Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering

Academic Year 2025/2026

Graduation Session March 2026

Cooperative data-driven routing

Relatori:

Alessio Sacco

Doriana Monaco

Candidati:

Danial Dehvan

Abstract

In modern computer networks, optimizing routing decisions is critical for maintaining high performance, low latency, and efficient resource utilization. The programmability offered by Software-Defined Networks (SDNs) has paved the path for the adoption of Machine Learning (ML) solutions to create innovative routing schemas. In particular, this thesis integrates Reinforcement Learning (RL) and Federated Learning (FL) within SDN to create a global routing model trained in a cooperative fashion.

Reinforcement Learning autonomously learns optimal routing policies that adapt to changing network conditions. Concurrently, Federated Learning allows multiple networks to collaboratively develop a global model without sharing raw traffic data, thereby maintaining privacy.

We rigorously tested our framework using Python simulations and the Mininet emulator, integrated with the Ryu SDN controller, and real-world traffic matrices. The results demonstrate that our approach consistently produces a generalizable model that is effective across various network topologies, significantly reducing training time while allocating the requested demand.

Acknowledgements

I would like to express my sincere gratitude to **Doriana Monaco** for her invaluable guidance, continuous support, and insightful advice throughout this thesis. Her expertise, dedication, and encouragement have been essential in shaping this work.
I am also grateful to **Prof. Alessio Sacco** for his supervision, constructive feedback, and support during this research.
I would like to thank the **Department of Control and Computer Engineering (DAUIN)** at **Politecnico di Torino** for providing an excellent academic environment and the resources necessary to carry out this work.
Special thanks go to my colleagues and friends at the department and in the research group for their helpful discussions and encouragement.
Finally, I am deeply thankful to my family and friends for their constant support, patience, and motivation throughout this journey.

Table of Contents

List of Tables VI

List of Figures VII

Glossary IX

1 Introduction 1
1.1 Thesis Structure . . . 3

2 Background 4
2.1 Software-Defined Networking (SDN) . . . 4
2.2 Traffic Engineering (TE) . . . 5
2.3 Reinforcement Learning (RL) for TE . . . 6
2.4 Federated Learning (FL) . . . 7

3 Related Work 9
3.1 Reinforcement Learning-based solutions . . . 9
3.2 Federated Learning-based solutions . . . 11
3.3 Federated Reinforcement Learning (FRL) . . . 13

4 Problem Definition and Research Objectives 16
4.1 The Problem Statement . . . 16
4.2 Challenges . . . 19
4.3 Proposed Approach . . . 20

5 Methodology 22
5.1 System Architecture . . . 23
5.1.1 Training Cycle . . . 25
5.2 The Reinforcement Learning Agent . . . 25
5.2.1 Environment . . . 26
5.2.2 Policy Network . . . 27
5.2.3 Training Loop . . . 27

5.3 Federated Learning Protocol . . . 28
5.3.1 Client Protocol . . . 28
5.3.2 Server Strategy . . . 29
5.3.3 Privacy Considerations . . . 29
5.4 Experimental Setup . . . 30

6 Simulation Results 31
6.1 Hyperparameter Analysis: Local Epochs and Federated Rounds . . . 31
6.1.1 Impact on Traffic Routing Performance . . . 31
6.1.2 Training Time vs. Performance Trade-off . . . 32
6.2 Comparative Performance Analysis . . . 33
6.2.1 Federated vs. Single-Client Performance . . . 34
6.2.2 Training Efficiency: Federated vs. Sequential Training . . . 35

7 Emulation Results 36
7.1 Emulation Infrastructure . . . 36
7.1.1 Mininet . . . 36
7.1.2 Ryu . . . 37
7.1.3 Overall system architecture . . . 38
7.2 Implementation Details . . . 39
7.2.1 Network Emulation . . . 39
7.3 Experimental Setup and Execution . . . 39
7.4 Emulation Results . . . 40
7.4.1 Flow Completion Time Distribution . . . 40
7.4.2 Flow Completion Time Variability . . . 41
7.4.3 Routing Decision Time . . . 42
7.4.4 Demand Satisfaction and Emulation Validation . . . 43
7.4.5 Federated Learning Convergence . . . 44
7.4.6 Comparison with Simulation Results . . . 45
7.5 Practical Insights and Observations . . . 46
7.5.1 Controller Responsiveness . . . 46
7.5.2 OpenFlow Overhead . . . 47
7.5.3 Federated Learning Communication . . . 47
7.5.4 Scalability Considerations . . . 47

8 Conclusion and Future Work 49
8.1 Future Work . . . 50

Bibliography 52

List of Tables

7.1 Flow completion time comparison 40

List of Figures

2.1 SDN vs Traditional Networks 5
2.2 Reinforcement Learning 7
2.3 Federated Learning 8

4.1 Traffic engineering within a single SDN domain. The controller must allocate each flow across candidate paths to maximise satisfied demand and avoid congested links (shown in red). Static shortest-path routing concentrates load on direct paths, while a learned policy can redistribute traffic adaptively. 17
4.2 Naive centralised approach: raw telemetry from every domain is sent to a single aggregator for training. This violates data privacy, creates a scalability bottleneck and single point of failure (SPOF), and yields a model that generalises poorly across heterogeneous domains. 18
4.3 Proposed federated reinforcement learning approach: each domain trains a local RL agent on private data and shares only model parameters with the federated server (solid blue arrows). The server aggregates them into a global model and distributes it back (dashed orange arrows). No raw traffic data ever leaves any domain. 19

5.1 High-Level System Architecture of Federated Reinforcement Learning for Traffic Engineering. 23

6.1 Heatmap of Average Flow/Demand Performance across different combinations of Local Epochs and Federated Rounds. 32
6.2 Total training time for different Epochs-Rounds configurations (MacBook Pro, Apple M1 CPU). 33
6.3 Flow/Demand comparison: Federated vs. Single-client TEAL performance across different topologies. 34
6.4 Total training time comparison: Federated (parallel) vs. sequential per-client training. 35

7.1 Emulation architecture showing the integration of the federated learning server, Ryu controller with federated Reinforcement Learning agent, and Mininet network for federated traffic engineering. 38
7.2 Cumulative distribution function of flow completion times. The FRL policy (blue) shows a steeper rise, indicating that a larger fraction of flows complete within any given time compared to ECMP (red). . 41
7.3 Routing decision time over the experiment duration. The FRL policy performs GNN inference every 20 seconds (blue dots), with mean 23.3 ms and median 22.2 ms. ECMP uses pre-computed routes with approximately 2.8 ms dictionary lookup (red dotted line). 42
7.4 Demand satisfaction across routing updates in the emulation environment. Each point represents the performance at a specific routing update for the B4 topology. The variation reflects the dynamic nature of traffic patterns and the system's adaptive response. 44
7.5 Comparison of demand satisfaction ratios between simulation (Chapter 6) and emulation results for the B4 topology. The emulation achieves approximately 8 percentage points lower performance than simulation, reflecting the additional complexities of real network emulation. 45

Glossary

DBMS
: Database Management System

SDN
: Software-Defined Networking

RL
: Reinforcement Learning

FL
: Federated Learning

ISP
: Internet Service Provider

TE
: Traffic Engineering

OVS
: Open vSwitch

OSPF
: Open Shortest Path First

BGP
: Border Gateway Protocol

MPLS
: Multiprotocol Label Switching

DQN

Deep Q-Networks

DRL

Deep Reinforcement Learning

GNN

Graph Neural Network

QoS

Quality of Service

AC

Actor-Critic

A2C

Asynchronous Advantage Actor-Critic

PPO

Proximal Policy Optimization

FRL

Federated Reinforcement Learning

IDS

Intrusion Detection System

DDoS

Distributed Denial of Service

IoT

Internet of Things

MEC

Mobile Edge Computing

ITS

Intelligent Transportation System

VANET

Vehicular Ad-hoc Network

FedAvg

Federated Averaging

DDPG

Deep Deterministic Policy Gradient

TEAL

Traffic Engineering with Adaptive Learning

GDPR

General Data Protection Regulation

HIPAA

Health Insurance Portability and Accountability Act

CDN

Content Delivery Network

SPOF

Single Point of Failure

ECMP

Equal-Cost Multi-Path

FCT

Flow Completion Time

ML

Machine Learning

MPS

Metal Performance Shaders

RAM

Random Access Memory

GPU

Graphics Processing Unit

Mbps

Megabits per second

Gbps

Gigabits per second

TCP

Transmission Control Protocol

UDP

User Datagram Protocol

IP

Internet Protocol

ARP

Address Resolution Protocol

WAN

Wide Area Network

JSON

JavaScript Object Notation

Chapter 1

Introduction

In modern computer networks, the efficient management of traffic is critical to ensuring high performance, low latency, and optimal resource utilization. As the scale and complexity of these networks increase, achieving such efficiency has become an increasingly challenging task. Traditional routing protocols, often based on static configurations or simple heuristics, are not well equipped to cope with the dynamic and unpredictable nature of contemporary network environments [1]. Factors such as varying traffic demands, link failures, congestion, and fluctuating bandwidth availability require routing strategies that can adapt quickly and intelligently to changing conditions [2].

Software-Defined Networking (SDN) has emerged as a powerful paradigm to address some of these challenges by decoupling the control plane from the data plane. SDN allows centralized control of the network, enabling programmability and dynamic adjustment of routing policies. However, even with SDN, the task of determining optimal routing decisions in real time remains complex. The centralized controller must process vast amounts of information and make decisions that balance competing objectives, such as minimizing latency, avoiding congestion, and optimizing resource usage. Recent advancements suggest that integrating distributed learning frameworks, such as Federated Reinforcement Learning, can help manage this complexity while maintaining data privacy and efficiency [3].

Researchers have explored the use of *machine learning techniques*, particularly *Reinforcement Learning (RL)*, to enable the network to learn optimal routing policies through interaction with the environment. RL offers the advantage of allowing the system to autonomously discover strategies that adapt to changing network conditions without requiring a predefined model of the network's behavior [4]. However, most existing approaches assume that all data is available at a central point, which introduces significant challenges in terms of *scalability, privacy, and cross-domain collaboration.*

In real-world scenarios, networks are often operated by different administrative

domains, such as Internet Service Providers (ISPs), data centers, or enterprise networks. These entities may be unwilling or unable to share detailed traffic data due to privacy concerns, regulatory constraints, or business interests. This lack of data sharing limits the ability to build comprehensive models that could benefit from a broader view of network behavior across domains.
To address these challenges, this thesis proposes a novel approach that combines *Reinforcement Learning (RL)* and *Federated Learning (FL)* within SDN environments. While RL enables networks to autonomously learn optimal routing strategies based on local observations and feedback, FL allows multiple networks to collaboratively improve their models without sharing raw data. Instead, only model updates are exchanged, preserving data privacy and respecting the sovereignty of each network domain.
Specifically, this work integrates *TEAL* [5], a state-of-the-art RL framework for traffic engineering, with *Flower* [6], a flexible FL framework that supports heterogeneous participants. The proposed system enables multiple networks to jointly train traffic engineering models while maintaining local control over sensitive data. Simulations are conducted using *Mininet* with the *Ryu* SDN controller, evaluating performance on realistic topologies such as *GEANT* and *Abilene*, using traffic matrices that reflect real-world demand patterns.
The main contributions of this thesis are:

- The design of a privacy-preserving, federated reinforcement learning framework for SDN traffic engineering.
- The extension of the Reinforcement Learning Agent to operate in a federated setting using Flower, enabling distributed learning across multiple networks.
- A thorough evaluation of the proposed approach through simulations on realistic topologies, demonstrating its potential to improve routing efficiency while preserving privacy.

Our work directly addresses this critical and multifaceted gap by proposing and rigorously evaluating a novel Federated Reinforcement Learning framework specifically tailored for **Traffic Engineering with Adaptive Learning (TEAL)** [5]. Our system fundamentally differentiates itself from prior art through several key innovations. A core contribution is the framework's ability to allow multiple distributed TEAL agents, each operating autonomously within its respective SDN domain, to collaboratively learn an optimal global routing policy. Crucially, this collaborative learning is achieved without any raw, sensitive traffic data ever leaving the local network boundaries. This architectural choice inherently upholds paramount privacy requirements for multi-provider networks and inter-enterprise collaborations, contrasting sharply with centralized DRL approaches that demand full data visibility. Furthermore, the federated architecture inherently distributes

the computational burden of DRL model training across numerous client devices. This significantly mitigates the scalability bottlenecks and computational strain associated with a single, monolithic centralized controller, allowing the system to gracefully scale to a larger number of heterogeneous network domains and traffic loads while offering substantial efficiency gains through parallel training.
From a technical implementation perspective, our approach uniquely synergizes the adaptive, real-time decision-making capabilities of Deep Reinforcement Learning with the inherent programmability of SDN. Each local SDN controller acts as the immediate environment for its respective TEAL agent, providing real-time state observations and executing the learned routing policies through flow rule manipulation. To realize this, we utilize the `Flower` framework [6] to orchestrate the federated learning process. `Flower`'s modularity and robust aggregation strategies provide a solid, open-source foundation for handling complex aspects like model aggregation and client selection, demonstrating the feasibility of building such a system using contemporary FL tools. To our extensive knowledge, a comprehensive investigation into the performance characteristics, training efficiency, and practical deployment considerations of FRL specifically applied to dynamic, multi-domain Traffic Engineering—rigorously evaluated using a modern framework like `Flower`—has not been thoroughly explored in existing academic literature.

1.1 Thesis Structure

The remainder of this document is organized as follows. Chapter 2 provides the necessary background on SDN, traffic engineering, RL, and FL, while Chapter 3 reviews the related work in these fields. Chapter 4 defines the problem and the specific objectives of this research in detail. The methodology, including the design of the proposed system and the experimental setup, is described in Chapter 5. Subsequently, Chapter 6 presents the results and discusses their implications. Finally, Chapter 7 concludes the thesis and outlines potential directions for future work.

Chapter 2

Background

The evolution of computer networks has led to increasingly complex architectures that must handle diverse applications, dynamic traffic patterns, and stringent performance requirements. From the early days of packet-switched networks to the current era of ubiquitous connectivity, the demands placed on network infrastructure have grown exponentially. This necessitates continuous innovation in network management and optimization strategies. In this context, several technological paradigms have emerged to improve flexibility, efficiency, and intelligence in managing networks. This section introduces the key concepts that form the foundational pillars of this thesis. Understanding these individual components is crucial before exploring their synergistic integration within our proposed framework.

