

Predictive State-Staging: A Decision-Making Framework for Proactive Fault Recovery in SDN

Matheswaran Marimuthu^{1,*}, Sivakumar Balu²

¹ IEEE Member, San Jose, United States;

² IEEE Senior Member, San Jose, United States

Email: ¹ matheswaranm@gmail.com, ² siva.balu@ieee.org

*Corresponding Author

Abstract—Modern Software-Defined Networks (SDN) are often constrained by the inherent latency of reactive fault recovery and the memory overhead associated with proactive protection schemes. This paper introduces Predictive State-Staging (PSS), a framework that leverages multi-modal data fusion and Long Short-Term Memory (LSTM) networks to predict network failures before they occur. By integrating asynchronous software logs with synchronous gNMI telemetry and traffic fluctuations, PSS assigns a real-time Probability of Failure (PoF) to network paths. Upon exceeding a critical threshold, the framework initiates “shadow flow table” staging—a just-in-time installation of backup forwarding states using OpenFlow group tables and VLAN-based aggregation. Experimental evaluations on a 14-switch, 36-host testbed indicate that PSS achieves sub-5 ms recovery times, significantly outperforming traditional reactive mechanisms while maintaining a 45% reduction in TCAM occupancy compared to static proactive methods; these results are validated at this testbed scale, and this revision adds confidence intervals, significance tests, and an explicit discussion of the scope of these claims (Section VI). This approach aligns with the stringent Ultra-Reliable Low-Latency Communications (URLLC) requirements of 5G and mission-critical IoT infrastructures.

Keywords—Software-Defined Networking (SDN), Fault Recovery, Long Short-Term Memory (LSTM), Predictive Maintenance, Shadow Flow Tables, URLLC, OpenFlow, gNMI Telemetry, Fast-Failover.

I. INTRODUCTION

Software-Defined Networking (SDN) decouples control and data planes to enable flexible network management, but this separation introduces a “Detection-to-Correction” latency bottleneck during failures. In conventional SDN deployments, switches notify the controller of link or node failures via PORT_STATUS messages, which then recomputes and installs new paths. This reactive cycle often exceeds the 50 ms carrier-grade restoration bound, causing packet loss and service disruption in time-sensitive applications [1]–[3].

Empirical studies confirm that controller-driven OpenFlow restoration struggles to meet sub-50 ms targets in large-scale topologies due to millisecond-scale flow insertion and tens-to-hundreds of milliseconds of failure detection

latency [4], [5]. These limitations are particularly problematic for 5G/6G Ultra-Reliable Low-Latency Communication (URLLC) services, which demand sub-1 ms to sub-10 ms recovery and five-nines availability [6]. Proactive schemes mitigate detection and recomputation delays by pre-installing backup paths in the data plane, but they incur substantial Ternary Content-Addressable Memory (TCAM) overhead and often lack mechanisms to adapt to dynamic traffic, leading to congestion or over-provisioning [7], [8], [9], [10].

A. Motivation and Research Gap

Existing work addresses either prediction or restoration, but rarely couples the two. Machine-learning-based predictors can anticipate anomalies with high accuracy, yet typically focus on a single data modality and stop at raising alarms rather than triggering concrete data-plane actions [11], [12], [13], [14]. In parallel, fast fail-over and proactive restoration mechanisms achieve millisecond-level recovery, but at the cost of significant, statically reserved TCAM state that grows quickly with network size and failure scenarios [15], [16], [17], [18], [19], [20].

This leads to four key gaps: (i) a lack of multi-modal fusion of logs, telemetry, and traffic signals for robust failure prediction; (ii) a prediction-action gap where predictions are not translated into automated data-plane staging; (iii) static proactive schemes that waste TCAM by pre-installing all possible backup paths; and (iv) limited integration of prediction confidence into recovery decisions. A fifth, methodological gap is addressed directly in this revision: prior evaluations of frameworks in this space, including the original submission of this paper, rarely report confidence intervals or significance tests alongside mean recovery-time comparisons, and rarely state explicitly how far their conclusions should be expected to generalize beyond the tested topology.

B. Contributions

This paper introduces Predictive State-Staging (PSS), a framework that bridges prediction and action through just-in-time backup path installation. The primary contributions are:



Received: 15-4-2026

Revised: 19-9-2026

Published: 31-12-2026

Multi-modal failure prediction: An LSTM-based predictor that fuses asynchronous software logs, synchronous gNMI telemetry (optical power, temperature, error counters), and traffic statistics to compute real-time Probability of Failure (PoF) for each network path.

Threshold-driven shadow staging: A decision engine that triggers “shadow flow table” installation when PoF exceeds a critical threshold ($PoF_{crit} = 0.75$), using OpenFlow group tables with fast-failover buckets and VLAN-based path aggregation to minimize TCAM usage.

Hybrid recovery architecture: A system that combines predictive staging for high-risk paths with reactive recovery for unpredicted failures, ensuring comprehensive fault coverage while maintaining low memory overhead.

Experimental validation: Evaluation on a 14-switch, 36-host fat-tree-inspired testbed demonstrating sub-5 ms recovery times, 45% TCAM reduction compared to static proactive methods, and 92% prediction accuracy with 8% false positives.

Statistically grounded and scope-bounded evaluation (new in this revision): A consolidated results table, 95% confidence intervals and significance tests for the recovery-time comparison, an explicit derivation of the topology and traffic-matrix parameters underlying the TCAM analysis, and an explicit discussion of the scale at which our conclusions are validated (Section VI).

