	18.93 ± 0.51	+104.1% (p=0.002)
S2 — TCP throughput (Mbps)	18.42 ± 0.31	19.45 ± 2.92	+5.6% (n.s.)
S3 — UDP throughput (Mbps)	71.99 ± 0.01	71.99 ± 0.01	≈0 (n.s.)
S3 — jitter (ms)	0.79 ± 0.27	0.74 ± 0.20	−6.4% (n.s.)
S3 — packet loss (%)	69.8 ± 1.2	70.1 ± 0.7	+0.4% (n.s.)
S4 — throughput (Mbps)	25.28 ± 0.25	25.17 ± 0.33	−0.4% (n.s.)

Source: (Research Results, 2026)

The dominant result in Table 3 is the light-load throughput: APR more than doubles the aggregate TCP goodput, from 9.28 to 18.93 Mbps (Wilcoxon p = 0.002, Cohen's d = 23.98). Hop-count SPF maps the four flows onto identical shortest paths and concentrates them on a single core link, whereas APR's load-aware placement distributes the flows across the four core switches, roughly doubling the usable bisection bandwidth..

Per-Scenario Analysis

As the number of concurrent flows increases (S2, eight TCP flows), the fabric becomes more uniformly loaded and the advantage narrows to +5.6%, which is not statistically significant ($p = 0.16$). Under the heavy-UDP scenario (S3) the offered load ($8 \times 9 = 72$ Mbps) far exceeds capacity, so both policies saturate: the aggregate rate, packet loss (≈70%), and jitter (≈0.75 ms) are statistically indistinguishable (loss $p = 0.92$, jitter $p = 0.63$). The mixed scenario (S4) shows the same pattern. Crucially, APR never degrades jitter or loss relative to SPF in any scenario, which is essential for QoS-sensitive traffic.

Control-Plane Overhead

Controller CPU was not instrumented in this study; APR's control-plane cost is limited to periodic per-port statistics polling every $\tau = 2$ s plus occasional flow updates, and a quantitative overhead characterization on larger topologies is left to future work.

Comparison with Prior Studies

Table 4 places the present work alongside several representative SDN-QoS studies. Because those studies use different controllers, topologies, and traffic, the comparison is indicative rather than head-to-head; the meaningful quantity here is the relative gain of APR over SPF on the same controller

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493

and topology, which reaches +104% for light TCP load and converges to the baseline under saturation

Table 4. Comparison with Prior Works on SDN QoS

Study	Topology	Ctrl.	Routing	Avg UDP
Abuabdallah [17]	Tree	ODL	Static	≈7.8
Hui et al. [19]	Tree/Linear	Multi	Static	7–8
Tahir et al. [8]	Custom+HTB	ODL	Static	≈8.0
R. et al. [16]	Custom	POX	Dynamic	≈8.3
Wiranata et al. [3]	Tree(3,2)	ODL	Static	8.79
This work	Fat-tree (k=4)	Ryu	APR	see Table 3

Source: (Research Results, 2026)

Technical Interpretation and Implications

The results delineate where adaptive path selection helps. When spare path diversity exists (light to moderate load), APR converts it into substantial throughput gains through load-aware initial placement; when the network is saturated, no routing policy can create capacity and APR safely matches the baseline. The hysteresis threshold governs a reactivity–stability trade-off: a smaller threshold increases responsiveness to congestion but, by migrating flows more often, raises jitter, whereas the conservative value used here (30%) preserves the throughput gains while keeping jitter and loss statistically identical to SPF. APR therefore behaves as a safe drop-in replacement for shortest-path forwarding that never performs worse on the measured QoS metrics.

Threats to Validity

Several limitations qualify these results. First, the evaluation uses Mininet emulation rather than physical hardware, so absolute values inherit the accuracy of tc-based link shaping; we mitigate this with deterministic shaping and receiver-side measurement. Second, APR's weights (α, β, γ) and the hysteresis threshold were set manually; automatic tuning is left to future work. Third, although the k=4 fat-tree (20 switches, 16 hosts) is substantially larger than our earlier single-path baseline, still larger and heterogeneous topologies remain to be studied, as does a direct measurement of controller CPU overhead. Finally, the throughput gains depend on available path diversity and therefore diminish under saturation. In particular, the present study compares APR only against the default shortest-path baseline on the same controller; a direct, controlled comparison with additional baselines such as ECMP and learning-based (deep-reinforcement-learning) routing is an important direction for future work.