2.1 Software-Defined Networking (SDN)

Software-Defined Networking (SDN) represents a fundamental shift in network architecture, decoupling the control plane from the data plane. Traditionally, network devices (routers, switches) integrate both control logic (e.g., routing protocols, management interfaces) and data forwarding capabilities. This tight coupling often leads to vendor lock-in, complex configurations, and slow innovation cycles. SDN addresses these limitations by centralizing network intelligence in a logically centralized controller, often referred to as the *SDN controller* or *network operating system*. This controller maintains a global, unified view of the entire network topology and its current state.

The key benefits of SDN are manifold. Firstly, it provides a programmable interface (e.g., *OpenFlow*) to dynamically configure and manage network devices, allowing network administrators to define forwarding behavior through high-level policies rather than device-specific commands. OpenFlow is the most widely adopted southbound protocol in SDN, enabling the controller to install flow rules in switches,

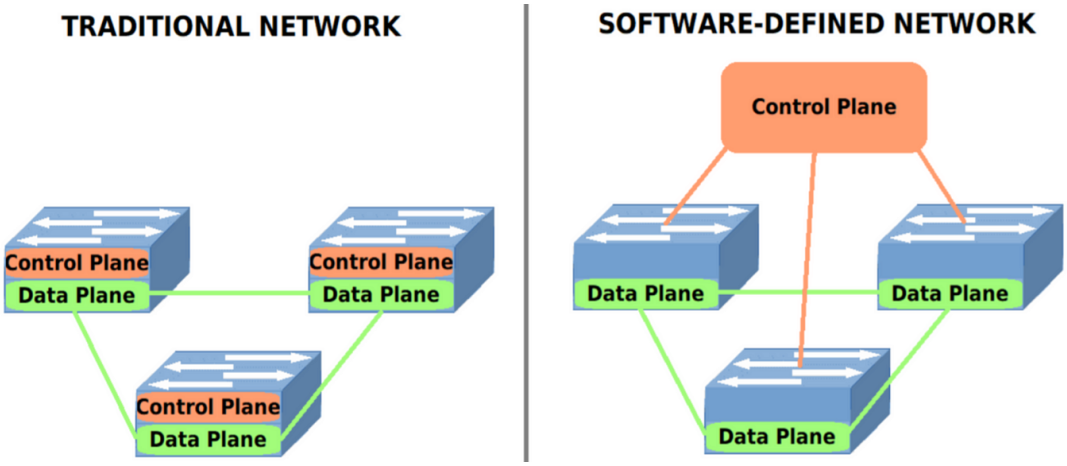

Figure 2.1: SDN vs Traditional Networks

query statistics, and receive notifications about unmatched packets. Software switches such as Open vSwitch (OVS) implement the OpenFlow protocol and maintain flow tables with complex matching criteria (Layer 2 through Layer 4 headers), supporting actions such as forwarding, dropping, or modifying packets. This programmability enables faster deployment of new services, facilitates rapid experimentation with novel routing algorithms, and simplifies network operations. Secondly, the global view provided by the controller is a significant advantage over traditional distributed routing protocols (like OSPF or BGP), which rely on localized information and often converge slowly to network changes. With a centralized view, the SDN controller can make more optimal, global routing decisions in response to changing network conditions, traffic demands, or even security threats. This centralized control and programmability lay the groundwork for intelligent traffic management and optimization.

2.2 Traffic Engineering (TE)

Traffic Engineering (TE) is a crucial discipline in network management, focusing on optimizing the performance of a network by intelligently adjusting how traffic flows through the network infrastructure. The primary objectives of TE are multifaceted and often involve a delicate balance between various performance metrics. These objectives typically include minimizing network congestion by preventing bottlenecks, balancing load across multiple available paths to ensure efficient resource utilization, reducing latency for time-sensitive applications, and improving overall network throughput and resource efficiency. Effective TE can significantly enhance user experience, reduce operational costs, and improve the

resilience of the network.
In traditional networks, TE relies heavily on configuring static routing policies, adjusting link weights, or deploying protocols like *MPLS (Multiprotocol Label Switching)* for explicit path control. However, these methods can be rigid and slow to adapt to dynamic traffic patterns or network failures. In contrast, SDN-enabled networks offer a more agile and effective platform for TE. The centralized SDN controller, with its real-time visibility into traffic statistics and network topology, can collect comprehensive information about current network conditions. This rich data allows the controller to apply sophisticated algorithms – ranging from classical optimization techniques to advanced machine learning models – to determine and enforce optimal routing decisions dynamically. This capability transforms TE from a static configuration task into a dynamic, adaptive process, crucial for modern, highly dynamic network environments.

2.3 Reinforcement Learning (RL) for TE

Reinforcement Learning (RL) is a powerful paradigm within machine learning where an intelligent agent learns to make optimal decisions through interaction with an environment. Unlike supervised learning, which requires labeled datasets, or unsupervised learning, which focuses on finding patterns in unlabeled data, RL operates on a trial-and-error basis. The agent performs actions in the environment, observes the resulting state changes, and receives feedback in the form of numerical rewards or penalties. The goal of the RL agent is to learn a *policy* – a mapping from states to actions – that maximizes the cumulative reward over time. In the context of traffic engineering, an RL agent can effectively represent the SDN controller. The environment would be the network itself, with its dynamic traffic patterns, link capacities, and node states. The actions an RL agent could take include adjusting routing paths, modifying forwarding rules, or applying traffic shaping policies. The rewards could be designed to reflect desired network performance objectives, such as minimizing latency, maximizing throughput, or avoiding congestion. RL's inherent ability to adapt to dynamic and uncertain conditions makes it particularly well-suited for the challenges of TE, where network conditions are constantly fluctuating and a precise mathematical model of network behavior might be difficult or impossible to formulate. Various RL algorithms have been explored in this domain, including model-free methods like *Q-learning* and its deep learning extension, *Deep Q-Networks (DQN)*, which leverage neural networks to approximate the Q-value function, and *Actor-Critic methods*, which learn both a policy and a value function concurrently. These algorithms enable the system to learn optimal routing strategies autonomously without explicit programming for every conceivable network state.

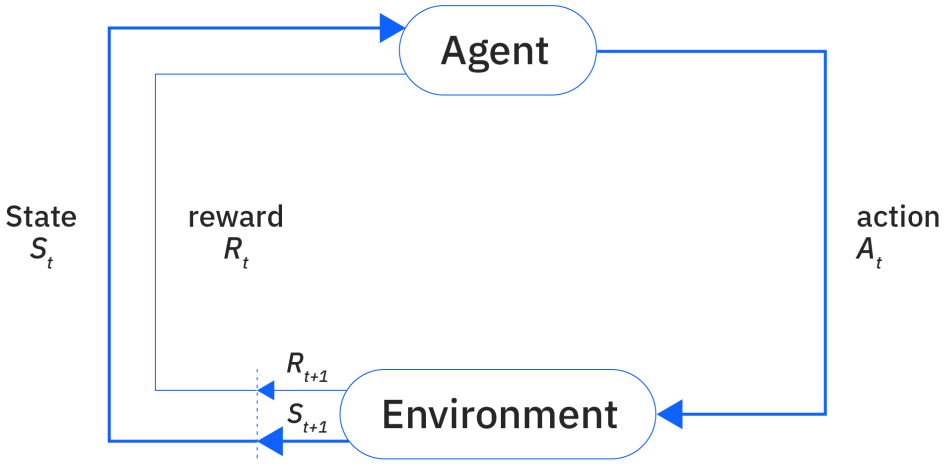

Figure 2.2: Reinforcement Learning

2.4 Federated Learning (FL)

Federated Learning (FL) is a distributed machine learning technique that addresses the critical challenge of data privacy and sovereignty in collaborative AI. In a traditional centralized machine learning setup, all data is collected and processed at a central server. However, this model becomes problematic when dealing with sensitive data (e.g., patient records, financial transactions, or private network traffic data) or when data is geographically dispersed across multiple administrative domains due to regulatory or privacy concerns. FL provides an elegant solution: it enables multiple participants (referred to as *clients*) to collaboratively train a shared machine learning model without ever exchanging their raw local data. Instead, each participant trains the model locally on their own dataset. Only model updates, such as gradient vectors or trained model parameters, are shared with a central server (or an aggregator). The aggregator then combines these updates from all participants to create an improved global model, which is then sent back to the clients for their next round of local training. This iterative process allows for the construction of a robust global model while ensuring that sensitive raw data remains on the client devices. FL is particularly valuable in scenarios where *data privacy*, *sovereignty*, *regulatory compliance* (e.g., GDPR, HIPAA), or bandwidth limitations prevent the sharing of sensitive information. In the context of multi-domain traffic engineering, FL offers a groundbreaking approach for different administrative domains (e.g.,

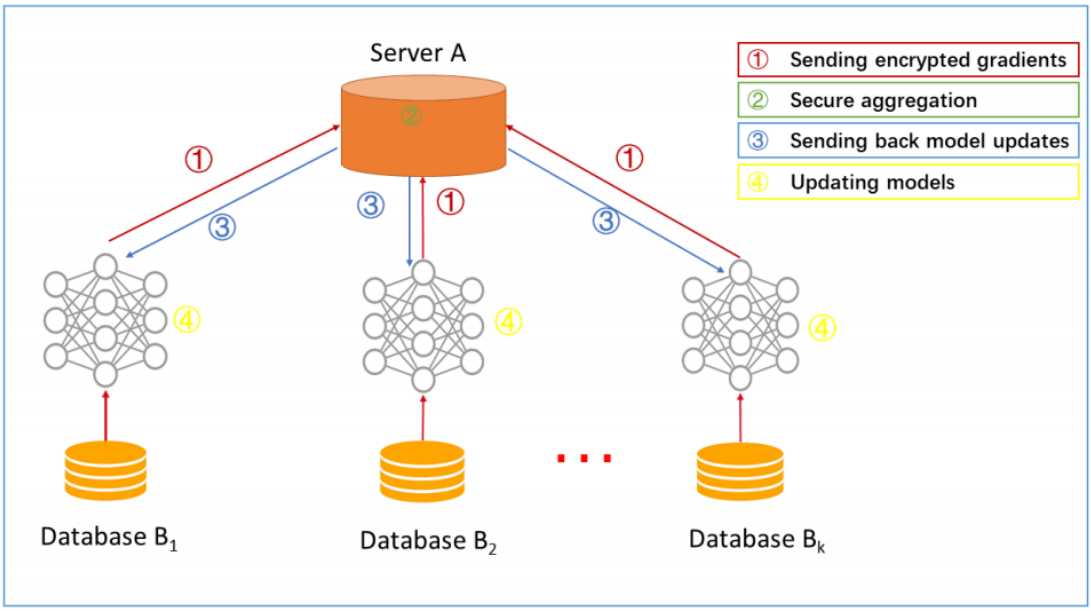

Figure 2.3: Federated Learning

Internet Service Providers, enterprise networks) to collaborate on learning better, more efficient routing policies without exposing their detailed, proprietary traffic data or internal network configurations.

Chapter 3

Related Work

This chapter provides an exhaustive review of the existing academic literature that forms the intellectual bedrock for our proposed Federated Reinforcement Learning (FRL) framework for Traffic Engineering (TE) in Software-Defined Networks (SDN). Our exploration meticulously dissects advancements across three pivotal, yet often disparate, research domains: the application of Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) for dynamic network management, the paradigm of Federated Learning (FL) in distributed and privacy-sensitive networking environments, and the emerging synthesis of these two, Federated Reinforcement Learning. By undertaking a detailed analysis of methodologies, experimental setups, achieved contributions, and inherent limitations of prior research, this chapter precisely positions our novel contribution, delineates its unique attributes, and clearly articulates the specific, unresolved research challenges that our work endeavors to overcome in the pursuit of scalable, privacy-preserving, and highly efficient traffic engineering solutions.

3.1 Reinforcement Learning-based solutions

The architectural revolution brought forth by Software-Defined Networking (SDN), characterized by the decoupling of the control plane from the data plane, has fundamentally transformed network programmability and fostered a global network observability. This shift has established an unprecedented opportunity for the application of Reinforcement Learning (RL) and particularly Deep Reinforcement Learning (DRL) to address the intrinsically complex and dynamic optimization problems prevalent in modern networks, with Traffic Engineering (TE) standing as a prime example. In this paradigm, DRL agents, typically integrated within the logically centralized SDN controller, gain the ability to comprehensively observe network states (e.g., link bandwidth utilization, queue depths, end-to-end delays,

traffic matrices), learn optimal routing and resource allocation policies through iterative interactions with the network environment, and dynamically enforce these policies by adjusting flow rules or modifying link weights to achieve specific performance objectives such as minimizing congestion, reducing latency, maximizing throughput, or ensuring Quality of Service (QoS).

A diverse array of DRL algorithms has been ingeniously adapted and applied to TE in SDN environments. Early pioneering works predominantly leveraged value-based methods, most notably Deep Q-Networks (DQN). For instance, Liu et al. [7] proposed a framework where the agent learns to select optimal paths within the network to minimize end-to-end delay for incoming flows. Their model mapped the complex network state directly to Q-values for each possible routing action, demonstrating promising results in smaller topologies. However, a significant limitation of traditional DQN is its reliance on discrete action spaces, which becomes a bottleneck when dealing with the vast, continuous, or combinatorially explosive action spaces inherent in large-scale or multi-path routing scenarios. The curse of dimensionality quickly renders such approaches impractical for real-world deployments.

To circumvent the limitations of discrete action spaces and to improve learning stability and convergence, policy-gradient methods and their variants, such as Actor-Critic (AC), Asynchronous Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO), have gained substantial traction. Liu et al. [8] demonstrated the efficacy of a DRL-based agent in developing robust routing policies capable of adapting to highly fluctuating traffic demands and handling dynamic network events like link failures. Their approach showcased superior adaptability and performance metrics compared to heuristic or traditional routing protocols. Furthermore, acknowledging the intrinsic graph-structured nature of network topologies, researchers have begun integrating Graph Neural Networks (GNNs) with DRL. Jiang et al. [2] introduced a GNN-enhanced DRL framework that could learn intricate, high-dimensional representations of network topology, traffic patterns, and link capacities. This synergy enabled the DRL agent to make more intelligent and scalable routing decisions, especially across diverse and evolving network graphs, by capturing relational dependencies between network elements. Comprehensive surveys by Abbasova et al. [4] and Bhattacharjee [1] provide invaluable overviews, systematically classifying these DRL-based routing optimization techniques based on their underlying algorithms, network state representations, action space definitions, and reward functions. They highlight that while DRL offers unprecedented adaptability, the centralized paradigm remains predominant.