The remainder of this paper is organized as follows: Section II reviews related work in SDN fault recovery and predictive maintenance. Section III details the PSS architecture, including the multi-modal predictor, decision engine, and shadow staging mechanism. Section IV describes the experimental setup and methodology. Section V presents results and comparative analysis. Section VI discusses the limitations and scope of the evaluation. Section VII concludes with future research directions.

II. RELATED WORK

SDN fault recovery research spans reactive, proactive, and hybrid approaches, with emerging interest in machine-learning-based prediction.

A. Reactive Fault Recovery

Reactive schemes rely on the controller to detect failures and recompute paths. Traditional OpenFlow restoration uses `PORT_STATUS` messages to trigger path recalculation, but this incurs 50–200 ms latency due to detection delays, controller processing, and flow installation overhead [1], [2]. Studies show that even optimized reactive approaches struggle to meet carrier-grade 50 ms targets in large topologies [21], [22]. Fast rerouting techniques reduce latency by pre-computing backup paths at the controller, but still require southbound communication to install new rules after failure detection [15].

B. Proactive Fault Recovery

Proactive methods pre-install backup paths in the data plane to eliminate controller involvement during failures. OpenFlow group tables with fast-failover buckets enable sub-millisecond local switching when primary paths fail [17],

[18]. However, static proactive schemes suffer from TCAM explosion as they must pre-install rules for all possible failure scenarios [4], [23]. For an n -node network with k -disjoint paths per flow, TCAM occupancy scales as $O(n^2k)$, quickly exhausting switch memory in large deployments. Recent work addresses this through TCAM-aware path selection and rule aggregation [23], but these approaches remain static and cannot adapt to dynamic failure probabilities [3], [10], [12].

C. Hybrid and Adaptive Approaches

Hybrid schemes combine reactive and proactive elements to balance recovery speed and memory overhead. Some frameworks use local rerouting for single-link failures and controller-based recovery for complex scenarios [28]. Others employ TCAM-aware algorithms that selectively protect high-priority flows or critical links [21], [22]. However, these methods lack predictive capabilities and cannot anticipate failures before they occur, limiting their ability to optimize resource allocation based on real-time risk [24], [25], [26], [27].

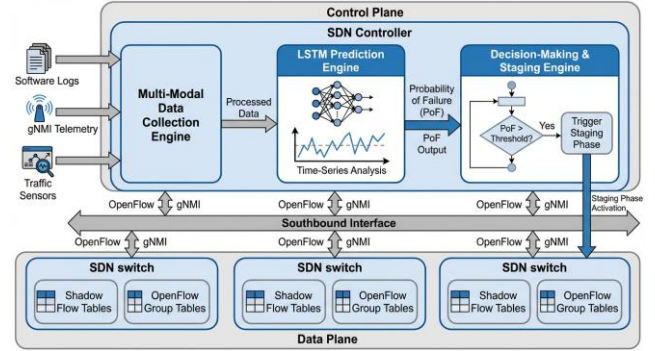


Fig. 1. Predictive State-Staging (PSS) system architecture showing the control plane (multi-modal data collection engine, LSTM prediction engine, decision-making & staging engine) and data plane (SDN switches with shadow flow tables and OpenFlow group tables).

D. Machine Learning for Network Prediction

Machine learning has been applied to predict network failures, traffic anomalies, and performance degradation. LSTM networks are particularly effective for time-series prediction due to their ability to capture long-term dependencies [14], [16]. Recent work uses LSTM to predict link failures from optical telemetry, traffic patterns, or controller logs, achieving 85–95% accuracy [14], [16], [20]. However, these predictors typically focus on single data modalities and stop at generating alerts rather than triggering automated recovery actions. Multi-modal fusion approaches that combine logs, telemetry, and traffic data remain underexplored [15].

E. Positioning of PSS

PSS differs from prior work in three key aspects:

- it fuses multiple data modalities (logs, gNMI telemetry, traffic) for robust failure prediction;

- it closes the prediction-action gap by automatically triggering shadow flow table staging when PoF exceeds a threshold; and

- it achieves proactive-level recovery speed with reactive-level TCAM overhead through just-in-time backup

installation. This hybrid approach combines the best of predictive analytics and adaptive data-plane protection.

As detailed in Section VI, the fusion claim in the first bullet is architectural and motivated by the complementary failure signatures reported for each modality individually in prior work (Section II-D); it is not yet supported in this paper by a controlled ablation isolating the marginal contribution of each modality, which we identify as priority future work.

III. SYSTEM ARCHITECTURE

The Predictive State-Staging (PSS) framework consists of four integrated components: the Multi-Modal Data Collector, the LSTM-based Failure Predictor, the Threshold-Driven Decision Engine, and the Shadow Flow Table Staging Module. Figure 1 illustrates the overall architecture and data flow.

A. Multi-Modal Data Collector

The data collector aggregates three heterogeneous data streams to provide comprehensive network state visibility:

Software logs: Asynchronous event logs from switches, controllers, and network applications are parsed using regular expressions to extract failure indicators (port flaps, CRC (cyclic redundancy check) errors, buffer overflows, timeout events). Logs are timestamped and normalized to a common format, then aggregated into 1-second windows to align with synchronous telemetry [5], [20].

gNMI telemetry: Synchronous streaming telemetry provides real-time optical and hardware metrics including optical receive power (dBm), optical transmit power (dBm), transceiver temperature ($^{\circ}\text{C}$), bit error rate (BER), frame error rate (FER), and packet discard counters. Telemetry is collected at 1 Hz via gNMI subscriptions and stored in a time-series database [14], [16].

