Dynamic Flow Scheduling in Leaf Spine Topology for Data Center Networks

Team 08

FIRST SEMESTER 2025-26

21 November 2025

UNDER THE SUPERVISION OF

Dr. Virendra Singh Shekhawat

Associate Professor

Department of Computer Science and Information System

TEAM MEMBERS

Ashish Shetty	2025H1030064P
S Rishab	2025H1030066P

Abstract

Data centers increasingly rely on leaf–spine topologies to provide high bandwidth, fault tolerance, and scalability. However, traffic in such networks is highly dynamic, with mice flows (short, latency-sensitive) and elephant flows (long, bandwidth-intensive) often competing for resources. Traditional routing mechanisms like Equal-Cost Multi-Path (ECMP) rely on static hashing, often routing multiple large flows onto the same link while others remain idle.

This project proposes a dynamic, intelligent routing framework leveraging Software-Defined Networking (SDN) and Deep Reinforcement Learning (DRL). The project implements an Actor-Critic RL agent within the Ryu controller to manage a Leaf-Spine topology emulated in Mininet. The system utilizes real-time network states, throughput, link utilization, utilization skew, and packet loss to dynamically reroute flows. Experimental results demonstrate that the RL-based approach achieves higher average throughput and better load balancing compared to ECMP, Static, and Greedy routing algorithms.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Modern Data Centers host diverse applications ranging from web services to distributed machine learning training. The traffic patterns in these networks are highly bimodal; the majority of flows are small latency-sensitive "mice," while a few large "elephant" flows consume the bulk of the bandwidth. Static routing protocols such as OSPF and standard load balancing techniques such as ECMP fails to account for the real-time utilization of links, leading to inefficient resource usage and congestion.

Regular routing techniques also do not take into account the context of flows or the condition of the links and switches. And in the case of dynamic single path routing, elephant flows with multiple parallel connections aren't taken into consideration. These limitations motivate the need for adaptive, context-aware routing that can respond to instantaneous network conditions. Reinforcement Learning (RL) provides a promising alternative because it can incorporate fine-grained link statistics, flow behavior, and historical performance into its decision process. By learning a policy that explicitly accounts for utilization, packet loss, and flow type, an RL-based controller can make proactive routing decisions, promoting large flows to weighted multi-path forwarding while letting small flows follow lightweight single-path rules. Such an approach has the potential to significantly reduce congestion, improve throughput fairness, and increase overall fabric efficiency in modern data-center networks.

1.2 Problem Statement

Data centers require flows that are both high bandwidth and low latency, no matter the type or context of the flow. It should be able to deliver Quality of Service to elephant flows, mice flows, interactive flows, etc. and should be able to dynamically handle link utilization between its switches. In a Leaf-Spine topology based Data Center Network the concept of using the shortest path fails as there are multiple equally shortest path between the hosts and equal cost multi-path doesn't take the link utilization into consideration. Hence a flow scheduling algorithm which would manage utilization based on the flow stats received from the switches is necessary.

1.3 Objectives

- To design a Leaf-Spine topology using Mininet.
- To generate artificial traffic using the iPerf3 to simulate the environment.
- To implement an SDN Controller using Ryu that gathers real-time port statistics.
- To develop an Actor-Critic Reinforcement Learning model to make intelligent routing decisions.
- To ensure Quality of Service (QoS)ness and fair among the different competing flows within the network.
- To compare the RL performance against ECMP, Static, and Greedy algorithms.

Chapter 2

Literature Review

2.1 Traditional and Heuristic Routing Algorithms

Traffic engineering in Data Center Networks (DCNs) has evolved from static configurations to dynamic, load-aware scheduling. This section reviews the fundamental algorithms currently deployed in production environments and identifies their limitations regarding "Elephant" flow management.

2.1.1 Static and Shortest Path Routing

The most fundamental approach to routing relies on static protocols or Shortest Path First (SPF) algorithms like OSPF (Open Shortest Path First). In these systems, the path between a source and destination is pre-determined based on hop count or link weights.

Mechanism: A flow is assigned a path solely based on network topology, ignoring current traffic conditions.
Limitation: Static routing suffers severely from "Head-of-Line Blockin". If a high-bandwidth flow saturates a specific link, all subsequent flows assigned to that link experience congestion, even if alternative paths in the Leaf-Spine topology are idle.

2.1.2 Equal-Cost Multi-Path (ECMP)

To utilize the bisection bandwidth of Leaf-Spine topologies, the industry standard is Equal-Cost Multi-Path (ECMP) routing[1].

Mechanism: ECMP distributes traffic across available paths using a hash of the packet header's 5-tuple (Source IP, Destination IP, Source Port, Destination Port, Protocol). This ensures that packets belonging to the same flow always take the same path, preventing TCP packet reordering.
Limitation: While ECMP provides excellent load balancing for many small "Mice" flows, it is "traffic-oblivious". It does not account for the size of the flow or the current utilization of the links. A phenomenon known as "Hash Collision" occurs when two or more "Elephant" flows coincidentally hash to the same physical link while other links remain underutilized. This results in significant throughput degradation for high-priority applications.

2.1.3 Heuristic and Greedy Approaches

With the advent of Software-Defined Networking (SDN), centralized controllers allowed for global visibility and dynamic re-routing. Several heuristic algorithms were proposed to address the limitations of ECMP.

The Greedy Algorithm: The "Greedy" or "Least-Loaded" approach represents a logical improvement over static hashing. The greedy algorithm makes locally optimal choices at each Hop, hoping to find a global optimum solution.
Mechanism: The controller periodically polls switch statistics. When a new flow arrives (or is detected), the controller assigns it to the path with the lowest instantaneous utilization.
Limitation: While effective in theory, Greedy algorithms often suffer from "Route Flapping" or oscillation. If Path A is empty, the algorithm shifts all new traffic to Path A, causing it to become congested immediately. In the next polling cycle, it shifts everything to Path B. This oscillation increases jitter and creates instability in TCP congestion control mechanisms.

Hedera[2] and Mahout[3]: Seminal works such as Hedera and Mahout introduced the concept of distinguishing flows.
Mechanism: These systems utilize a central scheduler to detect flows exceeding a specific bandwidth threshold (Elephants). Once detected, these flows are explicitly scheduled onto non-conflicting paths using algorithms like Global First Fit, while Mice flows are left to ECMP.
Limitation: These approaches typically rely on complex optimization solvers that may not converge quickly enough for highly dynamic traffic patterns, or they rely on slow polling intervals (e.g., 5 seconds) which may miss bursty traffic spikes.

2.2 Reinforcement Learning in Networking

Conventional routing algorithms, such as ECMP and Greedy heuristics, operate on rigid, pre-defined rules that often fail to adapt to the non-linear and highly dynamic nature of modern Data Center Networks (DCNs). Consequently, the research community has shifted toward Knowledge-Defined Networking (KDN), employing Machine Learning to enable autonomous network management. Specifically, Reinforcement Learning (RL) has emerged as a powerful paradigm for solving online routing optimization problems by formulating them as Markov Decision Processes (MDPs).

2.2.1 Real-Time DRL Optimization

Recent literature emphasizes the capability of Deep Reinforcement Learning (DRL) to handle complex network states that hand-crafted heuristics cannot manage. As discussed in *Towards Real-Time Routing Optimization with Deep Reinforcement Learning* **RT-RO**[4], DRL offers a distinct advantage in highly dynamic scenarios where traditional control systems fall short.
Mechanism: The mechanism relies on an agent that continuously interacts with the network environment, learning optimal decision-making policies through trial and error without requiring a pre-constructed mathematical model of the traffic pattern. This allows for real-time operation, addressing the computational latency issues often found in traditional optimization solvers.

2.2.2 DRL-R (Multi-Resource Optimization)

In the context of Software-Defined Data-Center Networks (SDN-DCN), simple bandwidth metrics are often insufficient. Liu et al. introduced **DRL-R**[5], a Deep Reinforcement Learning-based routing scheme that innovates by recombining multiple network resources, specifically bandwidth, cache, and computing power—into a unified state metric.
Mechanism: Their mechanism quantifies the contribution of cache in reducing delay, allowing the DRL agent deployed on the SDN controller to make routing decisions based on a "resource-recombined" state. They implemented and compared Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) agents, observing that policy-gradient methods (like DDPG) generally outperformed value-based methods (DQN) in terms of throughput and flow completion time.

