

Container-based Machine Learning-Driven Routing Framework for Adaptive Traffic Management in Real-World SDN Environments

Mariem Bouguerra
REGIM-Lab, ENETCOM
University of Sfax, Sfax, Tunisia
bouguerra.mariem94@gmail.com

Sofiane Khalfallah
PRINCE-Lab, ISSAT
University of Sousse, Sousse, Tunisia
sofiane.khalfallah@gmail.com

Hela Ltifi
REGIM-Lab, ENIS
University of Sfax, Sfax, Tunisia
hela.ltifi@gmail.com

Abstract—Software-Defined Networking (SDN) decouples the control plane from the data plane, enabling centralized programmability and paving the way for more intelligent, adaptive traffic management strategies. This paper presents MLRouter, a machine learning-based routing framework that enhances path selection by jointly considering traffic classification outputs and real-time network conditions. Unlike the majority of existing approaches, which are validated exclusively in simulation environments, the proposed solution is implemented and evaluated on a realistic experimental SDN testbed comprising virtual machines (VMs) running under KVM/QEMU, Open vSwitch (OVS) instances, and the Faucet SDN controller, with observability provided by Gauge and Prometheus.

The system is built incrementally across three layers: (i) a REST API-driven static routing layer that allows programmatic manipulation of Faucet routes without manual YAML editing; (ii) a classical dynamic routing layer based on OSPF implemented through FRRouting (FRR) running in Docker containers; and (iii) a Random Forest classifier that replaces OSPF path decisions by leveraging real-time telemetry. The ML module interacts with the Faucet controller through a RESTful API to apply routing updates dynamically, with an automatic fallback to OSPF in case of model failure or low prediction confidence.

Experimental results demonstrate that MLRouter reduces packet loss by approximately 50–60%, decreases end-to-end latency by roughly 30%, and increases throughput by 20–25% relative to OSPF, across varying traffic intensities and congestion scenarios. These gains confirm that integrating machine learning intelligence with SDN programmability constitutes a viable and effective pathway toward adaptive, efficient, and real-time routing in realistic networking environments.

Index Terms—Machine Learning, SDN, Docker-containers, FRRouting, REST-API OSPF.

I. INTRODUCTION

THE rapid proliferation of digital services — including mobile applications, real-time communication platforms, video streaming, and Internet-of-Things (IoT) deployments — has led to a sustained and dramatic increase in both the volume and heterogeneity of network traffic. This evolution places mounting pressure on modern network infrastructures, which must operate under increasingly dynamic and often unpredictable conditions where traffic patterns and service demands fluctuate continuously.

Traditional IP routing mechanisms are fundamentally limited in their ability to adapt to such environments.

In particular, they struggle to efficiently manage congestion, optimize resource utilization, and consistently guarantee Quality of Service (QoS) requirements in large-scale, heterogeneous networks [1]. Software-Defined Networking (SDN) has emerged as a transformative architectural paradigm to address these constraints. By splitting the control plane from the data plane, SDN enables centralized visibility and programmable control of forwarding behaviors.

It also allows the implementation of sophisticated traffic management policies. [2], [3].

Despite these advantages, many SDN-based routing solutions still rely on predefined, rule-based policies or simple heuristic strategies, limiting their responsiveness to real-time network variations [4].

Concurrently, traditional dynamic routing protocols such as Open Shortest Path First (OSPF) remain widely deployed, particularly in legacy and hybrid environments. [5]. Although OSPF efficiently computes shortest paths using link-state information, it operates without awareness of flow-level characteristics.

It also does not consider application-specific requirements, which can yield suboptimal performance under heterogeneous traffic.

To overcome these limitations, recent research has explored the integration of Machine Learning (ML) into network control and management tasks [6]. ML-based approaches enable systems to learn from historical measurements and support more adaptive, data-driven decision-making. However, most existing works focus on traffic classification and anomaly detection, with limited attention to routing optimization itself [7]. Furthermore, the vast majority of proposed solutions are validated exclusively in simulation or emulation environments, which may not fully capture the complexity of real-world network operation.