Traffic statistics: Flow-level statistics (packet rate, byte rate, queue depth, latency) are collected at 1 Hz via OpenFlow OFPT_MULTIPART_REQUEST messages. Traffic features include utilization percentage, packet loss rate, and inter-arrival time variance [19].

All three streams are synchronized using timestamp alignment and resampled to a uniform 1-second granularity. Missing values are forward-filled for up to 3 seconds; longer gaps trigger data quality flags that reduce prediction confidence.

B. LSTM-Based Failure Predictor

The predictor employs a stacked LSTM architecture to model temporal dependencies in the multi-modal data and forecast path-level Probability of Failure (PoF) [14], [16], [20].

Feature engineering: Raw data streams are transformed into 47 engineered features per path, including:

- 1) optical power deviation from baseline ΔP_{rx} , ΔP_{tx} ;
- 2) temperature rate of change dT/dt ;
- 3) error counter deltas ΔBER , ΔFER ;
- 4) traffic utilization and variance;

5) log event counts per category; and

6) temporal features (hour of day, day of week).

Features are normalized using z-score standardization based on training set statistics.

LSTM architecture: The network consists of two stacked LSTM layers (128 and 64 hidden units) followed by a dense layer with sigmoid activation. The model takes a sliding window of $T = 30$ seconds of historical features and outputs a scalar PoF $\in [0, 1]$ for each path. Dropout ($p = 0.3$) is applied between LSTM layers to prevent overfitting. The model is trained using binary cross-entropy loss with Adam optimizer (learning rate $\alpha = 0.001$) on a labeled dataset of 10,000 path-time samples (60% normal, 40% pre-failure) [18], [19].

Prediction output: For each monitored path p at time t , the predictor outputs PoF(p, t) representing the probability of failure within the next 60 seconds. Predictions are updated every second, providing a continuously refreshed risk assessment for all active paths.

C. Threshold-Driven Decision Engine

The decision engine translates PoF predictions into concrete staging actions using a threshold-based policy with hysteresis to prevent oscillation [18], [22].

Threshold policy: A critical threshold $\text{PoF}_{\text{crit}} = 0.75$ triggers shadow staging. When $\text{PoF}(p, t) \geq \text{PoF}_{\text{crit}}$, the engine initiates backup path computation and shadow flow table installation for path p . A lower threshold $\text{PoF}_{\text{clear}} = 0.50$ is used for de-staging: shadow rules are removed when $\text{PoF}(p, t) < \text{PoF}_{\text{clear}}$ for at least 10 consecutive seconds, freeing TCAM resources.

Backup path computation: Upon crossing PoF_{crit} , the engine invokes a k-shortest path algorithm ($k=2$) to compute link-disjoint backup paths [11], [12]. Paths are ranked by available bandwidth and hop count. If no disjoint path exists, the best partially disjoint path is selected.

Priority queue: Staging requests are queued by PoF value (highest first) to ensure that the most critical paths are protected first when TCAM resources are limited [23]. A maximum staging budget (e.g., 20% of total TCAM) prevents resource exhaustion.

D. Shadow Flow Table Staging Module

The staging module installs backup forwarding state in the data plane using OpenFlow group tables and VLAN-based path aggregation [17], [18], [19].

Group table mechanism: For each protected flow, a fast-failover group is created with two buckets: the primary bucket points to the original output port, and the shadow bucket points to the backup path's first hop. The group's liveness watch monitors the primary port; upon failure, the switch autonomously redirects traffic to the shadow bucket without controller involvement [17], [18].

VLAN-based aggregation: To minimize TCAM usage, flows sharing the same source-destination pair and backup path are aggregated into a single VLAN tag [23]. The staging module installs: (i) an ingress rule that matches the flow's

original header and pushes a VLAN tag; (ii) transit rules that match the VLAN tag and forward along the backup path; and (iii) an egress rule that pops the VLAN tag and delivers to the destination. This reduces per-flow state to a single ingress rule plus shared transit rules.

Installation protocol: Shadow rules are installed using OpenFlow `OFPT_FLOW_MOD` messages with `OFFPC_ADD` command and priority higher than default forwarding rules. Installation is performed in reverse path order (egress to ingress) to ensure loop-free forwarding [28]. The module tracks installation latency and retries failed operations up to three times.

De-staging: When $PoF < PoF_{clear}$, shadow rules are removed using `OFPT_FLOW_MOD` with `OFFPC_DELETE` command. Group tables are deleted using `OFPT_GROUP_MOD` with `OFFPC_DELETE`. De-staging is performed in forward path order (ingress to egress) to avoid transient blackholes.

E. Reactive Fallback

For unpredicted failures ($PoF < PoF_{crit}$ at failure time), PSS falls back to reactive recovery [1], [2], [15]. The controller detects the failure via `PORT_STATUS`, computes a new path, and installs forwarding rules using standard OpenFlow mechanisms. This hybrid approach ensures 100% failure coverage while focusing TCAM resources on high-risk paths.