2.2.3 Delay-Aware DRL Agent

Further validating this approach, recent studies[6] on Deep-RL for SDN routing optimization have demonstrated that agents can automatically adapt to current traffic conditions to propose tailored configurations.
Mechanism: The primary mechanism here is the minimization of a specific reward function based on network delay. By allowing the agent to observe the global state, the system provides significant operational advantages over traditional optimization algorithms, which often struggle to converge quickly during traffic bursts.

2.2.4 Proposed System: Actor-Critic for Elephant Flow Routing

Drawing from the insights above, this project implements a centralized control loop using the **Actor-Critic**[7] Reinforcement Learning architecture. While frameworks like DRL-R utilize DDPG, and others explore DQN, this project leverages Actor-Critic to combine the stability of Policy Gradient methods with the sample efficiency of Value-based methods.

This project specifically targets the "Elephant flow" problem. Unlike general traffic routing, here the agent utilizes a global view of the Leaf-Spine topology to detect high-bandwidth flows and dynamically re-route them based on a reward function that balances three competing objectives: maximizing throughput, minimizing link utilization skew (load balancing), and minimizing packet loss.

Chapter 3

Methodology and System Design

3.1 System Architecture

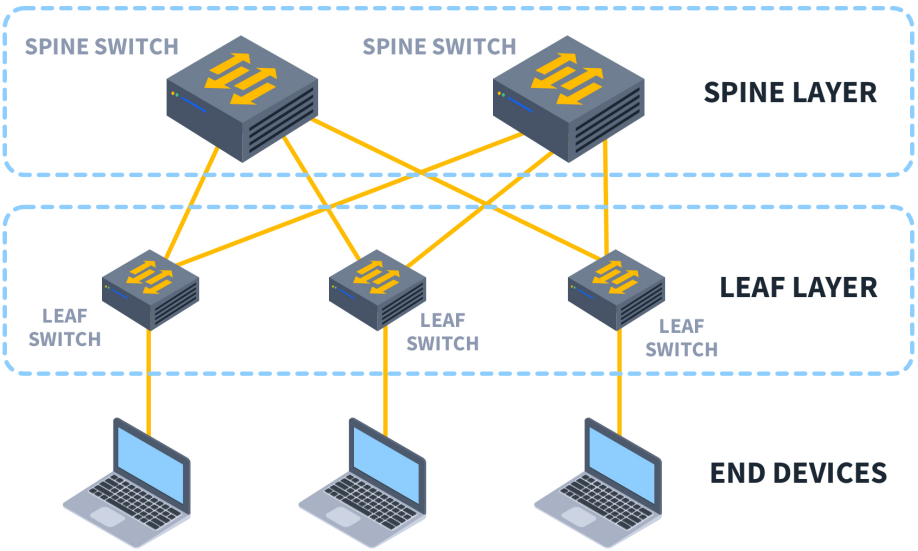

Figure 3.1: Leaf–Spine Topology

The system comprises three planes:

1. **Data Plane:** Emulated via Mininet using Open vSwitch (OVS). It consists of a leaf-spine topology network.
2. **Control Plane:** The Ryu SDN Controller. It communicates via OpenFlow 1.3 to install flow rules and query statistics.
3. **Intelligence Plane:** A PyTorch-based Deep Learning model embedded within the controller.

3.2 The RL Formulation

3.2.1 State Space (S_t)

The agent observes a state vector representing the condition of all available uplink paths from the leaf switch to the spine layer:

$$S_t = [U_1, L_1, U_2, L_2, \ldots, U_K, L_K]$$

where U_k denotes the normalized bandwidth utilization and L_k denotes the normalized packet loss rate of the k-th uplink path. This representation captures both congestion and reliability indicators for every possible next-hop spine switch.

3.2.2 Action Space (A_t)

The action corresponds to choosing one of the K uplinks (spine switches) for forwarding traffic:

$$A_t \in \{0, 1, \ldots, K-1\}.$$

The action is consumed differently depending on the flow type:

- **Mice Flows:** A single best path is selected using

  $$a_t = \arg\max_a \pi(a \mid S_t),$$

  ensuring low overhead and fast forwarding.

- **Elephant Flows:** Instead of a single action index, the full probability vector

  $$\pi(a \mid S_t) = [p_1, p_2, \ldots, p_K]$$

  is converted into OpenFlow group weights, enabling weighted load balancing across multiple uplinks.

Thus, the same policy is reused for both flow categories, but its output is interpreted differently based on flow size.

3.2.3 Reward Function (R_t)

The reinforcement learning agent is optimized using a weighted reward function designed to jointly encourage high throughput, balanced load distribution, and low packet loss:

$$R_t = \alpha \cdot \text{Throughput}_{\text{norm}} \; - \; \beta \cdot \text{Skew}_{\text{util}} \; - \; \gamma \cdot \text{Loss}_{\text{norm}}$$

where:

- $\text{Throughput}_{\text{norm}}$ is the normalized throughput of the flow,
- $\text{Skew}_{\text{util}}$ is the standard deviation of uplink utilizations (a measure of load imbalance),
- $\text{Loss}_{\text{norm}}$ is the normalized packet loss on the chosen path.

In this work, the reward weights are set to:

$$\alpha = 2, \qquad \beta = 1, \qquad \gamma = 1,$$

reflecting a stronger emphasis on maximizing throughput while still penalizing imbalance and packet loss.

3.3 RL Model Design

The reinforcement learning component in our system uses an Actor–Critic architecture to make routing decisions based on the real-time state of uplink ports on each leaf switch. The goal is to learn a policy that assigns flows to spine uplinks in a way that improves throughput while avoiding congestion.

3.3.1 Neural Network Architecture

Both the Actor and Critic are implemented using compact two-layer feed-forward neural networks.

Actor Network (Policy)

Input dim $= 2 * Numpaths(Spines)$, Hidden layers: $64 \rightarrow 64$, Output dim $= N$

The actor outputs logits that are converted to routing probabilities using a softmax function. The final probability vector:

$$\pi(a \mid s)$$

represents the routing distribution over uplinks.

Critic Network (Value Function)

Input dim $= 2 * Numpaths(Spines)$, Hidden layers: $64 \rightarrow 64$, Output dim $= 1$

The critic estimates the expected return for the current network state.

3.3.2 Action Selection

Given a state s, the actor computes:

$$\pi(a \mid s) = \text{softmax}(f_\theta(s)),$$

and an action is sampled from this distribution. Each action corresponds to choosing one spine uplink or multiple paths in case of elephant flows.

3.3.3 Training Objective

The critic is trained using a temporal-difference target:

$$y = r + \gamma V(s'),$$

and minimizes the squared error between $V(s)$ and y.

The actor uses an advantage-based policy-gradient update:

$$A = y - V(s),$$

which encourages actions that lead to higher throughput and balanced load.

An entropy regularization term is added to prevent premature convergence:

$$\mathcal{L}_{entropy} = -\beta\, H(\pi).$$

3.4 Flow Promotion Technique

In our RL-driven leaf–spine network controller, flow promotion is used to give large ("elephant") flows dedicated load-balanced forwarding treatment. The controller continuously polls OpenFlow flow-stats from all leaf switches and computes the byte delta for each active IPv4 flow. When this instantaneous throughput exceeds a configurable threshold (2 MB per polling interval), the controller marks the flow as an elephant. Instead of allowing the flow to use the default single uplink chosen by the RL policy for mice flows, the controller installs a SELECT group table on the ingress leaf. This group contains weighted buckets, each corresponding to one of the spine uplinks. The RL policy outputs a probability distribution over available uplinks, and these probabilities are converted into bucket weights to influence traffic splitting. As a result, elephant flows receive more stable and load-balanced forwarding, while smaller mice flows continue using simple one-shot flows. This flow promotion mechanism ensures that large transfers are dynamically steered over the most favorable paths based on real-time port utilization and loss measurements.