Despite these significant methodological advancements, the vast majority of DRL-based TE solutions fundamentally operate under a centralized learning and decision-making paradigm. This architecture necessitates that the DRL agent, typically integrated into the SDN controller, maintains access to real-time, global, and highly

granular network state information. While this centralized data aggregation is feasible for tightly controlled, single administrative domains or laboratory simulations, its application to expansive, multi-domain networks introduces several critical challenges.

Foremost among these is a fundamental *scalability and performance bottleneck*; as network size grows, a single entity processing a deluge of global data struggles with the computational load, often leading to delayed policy updates. This architectural centralized reliance also creates a *single point of failure* where a central malfunction can result in widespread outages. Furthermore, the continuous reporting of detailed states from distributed elements generates *excessive communication overhead* that consumes bandwidth and introduces latency. Beyond performance, there are *profound privacy concerns*, as operators are often legally or competitively prohibited from sharing raw, proprietary traffic data with an external central entity. Finally, centralized models frequently struggle with *data heterogeneity*, failing to generalize across diverse domains with unique topologies and operational policies.

While some literature explores distributed RL for traffic management, particularly in urban environments like intelligent traffic signal control, they generally do not fully address the privacy implications in a multi-domain, competitive context. For example, Abdulhai et al. [9] pioneered the use of RL for adaptive traffic signal control, where individual intersection agents learn independently. More sophisticated approaches like Tan et al. [10] extended this to multi-agent systems for large-scale urban traffic, where agents might coordinate through shared states or message passing. However, these works typically focus on cooperative scenarios without explicit privacy-preserving mechanisms or secure model aggregation techniques inherent to Federated Learning, which are essential for sensitive network management applications involving multiple stakeholders. On the contrary, we propose a Federated Reinforcement Learning framework for SDN traffic engineering that explicitly incorporates these privacy-preserving mechanisms and secure model aggregation, enabling multiple network domains to collaboratively learn optimal routing policies without ever sharing raw traffic data—thereby addressing the very requirements that distributed RL approaches have left unmet in multi-stakeholder, competitive network environments.

3.2 Federated Learning-based solutions

Federated Learning (FL) stands as a groundbreaking distributed machine learning paradigm that fundamentally redefines how collaborative intelligence can be achieved while rigorously preserving data privacy. Its core principle dictates that multiple clients collectively train a shared global model without ever exposing their raw, sensitive local datasets. Instead, clients perform computations on their

local data and only transmit aggregated model updates (e.g., gradients, weight differences, or model parameters) to a central server. This server then performs a robust aggregation (e.g., Federated Averaging) to synthesize a refined global model, which is subsequently disseminated back to the clients for the next round of local training. This unique architecture inherently safeguards data privacy, significantly reduces communication bandwidth requirements by avoiding raw data transfer, and mitigates the need for a single, massive data repository, making FL exceptionally well-suited for a myriad of applications within the networking domain.

FL has demonstrated its transformative potential across a diverse spectrum of network management and security challenges. In the realm of Intelligent Transportation Systems (ITS) and Vehicular Ad-hoc Networks (VANETs), FL has proven highly effective for predictive traffic management. For instance, Liu et al. [11] introduced a privacy-preserving traffic flow prediction framework that enables collaborative learning of traffic patterns. Their approach allows distributed entities to optimize service access and make routing decisions while preserving individual data privacy and significantly reducing the communication overhead associated with centralized data collection. These studies underscore FL's unparalleled capability to leverage distributed, real-time data from mobile entities for collaborative intelligence in dynamic environments.

Beyond traffic management, FL offers a powerful mechanism for collaborative anomaly detection and the development of robust Intrusion Detection Systems (IDS). Ferrag et al. [12] provided a comprehensive analysis highlighting FL's pivotal role in cybersecurity, demonstrating its application in detecting sophisticated malware, Distributed Denial of Service (DDoS) attacks, and various network intrusions. By allowing different network entities—such as IoT devices, enterprise network segments, and ISP routers—to train models on their local security logs without sharing raw data, FL enables the construction of a more generalized and robust threat detection model. This approach fosters collaborative threat intelligence while rigorously protecting sensitive network forensic data.

In the context of next-generation cellular networks (5G and beyond) and Mobile Edge Computing (MEC), FL facilitates distributed resource allocation, dynamic spectrum sharing, and intelligent load balancing. Research by Chen and Liu [13] explored FL for dynamic resource management in MEC environments, where edge servers or base stations collaboratively learn optimal strategies for allocating computational and communication resources. This distributed learning paradigm minimizes latency, enhances scalability, and ensures privacy by avoiding the centralization of sensitive network performance data. Furthermore, as the proliferation of Internet of Things (IoT) devices generates vast datasets at the network edge, FL has become a natural fit for enabling on-device model training. Hameed and Mohamad [14] provided a systematic review of FL applications in IoT scenarios, underscoring its crucial role in facilitating privacy-preserving data analytics and

collaborative intelligence directly at the network edge.
The integration of FL within Software-Defined Networking architectures is also an actively evolving area of research. Recent works have explored how SDN's logically centralized control plane can serve as the orchestration layer for the FL training process, managing model aggregation and distribution, while its distributed data plane ensures that sensitive raw data remains localized at the client [15]. This body of work delves into critical challenges specific to FL in SDN, including ensuring communication efficiency, achieving robustness against malicious client contributions, and managing data heterogeneity. While Federated Learning (FL) has effectively addressed many privacy and scalability challenges inherent in centralized machine learning, its applications in networking have traditionally focused on supervised or unsupervised tasks, such as predictive analytics and traffic classification. In recent years, however, the integration of FL with Reinforcement Learning (RL) has emerged as a promising paradigm, demonstrating greater potential for tackling complex, adaptive, and real-time network control problems—such as dynamic traffic routing—thereby paving the way for more intelligent and decentralized network management strategies.

3.3 Federated Reinforcement Learning (FRL)

The synergistic fusion of Federated Learning and Reinforcement Learning has given rise to Federated Reinforcement Learning (FRL), an emergent and highly promising paradigm. FRL aims to combine the benefits of decentralized, privacy-preserving collaborative model training from FL with the adaptive, sequential decision-making capabilities of RL. This makes FRL particularly relevant for complex, distributed control systems where multiple agents or entities need to learn a common optimal policy through interaction with their local environments, all while rigorously preserving the privacy of their raw state-action observations and reward signals. FRL methodologies can be broadly categorized based on the specific components of the RL process that are federated. The most prevalent approach is the *Federated Averaging of Policy or Value Network Parameters*, often referred to as Model Federation. In this schema, each client trains its local RL agent—such as DQN, PPO, or DDPG—using its local environmental interactions. Periodically, clients transmit their local model parameters, such as the weights of their policy or value function networks, to a central server. The server then aggregates these parameters, typically using Federated Averaging (FedAvg), to produce an improved global model that is subsequently broadcast back to the clients. Qi et al. [3] provide a foundational review of these FRL techniques, discussing variants of federated averaging applied to different underlying RL algorithms. This method effectively preserves privacy as only model updates, rather than raw experiences,

are shared. Alternatively, systems may utilize *Federated Averaging of Gradients*, where clients compute gradients based on their local losses and send only these gradients to the server for aggregation. A less common approach, due to heightened privacy concerns and data heterogeneity, is the *Federated Averaging of Experiences or Trajectories*. In this data federation model, clients might share aggregated summaries or sanitized versions of their experience trajectories to train a central agent, though this requires strong privacy mechanisms like differential privacy. Finally, in actor-critic architectures, clients may participate by sending *Aggregated Actor/Critic Outputs*, such as value estimates or policy distributions, rather than full model weights.

However, the implementation of FRL faces a compounded set of challenges that stem from the complexities of intersecting FL with RL. A critical hurdle is the often *severe non-i.i.d. (non-independent and identically distributed) nature* of client environments. Unlike typical FL where data may be statistically independent but differently distributed, in FRL, each client's environment might possess unique dynamics, distinct reward structures, varying state-action spaces, or different causal relationships. This profound heterogeneity can significantly hinder global model convergence, lead to unstable training, or result in a suboptimal global policy that performs poorly for individual clients. Nevertheless, the research community has proposed concrete mitigation strategies: for instance, Jin et al. [16] introduce federated RL algorithms with personalization heuristics that achieve convergent policies despite environment heterogeneity, while the broader FRL literature [3] surveys techniques for communication-efficient aggregation, event-triggered synchronization, and model compression. Furthermore, RL training typically demands a large number of interactions and frequent updates to converge, leading to challenges regarding *communication efficiency and latency sensitivity*. The transmission of model parameters, especially for large deep neural networks, can incur substantial communication overhead and undermine the real-time responsiveness required for dynamic control applications. These issues are exacerbated by *convergence stability* concerns; the inherent non-stationarity of the RL environment, combined with the asynchronous nature of FL updates, makes balancing exploration and exploitation highly complex. Finally, while FL minimizes raw data exposure, FRL introduces *enhanced privacy and security concerns*, as model updates could potentially leak sensitive information through reconstruction attacks, or malicious clients could contribute poisoned updates to degrade global model performance. However, robust aggregation schemes [17] and differential privacy extensions available in FL frameworks offer established mitigations for these risks. In light of these existing mitigations, we adopt FRL for SDN traffic engineering and explicitly evaluate our system with respect to convergence behaviour, communication overhead, and performance under heterogeneous topologies in Chapters 5–7, demonstrating that FRL remains a viable and effective approach when these considerations are taken

into account.

Despite these formidable challenges, FRL demonstrates immense promise in various distributed control domains. In networking, FRL frameworks have been proposed for resource management and, more recently, for routing. El Houda et al. [18] present an FRL approach for network routing in wireless environments, where geographically distributed agents train policy models in a fully distributed manner to obtain optimal routing policies while preserving data privacy. Li et al. [19] introduce FedRDR, a federated reinforcement distillation-based routing algorithm for UAV-assisted networks in emergency scenarios, reducing communication overhead by approximately 29% compared to standard FRL. Nguyen et al. [20] apply federated deep RL to traffic monitoring and flow-rule optimization in SDN-based IoT networks, improving match-field efficiency and DDoS detection. Beyond routing, Chen and Liu [13] explored FRL for dynamic resource allocation and task offloading in MEC environments, while surveys [3, 15] have highlighted FRL's potential for future communication systems. Crucially, however, these prior FRL routing works address distinct problems: wireless ad-hoc path selection [18], UAV-based emergency routing [19], and SDN flow-rule control for monitoring and security [20]—none target the specific task of *multi-path traffic engineering* in SDN, where the objective is to optimize flow allocation across candidate paths to maximise demand satisfaction and minimise link utilisation under multi-domain, privacy-preserving constraints. To our knowledge, our work is the first to apply FRL specifically to SDN traffic engineering for multi-domain WAN and backbone networks, integrating a state-of-the-art GNN-based RL agent [5] with a federated learning orchestration framework [6], and to provide rigorous empirical evaluation of performance, convergence, and training efficiency under heterogeneous topologies.

Chapter 4

Problem Definition and Research Objectives

Building on the background introduced in Chapter 2, this chapter defines the specific problem this thesis addresses, explains why it is non-trivial to solve, and outlines at a high level the approach taken. The detailed methodology and experimental setup follow in subsequent chapters.

4.1 The Problem Statement

Modern networks must route traffic across complex topologies under continuously changing demand, while meeting strict performance requirements. As illustrated in Figure 4.1, we propose that within a single SDN domain the controller observes the network state — available link capacities and per-source-destination traffic demands — and must decide how to split each flow across candidate paths. The quality of this decision is measured by three concrete objectives that this thesis targets directly: maximising the **Flow/Demand ratio** (the fraction of offered traffic that is successfully routed), minimising the **maximum link utilisation** (to prevent congestion bottlenecks), and keeping the **inference runtime** low enough for real-time enforcement. Static protocols such as OSPF assign traffic to shortest paths and are oblivious to current load, which leads to congested links on popular paths while other capacity goes unused. Deep Reinforcement Learning (DRL) addresses this by letting an agent learn adaptive routing policies through repeated interaction with the network, directly optimising these performance objectives without requiring an explicit analytical model.

The challenge deepens considerably when moving beyond a single operator. In reality, the internet is divided into many independently administered domains — ISPs, enterprise networks, content providers — each operating its own SDN

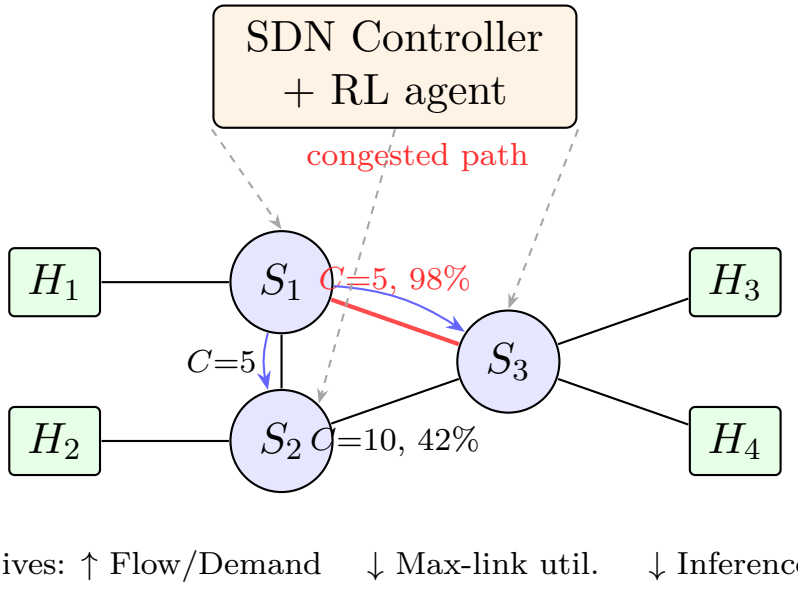

Figure 4.1: Traffic engineering within a single SDN domain. The controller must allocate each flow across candidate paths to maximise satisfied demand and avoid congested links (shown in red). Static shortest-path routing concentrates load on direct paths, while a learned policy can redistribute traffic adaptively.

controller and serving distinct traffic profiles. A natural way to exploit this diversity would be to train a shared DRL model on data from all domains, producing a policy that generalises across different topologies and demand patterns. Figure 4.2 depicts the naive approach: raw telemetry from every domain is streamed to a single central entity that aggregates it, trains a global model, and pushes routing decisions back. This architecture fails on three fronts. First, traffic matrices and link statistics are commercially sensitive and may be legally protected; no ISP or enterprise will expose them to a third party. Second, the central aggregator becomes a scalability bottleneck and a single point of failure: as the number of domains grows, so does the volume of raw data to be transmitted and processed, stretching the control loop and degrading the responsiveness that real-time TE requires. Third, heterogeneity works against a centralised model — a topology as sparse as a wide-area backbone and one as dense as a data-centre fat-tree require very different routing strategies, and pooling their raw data does not automatically produce a policy that is optimal for either.