In this context, the present paper adopts and substantially extends the MLRouter framework [11] by implementing it on a realistic experimental testbed composed of KVM/QEMU

virtual machines, Open vSwitch (OVS), and the Faucet SDN controller. The system is built in three incremental layers — REST-based static routing, dynamic OSPF via FRR, and ML-driven routing via Random Forest — enabling a rigorous and principled comparison. A key safety mechanism ensures automatic fallback to OSPF whenever the model confidence is insufficient or telemetry is unavailable, which is a design consideration absent from most related works.

The main contributions of this paper are as follows:

- A three-layer incremental architecture integrating REST-based static routing, OSPF dynamic routing, and ML-driven routing within the same real SDN testbed.
- A reproducible experimental methodology with standardized traffic generation (iperf3, ping), Prometheus-based telemetry collection, and multiple evaluation scenarios including nominal load, progressive congestion, and link failure.
- Empirical evidence from a real KVM-based testbed demonstrating consistent and significant performance improvements of the Random Forest routing policy over OSPF across packet loss, latency, and throughput.
- A safety-aware deployment design incorporating shadow mode validation, prediction confidence thresholding, hysteresis, and automatic OSPF fallback.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the proposed system architecture and methodology. Section IV discusses the experimental evaluation and results. Section V concludes the paper.

II. RELATED WORK

Research at the intersection of SDN and machine learning has expanded rapidly. This section briefly surveys the most relevant works and positions the present contribution relative to the literature.

SDN control frameworks. Zhang et al. [8] introduce Ryu, a high-level northbound API for SDN applications, evaluated mainly in Mininet with limited physical tests; it incorporates no machine learning or adaptive routing. **SDN routing approaches.** Pereira and Sousa [9] propose dynamic path selection based on predefined rules and load metrics, but entirely in simulation, with no ML model and no comparison against OSPF.

ML-based traffic classification. Jayasuriya et al. [10] and Parsaei et al. [7] apply SVM/Random Forest and Naïve Bayes classifiers in simulated or emulated SDN settings with high accuracy, but neither integrates classification outputs into actual routing decisions. **ML-based routing.** The most directly related prior work is MLRouter [11] itself, which uses a Random Forest classifier to select routing paths based on traffic type and congestion state, evaluated against OSPF purely in simulation; its real-network performance remains

unvalidated, which the present paper directly addresses. **ML-based network modeling.** Güemes-Palau et al. [12] propose RouteNet-Gauss, a GNN trained on hardware-testbed data for performance *prediction*, but it does not implement active routing nor compare against standard protocols.

ML for network resilience and congestion control. Danilchenko et al. [13] propose an online learning framework for predicting radio link failures in flying ad-hoc networks, addressing — in a different network context — the same convergence/anticipation problem that motivates our proactive rerouting approach. Maliha et al. [14] survey machine learning versus classical approaches for congestion control and scheduling in multipath TCP, reinforcing the broader trend, also observed in our own results, of ML-based methods outperforming classical protocol-level mechanisms under congestion.

Positioning. The literature reveals two persistent gaps: (i) most ML-based routing systems are validated exclusively in simulation, and (ii) direct comparative evaluations against standard protocols in real testbeds are rare. This work addresses both gaps simultaneously, as summarized in Table I.

TABLE I
COMPARISON OF RELATED WORKS

Work	ML	Real Testbed	Routing	vs. OSPF
[8]	No	Partial	No	No
[9]	No	No	Yes	No
[7]	Yes	No	No	No
[10]	Yes	No	No	No
[12]	Yes	Partial	No	No
[11]	Yes	No	Yes	Yes
[13]	Yes	No	No	No
[14]	Yes	No	No	No
Ours	Yes	Yes	Yes	Yes

III. PROPOSED METHODOLOGY

A. Context and Design Objectives

This work builds upon the MLRouter framework [11] and extends it by integrating it into a real experimental infrastructure. The overarching objective is to determine whether a data-driven routing policy, informed by real-time telemetry, can consistently outperform a standards-based OSPF policy under realistic and reproducible network conditions.

A key design principle is *incrementality*: the system is constructed in three layers, each adding a layer of capability while retaining the components of previous layers. This allows clear attribution of performance changes to specific architectural decisions and provides a stable fallback path at each stage. Another key principle is *safety*: the ML-driven routing module operates initially in shadow mode — logging decisions without applying them — and transitions to closed-loop control only after shadow mode validation, with automatic reversion to OSPF in case of anomaly.

B. Testbed Infrastructure

The experimental environment consists of a single Ubuntu/Debian host machine running all network components.