The flow promotion mechanism tightly couples RL output with OpenFlow group programming. The RL agent continuously observes per-uplink network conditions through port-stats, normalized into a state vector comprising utilization and loss features. When an elephant flow is detected, the controller queries the RL actor to compute action probabilities. Instead of choosing a single output port (as is done for mice flows), the system interprets the full probability distribution as a multi-action decision. These probabilities are converted into integer weights that populate a SELECT group. Each bucket in the group corresponds to a potential egress uplink, and the hardware load balancer chooses paths proportional to the RL policy's output. This allows the RL agent to express nuanced traffic-engineering strategies, distributing elephant flows across multiple spines while still biasing decisions toward paths that historically yield higher rewards. In this way, flow promotion becomes a direct actuation mechanism where the learned policy actively shapes the fabric's load distribution.

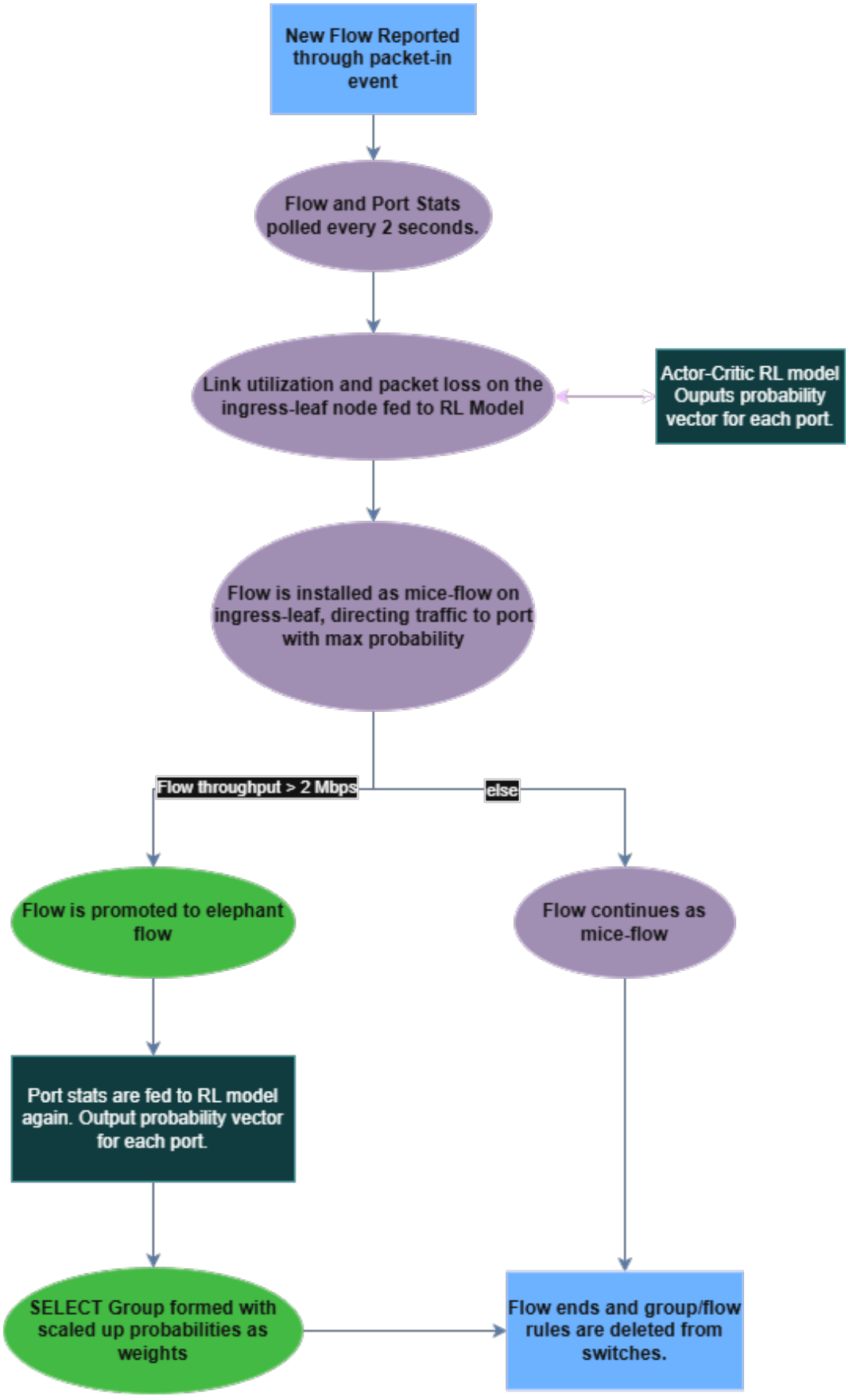

Figure 3.2: High-level overview of System Plan

3.5 Tools and Technologies Used

The implementation of the proposed Intelligent Routing System relies on a stack of open-source networking and machine learning tools.

3.5.1 Network Emulation: Mininet

Mininet[8] is used to emulate the Data Plane. It allows the creation of a realistic virtual network on a single kernel, supporting custom topologies and Open vSwitch (OVS) instances. It executes the topology.py script to instantiate switches, links, and hosts, providing an isolated environment for testing routing logic without physical hardware. Thus, the required topology is obtained.

3.5.2 SDN Controller: Ryu

Ryu[9] is a component-based software-defined networking framework written in Python. It serves as the Control Plane.It acts as the centralized brain of the network. It listens for Packet-In events, gathers port statistics, and installs flow rules via the OpenFlow protocol.

3.5.3 Machine Learning Framework: PyTorch

PyTorch[10] is an open-source machine learning library used to implement the Intelligence Plane. It hosts the Deep Reinforcement Learning (DRL) agent. It constructs the Actor-Critic neural network, performs the forward pass for policy inference (routing decisions), and executes the backpropagation algorithm (training) using the calculated reward signals.

3.5.4 Traffic Generation: iPerf3

iPerf3[11] is a standard tool for active measurements of the maximum achievable bandwidth on IP networks. It generates the synthetic workload to train and test the RL agent. It is scripted to generate dynamic flow patterns, including high-bandwidth TCP “Elephant” flows, replicating real-world Data Center congestion scenarios.

3.5.5 Python

Python[12] is the prgramming language used for implementation. Python version 3.9 is last version compatible with the Ryu. Matplotlib[13], which is Python package, is used for plotting the graphs.

Chapter 4

Implementation and Results

4.1 Experimental Setup

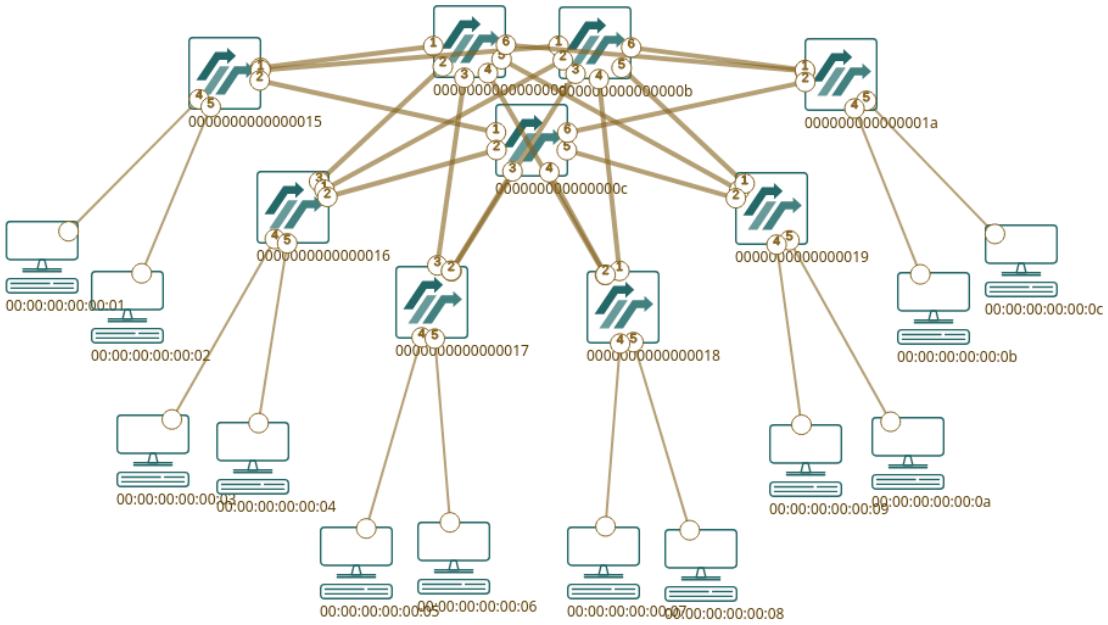

Figure 4.1: Primary Topology

- **Topology:** 3 Spines, 6 Leaves, 12 Hosts.
- **Traffic:** Iperf3 used to generate background UDP noise and a persistent TCP Elephant flow between Host 1 and Host 4.
- **Training:** The RL agent was trained for 1000 steps with a batch size of 8.