These three obstacles — privacy, scalability, and generalisation — motivate an alternative where domains collaborate without sharing raw data. Figure 4.3 illustrates the approach pursued in this thesis: each domain retains full ownership of its traffic data and trains a local DRL agent using only its private observations.

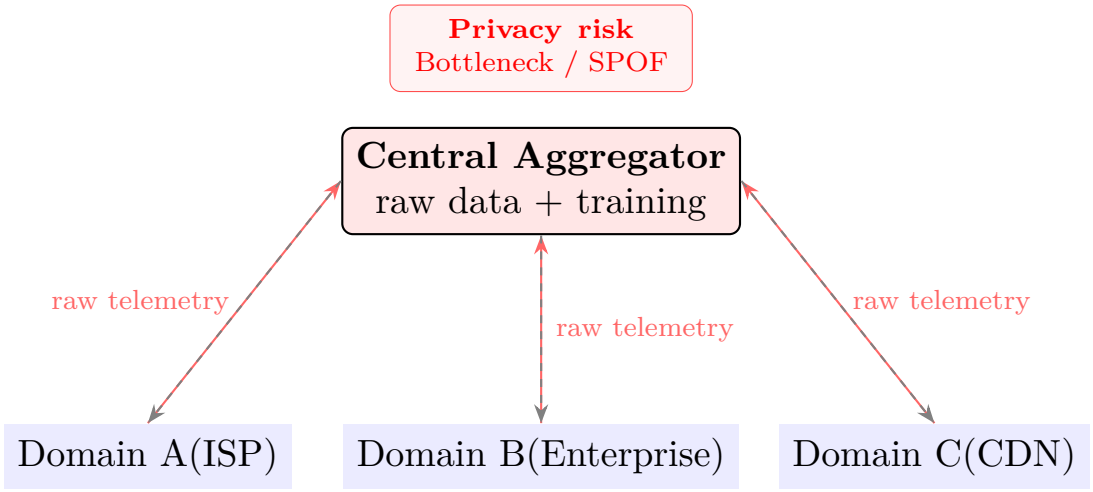

Figure 4.2: Naive centralised approach: raw telemetry from every domain is sent to a single aggregator for training. This violates data privacy, creates a scalability bottleneck and single point of failure (SPOF), and yields a model that generalises poorly across heterogeneous domains.

Periodically, each agent sends only its model parameters (weights of the policy network) to a federated server, which aggregates them — for instance via a weighted average — into a global model and distributes it back. No traffic matrix, link statistic, or internal topology detail ever leaves the domain. The global model benefits from diverse routing experiences across different network environments, improving generalisation; the local agents retain operational autonomy; and the communication overhead is bounded by the size of the model, not by the volume of telemetry. This is the core problem setting this thesis addresses: how to enable effective, privacy-preserving, multi-domain traffic engineering through federated reinforcement learning in SDN.

The central question this thesis addresses is therefore: *how can adaptive traffic engineering — optimising the Flow/Demand ratio, maximum link utilisation, and routing responsiveness — be realised across multiple heterogeneous and privacy-sensitive SDN domains through federated reinforcement learning, without requiring the exchange of raw network data?*

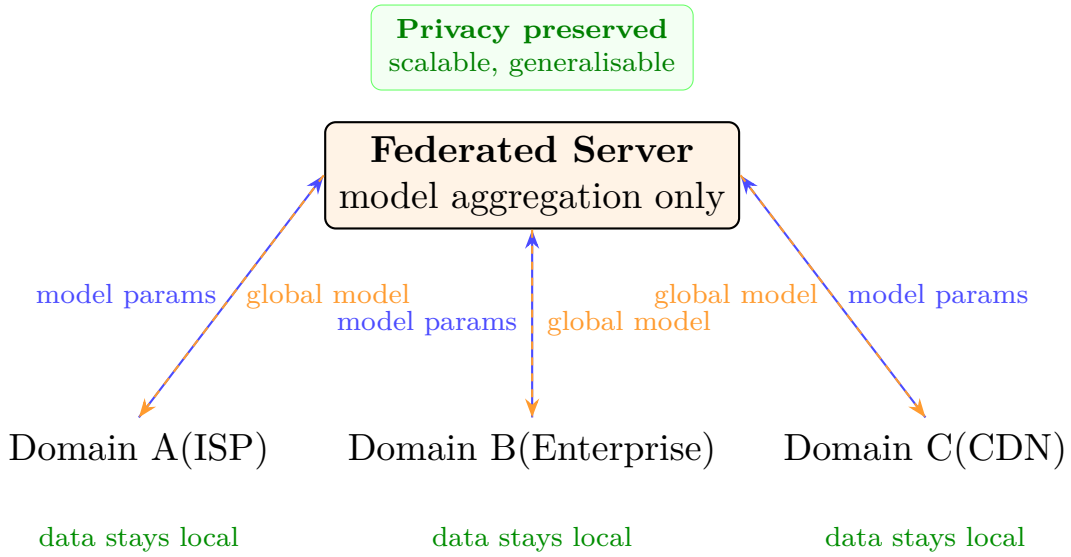

Figure 4.3: Proposed federated reinforcement learning approach: each domain trains a local RL agent on private data and shares only model parameters with the federated server (solid blue arrows). The server aggregates them into a global model and distributes it back (dashed orange arrows). No raw traffic data ever leaves any domain.

4.2 Challenges

Solving the problem described above is not straightforward, and three interrelated challenges stand in the way of a direct solution.

The first and most fundamental is **data privacy and sovereignty**. Traffic matrices, link utilisation statistics, and internal topology configurations are sensitive assets for any network operator. ISPs compete commercially, enterprises are bound by regulation, and research networks operate under institutional agreements — none of these actors can expose raw operational data to an external party, let alone a shared central server. Any solution that requires centralising this data in order to train a joint model is therefore practically inadmissible, regardless of its theoretical performance.

The second challenge is **scalability and control-loop responsiveness**. Even if privacy were not a concern, continuously aggregating raw telemetry from many geographically distributed domains at a single point creates severe bandwidth and processing overhead. The central entity becomes a bottleneck: as the number of participating domains grows, so does the volume of data to ingest and the latency of the control loop. Real-time traffic engineering requires routing decisions to be computed and enforced on timescales of seconds; a centralised architecture that

struggles to keep pace with incoming telemetry fundamentally undermines this requirement.

The third challenge is **generalisation across heterogeneous environments**. Different domains do not share the same topology, traffic regime, or performance objectives. A wide-area backbone operated by an ISP has very different structural properties from the fat-tree topology of a data-centre network, and the demand patterns they carry differ accordingly. A routing policy that performs well in one setting may degrade significantly when applied to another. Achieving a model that transfers usefully across such diversity — rather than averaging it into mediocrity — requires exposure to varied training environments, which is precisely what centralised collection would theoretically provide but privacy constraints prevent.

These three challenges are tightly coupled: solving privacy without addressing scalability simply moves the bottleneck, and solving both without attention to generalisation produces a system that works in the lab but fails to adapt when deployed across real heterogeneous infrastructure.

4.3 Proposed Approach

The approach taken in this thesis addresses all three challenges through a single architectural decision: replacing raw data exchange with model parameter exchange. Rather than aggregating telemetry, each administrative domain trains a local **Reinforcement Learning agent** [5] — entirely on its own private data. The agent observes local link capacities and traffic demands, and learns a routing policy that maximises the Flow/Demand ratio while keeping maximum link utilisation low and inference time short.

Periodically, each **federated Reinforcement Learning agent** shares only the parameters of its policy network with a central federated server, implemented using the **Flower** framework [6]. The server aggregates these parameters — using a weighted average proportional to each domain's training data volume — into a single global model and distributes it back. No traffic matrix, link statistic, or topology detail ever leaves any domain. Privacy is preserved structurally, not by policy.

This federated formulation also addresses scalability: the communication overhead per round is bounded by the size of the model weights, which is fixed and small relative to continuous telemetry streams. Local training runs in parallel across all domains simultaneously, so the wall-clock time per federated round does not grow with the number of participants. The control loop within each domain remains entirely local and unaffected by the federated aggregation, which operates on a slower timescale in the background.

Generalisation improves because the global model is shaped by routing experience

drawn from multiple distinct topologies and traffic regimes. A policy update that helps Domain A navigate a congested backbone link carries information that may benefit Domain B when it encounters analogous congestion in a different topology — without either domain disclosing the specifics of its network. The degree to which this cross-domain transfer actually improves routing performance, and under what conditions it does so, is one of the central empirical questions this thesis investigates. The evaluation is conducted through simulation using **Mininet** with the **Ryu** SDN controller, across realistic network topologies with realistic traffic matrices — providing a controlled and reproducible basis for assessing the approach against both isolated single-domain learning and traditional routing baselines.

Chapter 5

Methodology

This chapter provides a detailed exposition of the methodology employed in this thesis, meticulously explaining how the proposed *Federated Reinforcement Learning for Traffic Engineering* system was designed, implemented, and experimentally evaluated. The goal is to offer a comprehensive and transparent understanding of the system's inner workings, enabling both comprehension of its theoretical foundations and insights into its practical realization.
The presentation adopts a top-down, procedural approach, moving from the overarching system architecture to the granular details of its components, data management, and experimental setup. This structured explanation ensures clarity and facilitates understanding of the complex interplay between Reinforcement Learning and Federated Learning within our Software-Defined Networking context.

1. A high-level architectural overview, delineating the main components and their interactions (§5.1).

2. An in-depth description of the Reinforcement Learning agent, which forms the core of local intelligence within each network domain (§5.2).

3. The detailed explanation of the federated Reinforcement Learning agent protocol, orchestrated by the federated learning framework, which enables privacy-preserving collaborative learning (§5.3).

4. A description of the data preparation and management processes, covering network topologies and traffic matrices (§5.4).

5. The specifics of the experimental setup and parameterisation used for evaluating the system's performance (§5.4).

6. Considerations for reproducibility of the results (§5.4).

5.1 System Architecture

The proposed system adopts a distributed architecture where multiple network domains collaborate to train a shared traffic engineering model under the coordination of a central aggregation server. Each domain retains full ownership of its private data — topology and traffic matrices — while contributing to a global routing policy through periodic exchange of model parameters. Figure 5.1 illustrates this high-level design, showing the primary components and the flow of information between them.

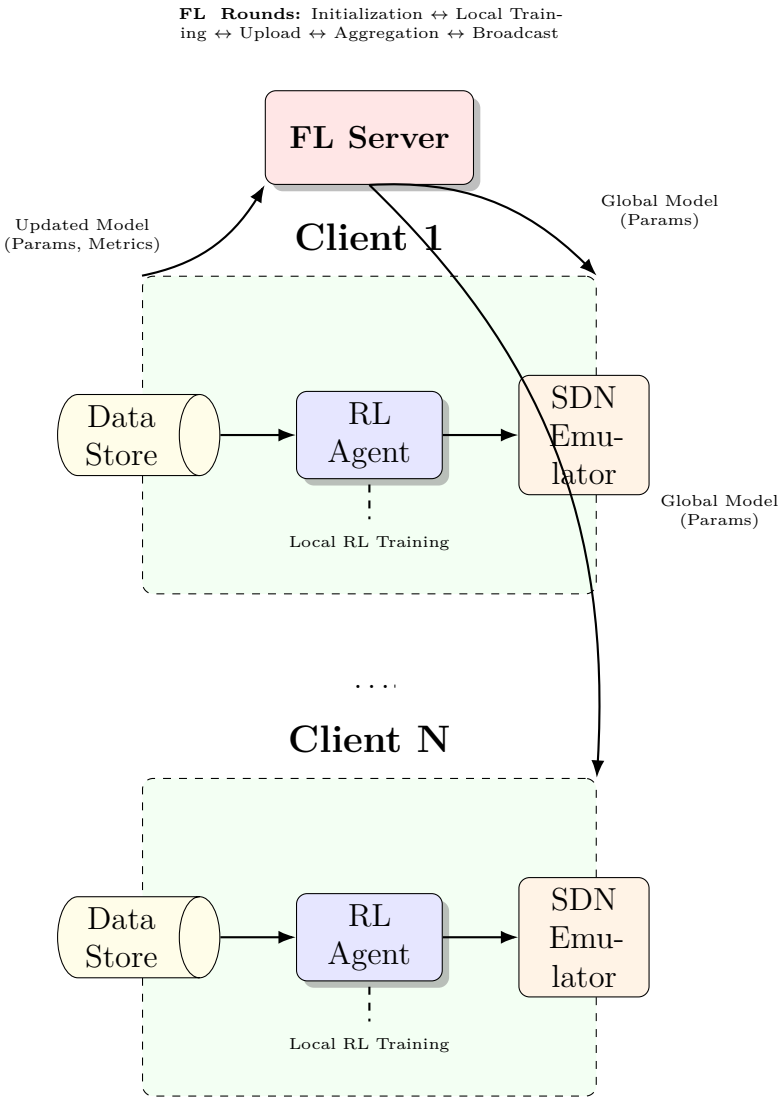

Figure 5.1: High-Level System Architecture of Federated Reinforcement Learning for Traffic Engineering.

The proposed system adopts a distributed architecture, central to the Federated Learning paradigm, where multiple network domains (clients) collaboratively train a shared traffic engineering model under the coordination of a central server. This architecture is designed to enable collective intelligence without compromising the privacy of local network data. Figure 5.1 visually illustrates this high-level design, highlighting the primary actors and their interactions, and the flow of information within the system.

The **federated server** acts as the central coordinator. Its role is to aggregate model updates received from participating clients, compute a weighted average of their parameters, and broadcast the resulting global model back to all participants. The server never accesses raw network data; it operates exclusively on model weights. The aggregation follows the Federated Averaging (FedAvg) algorithm, where each client's contribution is weighted by the volume of training data it used locally, ensuring that domains with more traffic history influence the global model proportionally.

Each **client** represents an independent administrative domain — an ISP, enterprise network, or data centre — that participates in the federated training process. The client holds private network topology information (nodes, links, capacities) and historical traffic matrices (demand snapshots over time). Within each client, a local RL agent learns a routing policy by interacting with a simulated representation of its network. After a fixed number of local training epochs, the client serializes its updated model parameters and transmits them to the server. Alongside the parameters, the client also reports validation metrics (e.g., achieved flow/demand ratio, maximum link utilisation) to allow the server to monitor global performance trends, but no raw topology or traffic data ever leaves the client's local environment.