¹<https://github.com/openvswitch/ovs>

²<https://github.com/FRRouting/frr>

The host is equipped with standard server-grade hardware and all components are deployed using open-source tools.

The OVS switch (switch0) is configured with a fixed datapath ID and connected to the Faucet controller via OpenFlow 1.3 on TCP port 6653. Four TAP interfaces connect the switch to the four virtual machines. Three VMs (VM-H1, VM-H2, VM-H3) are placed in the *office* VLAN (VID 100, subnet 10.0.100.0/24) and one VM (VM-H4) in the *guest* VLAN (VID 200, subnet 10.0.200.0/24). Faucet provides Layer-3 virtual IPs (`faucet_vips`) for inter-VLAN routing.

C. Layer 1 — REST API-Driven Static Routing

1) *Principle*: The first layer establishes programmatic control over Faucet’s routing table without requiring manual YAML editing. A lightweight REST API (implemented in FastAPI) manages a dedicated `routes.yaml` file included by the main `faucet.yaml` configuration. This layer serves two purposes: (i) it validates that all infrastructure components operate correctly in coordination; (ii) it establishes a stable, reproducible baseline for subsequent dynamic experiments.

2) *API Workflow*: The routing update workflow follows a strict, transactional sequence to ensure network stability at all times:

- 1) A `POST /routes` request is submitted with a JSON body specifying the destination prefix (`ip_dst`), next-hop address (`ip_gw`), and optionally a target VLAN identifier.
- 2) The API validates the JSON schema, checks route uniqueness, and verifies that the referenced VLAN exists in the current topology configuration.
- 3) The `routes.yaml` file is updated under a file lock, and a backup of the previous version is preserved.
- 4) `check_faucet_config` is executed; if validation fails, the system immediately rolls back to the last valid configuration.
- 5) If validation succeeds, Faucet is reloaded via `kill -HUP` or `systemctl reload faucet`.
- 6) The applied routing decision is logged with a timestamp for auditability.

This static routing layer highlights the inherent limitations of non-adaptive configurations: routes remain fixed regardless of traffic conditions, motivating the introduction of dynamic and ML-based mechanisms in subsequent layers.

D. Layer 2 — Dynamic Routing with OSPF via FRRouting

1) *Architecture*: The second layer replaces the manual REST-based route management with a standards-based dynamic routing protocol. An important architectural constraint is that Faucet does not natively support OSPF. The correct integration approach is therefore to deploy FRRouting (FRR) as an external routing service that forms OSPF adjacencies, computes shortest paths via the Dijkstra SPF algorithm, and redistributes learned routes toward Faucet through a supported redistribution mechanism (e.g., BGP peering between FRR and Faucet).

FRR runs inside Docker containers to ensure isolation and reproducibility. The `ospfd` and `bgpd` daemons are enabled in the FRR daemon configuration. OSPF adjacencies are formed over the networks corresponding to the office and guest VLANs, and selected routes are redistributed toward Faucet, which installs them as OpenFlow rules in OVS.

2) *Convergence Validation*: Link failure and degradation scenarios are tested using standard Linux traffic control tools. A link failure is simulated by taking an interface administratively down (`ip link set down`), and degradation is induced using `tc netem` to inject delay, jitter, loss, and rate limiting. The convergence behavior of OSPF is then observed through FRR’s LSDB updates and the subsequent update of Faucet flow rules.

This OSPF-based configuration serves as the primary performance baseline for layer 3: all routing decisions are based exclusively on static link costs and topology state, without any awareness of real-time traffic load or application-level requirements.

E. Layer 3 — ML-Driven Routing via Random Forest

1) *Data Collection and Feature Engineering*: The Random Forest classifier is trained on telemetry data collected from the testbed itself using Prometheus, Gauge, OVS port statistics, and `node_exporter`. Metrics are aggregated over a sliding window of 5–10 seconds to produce one feature vector per candidate path per window. The feature vector for each sample comprises the following attributes:

- **Protocol type**: categorical encoding of the application protocol (HTTP, FTP, SSH, DNS, SMTP).
- **Packet size**: mean and standard deviation of packet size per flow.
- **Inter-arrival time**: mean inter-packet arrival time, indicative of burstiness.
- **Link utilization**: current bandwidth occupancy of candidate paths in bits per second.
- **RTT and RTT p95**: mean and 95th-percentile round-trip time per path.
- **Packet loss rate**: observed drop fraction per candidate path.
- **Jitter**: UDP jitter measured by `iperf3`.
- **Interface errors and drops**: OVS port-level counters.