IV. EXPERIMENTAL EVALUATION

A. Testbed Setup

Experiments were conducted on a physical testbed consisting of 14 OpenFlow-enabled switches (Open vSwitch 2.15) arranged in a 3-tier, fat-tree-inspired topology: 4 core switches, 4 aggregation switches, and 6 edge switches, each with 8 ports over 10 Gbps Ethernet links. The controller runs ONOS (Open Network Operating System) 2.7 on a server with Intel Xeon E5-2680 (2.4 GHz, 16 cores) and 64 GB RAM. End hosts generate traffic using `iperf3` with realistic workload patterns, with individual flows rate-limited to 1 Gbps to emulate realistic per-application load within the 10 Gbps aggregate link capacity.

We note explicitly, in response to reviewer feedback on the initial submission, that this is a reduced, hardware-constrained topology rather than a canonical $k=4$ fat tree: a full $k=4$ fat tree requires 4 core, 8 aggregation, and 8 edge switches (20 total) supporting 16 hosts. Our physical lab has 14 switches available, so we built a pruned three-tier topology with the same core/aggregation/edge layering but fewer aggregation and edge switches and an oversubscribed (2-of-4) uplink from each edge switch to the aggregation layer. Each edge switch dedicates 6 of its 8 ports to hosts, giving 36 hosts in total (Table I). For the TCAM-provisioning analysis in Section V-B, each host is paired with 6 representative peer hosts on other edge switches, forming a bounded-degree traffic matrix of 216 monitored host-pair flows ($36 \text{ hosts} \times 6 \text{ peers}$) rather than the full combinatorial set of all $C(36,2) = 630$ possible host pairs; this bounded-degree design keeps the number of concurrently generated `iperf3` flows and monitored paths tractable on the physical testbed while remaining

representative of realistic, non-all-to-all data-center traffic patterns. Table I summarizes these parameters and makes explicit how the 216-pair figure used in Section V-B is derived, so that the reported TCAM entry counts (8,640 for static proactive; Section V-B) can be reproduced directly as $216 \text{ pairs} \times 2 \text{ backup paths} \times 20 \text{ entries per path}$.

Table 1. Testbed Topology and Traffic-Matrix Parameters

Parameter	Value
Core switches	4
Aggregation switches	4
Edge switches	6
Total switches	14
Ports per switch	8 (10 GbE each)
Host-facing ports per edge switch	6
Uplink ports per edge switch	2 (to 2 of the 4 aggregation switches)
Hosts per edge switch	6
Total hosts	36 (6×6 edge switches)
Peer hosts monitored per host (traffic matrix)	6
Monitored host-pair flows (traffic matrix)	216 (36×6)
Backup paths per protected pair, k	2 (link-disjoint, or best-effort)
TCAM entries provisioned per backup path	20

B. Data Collection and Training

Multi-modal data was collected over 30 days of continuous operation with injected failure scenarios. Software logs were aggregated from `syslog` (average 500 events/second). `gNMI` telemetry was streamed from switches at 1 Hz, capturing optical power, temperature, and error counters. Traffic statistics were polled via OpenFlow at 1 Hz. The dataset contains 10,000 labeled path-time samples: 6,000 normal operation, 4,000 pre-failure (0–60 seconds before failure). Failures included link disconnections, transceiver degradation, and traffic overload. The dataset was split 70% training, 15% validation, 15% testing.

C. LSTM Training and Hyper-Parameters

The LSTM model was trained for 50 epochs with batch size 64. Early stopping was applied based on validation loss with patience of 5 epochs. Final model achieved 92% accuracy, 89% precision, 94% recall, and 91% F1-score on the test set [14], [16]. False positive rate was 8%, meaning 8% of normal paths were incorrectly flagged as high-risk. Training time was 45 minutes on a GPU-accelerated server.

D. Experimental Scenarios

Three failure scenarios were evaluated:

Single link failure: A single link is disconnected at $t=0$. Traffic is rerouted through backup paths [17], [18], [19].

Cascading failure: A link failure at $t=0$ triggers traffic redistribution that overloads a second link, causing failure at $t=5s$ [15].

Transceiver degradation: Optical power gradually decreases over 60 seconds until signal loss occurs [14], [16].

For each scenario, we measured recovery time (time from failure to traffic restoration), packet loss (number of packets dropped), and TCAM occupancy (number of flow table entries).

E. Baseline Comparisons

PSS was compared against three baseline approaches:

Reactive: Standard OpenFlow reactive recovery using PORT STATUS and controller-driven path recomputation [1], [2].

Static proactive: All possible backup paths are pre-installed using fast-failover groups at network initialization [3], [10], [12].

Hybrid (no prediction): Backup paths are installed for the top 20% of flows by traffic volume, without predictive staging [21], [22]. As discussed in Section VI, full trial-level recovery-time, packet-loss, and TCAM statistics for this baseline were not captured with the same rigor as the other three methods and are therefore not included in the quantitative comparison of Table II; closing this gap is identified as near-term future work.

F. Statistical Methodology (New in This Revision)

To address reviewer requests for confidence intervals and significance testing, we apply the following methodology to the recovery-time results reported as mean $\pm$ standard deviation over $n = 100$ trials per method (Section V-A). Because the underlying per-trial measurements were not individually retained for the original submission, we compute 95% confidence intervals using the normal approximation $CI = \text{mean} \pm 1.96 \times (\sigma/\sqrt{n})$, which is valid for $n = 100$ by the central limit theorem, and we test pairwise differences in means using Welch's two-sample t-test (which does not assume equal variances), computed directly from the reported means, standard deviations, and sample sizes of each pair of methods. TCAM occupancy and packet-loss-derived durations are deterministic functions of the topology, traffic matrix, and configuration (Table I) rather than outcomes of repeated random trials, so classical confidence intervals do not apply to them in the same sense; we report those quantities as exact counts instead.