4.2 Performance Metrics

We evaluated the system based on:

1. **Throughput:** Measures the effective data transfer rate achieved by the flow. Higher throughput indicates that the selected path provides adequate bandwidth and minimal congestion.

2. **Packet Loss:** Inferred primarily from TCP retransmissions. Persistent loss suggests congestion, queue overflows, or suboptimal routing decisions.

3. **Round-Trip Time (RTT):** Represents end-to-end latency. Lower RTT reflects faster acknowledgment cycles, while elevated RTT often correlates with queue buildup.

4. **TCP Congestion Window (CWND):** Indicates how much data the sender can inject into the network without waiting for ACKs. A growing CWND implies stable conditions; stagnation or reduction indicates congestion.

5. **Jitter:** Measures the variation in RTT over time. High jitter typically reflects unstable paths, fluctuating queue lengths, or inconsistent scheduling across links.

4.3 Result Discussion

4.3.1 Throughput Comparison

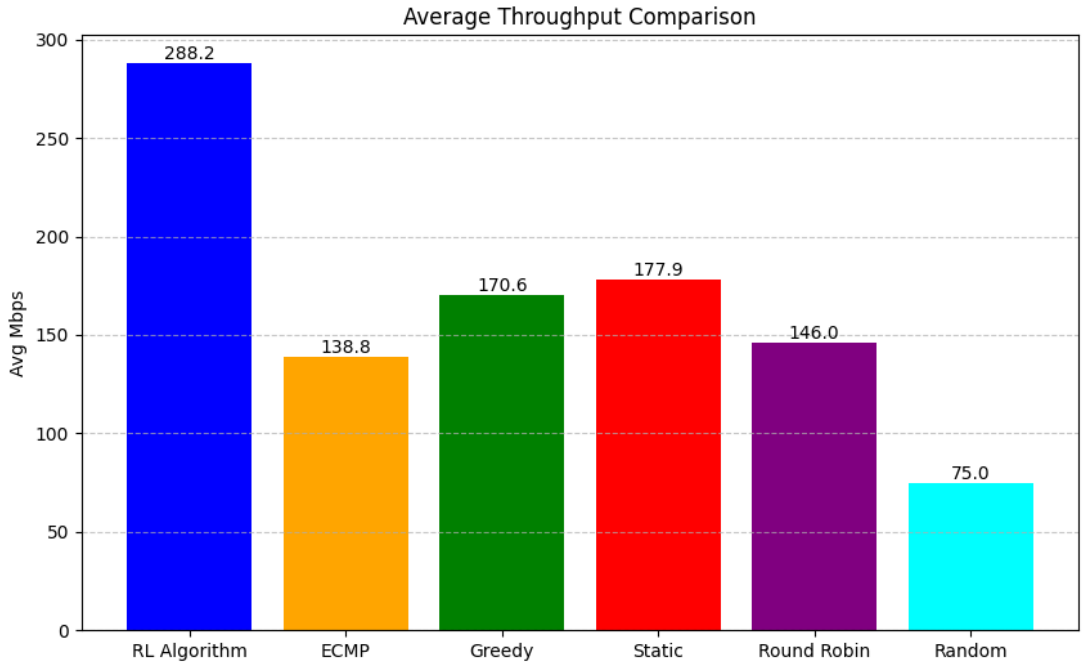

Figure 4.2: Throughput Comparison

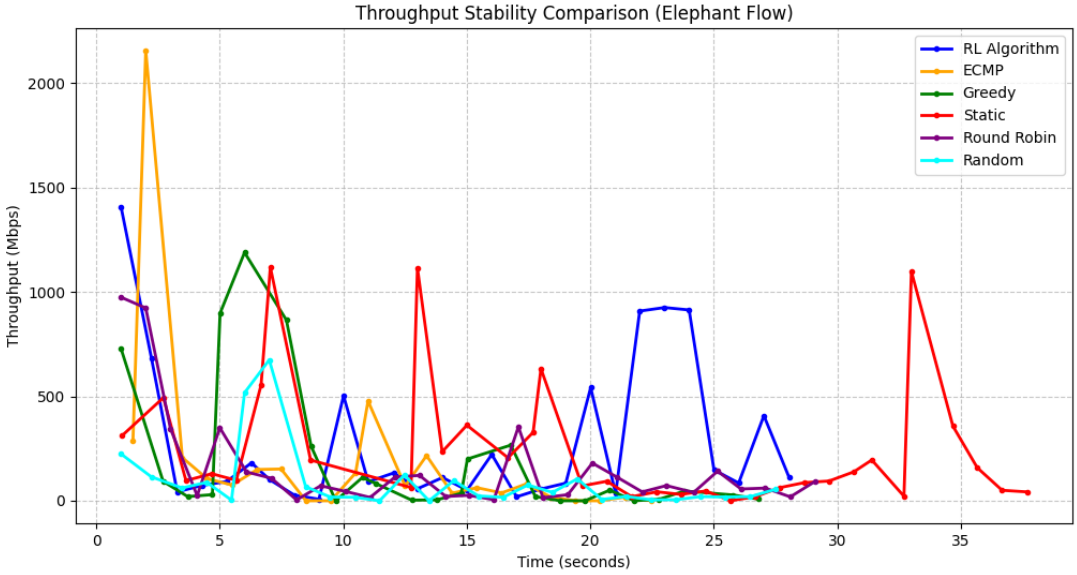

Figure 4.3: Throughput Timeseries of all methods

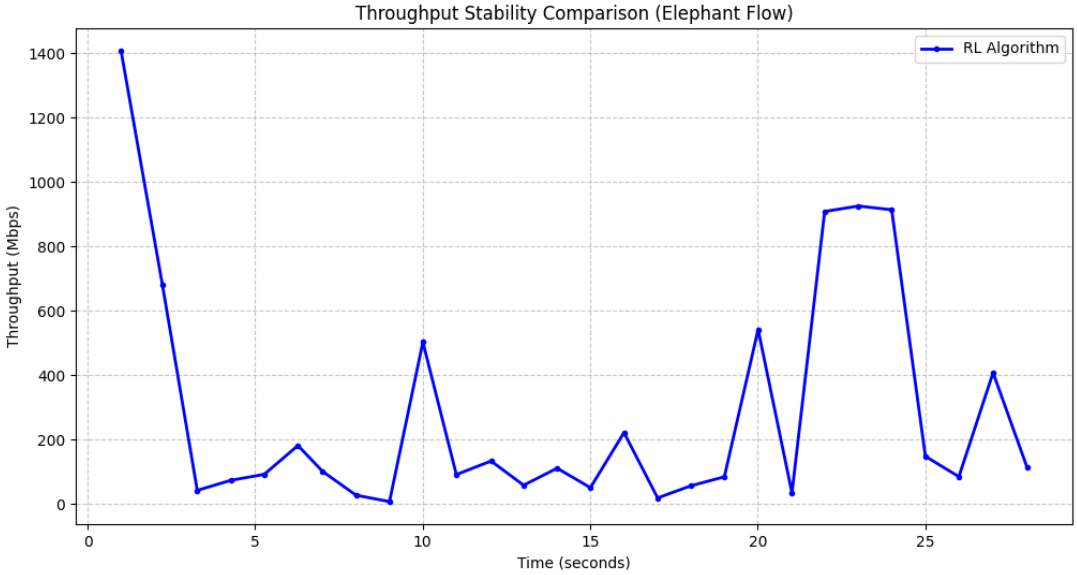

Figure 4.4: Throughput Timeseries for proposed method

The primary objective of this study was to evaluate the capacity of the Reinforcement Learning (RL) agent to maintain high data transfer rates for Elephant flows in the presence of network congestion. We compared the RL agent against five standard baselines: Equal-Cost Multi-Path (ECMP), Greedy , Static, Round Robin, and Random routing. The overall efficacy of each algorithm was measured by calculating the mean throughput of the monitored Elephant flow over the duration of the experiment. As illustrated in the figure, the RL agent was able to outperform other algorithms by a significant margin.