Several selected features directly represent Quality of Service (QoS) conditions, including delay, packet loss, jitter, and link utilization. These metrics allow routing decisions to account for current service quality rather than relying solely on shortest-path computation.

The target label is either the identifier of the optimal next-hop observed during data collection (classification mode) or the predicted future cost of each candidate path (regression mode). The classification formulation is adopted in this work, as it maps directly to the next-hop selection required by Faucet.

2) *Model Training and Validation*: The Random Forest ensemble is configured with 100 decision trees and operates with the standard Gini impurity criterion. A stratified 5-fold cross-validation procedure is used during training to mitigate

overfitting and ensure reliable generalization across traffic scenarios. Random Forest was selected because it offers a favorable balance between predictive accuracy, computational efficiency, robustness to noisy telemetry data, and model interpretability. These characteristics make it particularly suitable for real-time routing decisions in SDN environments where both responsiveness and explainability are important. The model is evaluated using accuracy and per-class F1-score, and feature importance scores are retained to maintain interpretability.

To account for traffic distribution shifts over time, the model is periodically retrained using newly collected measurements. This incremental retraining strategy improves prediction stability while maintaining robustness to topology and workload changes.

3) Deployment: Shadow Mode and Closed-Loop Control:

The ML module is deployed in two successive operational modes.

In *shadow mode*, the classifier generates routing decisions and logs them alongside the decisions that OSPF would have made, but does not apply any changes to the network. This layer enables validation of the model's decisions against observed ground truth without any risk of service disruption.

After shadow mode validation, the system transitions to *closed-loop control*: the classifier's predictions are applied to Faucet via the REST API, subject to the following safety conditions:

- 1) The model prediction confidence must exceed a defined threshold.
- 2) The decision must remain stable across at least two consecutive measurement windows (hysteresis).
- 3) The current path must violate at least one SLO threshold (latency, loss, or utilization).

If telemetry collection fails, the model cannot be loaded, or none of the safety conditions are met, the system automatically reverts to the last validated OSPF-based routing state. This fallback mechanism ensures that MLRouter never degrades network performance below the OSPF baseline.

4) *MLRouter-Faucet Communication*: For each routing decision, the ML module submits an HTTP POST request to the Faucet REST API containing the target IP prefix, the predicted optimal next-hop address, and optionally a VLAN identifier. Faucet translates this instruction into an OpenFlow flow rule and installs it on the target OVS switch without requiring any modification of the static YAML configuration. The full cycle — telemetry collection, feature extraction, prediction, API call, and rule installation — is designed to complete within a single measurement window of 5–10 seconds.

F. System Architecture Overview

The proposed architecture is organized into three functional planes. The **application plane** hosts the MLRouter module (Random Forest classifier, REST client, shadow/closed-loop controller) and the Faucet REST API layer. The **control plane** is managed by the Faucet SDN controller, which maintains

centralized visibility of the OVS topology and translates routing decisions into OpenFlow rules. The **data plane** consists of OVS switch0 interconnected to the four KVM virtual machine hosts via TAP interfaces. FRRouting runs alongside this architecture as an independent routing service, providing both the OSPF baseline and a fallback mechanism.