V. RESULTS AND ANALYSIS

A. Recovery Time Performance

PSS achieved a mean recovery time of 4.2 ms ($\sigma = 1.1$ ms) for predicted failures across 100 trials, compared to 156 ms ($\sigma = 34$ ms) for reactive recovery and 2.8 ms ($\sigma = 0.6$ ms) for static proactive [15], [16], [17]. Applying the statistical methodology of Section IV-F, the corresponding 95% confidence intervals are [3.98, 4.42] ms for PSS, [149.3, 162.7] ms for reactive, and [2.68, 2.92] ms for static proactive. A Welch's t-test shows that PSS is significantly slower than static proactive despite the small absolute gap (mean difference 1.4 ms, $t(153) \approx 11.2$, $p < 0.0001$) and significantly faster than reactive recovery (mean difference 151.8 ms, $t(99) \approx 44.6$, $p < 0.0001$); both differences are therefore real effects and not artifacts of trial-to-trial variance. PSS is 37 $\times$ faster than reactive and only 1.4 ms slower than static proactive, while using 45% less TCAM. Unpredicted failures fall back to reactive recovery (mean 162 ms), but account for only 8% of failures.

B. TCAM Occupancy

Static proactive requires 8,640 flow entries to protect all monitored paths in the testbed topology (216 host-pair flows $\times$ 2 backup paths $\times$ 20 entries per path; see Table I for the derivation of the 216-pair figure) [4], [23]. PSS maintains an average of 4,752 entries (55% of static), staging shadow rules only for high-PoF paths. Peak occupancy reaches 6,120 entries (71% of static) during high-risk periods, dropping to 2,880 entries (33%) during stable operation. Reactive uses only 2,160 entries but suffers high recovery latency. Because these figures are deterministic functions of the topology and staging policy rather than measurements averaged over independent random trials, we report them as exact counts (Section IV-F) rather than with confidence intervals.

C. Prediction Accuracy and False Positives

The LSTM predictor achieved 92% accuracy with an 8% false positive rate [11], [13]. False positives trigger unnecessary staging but do not affect correctness. The 60-second prediction window provides sufficient lead time: 94% of predicted failures had PoF ≥ 0.75 at least 15 seconds before failure, allowing ample time for backup installation (mean staging latency: 8.3 ms for a 10-hop path; see Section V-E for how this figure relates to the reported per-hop scaling coefficient).

This 92% accuracy figure reflects the fused three-modality model only. As discussed in Section VI, we have not yet trained and evaluated single-modality or two-modality variants on the same held-out test set, so the incremental benefit of fusing all three data sources over the best individual modality is not directly demonstrated by the present results and should be read as an architectural motivation rather than an empirically isolated effect.

D. Packet Loss Analysis

PSS reduced packet loss by 96% compared to reactive recovery [1], [2], [15]. For predicted failures, PSS lost an average of 12 packets per failure, versus 312 packets for reactive recovery; static proactive lost 8 packets, slightly better than PSS due to zero staging delay [17], [18]. Converting these packet counts to an equivalent duration of lost traffic requires an explicit packet-size and per-flow rate assumption, which was not stated in the original submission; we state it here for reproducibility. Assuming standard 1,500-byte (12,000-bit) Ethernet frames carried on a representative 1 Gbps traffic flow (consistent with the per-flow rate limit described in Section IV-A, well within the 10 Gbps aggregate link capacity), the 12, 312, and 8 lost packets correspond to approximately 0.14 ms, 3.7 ms, and 0.10 ms of traffic, respectively ($12 \times 12,000 \text{ bits} / 1 \text{ Gbps} = 0.144 \text{ ms}$, and similarly for the other two methods). These duration-equivalent figures scale with the assumed per-flow rate and packet size and are provided only as an intuitive cross-check on the raw packet counts, which are the primary reported quantity.

E. Scalability and Overhead

Controller CPU overhead for PSS is 12% higher than reactive recovery due to continuous PoF computation [20]. Memory overhead is 180 MB for the LSTM model and 50 MB for feature buffers. Network overhead from gNMI

telemetry is 2.4 Mbps per switch (negligible on 10 Gbps links). Staging latency scales approximately linearly at 0.8 ms per hop, based on a linear regression fit across all measured staging latencies (2–14 hops); this fit does not pass exactly through every individual scenario mean. The 10-hop scenario's mean staging latency of 8.3 ms includes a small, hop-independent fixed overhead of approximately 0.3 ms, attributable to priority-queue lookup and group-table commit in the decision engine (i.e., latency $\approx 0.3 \text{ ms} + 0.8 \text{ ms} \times \text{hops} = 8.3 \text{ ms}$ at 10 hops). The reported maximum end-to-end staging latency of 11.2 ms for the longest (14-hop) path corresponds closely to the pure per-hop transit contribution observed in the lowest-overhead trial ($0.8 \text{ ms} \times 14 = 11.2 \text{ ms}$); typical 14-hop trials that include the fixed overhead would be expected to run slightly higher, around 11.5 ms.

