

Adaptive Routing and Self-Healing for Fault-Tolerant SDN Mesh Networks

Aishwarya K. P.¹; Bhoomika V.²; Hamsini G. R.³; K. S. Sharadhi⁴

^{1;2;3;4}Department of Computer Science and Engineering, East West Institute of Technology Bangalore, India

Publication Date: 2026/09/29

Abstract: This paper presents AdaptiveNet SDN, a software-defined networking simulation framework designed to maintain communication when links, nodes, or groups of adjacent links fail. The system combines a multi-objective adaptive routing algorithm with a self-healing controller, heartbeat-based failure detection, bandwidth-aware admission control, and real UDP packet forwarding among virtual mesh nodes. Routing cost considers latency, reliability, energy use, and deadline miss probability, while an exponentially weighted moving average adjusts routing priorities according to observed network conditions. An eight-percent path retention rule reduces unnecessary route changes during small metric fluctuations. The framework was evaluated on six-node and twelve-node topologies using repeated failure experiments and a Dijkstra latency-based baseline. Across the current eighteen-condition experiment sweep, the twelve-node topology achieved complete recovery for single link, two independent link, and sequential failure scenarios for both algorithms. Node and correlated storm failures exposed the limits of topology redundancy. The experiments did not establish a statistically distinguishable advantage for adaptive routing on service recovery time or packet loss, which is reported as a limitation rather than hidden. The main contribution is therefore an integrated platform for studying adaptive routing and self-healing behavior under controlled network failures.

Keywords: Software-Defined Networking; Adaptive Routing; Self-Healing; Quality of Service; Fault Recovery; Dijkstra; UDP Mesh; Failure Detection.

How to Cite: Aishwarya K. P.; Bhoomika V.; Hamsini G. R.; K. S. Sharadhi (2026) Adaptive Routing and Self-Healing for Fault-Tolerant SDN Mesh Networks. *International Journal of Innovative Science and Research Technology*, 11(9), 1743-1748. <https://doi.org/10.38124/ijisrt/26sep807>

I. INTRODUCTION

Modern communication applications increasingly depend on networks that can continue service when individual links or devices become unavailable. Traditional shortest path routing is effective when the network state is stable, but a route selected using one dominant metric may become unsuitable when latency, congestion, delivery reliability, energy cost, and application deadlines change together. Software-defined networking offers a practical foundation for tackling this problem: by pulling routing decisions out of individual forwarding devices and into a programmable control layer, it lets a single policy govern how paths are chosen across the network.

This work develops AdaptiveNet SDN, a local SDN emulation platform in which twelve virtual nodes exchange packets through real UDP sockets. The controller maintains the topology, active flows, link states, and recovery events, while a web interface provides live visibility into the mesh. The system was designed specifically to study self-healing behavior rather than to claim deployment as a production SDN controller. It includes a Dijkstra baseline and a multi-objective Adaptive Routing Algorithm, referred to as ARA throughout this paper.

The central research question is whether an adaptive multi-metric routing mechanism can maintain service while network conditions and failures change. The implementation addresses this question through four connected mechanisms: adaptive routing weights, path hysteresis, fault detection and recovery, and capacity-aware path admission. The evaluation compares ARA with latency-based Dijkstra routing under controlled link, node, sequential, and correlated failure scenarios.

The paper makes three practical contributions. First, it integrates routing, failure handling, and packet forwarding in one executable environment. Second, it adapts QoS priorities from observed network trends instead of keeping one fixed weight vector for every network condition. Third, it reports repeated experimental results together with limitations in reproducibility and topology redundancy, avoiding claims that are not supported by the measured data.

II. RELATED WORK

Prior surveys frame software-defined networking around one central idea: control logic is pulled out of individual switches and placed in a programmable layer that can be reconfigured through software rather than device-by-device

changes. OpenFlow was among the first widely adopted protocols to put this idea into practice, giving an external controller a standard way to push forwarding rules into switches and update them on the fly. The present work follows the programmable control idea at the simulation level, but it does not implement OpenFlow, Open vSwitch, Mininet, or a production SDN controller. Instead, the controller directly manages an emulated mesh and uses real local UDP sockets for packet transport.

Shortest path routing provides a natural baseline for the proposed system. The well-known Dijkstra algorithm computes lowest cost paths through a graph whose edges all carry non-negative weights, and remains the standard reference point for shortest path routing. In this project, Dijkstra is retained as a reference method and uses link latency as its routing cost. ARA extends the same graph search process by replacing the single latency metric with a normalized composite cost that represents several QoS concerns.