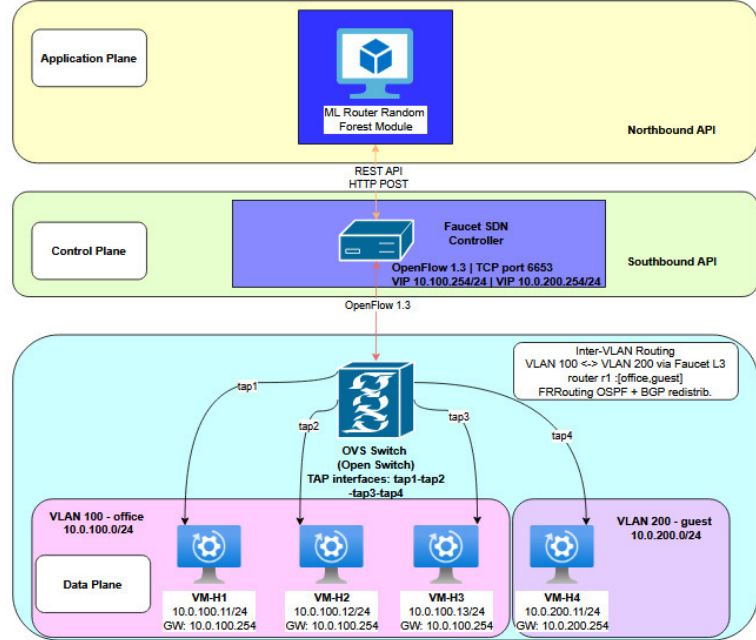


Fig. 1. System Architecture

IV. PERFORMANCE EVALUATION

A. Experimental Setup and Traffic Generation

Both routing configurations — MLRouter and OSPF/FRR — are evaluated on the same physical infrastructure and under identical network conditions. The OSPF baseline uses FRR with default link costs and no traffic awareness. The MLRouter configuration uses the closed-loop Random Forest controller, with automatic fallback to OSPF enabled.

Network traffic is generated directly by the KVM virtual machines. Heterogeneous workloads are produced to emulate realistic mixed application scenarios:

- **TCP bulk transfer** (HTTP/FTP emulation): `iperf3 -c <dst> -t 60 -i 1 -J` for 60-second sessions with 1-second reporting intervals.
- **UDP constant-rate traffic** (real-time application emulation): `iperf3 -u -b 50M -t 60 -i 1 -J` for jitter and loss measurement.
- **ICMP probing**: `ping -i 0.2 -c 300` for continuous RTT monitoring.

Traffic intensity is progressively increased by increasing the offered load in steps, enabling evaluation under light, moderate, and heavy congestion conditions. Link degradation is injected using `tc netem` (20 ms delay, 5 ms jitter, 1% loss, 50 Mbit/s rate cap) to simulate link saturation. Link failures

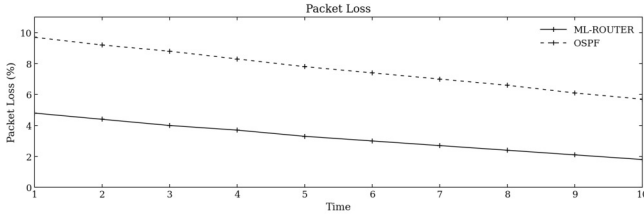


Fig. 2. Packet Loss Comparison Curve

are triggered by taking the relevant interface administratively down for 20 seconds and then restoring it.

B. Evaluation Metrics

Three primary metrics are evaluated, all collected directly from the experimental environment:

- **Packet loss:** the fraction of transmitted packets that fail to reach the destination, monitored via iperf3 JSON output and OVS port drop counters.
- **End-to-end latency:** the mean round-trip time (RTT) measured by continuous ICMP probing, reflecting the responsiveness of each routing strategy under current network conditions.
- **Throughput:** the rate of successfully delivered data (goodput) measured in packets per second by iperf3, indicating effective bandwidth utilization.

Each reported curve represents the average over multiple experimental runs conducted under the same scenario, ensuring statistical reliability of the reported results.

C. Results and Discussion

1) *Packet Loss:* It is worth noting that OSPF does not incorporate any learning mechanism. The observed variations in packet loss and latency are not caused by route adaptation but rather by transient network effects, including TCP congestion-window evolution, ARP/MAC table stabilization, and progressively changing congestion conditions during the experiments.

Figure 2 presents the packet loss comparison over the observation period. OSPF/FRR begins at approximately 9.8% and gradually decreases to around 5.8%, reflecting its limited ability to respond to evolving congestion. MLRouter, by contrast, maintains significantly lower values throughout, starting at approximately 4.8% and declining steadily to below 2%. This corresponds to an overall reduction of approximately 50–60% relative to OSPF.

This improvement is attributed to the Random Forest classifier's ability to recognize characteristic traffic patterns and proactively redirect flows before severe packet loss occurs. OSPF reacts only after link costs change or adjacencies reconverge, introducing a delay during which packets are dropped or queued beyond their deadline. MLRouter's data-driven approach circumvents this convergence latency by anticipating congestion based on observed telemetry features.