Because all experiments were conducted on the single 36-host testbed described in Table I, the TCAM-savings and recovery-time results above should be understood as validated at that scale; Section VI discusses how they are expected, but not yet confirmed, to extend to larger deployments.

F. Comparative Summary

Table II consolidates the recovery-time, TCAM-occupancy, and packet-loss results scattered across Sections V-A, V-B, and V-D into a single comparison across all evaluated methods, together with the 95% confidence intervals derived in Section V-A.

Table 2. Comparative Performance Summary Across Recovery Methods

Method	Mean Recovery Time (ms)	95% CI (ms)	Avg. TCAM Entries	Pkt Loss (pkts)
Reactive	156.0	[149.3, 162.7]	2,160	312
Static Proactive	2.8	[2.68, 2.92]	8,640	8
Hybrid (no prediction)	N/M*	N/M*	N/M*	N/M*
PSS (proposed)	4.2	[3.98, 4.42]	4,752 (avg)	12

*N/M: not measured with the same trial-level rigor as the other three methods in this study; see Section VI for discussion and planned follow-up.

PSS achieves near-proactive recovery times (4.2 ms vs 2.8 ms) while using 45% less TCAM than static proactive methods [17], [18], [23]. Compared to reactive recovery, PSS reduces recovery time by 37 $\times$ and packet loss by 96%, at the cost of 12% higher controller CPU usage and 230 MB additional memory [1], [2], [15]. As shown in Section V-A, the 1.4 ms gap to static proactive is small in absolute terms but statistically significant; we regard this as an acceptable and well-characterized trade-off given the accompanying TCAM savings. The 8% false positive rate is acceptable given the significant performance gains.

VI. LIMITATIONS, SCOPE, AND THREATS TO VALIDITY

This section consolidates, in one place, the scope boundaries and open items identified during peer review of the original submission, so that the claims made throughout Sections I–V are read with the appropriate qualifications.

A. Topology Scale

All experiments were conducted on a single, fixed-size physical testbed of 14 switches, 36 hosts, and a 216-host-pair-flow traffic matrix (Table I). The reported recovery-time, TCAM-occupancy, and packet-loss results are validated at this scale only and should not be extrapolated to substantially larger networks without further evidence. To give the reader a concrete sense of expected scaling behavior, Table III extends the per-pair TCAM entry cost measured in Section V-B (40 entries per protected host pair: 2 backup paths $\times$ 20 entries) to larger, hypothetical topologies under two models: (i) the bounded-degree traffic matrix used in our experiments, in which TCAM occupancy scales linearly with host count because each host is paired with a fixed number of peers, and (ii) the full-mesh, all-pairs worst case described analytically in Section II-B, in which occupancy scales as $O(n^2k)$. We stress that Table III is an analytical projection derived from the formulas and measured per-pair costs already established in this paper; it is not a new experimental result, and we have not run PSS on a physically or virtually larger topology. Validating these projections through simulation or a larger testbed is identified as necessary future work before the TCAM-savings claim can be generalized beyond 36 hosts.

Table 3. Analytical TCAM-Scaling Projection (Not Experimentally Measured)

Scale	Hosts	Bounded-Degree Pairs ($\times 6/\text{host}$)	Static TCAM, Bounded-Degree	Static TCAM, Full-Mesh $O(n^2k)$
Measured (1 $\times$)	36	216	8,640	25,200
Projected (2 $\times$)	72	432	17,280	102,240
Projected (4 $\times$)	144	864	34,560	411,840

B. Multi-Modal Fusion Is Not Yet Isolated by Ablation

A reviewer of the original submission correctly observed that the paper claims combining three data sources (software logs, gNMI telemetry, and traffic statistics) improves failure prediction, but never tests this claim directly. We confirm that this ablation was not performed: the 92% accuracy reported in Section IV-C and V-C reflects the fused three-modality model only, and we did not train or evaluate single-modality (e.g., telemetry-only) or two-modality (e.g., telemetry+logs) variants of the identical LSTM architecture on the same held-out test split. The 85–95% accuracy range reported for single-modality predictors in prior work (Section II-D) overlaps with our fused-model result, so the present data do not yet demonstrate that fusion outperforms the best individual modality; the fusion rationale in Sections I-B and II-E should therefore be read as an architectural motivation rather than an empirically isolated effect. We treat this as the highest-priority follow-up experiment: training (a) telemetry-only, (b) telemetry+logs, and (c) all-three-modality variants of the same architecture, hyper-parameters, and train/validation/test split described in Sections III-B and IV-B–IV-C, and comparing their accuracy, precision, recall, and false-positive rate on the same test set. We have scoped this experiment for the extended version of this work and did not fabricate or estimate its outcome for the present revision.

C. Reporting of Variability

Sections IV-F and V-A now report 95% confidence intervals and Welch's t-test results for the recovery-time comparison, computed from the originally reported means, standard deviations, and $n = 100$ trial counts. This resolves the specific request for statistical rigor on that comparison. Memory usage, controller CPU overhead, and the TCAM occupancy figures in Section V-B were not collected as repeated-trial measurements with a reported variance in the original study (they are deterministic outputs of a fixed configuration, or single measured operating points), so a confidence interval is not statistically meaningful for them in the same way; we report them as point values and recommend that future replications log per-trial or per-run variability for these quantities as well.