Several recent studies look for ways to extend or adapt Dijkstra-style computation for larger SDN topologies rather than replacing it outright. Turlykozhaeva et al. partition an SDN graph into subnets using box-covering methods and then apply Dijkstra between and within the resulting subnets, reporting shorter running times than plain Dijkstra as the number of nodes grows [1]. Kamarudin et al. describe QSroute, a QoS-aware routing concept that admits a path only when it satisfies independent threshold constraints on available bandwidth, packet delay, and packet loss, though the authors present it as an architecture proposal rather than an implemented and measured system [2]. ARA shares the underlying goal of moving past a single latency metric, but instead of subnet partitioning or independent thresholds it folds latency, reliability, energy, and deadline costs into one normalized composite value, and the mechanism is evaluated on an executing UDP-based mesh rather than left as an unimplemented design.

Learning-based approaches have also been proposed for adapting routing behavior to changing network conditions. Gururama Senthilvel et al. combine a convolutional neural network for congestion prediction with Q-learning for path selection in wireless mesh networks, reporting improved packet delivery ratio and latency over AODV and DSR baselines in simulation [3]. AdaptiveNet SDN instead adjusts its routing weights with an exponentially weighted moving average of observed error rate, utilization, and latency, which keeps the adaptation lightweight and interpretable at the cost of the predictive lookahead a trained model can offer. Fast recovery has also been studied closer to the data plane: De Lima et al. propose RESISTING, a P4-based fast reroute mechanism that redistributes flows evenly across the surviving links of an ECMP group after a failure, and show that this avoids the overload a single backup path produces under multiple simultaneous failures [4]. The self-healing engine in this paper performs recovery at the control layer through re-routing and heartbeat-based detection rather than in programmable switch hardware, so its recovery timescale is milliseconds rather than the sub-microsecond range reported for P4 recirculation, but it

pursues the same goal of avoiding a single overloaded fallback path.

Comparative studies of classical routing algorithms continue to inform which shortest path method suits a given network. Tammanashastri et al. compare RIP-style Bellman-Ford routing with OSPF-style Dijkstra routing across small Cisco Packet Tracer topologies and report that Dijkstra-based OSPF scales better as router count increases, while Bellman-Ford-based RIP stays competitive only in the smallest topologies [5]. This finding is consistent with the choice made here to retain Dijkstra, rather than Bellman-Ford, as the shortest path baseline against which the adaptive algorithm is measured.

The project also relates to research on resilient and dependable SDN systems, where controller availability, failure detection, and rapid route reconfiguration are important design concerns. Rather than claiming distributed controller consensus, AdaptiveNet SDN models controller failover locally and allows nodes to compute a fallback next hop when the controller is marked offline. This distinction is important because the implementation is a research simulator and not a geographically distributed control plane.

Unlike a literature-only proposal, the present study evaluates an executable implementation with repeated trials. The project documentation explicitly distinguishes implemented features from features that remain outside scope, including predictive failure forecasting, distributed consensus, and production-grade load balancing. This separation is used throughout the paper to keep the experimental claims aligned with the actual system.

➤ System Architecture

AdaptiveNet SDN is organized into a control layer, forwarding layer, simulation layer, and visualization layer. The backend is implemented with Node.js and Express, while WebSocket messages synchronize state with a React-based interface. The controller maintains node and link status, routing flows, failure events, and recovery actions. A dynamic topology view allows a user to inject node or link failures and observe the resulting path changes.

The forwarding layer uses real UDP datagrams between twelve virtual nodes on the local machine. Each packet contains a fixed binary header carrying fields such as flow identifier, sequence number, timestamp, time-to-live, packet type, and payload length. Heartbeat and heartbeat acknowledgement packets are handled through the same socket-based communication model. This gives the simulator real packet transport behavior without requiring physical network hardware.

Traffic is grouped into three QoS classes. PREMIUM traffic is configured with stronger emphasis on reliability and deadline behavior, STANDARD traffic balances the major metrics, and BULK traffic gives greater importance to energy cost. The controller also maintains bandwidth commitments for active flows. A link whose remaining capacity cannot satisfy a new flow is excluded from the candidate graph instead of merely receiving a higher routing penalty.