CONCLUSION

This study designed, implemented, and evaluated Adaptive Path Routing (APR), a QoS-aware routing application for Software-Defined Networking built on the Ryu controller, which selects end-to-end paths by minimizing a normalized composite cost over residual bandwidth, one-way delay, and packet loss. On a k=4 fat-tree (20 switches, 16 hosts) emulated in Mininet, and compared against shortest-path-first forwarding on the same controller over ten repetitions per scenario, APR more than doubled aggregate TCP goodput under light load (+104.1%, $p<0.01$) by spreading flows across the core tier through load-aware path placement. Under saturation, where no routing policy can create capacity, APR matched the baseline; owing to a 30% hysteresis threshold it never increased jitter or packet loss, making it a safe drop-in replacement for shortest-path forwarding. The main insight is that the benefit of adaptive routing is governed by the available path diversity and by a reactivity–stability trade-off controlled by the hysteresis threshold. Practically, operators can deploy APR on a mainstream open-source controller to recover otherwise-wasted bisection bandwidth without degrading latency stability. Future work includes quantifying control-plane overhead, automatic tuning of the cost weights and hysteresis (e.g., via reinforcement learning), evaluation on larger and heterogeneous topologies, and validation on physical hardware.

REFERENCE

[1] V. A. Shirsath and M. M. Chandane, "Beyond the Basics: An In-Depth Analysis and Multidimensional Survey of Programmable Switch in Software-Defined Networking," *Int. J. Networked Distrib. Comput.*, vol. 13, no. 1, p. 8, Jun. 2025, doi: 10.1007/s44227-024-00049-6.

[2] M. Rostami and S. Goli-Bidgoli, "An overview of QoS-aware load balancing techniques in SDN-based IoT networks," *J. Cloud Comput.*, vol. 13, no. 1, p. 89, Apr. 2024, doi: 10.1186/s13677-024-00651-7.

[3] A. D. Wiranata, S. Sunardi, and I. Riadi, "Tinjauan Sistematis Quality Of Service Pada Layanan Jaringan Software Defined Networking," *Infotech J. Technol. Inf.*, vol. 11, no. 2, pp. 247–252, Nov. 2025, doi: 10.37365/jti.v11i2.422.

[4] A. Asruddin and A. D. Wiranata, "PERFORMANCE ANALYSIS OF THE OPENDAYLIGHT SDN CONTROLLER ON TREE

TOPOLOGY: THROUGHPUT, JITTER, AND PACKET LOSS," *INFOTECH J.*, vol. 11, no. 2, pp. 408–415, Dec. 2025, doi: 10.31949/infotech.v11i2.16053.

[5] S. Kaczmarek and J. A. Litka, "Impact of SDN Controller's Performance on Quality of Service," *IEEE Access*, vol. 12, pp. 8262–8282, 2024, doi: 10.1109/ACCESS.2024.3352433.

[6] V. Sai R., P. U., S. Ch., S. K. Panda, and G. V., "An Enhanced Dynamic Multi-Path Routing Algorithm for Software Defined Networks," *Procedia Comput. Sci.*, vol. 233, pp. 12–21, 2024, doi: 10.1016/j.procs.2024.03.191.

[7] S. Aouad, I. El Meghrouni, Y. Sabri, A. Hilmani, and A. Maizate, "Quality of services in software defined networking: challenges and controller placement problems," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 33, no. 2, p. 951, Feb. 2024, doi: 10.11591/ijeecs.v33.i2.pp951-959.

[8] M. Tahir, S. Hendra Sugara, and N. Estu Harsiwi, "Implementasi Quality of Service pada Manajemen Jaringan di SMKN Negeri 3 Bangkalan Menggunakan Teknologi Opendaylight Controller Menggunakan Metode HTB," *J. Ilm. Edutic Pendidik. Dan Inform.*, vol. 9, no. 1, pp. 63–76, Nov. 2022, doi: 10.21107/edutic.v9i1.17094.