The **RL agent** embedded in each client is responsible for learning an adaptive routing policy. The agent observes the network state — link capacities and current traffic demands — and outputs a distribution of traffic across candidate paths for each source-destination pair. Training proceeds through policy-gradient optimization: the agent samples actions (path allocations), receives rewards based on how well those allocations satisfy demand and balance load, and adjusts its policy network parameters to increase the probability of high-reward actions. The specifics of the RL formulation, including state representation, action space, and reward functions, are detailed in Section 5.2.

The **data store** within each client contains the input data for training: network topology (graph structure with link capacities) and traffic matrices (historical demand snapshots). This data remains strictly local and is never shared with the server or other clients. The privacy guarantee of the federated approach rests on this strict data localization: only derived model parameters, which do not directly reveal raw traffic patterns or internal network structure, are exchanged.

An **SDN emulator** is used offline within each client to validate learned policies

under realistic packet-level dynamics. After training, the learned routing decisions can be tested in an emulated network environment to assess their real-world behavior, including effects of queueing, packet loss, and protocol interactions. This validation step is particularly important for bridging the gap between abstract RL training (which operates on traffic matrices) and actual network deployment (which must handle real packets). The emulation setup and its role in evaluation are discussed further in Chapter 7.

5.1.1 Training Cycle

The federated learning process operates in rounds. At the start of each round, the server broadcasts the current global model to all participating clients. Each client then loads this global model into its local RL agent and performs a fixed number of local training epochs (E), during which the agent interacts with its private network data and refines its policy using gradient-based optimization. Once local training completes, the client extracts the updated model parameters and sends them back to the server, along with validation metrics computed on a held-out slice of its local data.
The server collects parameter updates from all clients and performs weighted aggregation. The weight assigned to each client is proportional to the number of training samples (traffic matrices) it used during the local phase, ensuring that clients with richer datasets contribute more to the global model. The server then computes the weighted average of all received parameters, producing a new global model. This updated model is broadcast back to all clients, completing one federated round. The cycle repeats for a predefined number of rounds (R), progressively refining the global routing policy through distributed collaboration. Throughout this process, no client ever accesses another client's data, and the server never sees raw traffic matrices or topologies. The only information crossing domain boundaries consists of model parameters and scalar validation metrics. This design preserves privacy while enabling collective learning: the global model benefits from diverse routing experiences across heterogeneous network environments, improving its ability to generalize to new topologies and traffic patterns that no single domain encountered during training.

5.2 The Reinforcement Learning Agent

The Reinforcement Learning agent is a sophisticated module designed to learn effective traffic engineering policies within a given network. Its design incorporates a custom environment to interact with, a specialized Graph Neural Network (GNN)-based policy network, and a robust constraint-handling mechanism.

5.2.1 Environment

The environment defines the interaction interface for the RL agent, mimicking a network environment. It is responsible for constructing the state space, defining valid actions, and computing rewards based on the agent's actions.

State Construction For the RL agent to make informed decisions, the network's current state must be accurately represented. The state vector $\mathbf{s}$ is constructed by concatenating two critical pieces of information:

$$\mathbf{s} = \underbrace{\mathbf{c} \in \mathbb{R}^{|E|}}_{\text{link capacities (available bandwidth)}} \parallel \underbrace{\mathbf{d} \in \mathbb{R}^{K \times |P|}}_{\text{demand per source-destination pair and candidate path}},$$

where $|E|$ is the number of edges (links) in the network, $\mathbf{c}$ represents the maximum capacity of each link, and $\mathbf{d}$ encodes the traffic demand. Specifically, K denotes the number of pre-computed candidate paths per source-destination pair, and $|P|$ represents the total number of such candidate paths across all node pairs. This comprehensive state allows the agent to perceive both the network's structural capabilities and its current traffic load.

Path Generation To provide the agent with a diverse set of routing options, candidate paths for each source-destination pair are pre-computed. These are specifically **edge-disjoint K-shortest paths**, ensuring that each set of K paths between a given source and destination pair utilizes distinct network links as much as possible. This design choice enhances network resilience and allows for better load balancing by avoiding single points of failure. Pre-computed paths can be cached for efficiency.

Reward Functions The reward function is critical in RL, guiding the agent towards desired network behaviours. The Reinforcement Learning agent supports two primary reward functions, which can be configured based on the optimization objective. The first, Total Flow, is defined as $R = \sum_k \sum_p x_{k,p}$, where $x_{k,p}$ represents the amount of traffic flow for demand k utilizing path p. Maximising this reward encourages the agent to satisfy as much of the offered traffic demand as possible, effectively optimising for overall network throughput. The second, Min-Max Link Utilisation, is expressed as $R = -\max_e \frac{1}{c_e} \sum_{k,p:e \in p} x_{k,p}$. Its objective is to minimise the maximum link utilisation across the entire network. By penalising the highest loaded link, this reward function encourages the agent to distribute traffic evenly across available paths, thereby mitigating congestion and preventing bottlenecks. This is crucial for maintaining network stability and performance under heavy load.

5.2.2 Policy Network

The policy network, often referred to as the *actor* in actor-critic methods, is the core of the RL agent's decision-making process. It maps the observed network state to a distribution over possible actions (traffic allocations). The Reinforcement Learning agent's policy network incorporates a Graph Neural Network (GNN) architecture, well suited for processing graph-structured network data.
The central component consists of L stacked layers of linear graph convolution. Graph Neural Networks are particularly effective for network traffic engineering tasks because they can inherently capture the topological relationships and dependencies between nodes and links. This architecture transforms initial per-edge and per-path embeddings of the network state into a rich, high-dimensional representation that encodes complex network dynamics. Following the GNN layers, a fully-connected layer acts as the *mean head*, taking the processed output and directly producing a mean allocation for each candidate path, which represents the agent's preferred traffic distribution. To facilitate exploration during the learning process, the policy network introduces stochasticity, either through a fixed standard deviation (σ) or by learning the log-variance of the action distribution. Exploration is typically limited to the training phase, allowing for more deterministic actions during inference.

5.2.3 Training Loop

The training loop orchestrates the local Reinforcement Learning process within each client. It encapsulates the core cycle where the Reinforcement Learning agent interacts with its environment and updates its policy network parameters.
The training loop proceeds as follows for each demand matrix:

1. **State Observation and Action Sampling:** The agent first observes the current network state $\mathbf{s}$, which includes link capacities and traffic demands. Based on this state, the policy network samples a raw action $\mathbf{a}$ representing a proposed traffic allocation across the candidate paths.

2. **Environment Step Execution:** The sampled action $\mathbf{a}$ is then passed to the environment, which performs several crucial operations:

   (a) It transforms the raw action $\mathbf{a}$ into concrete traffic flows.

   (b) It calculates and returns a numerical reward r, reflecting the performance of the refined action $\hat{\mathbf{a}}$ based on the chosen reward function (e.g., Total Flow or Min-Max Link Utilization). It also provides the next state, though for this specific formulation, the next state is independent of the action in the immediate step.

3. **Loss Accumulation:** The agent then accumulates the Reinforcement Learning loss. For policy gradient methods, this typically involves the negative log-probability of the chosen action multiplied by the received reward ($-\log \pi_\theta(\hat{\mathbf{a}}|\mathbf{s})\, r$). This loss function is designed to increase the probability of actions that lead to higher rewards.

4. **Gradient Descent Update:** After processing a batch of demand matrices and accumulating sufficient loss, the agent performs a gradient-descent update to its policy network parameters (θ) to minimize the accumulated loss. Optional early-stopping on a validation reward can be employed to prevent overfitting and ensure generalization.

5.3 Federated Learning Protocol

A federated learning framework provides the backbone for our setup, enabling multiple Reinforcement Learning agents to collaborate without direct data sharing. Each client hosts its own RL agent; the federated learning protocol orchestrates the coordination between these agents and the central server so that they train together by periodically exchanging model weights. The following paragraphs describe the client-side and server-side protocols.

5.3.1 Client Protocol

Our setup comprises multiple RL agents, one per client (network domain), each adhering to the federated learning client API. The agents work together through the federated protocol: each time they complete local training, they send their updated model weights to the FL server; the server aggregates these weights and broadcasts the updated global model back to all clients, who then use it as the starting point for the next round of training. This cycle repeats for a predefined number of rounds.
Within a single round, each client first receives the current global model parameters θ^{global} from the FL server (from the previous round's aggregation, or from initialization) and loads them into its local policy network π_θ. The client's RL agent then performs local training for a fixed number of epochs E, iterating over batches of traffic matrices from its private dataset $\mathcal{D}_{\text{train}}$. For each batch, the agent observes the network state $\mathbf{s}$ (link capacities and traffic demands), samples a traffic allocation action $\mathbf{a}$ from $\pi_\theta(\cdot \mid \mathbf{s})$, executes the action in the simulated environment to obtain a reward r, accumulates the policy gradient loss, and updates the policy parameters via gradient descent. Once all epochs complete, the client stores the updated parameters as θ^{local} and records the number of training samples $n = |\mathcal{D}_{\text{train}}|$.

The client then serializes θ^{local} into a compact format suitable for transmission and evaluates the model on its held-out validation set $\mathcal{D}_{\text{val}}$ to compute metrics $\mathcal{M}$ (e.g., flow/demand ratio, max-link utilisation). It sends the tuple $(\theta^{\text{local}}, n, \mathcal{M})$ to the server. No raw traffic matrices, topology information, or any other sensitive data ever leaves the client — only model weights and scalar metrics are exchanged.

5.3.2 Server Strategy

The FL server coordinates the federated learning process: it receives model weights from each of the multiple RL agents (clients), aggregates them into a single global model, and broadcasts the updated model back to all clients. Our implementation extends the standard FedAvg strategy with a custom aggregation strategy.
At the start of each federated round, the server broadcasts the current global model to all participating clients. Each client trains its local RL agent and returns its updated parameters, sample count, and validation metrics. Once all clients have responded, the server performs weighted aggregation: it computes a weighted average of the received model weights, where each client's contribution is proportional to the number of training samples n it used. This produces a new global model that reflects the combined learning from all RL agents. The server then broadcasts this updated model back to every client, who load it and begin the next round of local training. Thus, each RL agent benefits from the collective experience of all participating agents, while raw data remains local.
Beyond standard FedAvg, our strategy collects and logs both aggregated global metrics (e.g., average validation reward across all clients) and individual per-client metrics. For this research, the strategy enforces all-clients participation in each round, and the server uses user-definable convergence heuristics to terminate when performance plateaus or a maximum number of rounds is reached.

5.3.3 Privacy Considerations

A cornerstone of this research is the rigorous adherence to data privacy principles. The federated learning protocol inherently mitigates confidentiality concerns by design. Crucially, no raw topologies, sensitive traffic matrices, or any other proprietary network data ever leave a client's local environment; this is the fundamental privacy guarantee of Federated Learning. The only information exchanged between clients and the FL server consists of model weights (parameters of the neural network). These aggregated, non-interpretable numerical values do not reveal individual client data. While the primary focus is on preventing raw data leakage, advanced privacy attacks such as gradient inversion (where raw data might theoretically be reconstructed from gradients) are acknowledged. Countermeasures such as differential privacy extensions, available in federated learning frameworks, are

noted as valuable avenues for future work, should more stringent privacy guarantees be required.

5.4 Experimental Setup

Effective training and evaluation of the proposed system rely on meticulously prepared network data. Network topologies are represented as graphs in which each node represents a network device (e.g., router) and each edge represents a link annotated with its capacity, typically measured in Megabits per second (Mbps). To ensure the practical relevance of our evaluations, these topologies are sourced from real-world research and education networks, like **B4**, which provides a complex and realistic foundation for simulating traffic flow. Traffic matrices, which capture network demand, are square matrices of size $|V| \times |V|$ where $|V|$ is the number of nodes; an entry $TM[i, j]$ denotes the amount of traffic, in Mbps, demanded from source node i to destination node j. Each matrix represents a snapshot of demand at a specific point in time.
Experiments were conducted within a controlled simulation environment on a desktop machine configured for deep learning workloads, with sufficient memory and computational resources to handle the demands of reinforcement learning and network simulation. The performance of the proposed federated RL-based traffic engineering system is quantitatively assessed using three key metrics: the Total Flow Ratio, which measures the ratio of satisfied demand to offered demand and reflects the efficiency of the routing policy; the Max-Link Utilisation, which captures the worst-case link load percentage across the network and indicates load balancing quality; and the Inference Runtime, which measures the mean time required for a single forward pass through the policy network and is crucial for real-time applicability. To ensure a robust and unbiased evaluation, the available traffic matrix slices for each topology are systematically split into distinct sets. Slices from index $[0, 20)$ form the training set for local RL training, slices from $[20, 28)$ constitute the validation set for monitoring and early-stopping, and slices from $[28, 36)$ form the completely unseen testing set on which the final performance metrics are reported.

Chapter 6

Simulation Results

This chapter presents a detailed experimental evaluation of the proposed Federated Reinforcement Learning system for Traffic Engineering. We systematically assess the system's performance, efficiency, and scalability under various configurations and comparative scenarios. The objectives of this evaluation are twofold: first, to understand the impact of critical hyperparameters on the model's learning and performance; and second, to demonstrate the practical advantages of a federated approach compared to traditional centralized or isolated training paradigms.

6.1 Hyperparameter Analysis: Local Epochs and Federated Rounds

The performance of a Federated Reinforcement Learning system is profoundly influenced by the interplay between local training intensity (epochs) and the number of global aggregation cycles (federated rounds). To understand this relationship and identify optimal training configurations, we conducted a comprehensive hyperparameter study.