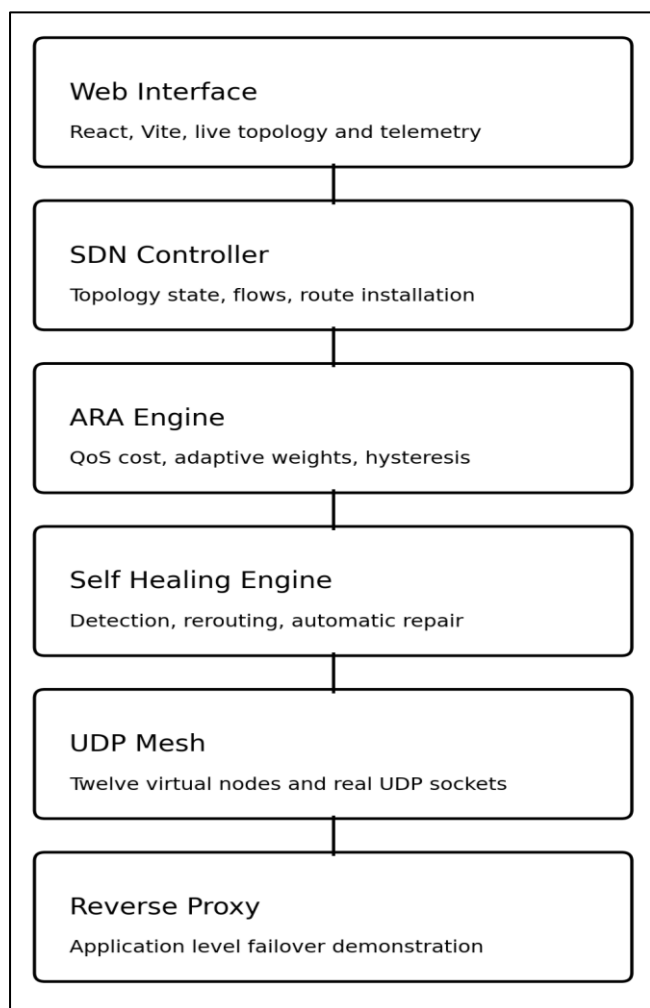


Fig 1 Major Software and Network Components of AdaptiveNet SDN.

III. ADAPTIVE ROUTING ALGORITHM

ARA evaluates every usable link using four normalized components. For a link e , the composite cost is represented as the weighted sum of latency, reliability, energy, and deadline costs. The implementation normalizes every component to the range from zero to one so that one metric cannot dominate simply because it has a larger numerical scale. The effective weights are non-negative and normalized to sum to one.

Latency cost is derived from the measured link delay and a queueing multiplier based on utilization. Reliability cost is derived from the probability of successful delivery after accounting for both the configured error rate and congestion-related tail drops. Energy cost uses a CPU-based proxy together with link bandwidth. Deadline cost estimates the probability that the current delay will exceed the service-level target using a bounded Gaussian cumulative distribution approximation.

The baseline QoS vectors are different for PREMIUM, STANDARD, and BULK traffic. The adaptive component then modifies these vectors using an exponentially weighted moving average with a smoothing factor of 0.15. When the observed error trend increases, the reliability component receives more weight and the energy component receives less. When

utilization rises, the deadline component receives more weight while raw latency is reduced slightly. Weight changes are bounded and each component has a minimum floor, preventing the vector from collapsing onto one metric.

ARA also includes path hysteresis. Links belonging to a flow's current path receive a fixed retention factor of 0.92 during route comparison. The intention is to prevent repeated switching between paths whose costs differ only because of small utilization fluctuations. The retention factor is never allowed to preserve a failed route because failed nodes and links are removed from the candidate graph before route evaluation.

The same heap-backed shortest path routine supports both ARA and the baseline. A binary min heap and lazy deletion provide $O((V + E) \log V)$ graph search behavior. For the baseline configuration, the route cost is link latency. For ARA, the route cost is the normalized multi-metric composite value. Capacity admission is applied before the link enters the candidate graph, ensuring that an infeasible link is not selected solely because it appears cheap.

➤ Self-Healing and Failure Handling

The self-healing engine treats fault injection and fault detection as separate concepts. A link failure removes the affected connection from route computation, causing active flows to search for another feasible path. A node failure disables the node and all incident links. When automatic healing is enabled, a recovery timer restores the failed element after the configured delay. The default link and node repair delay used by the project is approximately two seconds.