[9] N. M. Yungaicela-Naula, C. Vargas-Rosales, J. A. Pérez-Díaz, and D. F. Carrera, "A flexible SDN-based framework for slow-rate DDoS attack mitigation by using deep reinforcement learning," *J. Netw. Comput. Appl.*, vol. 205, p. 103444, Sep. 2022, doi: 10.1016/j.jnca.2022.103444.

[10] M. K. Mohammed, M. K. Awad, E. M. Alotaibi, and R. Mohammadi, "An Implementation of Deep Reinforcement Learning-Based Routing Framework for Open-Network Operating System-Controlled and Mininet-Emulated Software-Defined Networking," *IET Netw.*, vol. 14, no. 1, p. e70016, Jan. 2025, doi: 10.1049/ntw2.70016.

[11] L. P. Aguirre Sanchez, Y. Shen, and M. Guo, "MDQ: A QoS-Congestion Aware Deep Reinforcement Learning Approach for Multi-Path Routing in SDN," *J. Netw. Comput. Appl.*, vol. 235, p. 104082, Mar. 2025, doi: 10.1016/j.jnca.2024.104082.

[12] Y. Wang, M. Othman, W. O. Choo, R. Liu, and X. Wang, "DFRDRL: a dynamic fuzzy routing algorithm based on deep reinforcement learning with guaranteed latency and bandwidth for software-defined networks," *J. Big Data*, vol. 11, no. 1, p. 150, Oct. 2024, doi: 10.1186/s40537-024-01029-x.

[13] Y. He, G. Xiao, J. Zhu, T. Zou, and Y. Liang, "Reinforcement learning-based SDN routing scheme empowered by causality detection and GNN," *Front. Comput. Neurosci.*, vol. 18, p. 1393025, Apr. 2024, doi: 10.3389/fncom.2024.1393025.

[14] W. Jiang *et al.*, "Graph Neural Networks for Routing Optimization: Challenges and Opportunities," *Sustainability*, vol. 16, no. 21, p. 9239, Oct. 2024, doi: 10.3390/su16219239.

[15] S. Faezi and A. Shirmarz, "A Comprehensive Survey on Machine Learning using in Software Defined Networks (SDN)," *Hum.-Centric Intell. Syst.*, vol. 3, no. 3, pp. 312–343, Jun. 2023, doi: 10.1007/s44230-023-00025-3.

[16] N. Gupta, M. S. Maashi, S. Tanwar, S. Badotra, M. Aljebreen, and S. Bharany, "A Comparative Study of Software Defined Networking Controllers Using Mininet," *Electronics*, vol. 11, no. 17, p. 2715, Aug. 2022, doi: 10.3390/electronics11172715.

[17] A. G. Abuabdallah, "Performance-based evaluation of SDN controllers: OpenDayLight and RYU," *J. Electr. Syst. Inf. Technol.*, vol. 13, no. 1, p. 35, Apr. 2026, doi: 10.1186/s43067-026-00343-z.

[18] D. J. Hamad, K. G. Yalda, and N. ÈšÄfpuÈ™, "Performance Assessment of QoS metrics in Software Defined Networking using Floodlight Controller," *JOIV Int. J. Inform. Vis.*, vol. 7, no. 3, p. 631, Sep. 2023, doi: 10.30630/joiv.7.3.1288.

[19] D. Hui, S. H. Wei, L. O. Bi, X. Zhu, and S.-K. Yoon, "Performance Evaluation of Ryu, OpenDayLight and Floodlight Controllers in Diverse Software-Defined Networking Topologies," *J. Adv. Res. Des.*, vol. 132, no. 1, pp. 103–114, May 2025, doi: 10.37934/ard.132.1.103114.

[20] M. N. A. Sheikh, I.-S. Hwang, M. S. Raza, and M. S. Ab-Rahman, "A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers via Mininet Emulator," *Computers*, vol. 13, no. 4, p. 85, Mar. 2024, doi: 10.3390/computers13040085.

[21] A. A. Bahashwan, M. Anbar, S. Manickam, T. A. Al-Amiedy, M. A. Aladaileh, and I. H. Hasbullah, "A Systematic Literature Review on Machine Learning and Deep Learning Approaches for Detecting DDoS Attacks in Software-Defined Networking," *Sensors*, vol. 23, no. 9, p. 4441, May 2023, doi: 10.3390/s23094441.

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8493