2) *Latency:* Figure 3 illustrates the latency evolution over the observation period. OSPF/FRR ranges from approximately 92 ms down to 67 ms, while MLRouter achieves a consistently lower and smoother trajectory, decreasing from approximately 68 ms to 45 ms — an average reduction of approximately 30%.

The smoother profile of MLRouter's latency curve is particularly noteworthy: it reflects the stability of the adaptive rerouting mechanism. By preemptively shifting flows away from congested paths, MLRouter avoids the transient latency spikes that arise in OSPF during periods of link renegotiation or topology reconvergence. This characteristic is especially valuable for delay-sensitive applications such as real-time voice and video, where latency spikes — even brief ones — are perceptible and disruptive.

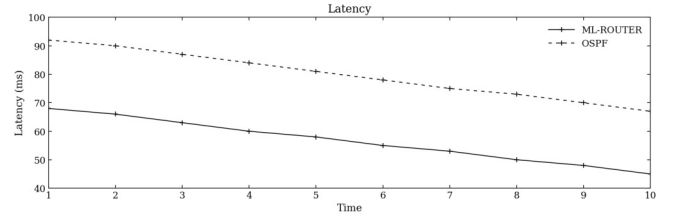


Fig. 3. Latency comparison curve

3) *Throughput:* Figure 4 shows the throughput evolution for both approaches. MLRouter exhibits a continuous and consistent increase from approximately 110 packets per second (pps) to nearly 195 pps. OSPF/FRR also improves, but more modestly, from approximately 95 pps to 158 pps. The resulting throughput advantage of MLRouter amounts to approximately 20–25% relative to the OSPF baseline.

This gain reflects MLRouter's ability to distribute traffic across multiple available paths based on real-time utilization information, rather than committing all flows to a single shortest path regardless of its current load. The widening performance gap at higher load levels confirms that the benefit of intelligent load distribution becomes increasingly significant as traffic volume grows — precisely the conditions under which traditional protocols are most constrained.

4) *Summary:* Table II consolidates the quantitative performance improvements achieved by MLRouter relative to OSPF across all three metrics.

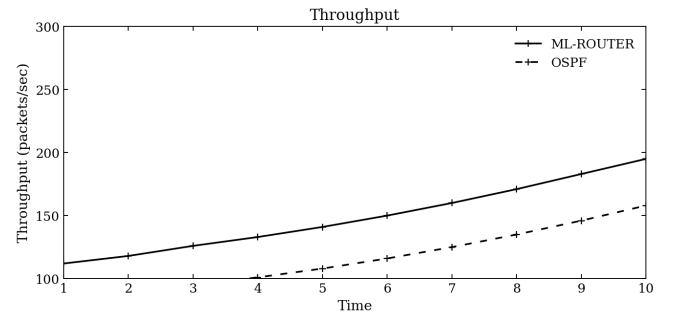


Fig. 4. Throughput Comparison Curve

TABLE II
PERFORMANCE IMPROVEMENT OF MLROUTER OVER OSPF/FRR

Metric	OSPF Range	MLRouter Range	Improvement
Packet Loss	5.8–9.8%	2.0–4.8%	≈50–60%
Latency	67–92 ms	45–68 ms	≈30%
Throughput	95–158 pps	110–195 pps	≈20–25%

These results confirm that MLRouter consistently and significantly outperforms classical OSPF-based routing in a real SDN testbed, across all three evaluated dimensions.

5) *Discussion of Limitations:* Several limitations of the present evaluation must be acknowledged. First, the experimental testbed remains relatively small in scale, consisting of four VM hosts and a single OVS switch. While the results consistently demonstrate improvements in packet loss, latency, and throughput, the scalability of the proposed approach to larger and more complex topologies—including multiple switches, a higher number of hosts, and a greater volume of concurrent flows—remains to be validated.

Second, the current evaluation was conducted over a limited observation period, and the number of independent experimental runs as well as associated confidence intervals have not yet been reported. This introduces a methodological limitation that may affect the statistical robustness of the results and will be addressed in future work through extended and repeated experiments over longer durations.

Finally, the current evaluation does not yet quantify the computational overhead introduced by the machine learning component, including CPU usage, memory consumption, and API round-trip latency. These aspects are essential for deployment-oriented assessment and will be investigated in future studies. In addition, reinforcement learning techniques will be explored to enhance adaptability in highly dynamic network environments and to further improve long-term operational behavior under diverse traffic conditions.