A heartbeat-based detector provides a non-oracle detection mode. Heartbeats are sent at a configured interval and the detector records only acknowledgement packets that actually arrive through the UDP mesh. When the elapsed time since the last acknowledgement exceeds the failure timeout, the monitored element changes to the DOWN state. The default configuration uses a 500-millisecond heartbeat interval and a 1600-millisecond failure timeout. The live application normally uses an oracle mode for immediate visualization, while heartbeat mode is available when detection behavior itself is being studied.

The framework also models correlated failure storms. A storm takes down an active node and a randomly selected portion of its active incident links as one event. This is more demanding than a single link cut because the failure can remove several alternate routes at once. The results show that topology redundancy, rather than the routing cost function alone, becomes the dominant factor under these conditions.

Controller failure is modeled as a simulated failover mechanism. When the controller is marked offline, nodes can calculate a local next hop using the topology information already available to them. A timer later restores the controller state. This demonstrates continued routing behavior during a control plane interruption, but it should not be interpreted as a distributed consensus system because there is no Raft or Paxos state replication between independent controllers.

➤ Implementation and Application Demonstration

The backend combines Express, WebSocket synchronization, routing logic, packet forwarding, failure detection, and experiment services. The frontend is implemented using React and Vite and presents the live mesh, event stream, routing statistics, and test controls. The project contains an automated test suite covering routing, topology, packet framing, failure and healing, heartbeat detection, path switching, cost normalization, admission control, and related controller behavior.

A notable end-to-end demonstration is the self-healing reverse proxy. One mesh node acts as an ingress adapter and two other mesh nodes connect to independent HTTP service replicas. A browser request is converted into an application packet, forwarded through the UDP mesh, delivered to a selected egress node, and then served by the corresponding HTTP backend. Before each request, the system probes backend health and can move the next request to the surviving service if a backend process, egress node, or required link becomes unavailable.

This design is intentionally implemented through the same forwarding and routing pipeline used by ordinary flows. Consequently, the reverse proxy is not a separate shortcut demonstration; it exercises route selection, packet delivery, failure handling, and application response transfer together.

IV. EXPERIMENTAL METHODOLOGY

The evaluation uses two topology sizes: the primary twelve-node mesh and a smaller six-node mesh. ARA is compared with latency-based Dijkstra under single link, single node, two independent link, sequential failure, and correlated

storm conditions. Healthy and congested cases are also included as reference conditions. The executed baseline sweep contains eighteen conditions with thirty trials per condition, giving 540 trial runs. The experiments use the same topology, failure schedule, seed selection, and measurement window for paired ARA and Dijkstra comparisons.

The main measurements are recovery success rate, service recovery time, packet loss percentage, estimated throughput, reroute count, control event count, and no-route events. The current baseline sweep uses oracle detection, so detection latency is zero by construction and should not be interpreted as a measured heartbeat detection performance result. A separate heartbeat mode exists in the implementation for future detection latency experiments.

The project also records an important reproducibility limitation. Scenario selection is seeded, but topology error rates, packet drop decisions, and the subset of links selected during a correlated storm still use unseeded random behavior. Therefore the structural failure outcomes are repeatable, while packet-level measurements can vary between repeated sweeps with the same scenario seeds. The reported results below are from the current on-disk experiment run.

V. RESULTS AND DISCUSSION

The twelve-node topology recovered from every single link, two independent link, and sequential failure trial for both routing methods. Single node failures were harder because removing one node simultaneously removes all incident connections. The correlated storm scenario was harder still because it combines node loss with multiple adjacent link losses.

Table 1 Recovery Success Observed in the Executed Baseline Sweep.

Case	ARA	Dijkstra
Link 12N	100.0%	100.0%
Node 12N	63.3%	83.3%
2 Links 12N	100.0%	100.0%
Sequential 12N	100.0%	100.0%
Storm 12N	50.0%	66.7%
Node 6N	46.7%	46.7%

For single link failures, both algorithms found alternate routes in all trials. This indicates that the twelve-node mesh has sufficient path redundancy for this failure type at the tested traffic load. In contrast, the lower recovery rates for node and storm failures indicate that some failure states leave no feasible end-to-end route regardless of how the remaining paths are ranked. The identical 46.7 percent recovery rate for both algorithms on six-node single node failures reinforces this interpretation: once topology redundancy is exhausted, a more sophisticated cost function cannot create a path that does not exist.