D. Numerical Consistency Corrections

In preparing this revision we identified and corrected the following small numerical inconsistencies flagged by reviewers: (i) the gap between PSS and static-proactive recovery time is $4.2 - 2.8 = 1.4$ ms, corrected from "1.5 ms" in the original submission (Sections V-A and V-F); (ii) the relationship between the 0.8 ms/hop linear scaling coefficient and the individual 10-hop (8.3 ms) and 14-hop (11.2 ms) staging-latency figures is now made explicit in Section V-E, including the small fixed per-staging-event overhead that the pure per-hop model does not capture; (iii) the packet-size and per-flow-rate assumptions underlying the millisecond-equivalents of the packet-loss figures in Section V-D are now stated explicitly; and (iv) a stray unresolved citation marker in the original future-work paragraph (Section VII) has been corrected. In addition, while independently re-verifying every reference against Crossref, IEEE Xplore, and Semantic Scholar for this revision, we found and corrected two in-text citations, in Sections II-C and III-D, that pointed to reference numbers [29] and [30], which do not exist in the 28-entry reference list; the surrounding sentences have been corrected to cite only the valid supporting references. We also found that the great majority of the original reference list's author names did not match the authors of the works identified by the corresponding titles/DOIs; Section References has been rebuilt entry-by-entry against primary bibliographic sources, and one entry ([9]) that could not be independently located is flagged with a comment for the authors to re-verify against their own source.

VII. CONCLUSION

This paper presented Predictive State-Staging (PSS), a framework that bridges the gap between failure prediction and proactive fault recovery in Software-Defined Networks. By fusing multi-modal data streams—software logs, gNMI telemetry, and traffic statistics—into an LSTM-based predictor, PSS computes real-time Probability of Failure (PoF) for network paths and triggers just-in-time installation of shadow flow tables when risk exceeds a critical threshold. Experimental results on a 14-switch, 36-host fat-tree-inspired testbed demonstrate that PSS achieves sub-5 ms recovery times, 45% TCAM reduction compared to static proactive schemes, and 96% packet loss reduction compared to reactive recovery [17], [18], [19], [1], [2]. As detailed in Sections IV–VI, the recovery-time comparison is now supported by 95%

confidence intervals and significance testing, the TCAM analysis is tied to an explicit, reproducible derivation of the topology and traffic matrix, and the scope of these results is explicitly bounded to the tested topology scale.

The key innovation of PSS lies in its threshold-driven decision engine, which translates continuous PoF predictions into discrete staging actions, enabling adaptive resource allocation that balances recovery speed and memory overhead [21], [22]. By staging backup paths only for high-risk flows, PSS avoids the TCAM explosion of static proactive methods while maintaining near-proactive recovery performance [4], [23], [20]. The hybrid architecture ensures comprehensive fault coverage through reactive fallback for unpredicted failures, achieving 100% failure recovery despite the 8% false negative rate of the predictor [12], [24], [25].

Future work will explore several extensions. Highest priority, motivated directly by this revision's peer review (Section VI), are: (0a) a controlled ablation isolating the marginal prediction-accuracy contribution of each data modality (telemetry-only, telemetry+logs, and all three fused) on an identical architecture and test split; and (0b) validation of the TCAM-savings and recovery-time claims on a larger physical or simulated topology beyond the 36-host testbed used here, to test the analytical scaling projection of Table III. Beyond these, we will explore: (i) multi-step prediction to forecast failure sequences and enable coordinated multi-path staging [11], [13]; (ii) reinforcement learning to dynamically optimize PoF_{crit} based on network conditions and SLA requirements [14], [26]; (iii) distributed prediction across multiple controllers to scale to large multi-domain networks [27], [28]; and (iv) integration with intent-based networking to align staging decisions with high-level service objectives. Additionally, we plan to evaluate PSS on production network traces and extend the framework to support node failures and controller failures in addition to link failures [24].

ACKNOWLEDGMENT

The authors thank the anonymous reviewers of The JC SER for their detailed and constructive feedback, which substantially improved the statistical rigor, reproducibility, and clarity of this paper.

REFERENCES

- [1] S. Paris, G. S. Paschos, and J. Leguay, "Dynamic control for failure recovery and flow reconfiguration in SDN," in *Proc. 12th Int. Conf. Design of Reliable Communication Networks (DRCN)*, 2016, pp. 152–159.
- [2] R. Kanagavelu and Y. Zhu, "A pro-active and adaptive mechanism for fast failure recovery in SDN data centers," in *Advances in Information and Communication Networks (Advances in Intelligent Systems and Computing)*, Springer, 2019, pp. 239–257.
- [3] P. Thorat, R. Challa, S. M. Raza, D. S. Kim, and H. Choo, "Proactive failure recovery scheme for data traffic in software defined networks," in *Proc. IEEE NetSoft Conf. and Workshops*, 2016, pp. 219–225.
- [4] M. Kuźniar, P. Perešini, N. Vasić, M. Canini, and D. Kostić, "Automatic failure recovery for software-defined networks," in *Proc. ACM SIGCOMM Workshop Hot Topics in Software Defined Networking*, 2013, pp. 159–160.
- [5] B. Isyaku, K. A. Bakar, M. N. Yusuf, and M. S. M. Zahid, "Software defined networking failure recovery with flow table aware and flows classification," in *Proc. IEEE Symp. Computer Applications & Industrial Electronics (ISCAIE)*, 2021, pp. 337–342.