4.3.2 Latency Comparison

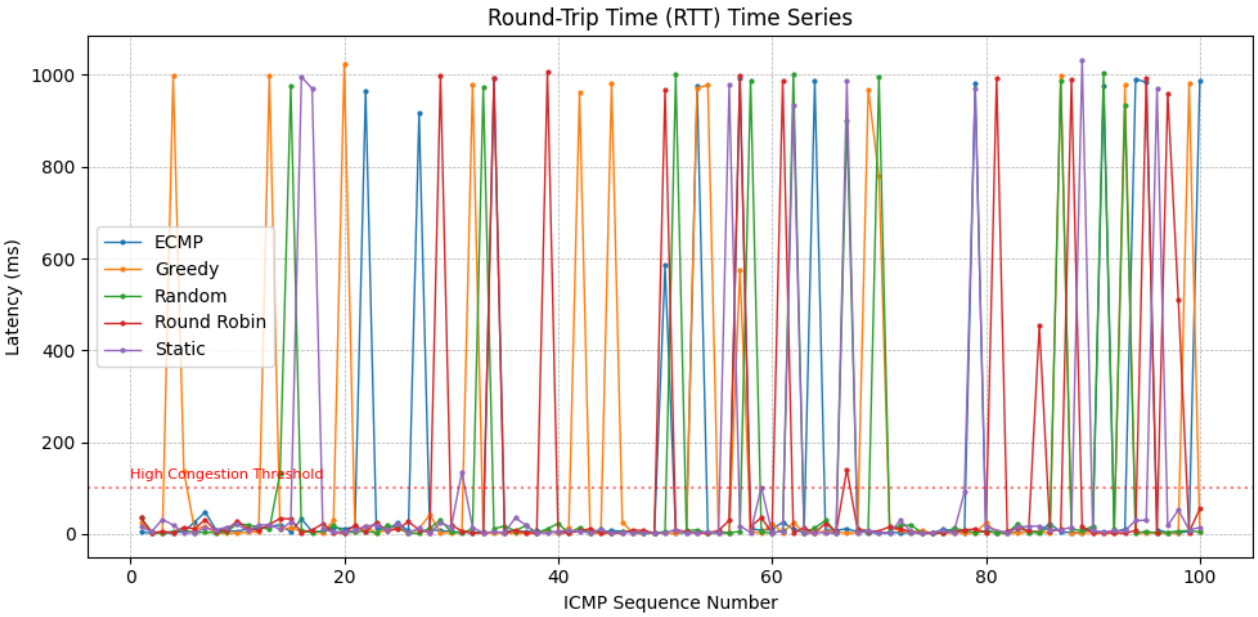

Figure 4.5: Latency Comparison of other methods

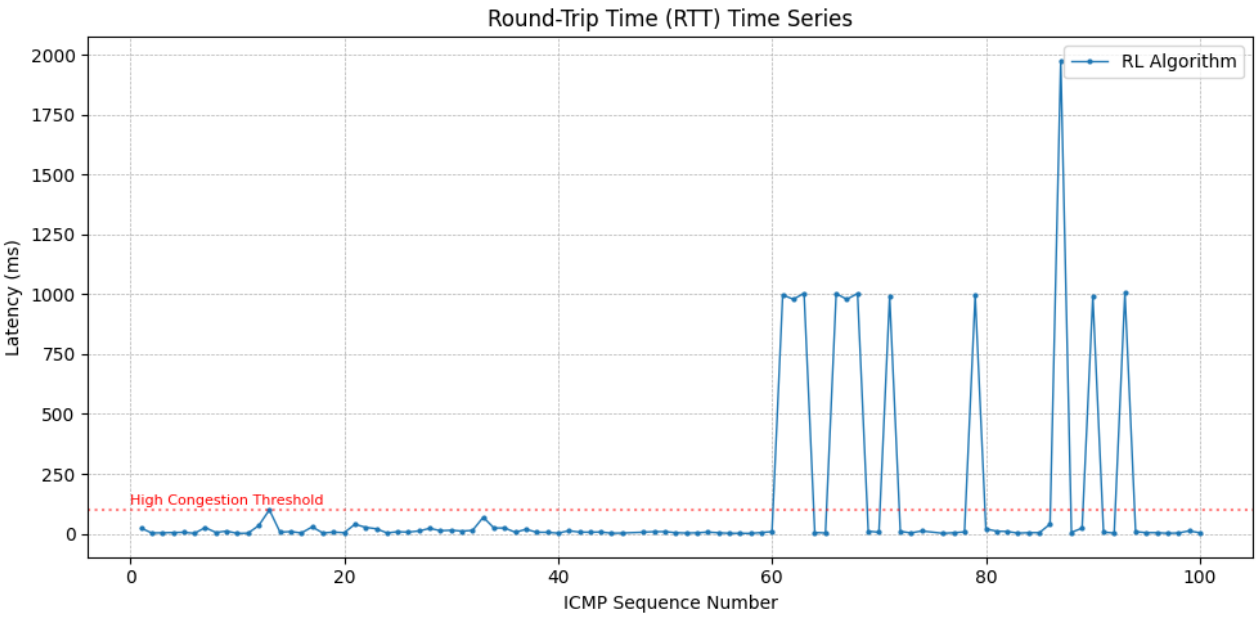

Figure 4.6: Latency of proposed model

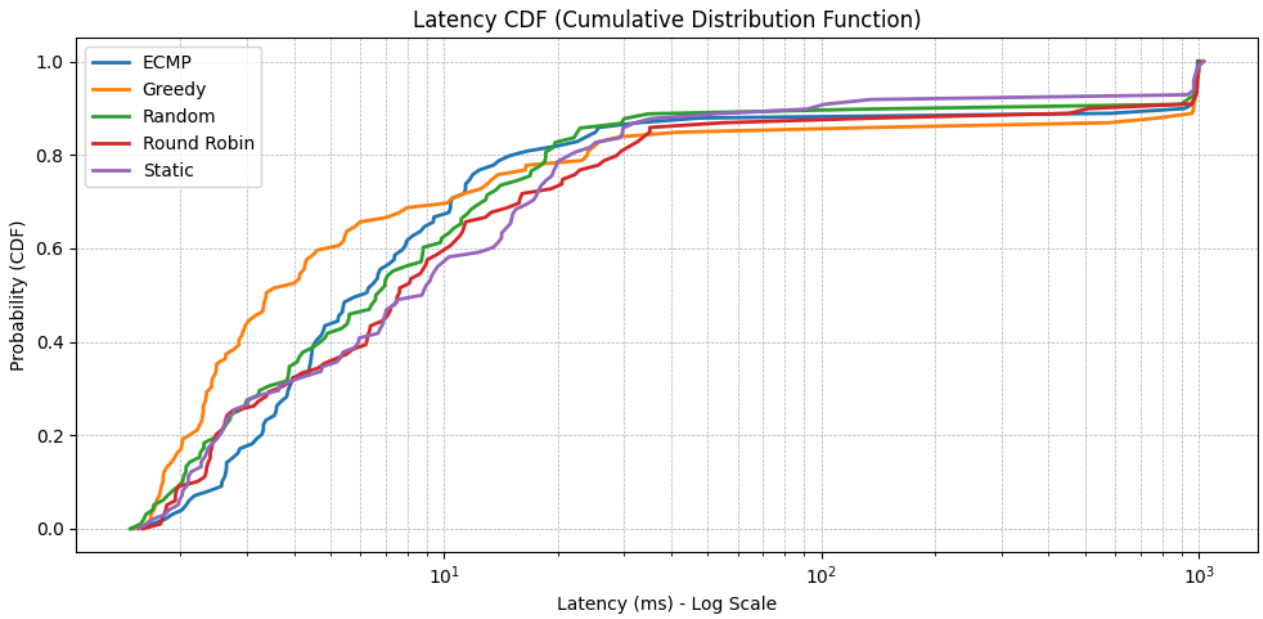

Figure 4.7: Latency cdf of other methods

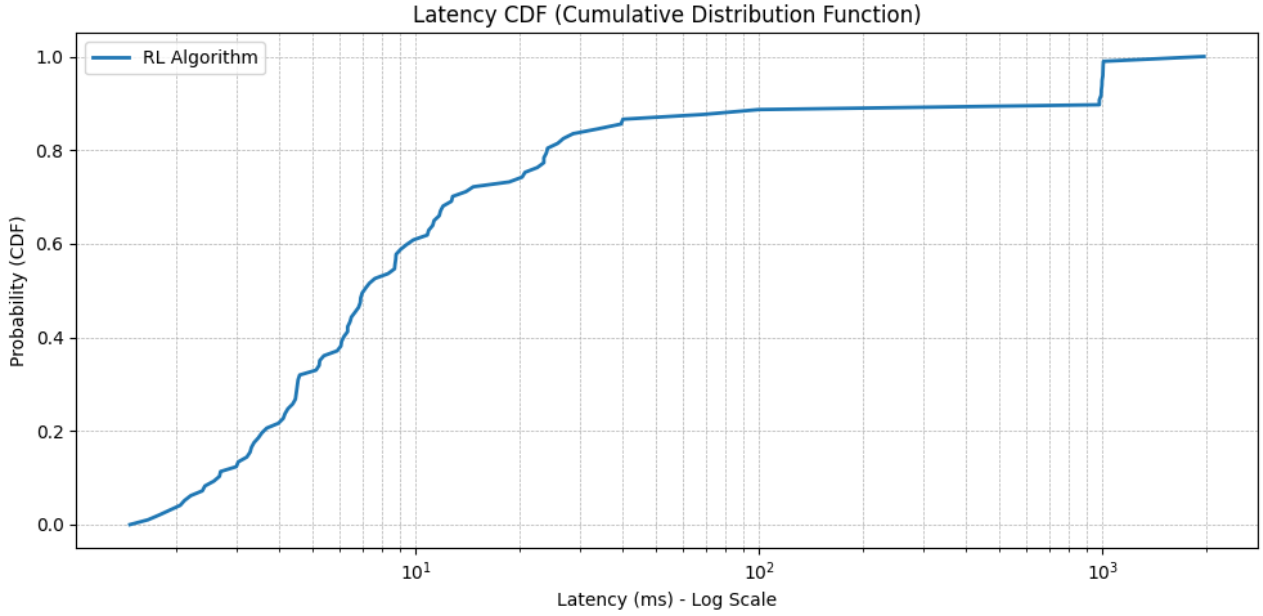

Figure 4.8: Latency cdf of proposed method

Even with such a high throughput, RTT of the connection managed by the RL agent was initially very low, but it increased due to exploration by the agent. But still it performed