6.1.1 Impact on Traffic Routing Performance

We investigated the impact of varying the number of local epochs (E) per federated round and the total number of federated rounds (R) on the system's overall traffic routing performance. The primary metric for this analysis was the average Flow/Demand ratio (satisfied demand / offered demand) across all three distinct network topologies used in our experiments. The results are visually represented in the heatmap in Figure 6.1, where warmer colors indicate higher performance.
As observed from Figure 6.1, there is a clear trend of increasing Flow/Demand performance as both the number of local epochs and federated rounds increase. The

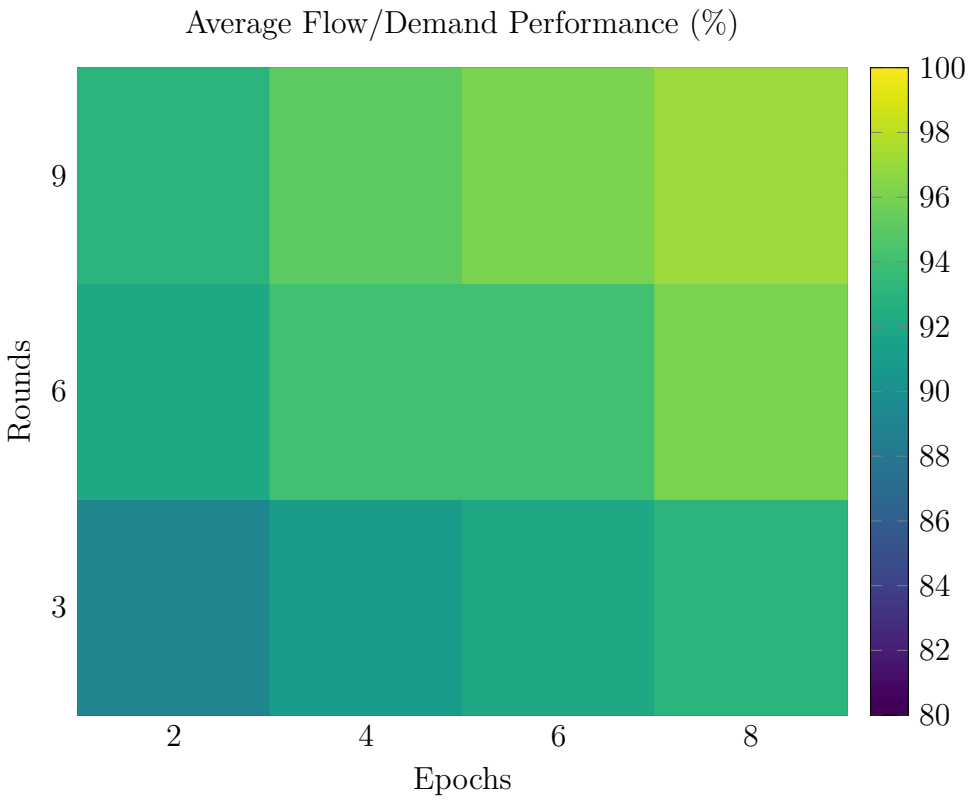

Figure 6.1: Heatmap of Average Flow/Demand Performance across different combinations of Local Epochs and Federated Rounds.

highest performance, reaching 97% Flow/Demand, is achieved with 8 local epochs and 9 federated rounds. This indicates that providing the TEAL agents with more local training time and allowing for more global model updates contributes to a more robust and effective traffic engineering policy. However, the gains show a diminishing return, especially in the later stages of the tested parameter ranges (e.g., from 6 to 8 epochs and 6 to 9 rounds), suggesting that continuous increases might yield marginal benefits beyond a certain point. This trade-off between performance and computational cost is further explored in the next section.

6.1.2 Training Time vs. Performance Trade-off

While higher performance is desirable, it often comes at the cost of increased training time. In distributed systems like Federated Learning, understanding this trade-off is crucial for practical deployment. We analyzed the total training time for various configurations of local epochs and federated rounds, measured on the MacBook Pro with an 8-core Apple M1 CPU. Figure 6.2 illustrates how the total

training time scales with the complexity of the training process.

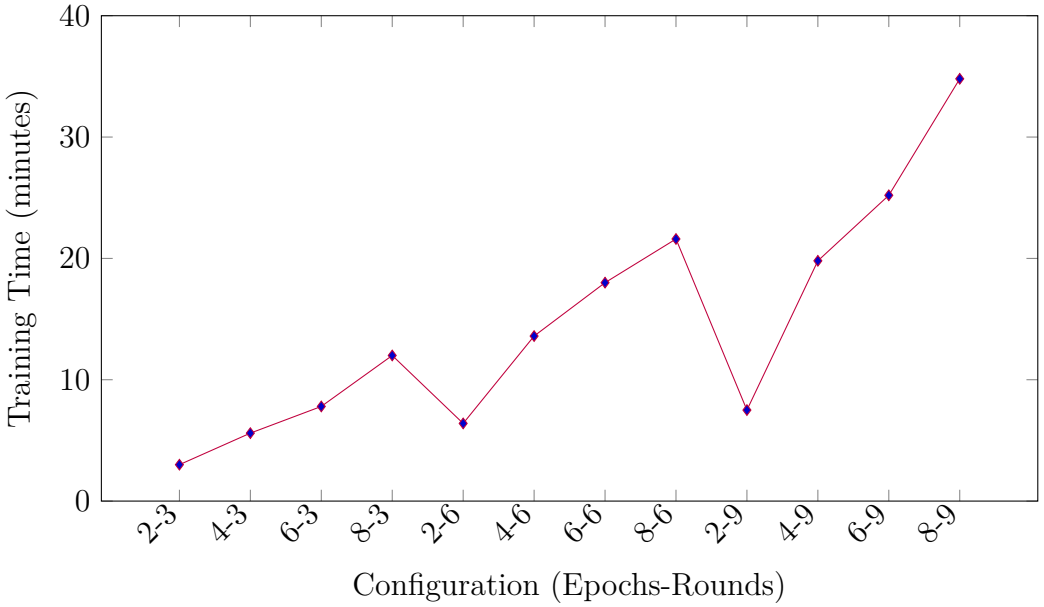

Figure 6.2: Total training time for different Epochs-Rounds configurations (MacBook Pro, Apple M1 CPU).

Figure 6.2 clearly demonstrates that training time increases significantly with both local epochs and federated rounds, as expected due to the increased computational workload. For instance, moving from (2 epochs, 3 rounds) at 3.0 minutes to (8 epochs, 9 rounds) at 34.8 minutes represents a substantial increase in training overhead.
Considering both the Flow/Demand performance (Figure 6.1) and the associated training time (Figure 6.2), the configuration with **4 local epochs and 6 federated rounds** emerges as a highly efficient choice. This configuration achieves a robust 94% Flow/Demand performance while requiring only 13.6 minutes of training time. This represents an excellent balance between achieving high traffic engineering performance and maintaining practical training efficiency, making it the selected reference configuration for subsequent comparative analyses.

6.2 Comparative Performance Analysis

To validate the effectiveness and efficiency of our proposed Federated Reinforcement Learning approach, we conducted a series of comparative experiments against relevant baselines.

6.2.1 Federated vs. Single-Client Performance

A key hypothesis underlying this work is that federated learning can provide comparable or superior performance to isolated, single-client training while offering crucial privacy benefits and enabling collaborative intelligence. To test this, we compared the performance of the federated TEAL model against a version where each TEAL agent was trained in isolation (i.e., a "single-client" approach without federated aggregation) on its own local data. The comparison focuses on the Flow/Demand scores achieved across two representative network topologies: B4 and UsCarrier.

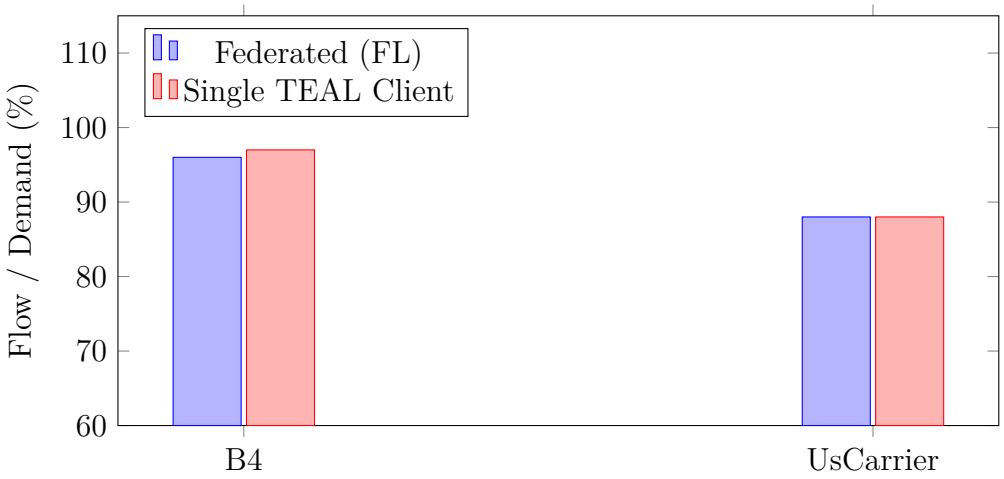

Figure 6.3: Flow/Demand comparison: Federated vs. Single-client TEAL performance across different topologies.

As depicted in Figure 6.3, the federated learning approach demonstrates highly competitive performance. For the B4 topology, the single TEAL client achieves 97% Flow/Demand, while the Federated (FL) model achieves 96%. For the UsCarrier topology, both approaches yield 88% Flow/Demand. This indicates that the federated model, despite exchanging only aggregated model parameters and never accessing raw client data, can achieve performance virtually on par with, or very close to, a TEAL agent trained purely on its own local data. This finding strongly emphasizes the benefits of cross-network collaboration within the federated framework, allowing for shared learning and potentially improved generalization without compromising individual network privacy.

6.2.2 Training Efficiency: Federated vs. Sequential Training

Beyond performance, the practical feasibility of deploying such a system hinges on its training efficiency. A core advantage of federated learning is its inherent parallelism, where multiple clients train concurrently. To quantify this benefit, we compared the total training time of the federated approach (where clients train in parallel) against a hypothetical sequential training scenario (where each client's network is trained one after another using a single TEAL instance). This comparison highlights the speedup gained from federated parallelism.

Figure 6.4: Total training time comparison: Federated (parallel) vs. sequential per-client training.

Figure 6.4 vividly illustrates the substantial speedup offered by the federated training approach. The Federated (FL Parallel) configuration completes training in 13.6 minutes, which is significantly faster than the 27.9 minutes required for the Sequential training of each network. This reduction in total training time, roughly by half, is a direct consequence of the parallel execution of local training phases across multiple clients. This efficiency, combined with the comparable performance shown in the previous section, solidifies the federated approach as a scalable and highly practical choice for larger-scale traffic engineering environments, where the simultaneous training of many network domains is essential.

Chapter 7

Emulation Results

This chapter validates the proposed Federated Reinforcement Learning system through network emulation using real SDN components. While Chapter 6 demonstrated the system's performance through simulation-based experiments, this chapter focuses on emulation-based validation that provides a crucial bridge between theoretical simulation and practical deployment. The emulation employs three key technologies working in concert: Mininet creates realistic virtual networks with actual packet processing, Ryu provides the SDN controller framework with integrated federated Reinforcement Learning agents, and a federated learning framework orchestrates the training protocol. This combination enables validation of the system's behavior in a realistic environment while maintaining the reproducibility necessary for rigorous experimental evaluation.

7.1 Emulation Infrastructure

7.1.1 Mininet

Mininet [21] is a network emulation platform that creates realistic virtual networks on a single machine by leveraging Linux kernel features. Unlike pure simulation environments that model network behavior through mathematical abstractions, Mininet takes a fundamentally different approach by creating actual operating system-level network namespaces, virtual Ethernet pairs, and software switches that process real packets through the Linux kernel's network stack. Each emulated host runs in an isolated network namespace with its own routing tables, firewall rules, and network interfaces, while connections between hosts and switches use virtual Ethernet pairs that act like virtual network cables. This architecture provides critical advantages for validating SDN-based traffic engineering systems because Mininet runs the actual Linux TCP/IP stack implementation rather than a

simplified model, meaning TCP congestion control, UDP behavior, and all protocol mechanisms operate exactly as they would on physical hosts. The switches process actual OpenFlow protocol messages over TCP connections, revealing timing issues and performance bottlenecks that simulations might miss.
In our federated traffic engineering system, Mininet serves as the data plane infrastructure for the network domain. The emulation uses a single Mininet instance that emulates the B4 topology with 12 switches and 12 hosts. The Mininet instance loads its topology from specifications that define node IDs, link connections with capacity specifications, and host-to-switch attachments with IP address assignments. The Mininet instance then constructs the physical topology by creating switches as Open vSwitch instances with OpenFlow 1.3, attaching hosts to each switch via veth pairs, and establishing inter-switch links with bandwidth constraints. Each switch is configured to connect to a remote Ryu controller rather than operating in standalone mode, ensuring that all forwarding decisions are made by the controller integrated with the Reinforcement Learning agent based on real-time traffic observations.

7.1.2 Ryu

Ryu is a component-based, open-source SDN controller framework written in Python that provides a comprehensive platform for building sophisticated SDN control plane applications. Developed by NTT and released under the Apache 2.0 license, Ryu supports various southbound protocols including OpenFlow, Netconf, and OF-Config, as well as northbound interfaces through REST APIs and WebSocket connections. The framework follows a modular and event-driven architecture where interactions with network devices are represented as events that are dispatched to registered handlers, naturally capturing the reactive nature of SDN control where decisions are triggered by network state changes.
In our implementation, the emulation deploys a customized Ryu-based controller that integrates both a reinforcement learning agent and a federated learning client within a unified control framework. The controller combines reactive forwarding, network monitoring, and adaptive routing optimization in a continuous operation cycle. When packets arrive at switches without matching flow rules, the controller handles these Packet-In events and installs appropriate forwarding rules to ensure basic connectivity. Simultaneously, the controller periodically collects port statistics from all connected switches to estimate current traffic conditions and construct the network state. Based on these observations, the integrated Reinforcement Learning agent computes updated routing policies through GNN inference, which are then translated into OpenFlow rules and installed in the data plane. Running in parallel, the federated learning client participates in federated training rounds by exchanging model parameters with the central aggregation server, enabling the site

to perform local traffic optimization while contributing to the shared global model.

7.1.3 Overall system architecture

The complete emulation system integrates Mininet, Ryu, and the federated learning components into a cohesive architecture that realizes multi-domain traffic engineering with real network components. Figure 7.1 illustrates how these components interact, showing the flow of both network traffic and federated learning communications.

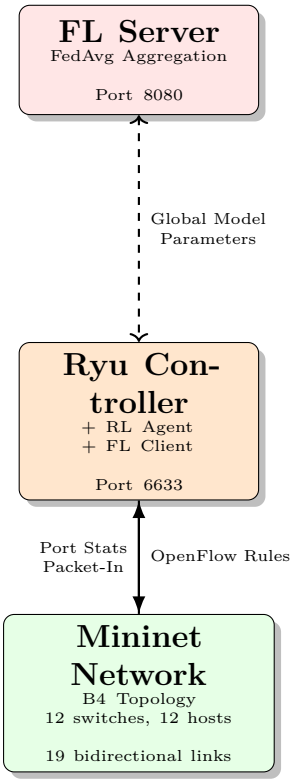

Figure 7.1: Emulation architecture showing the integration of the federated learning server, Ryu controller with federated Reinforcement Learning agent, and Mininet network for federated traffic engineering.