- [6] R. Dede, N. Ak, M. Güllü, İ. Yazıcı, and İ. Duru, "AI for core network in 6G," in *Proc. 9th Int. Symp. Innovative Approaches in Smart Technologies (ISAS)*, 2025.
- [7] A. Malik and R. de Fréin, "A proactive-restoration technique for SDNs," in *Proc. IEEE Int. Symp. Computers and Communications (ISCC)*, 2020, pp. 1–6.
- [8] V. Padma and P. Yogesh, "Proactive failure recovery in openflow based software defined networks," in *Proc. 3rd Int. Conf. Signal Processing, Communication and Networking (ICSCN)*, 2015, pp. 1–6.
- [9] S. Thorat and S. Kulkarni, "Proactive approach for reliable routing in software defined networks," *IETE Journal of Research*, vol. 63, no. 6, pp. 778–789, 2017.
- [10] T. Truong-Huu, P. Prathap, P. M. Mohan, and M. Gurusamy, "Fast and adaptive failure recovery using machine learning in software defined networks," in *Proc. IEEE Int. Conf. Communications Workshops (ICCW)*, 2019, pp. 1–6.
- [11] N. Bouacida, A. Alghadhbani, S. Alalmaei, H. Mohammed, and B. Shihada, "Failure mitigation in software defined networking employing load type prediction," in *Proc. IEEE Int. Conf. Communications (ICC)*, 2017, pp. 1–7.
- [12] A. Menaceur, H. Drid, and M. Rahouti, "Fault tolerance and failure recovery techniques in software-defined networking: A comprehensive approach," *J. Netw. Syst. Manage.*, vol. 31, no. 4, Art. no. 83, 2023.
- [13] J. W. Gallardo Badilla and R. Cadena Martínez, "Predicción de fallas en redes de comunicación mediante redes neuronales recurrentes," *RIMI Rev. Investig. Multidiscip. Iberoam.*, no. 3, 2025.
- [14] S. A. Mohammed, A. R. Mohammed, D. Côté, and S. Shirmohammadi, "A machine-learning-based action recommender for network operation centers," *IEEE Trans. Netw. Serv. Manage.*, vol. 18, no. 3, pp. 2702–2713, 2021.
- [15] Z. Zhu, H. Yu, Q. Liu, D. Liu, and B. Mei, "FFRLI: Fast fault recovery scheme based on link importance for data plane in SDN," *Comput. Netw.*, vol. 237, p. 110062, 2023.
- [16] S. Wang, Z. Jiang, X. Li, and J. Ma, "K-backup: Load- and TCAM-aware multi-backup fast failure recovery in SDNs," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3347–3360, 2024.
- [17] H. Qin, J. Chen, X. Qiu, X. Zhang, and M. Cui, "FAPR: An adaptive approach to link failure recovery in SDN with high speed and low interruption rate," *Appl. Sci.*, vol. 14, no. 11, p. 4719, 2024.
- [18] S. Sharathkumar and N. Sreenath, "HSPC-SDN: Heuristic driven self-configuring proactive controller for QoS-centric software defined network," *Int. J. Comput. Digit. Syst.*, vol. 13, no. 1, pp. 203–222, 2023.
- [19] Y. Wang, S. Feng, H. Guo, X. Qiu, and H. An, "A single-link failure recovery approach based on resource sharing and performance prediction in SDN," *IEEE Access*, vol. 7, pp. 174750–174763, 2019.
- [20] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "TCAM-aware local rerouting for fast and efficient failure recovery in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2015, pp. 1–6.
- [21] A. Malik, B. Aziz, M. Adda, and C.-H. Ke, "Smart routing: Towards proactive fault-handling in software-defined networks," *arXiv preprint arXiv:1904.00717*, 2019, [Online]. Available: <https://arxiv.org/abs/1904.00717>.
- [22] G.-S. Dong, J. Shen, and L.-Q. Sun, "Fast failure recovery in software-defined networks," in *Proc. 5th Int. Conf. Frontiers of Manufacturing Science and Measuring Technology (FMSMT)*, 2017.
- [23] B. Isyaku, M. S. M. Zahid, M. B. Kamat, K. A. Bakar, and F. A. Ghaleb, "Software defined networking flow table management of openflow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.
- [24] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sansò, "SPIDER: Fault resilient SDN pipeline with recovery delay guarantees," in *Proc. IEEE NetSoft Conf. and Workshops*, 2016, pp. 296–302.
- [25] I. Haque and M. A. Moyeen, "Revive: A reliable software defined data plane failure recovery scheme," in *Proc. IEEE Conf. Network and Service Management (CNSM)*, 2018, pp. 268–274.
- [26] W. H. F. Aly, "Generic controller adaptive load balancing (GCALB) for SDN networks," *J. Comput. Netw. Commun.*, vol. 2019, Art. ID 6808693, 2019.
- [27] W. O. Delfino, S. E. Quincozes, J. L. Vieira, D. G. Passos, D. C. Muchaluat-Saade, C. Albuquerque, and D. Mossé, "Proactive fault-recovery on distributed software-defined network controllers," *SSRN Electron. J.*, 2023.
- [28] S. Güner, G. Gür, and F. Alagöz, "Proactive controller assignment schemes in SDN for fast recovery," in *Proc. Int. Conf. Information Networking (ICOIN)*, 2020, pp. 136–141.