better than most algorithm. The Latency CDF graph shows that the proposed model is right in the middle.

4.3.3 Packet Retransmition

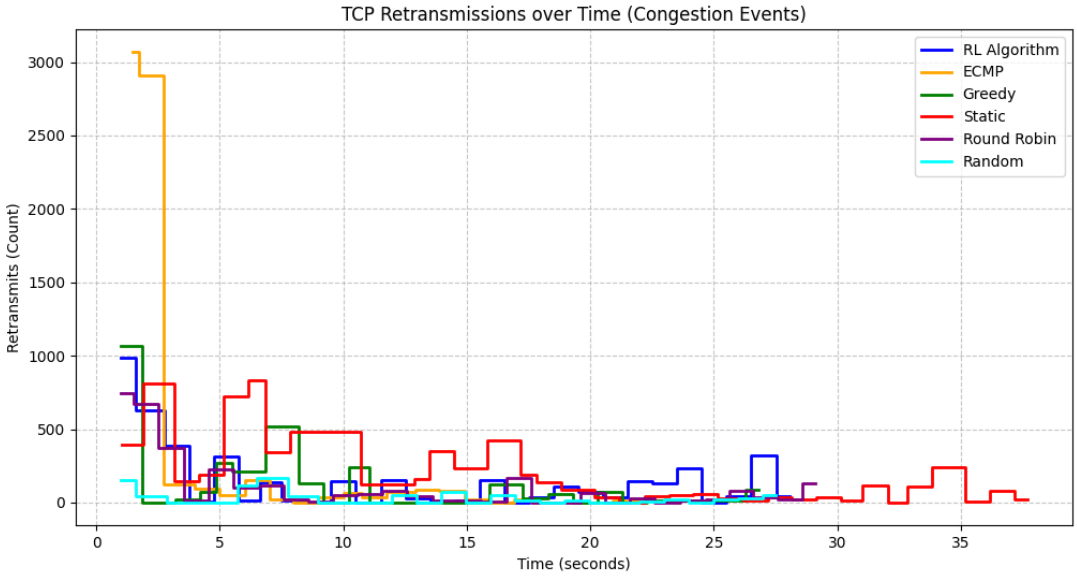

Figure 4.9: Retransmition Timeseries

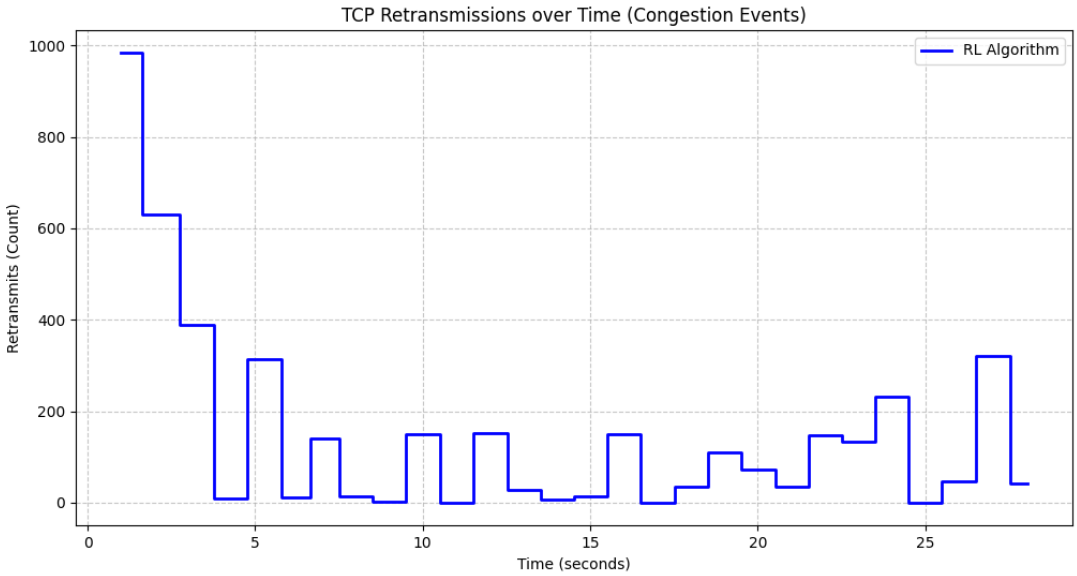

Figure 4.10: Retransmition Timeseries for proposed method

Despite the high throughput, the proposed model has very low retransmission rate. As, the model learns environment better, the retransmission rate decreases. It remains constant except when model tries to explore better outcomes.