Service recovery time also remained close between the two methods. On the twelve-node topology, the mean service recovery time for single link failure was 33.2 milliseconds for ARA and 41.5 milliseconds for Dijkstra. For single node

failure, the corresponding means were 38.3 and 40.6 milliseconds. For sequential failures, they were 34.1 and 34.7 milliseconds. These values show that ARA can recover quickly, but the experiment does not establish a statistically significant advantage over the baseline.

Packet loss results must be interpreted with the reproducibility caveat described earlier. In the current run, the twelve-node single link case produced mean packet loss of about 1.07 percent for ARA and 1.83 percent for Dijkstra. In the two independent link case the means were about 1.52 percent and 1.01 percent, respectively. Because packet-level randomness is not fully controlled by the scenario seed, these differences should not be presented as proof that one algorithm is universally better.

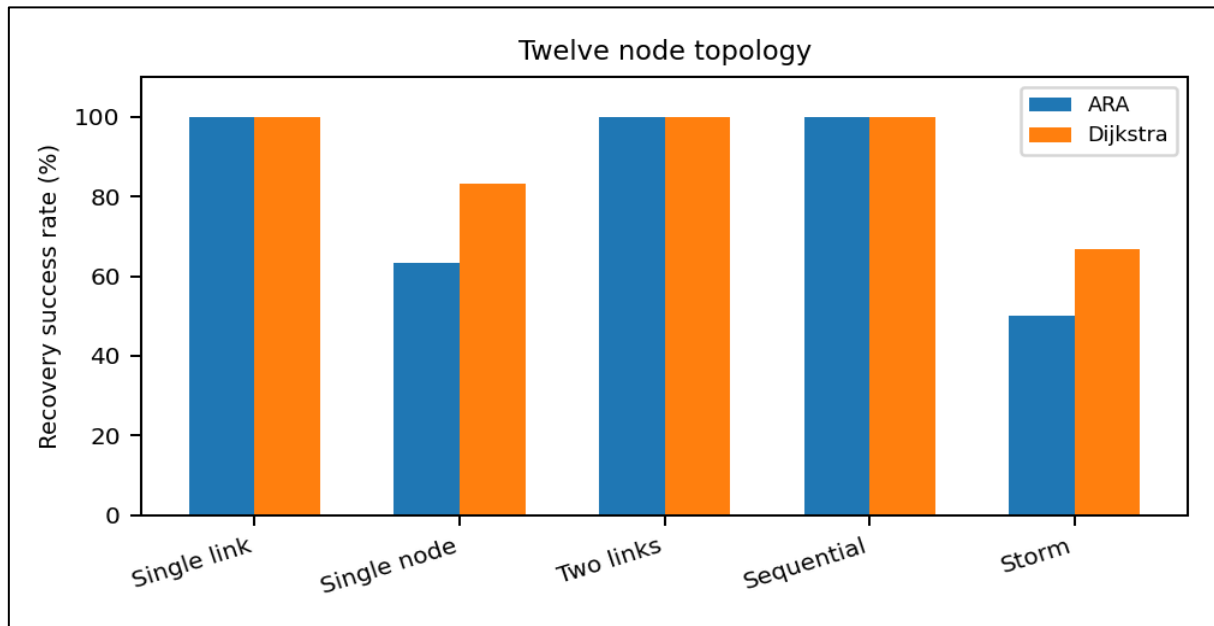


Fig 2 Recovery Success for ARA and Latency-Based Dijkstra in the Current Twelve-Node Sweep.

Control event counts also illustrate why recovery success and event count should not be interpreted independently. For single node failures, ARA averaged 8.0 control events while Dijkstra averaged 13.0. The difference partly follows the fact that a successful recovery generates additional route installation events, whereas an unrecoverable flow terminates earlier. For correlated storms, the event counts were 22.5 for ARA and 28.7 for Dijkstra, reflecting the larger number of state changes created by multi-element failures.

Overall, the baseline study does not demonstrate a statistically distinguishable improvement in service recovery time or packet loss for ARA over Dijkstra. Confidence interval overlap checks were performed for the five major twelve-node failure scenarios and did not separate the two methods on those metrics. This result is valuable because it identifies what the current experiment can and cannot support. The contribution of the system is better framed as an adaptive and self-healing research platform with measurable behavior, rather than as a proven replacement for shortest path routing.