V. CONCLUSION

This paper presented MLRouter, an intelligent SDN-based routing framework that incorporates a Random Forest classifier for real-time, traffic-aware path selection. The system was designed, implemented, and evaluated in three incremental layers — REST-based static routing, OSPF dynamic routing via FRRouting, and ML-driven routing.

All layers were deployed on a realistic experimental testbed comprising KVM virtual machines, Open vSwitch, and the Faucet SDN controller with Prometheus-based telemetry. Unlike most existing works relying on simulation environments, this study provides a more practically grounded validation by conducting all experiments in a real network environment.

Experimental results demonstrate that MLRouter consistently outperforms OSPF/FRR in packet loss (50–60% reduction), latency (≈30% reduction), and throughput (≈20–25% increase). These improvements are primarily attributed to the classifier’s ability to anticipate congestion from real-time

telemetry and apply proactive rerouting before performance degradation occurs.

This capability fundamentally exceeds what link-state-based protocols can achieve within their convergence constraints. The safety-aware deployment design — including shadow mode validation, confidence thresholding, hysteresis, and automatic OSPF fallback — ensures that the system never degrades below the conventional routing baseline.

Future work will investigate reinforcement learning approaches capable of continuously adapting routing policies through interaction with changing network conditions. Such approaches may provide improved adaptability in highly dynamic SDN environments where traffic characteristics evolve over time.

This aims to bring the system closer to production-scale applicability.

REFERENCES

- [1] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*. Elsevier, 2007.
- [2] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmoly, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [4] V. Srivastava and R. S. Pandey, “Load balancing for software-defined network: A review,” *International Journal of Computers and Applications*, vol. 44, no. 8, pp. 746–759, 2022.
- [5] J. Moy, “OSPF Version 2,” RFC 2328, Internet Engineering Task Force (IETF), Apr. 1998. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc2328>
- [6] S. Faezi and A. Shirmarz, “A comprehensive survey on machine learning usage in software-defined networks (SDN),” *Human-Centric Intelligent Systems*, vol. 3, no. 3, pp. 312–343, 2023.
- [7] M. R. Parsaei, M. J. Sobouti, R. Javidan et al., “Network traffic classification using machine learning techniques over software-defined networks,” *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 7, 2017.
- [8] S. Zhang, Y. Shen, M. Herlich et al., “Ryu: Using a high-level northbound API for control messages in software-defined network,” in *Proc. 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2015, pp. 115–120.
- [9] G. Pereira and P. Sousa, “A routing approach in SDN environments,” in *Proc. 17th Iberian Conference on Information Systems and Technologies (CISTI)*, 2022, pp. 1–6.
- [10] M. P. K. Perera Jayasuriya, K. Piamrat, and S. Hamma, “Network traffic classification using machine learning for software-defined networks,” in *Machine Learning for Networking (MLN 2019)*, Springer, 2020, pp. 28–39.
- [11] M. Bouguerra, S. Khalfallah, and H. Ltifi, “MLRouter: A learning-based traffic classification within congestion-aware link-state routing protocol in SDN,” in *Proc. International Conference on Innovation in Networking and Intelligent Systems (INISTA)*, 2025.
- [12] C. Güemes-Palau, M. Ferriol-Galmés, J. Paillisse-Vilanova et al., “RouteNet-Gauss: Hardware-enhanced network modeling with machine learning,” *IEEE Transactions on Networking*, 2026.
- [13] K. Danilchenko, N. Lazmi, and M. Segal, “Online learning framework for radio link failure prediction in FANETs,” in *Proceedings of the 18th Conference on Computer Science and Intelligence Systems (FedCSIS)*, Warsaw, Poland, 2023, ser. Annals of Computer Science and Information Systems, vol. 35, pp. 41–48.
- [14] M. Maliha, G. Habibi, and M. Atiquzzaman, “A survey on congestion control and scheduling for multipath TCP: Machine learning vs classical approaches,” in *Proceedings of the 18th Conference on Computer Science and Intelligence Systems (FedCSIS)*, Warsaw, Poland, 2023, ser. Annals of Computer Science and Information Systems, vol. 35, pp. 49–61.