4.3.4 TCP congestion window and Jitter

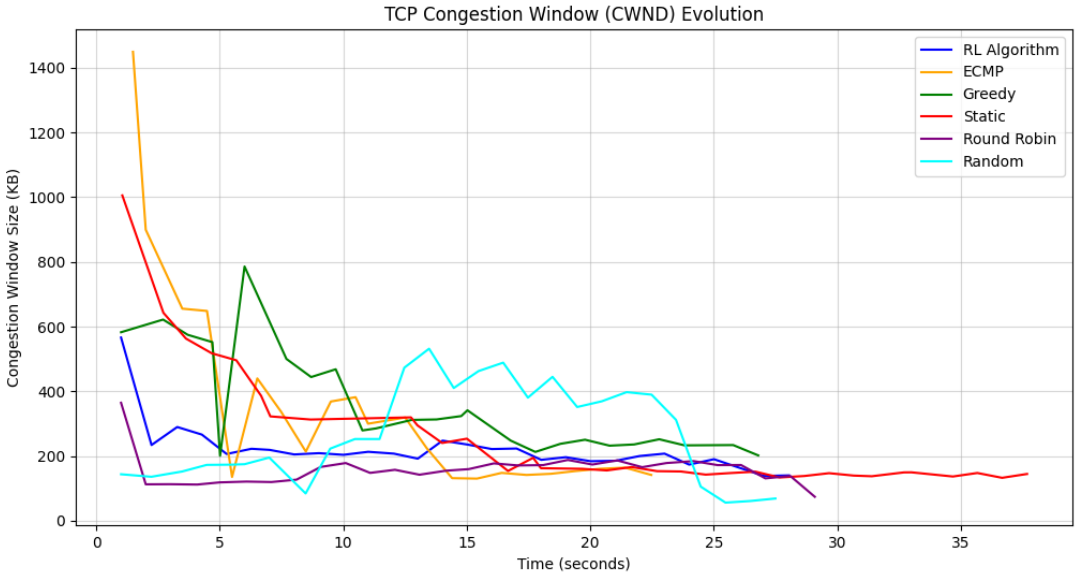

Figure 4.11: TCP CWND for all methods

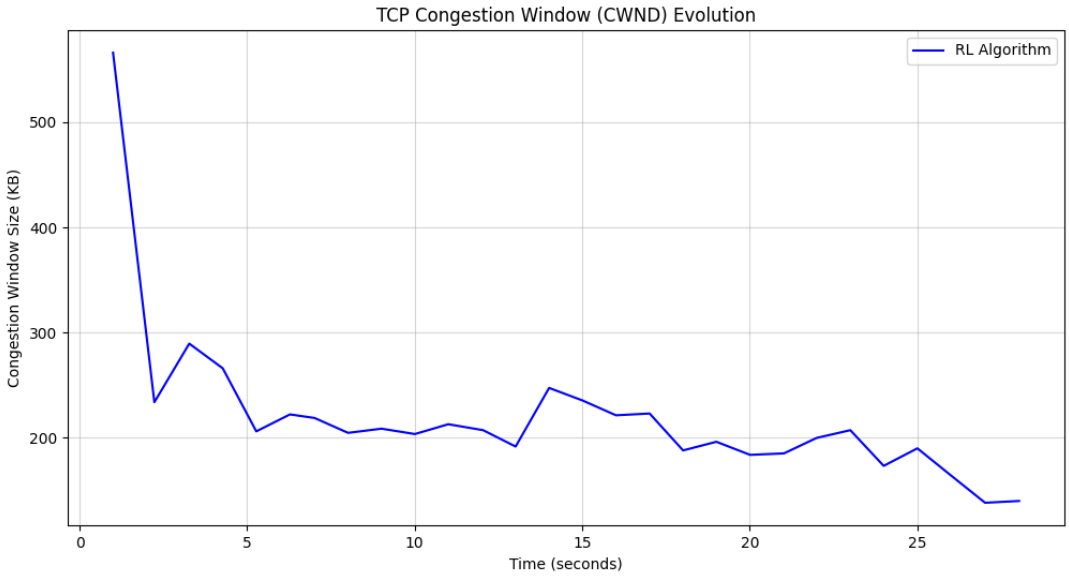

Figure 4.12: TCP CWND for proposed method

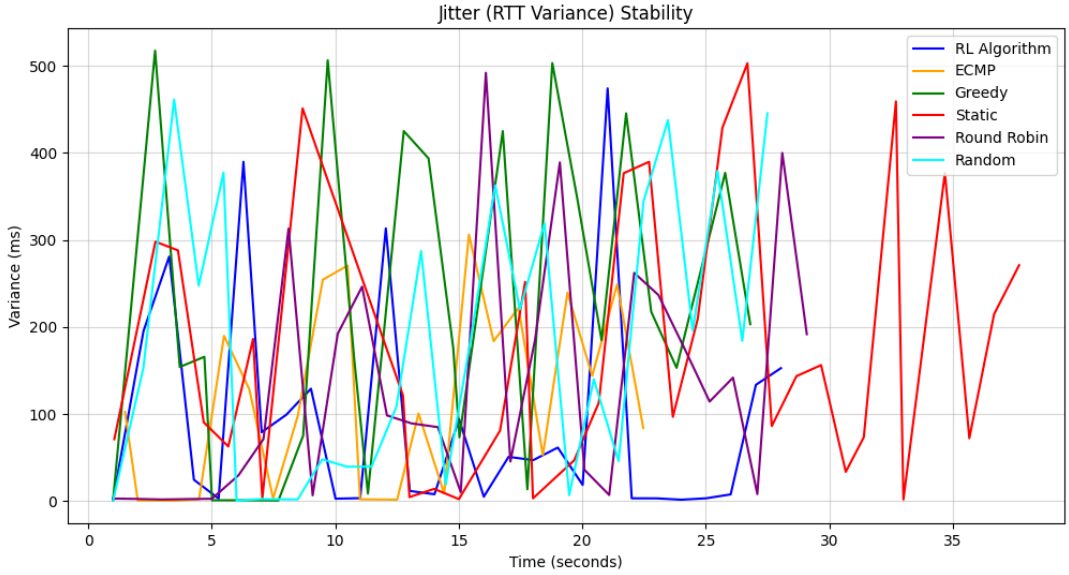

Figure 4.13: TCP jitter for all methods

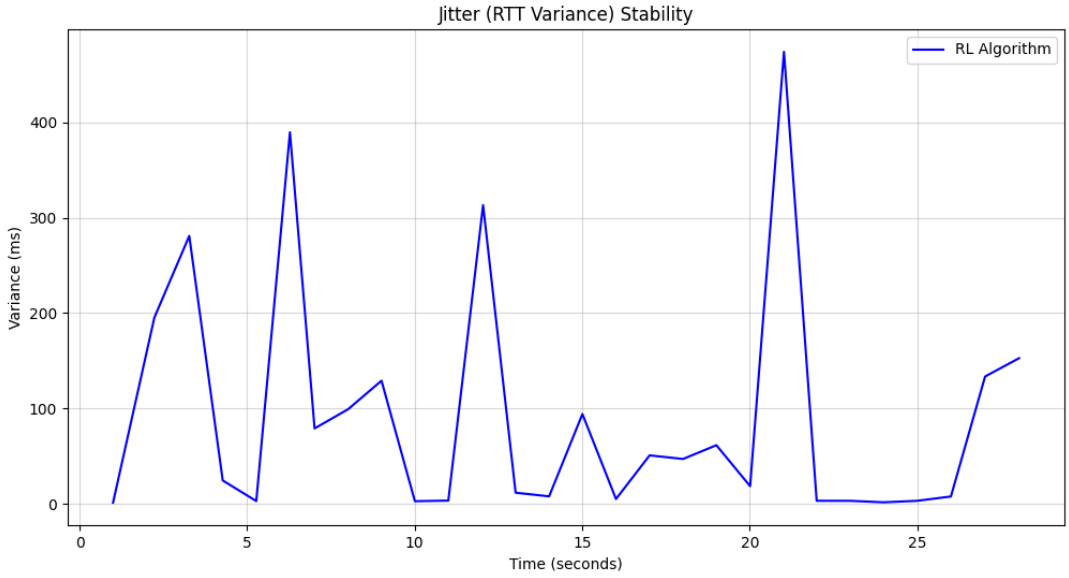

Figure 4.14: TCP jitter for proposed method

The proposed model has very constant TCP congestion window suggesting that model penalizes packet drop, which has resulted in less packet drops. In terms of Jitter, the proposed model demonstrate slightly better than Greedy, Random and Static algorithm.