VI. LIMITATIONS AND FUTURE ENHANCEMENTS

The current system is a single-process local simulation. Although it uses genuine UDP sockets, it does not reproduce geographic propagation delay, heterogeneous physical links, NAT traversal, or independent controller failures across multiple machines. The controller failover mechanism is simulated and does not provide distributed state replication or consensus.

The adaptive weight mechanism is reactive rather than predictive. Its EWMA responds to observed error rate, utilization, and latency trends but does not forecast an upcoming failure. Future work can introduce prediction models trained on historical link quality and evaluate whether preemptive rerouting improves service continuity.

Failure storm topology is randomly selected around an affected node and therefore does not model a specific physical disaster pattern. The reverse proxy demonstration is also intentionally small, using one ingress and two egress service replicas. A production extension could support dynamic service discovery, weighted load balancing, session affinity, and graceful backend draining.

Admission control currently considers bandwidth availability but does not guarantee queueing delay, jitter, or priority-based preemption. In addition, flows are admitted sequentially in array order. A future scheduler could reserve capacity by QoS priority and evaluate order-independent admission decisions. Finally, packet-level reproducibility should be improved by propagating a single seeded random source through topology generation, packet loss, and storm selection before stronger statistical comparisons are attempted.

VII. CONCLUSION

This paper presented AdaptiveNet SDN, a software-defined mesh simulation framework that combines adaptive multi-objective routing with automatic recovery from communication failures. The system integrates ARA cost computation, adaptive QoS weighting, path hysteresis, heartbeat-based detection, bandwidth-aware admission, real UDP forwarding, correlated failure scenarios, and an application-level reverse proxy demonstration.

The executed experiments show that the twelve-node mesh can recover completely from the tested single link, dual independent link, and sequential failure scenarios, while node and correlated storm failures expose the limits of available topology redundancy. The comparison with latency-based Dijkstra did not establish a statistically distinguishable performance advantage for ARA on service recovery time or packet loss. Rather than weakening the project, this finding clarifies the role of the framework: it provides a controlled

environment for studying adaptive routing and resilience and exposes the conditions under which routing intelligence is useful versus the conditions where additional physical or logical redundancy is the real requirement.

Future versions can extend the platform toward distributed controller replication, predictive failure avoidance, more realistic traffic models, deterministic packet-level experiments, larger topologies, and statistically stronger evaluation. These additions would enable the framework to move from a local research simulator toward a broader testbed for resilient SDN routing.

ACKNOWLEDGMENT

The authors acknowledge the academic guidance and project support received during the development and evaluation of the AdaptiveNet SDN implementation.

REFERENCES

- [1]. D. Turlykozhaeva, A. Akhmetali, S. Akhtanov, A. Bolysbay, N. Ussipov, and Y. Shabdan, "Routing algorithm for software defined network based on box-covering algorithm," 2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM), 2023, doi: 10.1109/WINCOM59760.2023.10322960.
- [2]. I. E. Kamarudin, M. I. U. Ong, M. A. Ameen, and A. Zabidi, "QSroute: A QoS aware routing scheme for Software Defined Networking," 2023 IEEE 8th International Conference on Software Engineering and Computer Systems (ICSECS), 2023, pp. 388-391, doi: 10.1109/ICSECS58457.2023.10256362.
- [3]. P. Gururama Senthilvel, T. Thirupurasundari D R, R. Kumar K, H. Kumar Palani, and K. Sundravadevelu, "Enhancing wireless mesh network performance through AI-based routing algorithms," 2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI), 2023, pp. 931-935, doi: 10.1109/ICCSAI59793.2023.10421320.
- [4]. D. De Lima, F. Vogt, A. Teixeira da Silva, F. Rodriguez Cesen, and C. Esteve Rothenberg, "RESISTING: A new fast-reroute mechanism with packet distribution on P4-programmable switches," 2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2024, doi: 10.1109/NFV-SDN61811.2024.10807507.
- [5]. P. Ravi Tammanashastri, S. M. Rajagopal, and S. Shivadutt Bananjee, "Comparative analysis of Bellman-Ford and Dijkstra algorithms in Software Defined Networking," 2024 5th International Conference on Data Intelligence and Cognitive Informatics (ICDICI), 2024, doi: 10.1109/ICDICI62993.2024.10810921.