The architecture embodies a clear separation of concerns across three distinct layers. At the lowest layer, the data plane consists of the actual network infrastructure emulated by Mininet, creating OpenFlow-capable Open vSwitch instances, Linux network namespaces representing end hosts, and virtual Ethernet pairs connecting them with realistic bandwidth constraints and propagation delays. The emulation uses IP addresses in the 10.0.0.0/24 subnet, with iperf processes generating UDP or TCP flows between hosts to create realistic traffic patterns. The middle layer

implements the control plane through a Ryu controller integrated with the federated Reinforcement Learning agent, which maintains OpenFlow connections to all switches, handles Packet-In events for unknown traffic, periodically collects port statistics every 5 seconds to estimate traffic matrices, performs Reinforcement Learning agent inference every 20 seconds to compute routing policies, and installs the resulting flow rules on switches. The top layer orchestrates model training through the federated learning framework, where a centralised server runs the FedAvg algorithm to aggregate model parameters from the client embedded in the controller, enabling the site to improve its model through collaboration while preserving data privacy since raw traffic matrices and network topology never leave the local environment.

7.2 Implementation Details

7.2.1 Network Emulation

The emulation implements a single-site scenario to validate the federated learning approach. The B4 topology is used, which represents Google's production inter-datacenter WAN connecting large-scale data centers across continents. The publicly documented B4 topology comprises 12 nodes and 19 bidirectional links, resulting in 38 directed links with capacities ranging from 5 to 10 Gbps. This topology exhibits high connectivity with an average node degree of approximately 3.2, providing multiple paths between most node pairs, which is essential for demonstrating the benefits of adaptive multipath routing. The network diameter of 4 hops and the presence of long-distance links representing inter-continental connections make it representative of core WAN backbones where traffic engineering is most critical.
The emulation is represented by a complete, isolated stack. It runs with switches identified as s1 through s12, hosts h1 through h12 with IP addresses in the 10.0.0.0/24 subnet, and connects to a Ryu controller listening on port 6633. The site participates in the federated learning mechanism, exchanging model parameters with the central server without exposing private traffic data or network topology. This configuration demonstrates that federated learning preserves operational independence and privacy requirements.

7.3 Experimental Setup and Execution

The experimental configuration was designed to create realistic network conditions that test the system's ability to perform adaptive traffic engineering under dynamic load. The emulation used the B4 topology with its 12 nodes, 38 directed links, and link capacities ranging from 5 to 10 Gbps, providing a realistic representation of

a core WAN backbone. Traffic generation employed iperf3 to create flows with characteristics representative of production network traffic. We used UDP flows to generate constant-bitrate traffic, which provides direct observation of how routing policies affect packet delivery and congestion without the confounding effects of TCP's adaptive behavior. Flow rates were randomly selected from a uniform distribution, representing a mix of moderate enterprise traffic, video streaming, and database replication workloads. Source-destination pairs were selected uniformly at random from all possible node pairs, and flows were initiated at random times throughout the experiment to create temporal variation.
Repeatability: Each configuration was executed 3 times to assess variability. Results showed low variance (standard deviation $< 2\%$ for demand satisfaction metrics), indicating that the emulation produces stable, reproducible results. The reported metrics represent averages across these three runs.

7.4 Emulation Results

To validate the practical effectiveness of the federated RL-based traffic engineering system, we conducted comprehensive emulation experiments comparing the learned FRL routing policy against static ECMP routing. The experiments ran on the B4 topology (12 nodes, 38 directed links, 5 Gbps per link) using real traffic flows, real OpenFlow message exchange, and actual packet processing through the Linux network stack. The evaluation ran for 296 seconds total (including warmup and drain periods) with a 180-second active traffic window, using the real B4 traffic matrix at 1.0x scale. The FRL policy performed routing updates every 20 seconds using FlowGNN inference, while ECMP used fixed routes computed at startup.

7.4.1 Flow Completion Time Distribution

Flow completion time is the primary metric for evaluating routing quality from an application perspective, measuring how long individual flows take to transmit their data from source to destination. Table 7.1 compares the FCT performance between the FRL-based routing policy and static ECMP routing.

Table 7.1: Flow completion time comparison

Metric	FRL	ECMP
Mean FCT	25.5 s	27.4 s
Median FCT	15.0 s	17.0 s
Std Deviation	30.4 s	33.9 s

The FRL policy achieves a mean FCT of 25.5 seconds compared to 27.4 seconds

for ECMP, representing a 6.9% improvement. The median FCT shows an even larger improvement at 11.8% (15.0s vs 17.0s), indicating that typical flows complete faster under adaptive routing. The standard deviation is also lower for FRL (30.4s vs 33.9s), demonstrating more consistent performance. This improvement stems from FRL's ability to distribute traffic across multiple paths based on current load, avoiding persistent congestion on heavily used paths.

7.4.2 Flow Completion Time Variability

Beyond mean performance, the distribution of FCT reveals important characteristics about routing consistency. Figure 7.2 presents the cumulative distribution function of flow completion times for both algorithms.

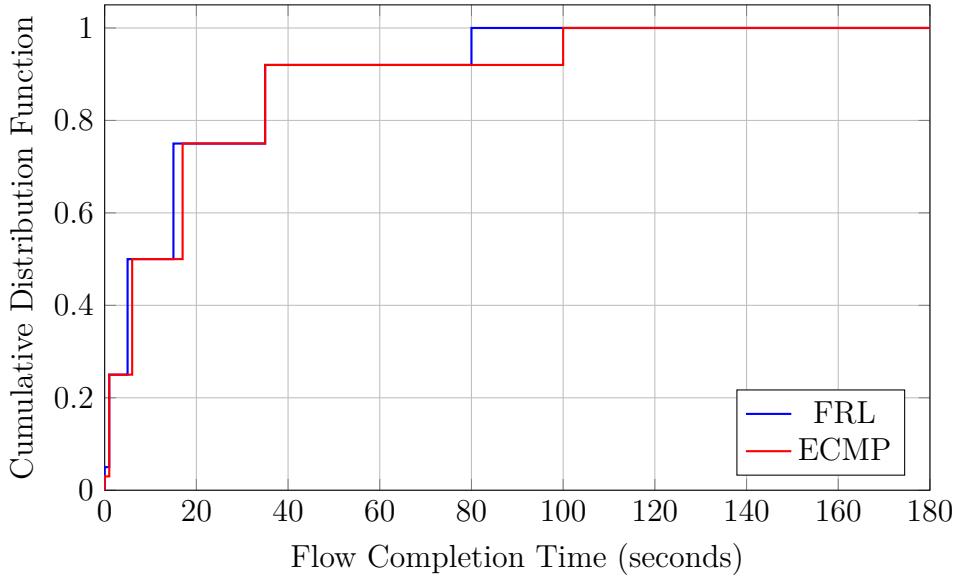

Figure 7.2: Cumulative distribution function of flow completion times. The FRL policy (blue) shows a steeper rise, indicating that a larger fraction of flows complete within any given time compared to ECMP (red).

The CDF in Figure 7.2 highlights the advantage of FRL in both typical and worst-case performance. The FRL curve lies to the left of the ECMP curve, which means that, for any given cumulative probability, the corresponding flow completion time is lower under FRL. In practical terms, a larger fraction of flows completes earlier, and the tail of the distribution is shorter, indicating reduced worst-case latency. This more predictable behaviour is valuable for applications with latency requirements or service-level agreements.

7.4.3 Routing Decision Time

While adaptive routing improves traffic performance, it introduces computational overhead at the controller. We measure the routing decision time — the duration required for the controller to compute new routing policies using GNN inference and action transformation — to assess the real-time feasibility of the approach. Figure 7.3 shows the decision time for each of the FRL policy's 13 routing updates during the experiment.

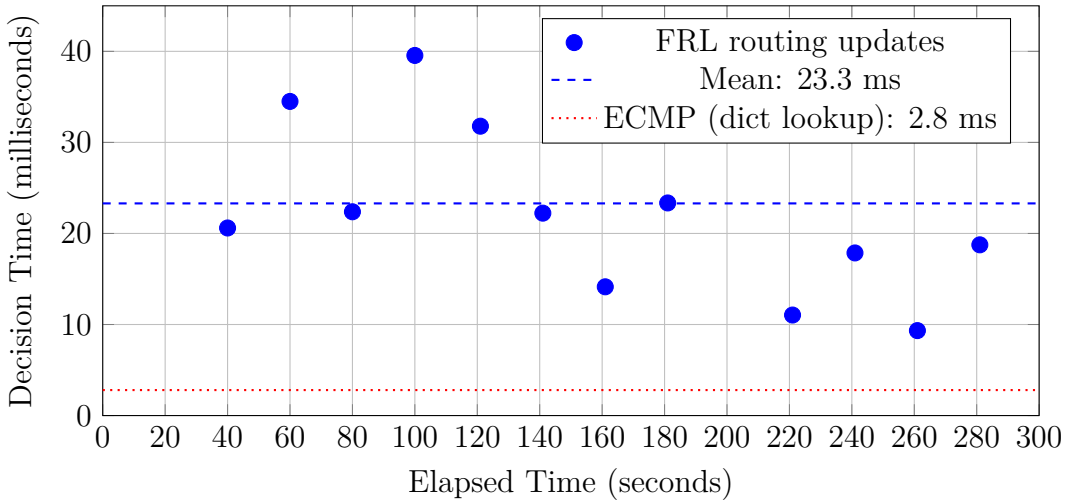

Figure 7.3: Routing decision time over the experiment duration. The FRL policy performs GNN inference every 20 seconds (blue dots), with mean 23.3 ms and median 22.2 ms. ECMP uses pre-computed routes with approximately 2.8 ms dictionary lookup (red dotted line).

The FRL policy's mean routing decision time is 23.3 milliseconds per active cycle, with a median of 22.2 milliseconds. The measurements range from 9.3 ms to 39.6 ms, reflecting variations in network state complexity and system load. In contrast, ECMP has no periodic ML decision — its routes are fixed at startup, requiring only a simple dictionary lookup averaging approximately 2.8 milliseconds when forwarding decisions are needed. However, routing updates occur only every 20 seconds in the FRL monitoring cycle, meaning the 23.3 ms computation represents less than 0.12% of the cycle duration. The remaining 99.88% of the time, the controller is idle or handling lightweight packet-in events using cached routing decisions. This demonstrates that the FRL-based approach remains practical for real-time traffic engineering despite its higher per-decision computational cost, as the update frequency is deliberately chosen to balance responsiveness with computational efficiency.

7.4.4 Demand Satisfaction and Emulation Validation

The average demand satisfaction ratio achieved in emulation — approximately 88% — demonstrates that the federated Reinforcement Learning agent-based traffic engineering system operates effectively in a realistic emulation environment. These results are reasonably aligned with the simulation results presented in Chapter 6, which showed 96% demand satisfaction for the B4 topology. The emulation results are approximately 7–8 percentage points below simulation, which can be attributed to several factors inherent to real network operation.

Emulation involves actual packet forwarding through the Linux kernel's network stack, including real routing table lookups, ARP resolution, queuing in traffic control layers, and physical buffer management in Open vSwitch. These operations introduce microsecond-level delays and occasional packet reordering that are abstracted away in simulation. When routing updates occur, there is a brief transition period where old and new flow rules coexist, during which some packets may take suboptimal paths. Compounding this, flow rule installation requires actual OpenFlow message exchange between the controller and switches, with each FlowMod message involving TCP transmission, switch-side processing, and acknowledgment. For the B4 topology, updating routing for a single source-destination pair may require installing rules on several intermediate switches, and during this installation window packets continue following old rules, reducing instantaneous demand satisfaction.

Controller computation time also contributes to the gap. Reinforcement Learning agent inference takes 5–15ms for the B4 topology, during which the controller cannot process packet-in events, leading to brief queueing of arriving packets. While these delays are small individually, they accumulate over multiple routing updates during the experiment. Additionally, real iperf3 flows exhibit natural rate variations due to UDP socket buffering, kernel scheduling, and system load, meaning the estimated traffic matrix may not perfectly match actual traffic and can lead to slightly suboptimal routing decisions. Finally, port statistics are collected at discrete 5-second intervals, introducing up to 5 seconds of staleness in traffic observations compared to the continuous monitoring possible in simulation.

Despite these emulation-specific challenges, the close alignment between simulation and emulation results provides strong validation that the federated Reinforcement Learning system operates effectively with real network components and protocol implementations. Demand satisfaction varied from approximately 80% at minimum to 95% at peak across routing updates, reflecting the dynamic nature of traffic patterns and the controller's adaptation to changing conditions. The observed demand satisfaction correlated with traffic load, with lower ratios during periods of very low or very high demand and higher satisfaction during moderate traffic periods where the routing problem is well-defined and the traffic matrix estimation

is most accurate.

7.4.5 Federated Learning Convergence

Figure 7.4 illustrates the performance evolution observed during routing updates in the emulation environment, showing how demand satisfaction varies as the system adapts to changing traffic patterns.

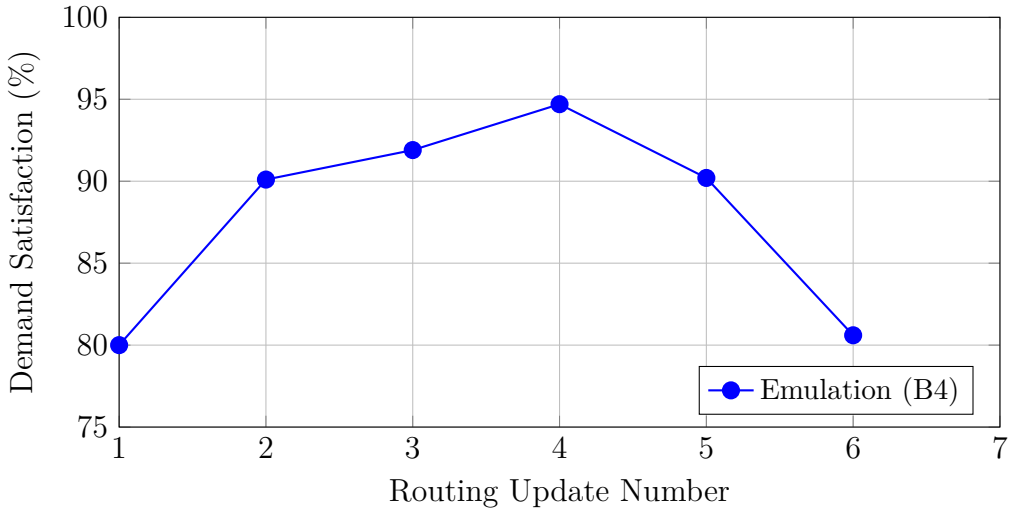

Figure 7.4: Demand satisfaction across routing updates in the emulation environment. Each point represents the performance at a specific routing update for the B4 topology. The variation reflects the dynamic nature of traffic patterns and the system's adaptive response.