4.3.5 RL Agent Training

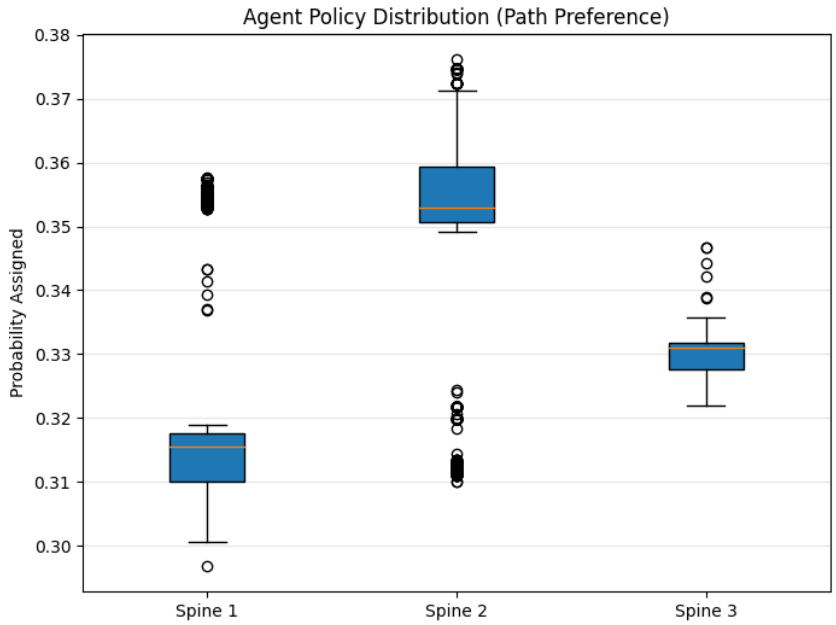

Figure 4.15: RL Agent Policy Boxplot

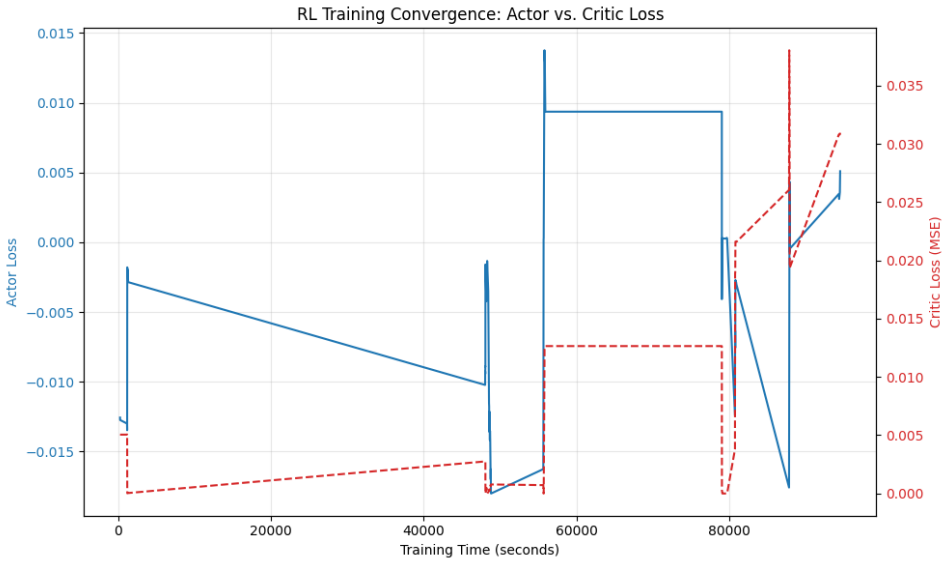

Figure 4.16: RL Agent Loss Convergence

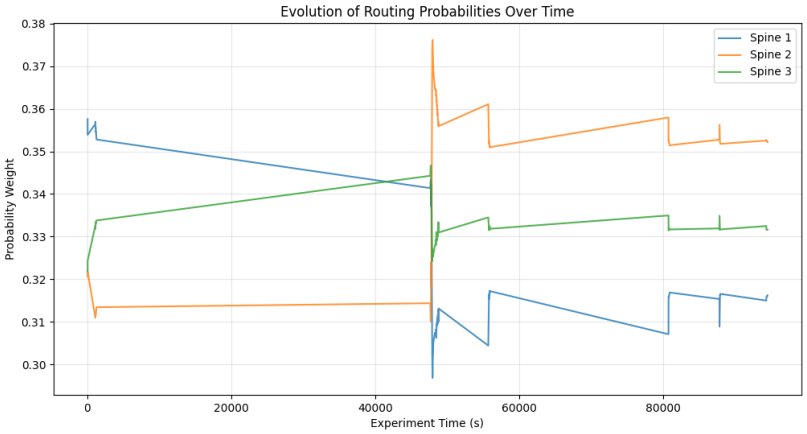

Figure 4.17: RL Agent Probability Evolution

The Figure 4.15 shows the box plot of the probability distribution for the three spines. The second one is clearly preferred by the model. It is followed by the third spine, which also has the least range and outliers out of the IQR. The first spine is least preferred by the model.

The Figure 4.17 shows the similar trends with Spine1 losing the favourability towards the end. The third spine shows the least variation.

The figure 4.16 illustrates the agent's learning dynamics. The initial phase shows stable but slow learning. At around 55,000 seconds, a significant network event triggered a sharp increase in both Actor and Critic losses, forcing the agent out of a local optimum. The subsequent volatility indicates active learning, where the Critic constantly re-evaluates state values in response to the Actor's changing policy. The fact that the losses do not simply explode but oscillate within a bounded range suggests the agent is successfully updating its policy without diverging, actively adapting to the dynamic traffic conditions

Chapter 5

Conclusion

This project demonstrates that Deep Reinforcement Learning (DRL) can serve as a practical and effective approach to dynamic traffic engineering in Software-Defined Networks in context of Data Center Network. By integrating an Actor-Critic agent directly into the Ryu controller, we enabled the control plane to make real-time routing decisions based on live network statistics such as port utilization and packet loss. The system successfully identifies high-bandwidth elephant flows, promotes them through selective group routing, and distributes them across multiple spine links based on predicted performance. Unlike traditional mechanisms such as ECMP, which assume uniform path quality and lack responsiveness to congestion, the DRL-based method adapts continuously to shifting network conditions.

Experimental results in our Mininet-based leaf-spine topology show that the RL agent learns to avoid hotspots, reduce flow collisions, and improve throughput across competing flows. By using measured rewards derived from throughput, utilization skew, and loss, the agent evolves policies that outperform static baselines and heuristic-driven routing. Moreover, the design supports real-time inference without imposing excessive overhead on the controller, demonstrating the feasibility of embedding learning-based logic into SDN control systems.

For future work, an idea could be extending the RL agent from per-leaf local decisions to a multi-agent or centralized global policy could unlock additional performance gains. Incorporating additional metrics such as latency, queue occupancy, or energy consumption may improve the agent's situational awareness. On the deployment side, migrating from Ryu to P4-enabled programmable switches would allow running learned policies directly in the data plane, reducing decision latency and enabling line-rate execution. Finally, evaluating the system at larger scales and under more diverse traffic patterns would help validate its robustness for real data center workloads.

References

[1] C. Hopps, *Rfc2992: Analysis of an equal-cost multi-path algorithm*, USA, 2000. DOI: 10.17487/RFC2992

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10, San Jose, California: USENIX Association, 2010, p. 19.

[3] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 1629–1637. DOI: 10.1109/INFCOM.2011.5934956

[4] P. Almasan, J. Suárez-Varela, B. Wu, S. Xiao, P. Barlet-Ros, and A. Cabellos-Aparicio, "Towards real-time routing optimization with deep reinforcement learning: Open challenges," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6. DOI: 10.1109/HPSR52026.2021.9481864

[5] W.-x. Liu, J. Cai, Q. C. Chen, and Y. Wang, "Drl-r: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks," *Journal of Network and Computer Applications*, vol. 177, p. 102 865, 2021, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102865 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520303313

[6] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Drsir: A deep reinforcement learning approach for routing in software-defined networking," 4, vol. 19, 2022, pp. 4807–4820. DOI: 10.1109/TNSM.2021.3132491

[7] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012. DOI: 10.1109/TSMCC.2012.2218595

[8] "Mininet: Network simulator. "[Online]. Available: https://mininet.org/

[9] "Ryu: Sdn framework. "[Online]. Available: https://ryu-sdn.org/

[10] "Pytorch. "[Online]. Available: https://pytorch.org/

[11] "Iperf. "[Online]. Available: https://iperf.fr

[12] "Python. "[Online]. Available: https://www.python.org

[13] "Matplotlib: A python plotting library. "[Online]. Available: https://matplotlib.org