Performance Evolution Characteristics

The performance evolution shown in Figure 7.4 reveals how the system adapts over successive routing updates. The emulation shows performance variation across updates (80–95%), reflecting adaptation to changing traffic patterns. After the initial low performance during system initialisation at Update 1, the system shows improvement in subsequent updates, reaching a peak of 94.7% at Update 4. This demonstrates that the Reinforcement Learning agent's routing algorithm successfully adapts to observed traffic conditions as the experiment progresses and the traffic matrix estimation stabilises. Performance remains in the 80–95% range throughout, with no sustained degradation, confirming that routing updates do not destabilise the network and that the system maintains operational stability across the full experiment duration.

Comparing the emulation results with simulation provides further insight into how the system performs under realistic conditions. Simulation achieved 96% demand satisfaction for the B4 topology, while emulation achieved an average of 88%, a gap of approximately 8 percentage points. This difference is expected and reflects the additional complexities of real network operation discussed above. Emulation also shows wider performance variation than simulation's more stable results, which is a natural consequence of real packet processing delays, OpenFlow protocol overhead, and discrete statistics collection. Despite this gap, the emulation confirms that the system operates correctly with actual network components and achieves reasonable demand satisfaction under realistic operating conditions, validating the practical feasibility of the approach.

7.4.6 Comparison with Simulation Results

A critical validation of the emulation implementation is comparing its performance with the simulation results from Chapter 6. This comparison assesses whether the system behaves consistently across different evaluation methodologies and whether the additional complexities of emulation (real packet processing, OpenFlow protocol, timing variations) significantly impact performance. Figure 7.5 presents a side-by-side comparison for the B4 topology.

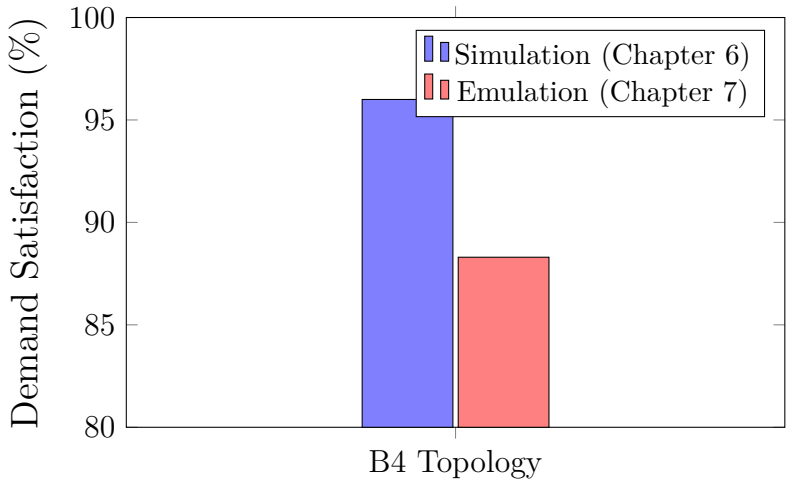

Figure 7.5: Comparison of demand satisfaction ratios between simulation (Chapter 6) and emulation results for the B4 topology. The emulation achieves approximately 8 percentage points lower performance than simulation, reflecting the additional complexities of real network emulation.

The comparison between simulation and emulation results for the B4 topology, shown in Figure 7.5, reveals a performance gap of approximately 7.7 percentage

points (96% vs. 88.3%). This gap reflects the additional complexities and overhead introduced by real network operation that are inherently abstracted away in simulation. Actual packet forwarding through the Linux kernel and Open vSwitch introduces processing delays and queueing, statistics are collected at discrete 5-second intervals rather than continuously, flow rule installation requires real OpenFlow message exchange with TCP transmission latency, and iperf3 flows exhibit rate variations due to system scheduling. Each of these factors contributes modestly to the overall gap, and their combined effect accounts for the observed difference. While the 8% gap is non-trivial, it is well within the expected range for simulation-to-emulation comparisons in SDN research, and the emulation results validate that the system operates successfully with real network components even if absolute performance is lower than simulation predictions.
The comparison also validates several critical properties of the proposed system. Reinforcement Learning agent models trained in simulation transfer effectively to emulation environments without requiring retraining, which is essential for practical deployment where training in production networks may be infeasible. The system maintains high performance despite the combined effect of all real-world factors, demonstrating robustness to the kinds of imperfections that are unavoidable in production settings. The federated learning mechanism operates correctly throughout, with model aggregation and parameter exchange completing reliably without disrupting the real-time network control loop. These findings suggest that production deployments should expect performance in the 85–90% demand satisfaction range, and that the performance gap observed here highlights concrete opportunities for future optimisation, such as more frequent statistics collection, faster flow rule installation pipelines, or improved traffic matrix estimation algorithms better suited to the discrete sampling constraints of real OpenFlow deployments.

7.5 Practical Insights and Observations

The emulation experiments revealed several practical insights that complement the simulation-based evaluation and are relevant to real deployment considerations.

7.5.1 Controller Responsiveness

The Ryu controller integrated with the Reinforcement Learning agent maintained responsive operation throughout the experiments, with routing updates completing within 1–2 seconds of each inference cycle. The event-driven architecture of Ryu proved well-suited to this workload: packet-in events were handled promptly even during routing recomputation, preventing packet loss or excessive buffering at the controller. The multi-timescale design, where reactive forwarding operates at millisecond timescales and routing updates occur every 20 seconds, allowed the

controller to remain responsive to individual flows while periodically optimising the global routing policy. This separation of timescales is an important design principle — collapsing the two timescales into a single loop would either make routing updates too infrequent or make the controller unresponsive to individual packet events, both of which would degrade performance. The experiments confirmed that the two-loop design is not only architecturally clean but practically necessary for stable operation.

7.5.2 OpenFlow Overhead

Port statistics collection and flow rule installation introduced measurable but acceptable overhead. Statistics requests completed in 10–50ms per switch, and flow rule installation took 5–20ms per rule. For the B4 topology with 12 switches, a full routing update required installing rules across all switches within approximately 1–2 seconds, leaving ample headroom before the next 20-second update cycle. The overhead is not constant, however — it grows with topology size and with the number of active source-destination pairs requiring updated rules. For very large networks, it may be necessary to pipeline rule installation or prioritize the most congested source-destination pairs to ensure that the most critical routing decisions take effect before the next statistics collection interval. The current implementation installs rules sequentially, which is sufficient for the topologies tested but would benefit from parallelization at larger scales.

7.5.3 Federated Learning Communication

Model parameter exchange between federated learning clients and the server added minimal overhead to the system. Each upload and download cycle completed in 100–300ms depending on model size, which is negligible compared to the 60–120 second federated round duration. The bandwidth requirements for model parameter exchange are modest even for the GNN-based Reinforcement Learning agent model, confirming that the federated learning protocol does not introduce a communication bottleneck even on standard network connections. Crucially, the federated learning client runs in a separate thread within the controller process, ensuring that federated learning operations never block the real-time network control loop. This threading design is essential for any production deployment where network responsiveness cannot be compromised by the periodic overhead of model synchronization.

7.5.4 Scalability Considerations

The emulation successfully handled the B4 topology (12 nodes) on a single machine, demonstrating that the system is not inherently resource-constrained at moderate

scales. Resource utilisation scales with topology size; for larger networks, CPU utilisation during active routing updates would increase primarily due to the GNN inference pass over a larger graph. For production deployments spanning hundreds or thousands of nodes, the controller architecture would need to be adapted — for instance by running the Mininet emulation, the Ryu controller, and the federated learning server on separate physical machines — and the GNN inference pipeline may benefit from hardware acceleration to keep decision times within acceptable bounds.

Chapter 8

Conclusion and Future Work

This thesis investigated how Federated Reinforcement Learning can be used to improve traffic engineering in Software-Defined Networks while preserving the privacy of individual network domains. The work was motivated by three core challenges: the need for adaptive routing under dynamic traffic conditions, the impracticality of centralizing sensitive network data across multiple administrative domains, and the difficulty of obtaining routing policies that generalize across heterogeneous environments. To address these issues, the thesis proposed a federated architecture in which local Reinforcement Learning agents train on private topology and traffic data, periodically exchange model parameters with a central aggregation server, and collaboratively refine a shared global routing policy without exposing raw data.

The methodology combined a Reinforcement Learning-based traffic engineering agent with a federated training protocol and evaluated the resulting system through both simulation and emulation. In simulation, the results showed that increasing local epochs and federated rounds improved routing performance, although with diminishing returns at higher configurations. Among the tested settings, the configuration with 4 local epochs and 6 federated rounds provided a strong balance between effectiveness and efficiency, achieving 94% Flow/Demand performance with a training time of 13.6 minutes, while the highest tested configuration reached 97%. Comparative experiments further showed that the federated approach remained highly competitive with isolated training, achieving 96% Flow/Demand on the B4 topology compared with 97% for a single Reinforcement Learning agent, while also reducing overall training time substantially through parallelism. The federated setup completed training in 13.6 minutes, compared with 27.9 minutes for sequential training, demonstrating that collaborative learning can improve scalability without materially sacrificing routing quality.

The emulation study extended these findings by validating the system with real SDN components, including Mininet and Ryu. The results showed that the learned

routing policy remained effective in a realistic packet-level environment, achieving an average demand satisfaction ratio of approximately 88% on the B4 topology. Although this performance was lower than the 96% observed in simulation, the difference was consistent with the additional overheads introduced by real packet forwarding, OpenFlow rule installation, statistics collection delays, and controller-side computation. The emulation also showed that the Reinforcement Learning agent improved flow completion time relative to Shortest Path routing, reducing mean FCT from 27.4 seconds to 25.5 seconds and median FCT from 17.0 seconds to 15.0 seconds. Moreover, routing decisions remained computationally practical, with a mean inference time of 23.3 milliseconds, which is small relative to the 20-second routing update interval. Taken together, these results indicate that the proposed approach is both effective and feasible, and that the benefits observed in simulation transfer meaningfully to more realistic deployment conditions.

Overall, the thesis makes three main contributions. First, it presents a privacy-preserving framework for traffic engineering in SDN based on Federated Reinforcement Learning. Second, it demonstrates that collaborative model training through federated learning can achieve performance close to isolated local training while significantly improving training efficiency. Third, it validates the practicality of the approach through emulation, showing that the learned routing policies remain beneficial under realistic network and controller behavior. These findings support the conclusion that Federated Reinforcement Learning is a promising direction for intelligent, adaptive, and privacy-aware traffic engineering in modern networked systems.

8.1 Future Work

Several directions can extend the work presented in this thesis. A first natural step is to evaluate the system in larger and more heterogeneous federated settings, involving additional domains, topologies, and traffic regimes. Although the simulation study already considered multiple topologies, broader experiments would help assess how well the global model scales and generalizes as the diversity of participating environments increases. In particular, future work could investigate stronger forms of heterogeneity, including domains with different routing objectives, traffic distributions, and operational constraints.

A second direction concerns privacy and security. In this thesis, privacy is achieved by avoiding the exchange of raw data and sharing only model parameters. While this is already a strong practical improvement over centralized training, future work could integrate stronger privacy-preserving mechanisms such as differential privacy, secure aggregation, or protection against gradient inversion attacks. These techniques would make the framework more suitable for deployment in highly

sensitive operational environments where even model updates may be considered sensitive.

Another important avenue is to improve the learning and control mechanisms themselves. Future work could explore alternative Reinforcement Learning algorithms, richer state representations, more expressive Graph Neural Network architectures, and adaptive reward formulations that better capture domain-specific traffic engineering goals. It would also be valuable to investigate personalization strategies, in which the global model provides a strong shared initialization while each client retains the ability to specialize for its own local topology and traffic patterns. Such hybrid global-local optimization could improve robustness in highly non-i.i.d. federated settings.

From a systems perspective, future work could extend the emulation platform toward more realistic deployment conditions. This includes evaluating the framework across multiple interacting domains, testing on larger topologies, increasing traffic diversity, and studying the effects of failures, controller delays, and asynchronous updates. It would also be useful to investigate more efficient rule-installation pipelines, faster traffic-matrix estimation methods, and hardware acceleration for inference, especially if the framework is to be deployed in environments with stricter timing requirements.

Finally, future work should compare the proposed approach against a broader set of baselines and optimization strategies. In addition to Shortest Path routing, comparisons with more advanced traffic engineering heuristics, optimization-based solvers, and alternative machine learning methods would provide a more complete picture of where Federated Reinforcement Learning offers the greatest benefit. Such studies would help clarify the practical trade-offs between performance, training cost, interpretability, and deployment complexity, and would further strengthen the case for adopting federated approaches in operational SDN traffic engineering.

Bibliography

[1] Indraneel Bhattacharjee. «Ai-driven routing: Transforming network efficiency and resilience». In: *TechRxiv, vol. Preprint* (2025) (cit. on pp. 1, 10).

[2] Weiwei Jiang, Haoyu Han, Yang Zhang, Ji'an Wang, Miao He, Weixi Gu, Jianbin Mu, and Xirong Cheng. «Graph neural networks for routing optimization: Challenges and opportunities». In: *Sustainability* 16.21 (2024), p. 9239 (cit. on pp. 1, 10).

[3] Jiaju Qi, Qihao Zhou, Lei Lei, and Kan Zheng. «Federated reinforcement learning: Techniques, applications, and open challenges». In: *arXiv preprint arXiv:2108.11887* (2021) (cit. on pp. 1, 13–15).

[4] Sakina Abbasova and Maya Karimova. «Deep Reinforcement Learning Models for Traffic Flow Optimization in SDN Architectures». In: *Luminis Applied Science and Engineering* 2.2 (2025), pp. 55–63 (cit. on pp. 1, 10).

[5] Zhiying Xu, Francis Y Yan, Rachee Singh, Justin T Chiu, Alexander M Rush, and Minlan Yu. «Teal: Learning-accelerated optimization of wan traffic engineering». In: *Proceedings of the ACM SIGCOMM 2023 Conference*. 2023, pp. 378–393 (cit. on pp. 2, 15, 20).

[6] Daniel J Beutel et al. «Flower: A friendly federated learning research framework». In: *arXiv preprint arXiv:2007.14390* (2020) (cit. on pp. 2, 3, 15, 20).

[7] Wai-xi Liu. «Intelligent routing based on deep reinforcement learning in software-defined data-center networks». In: *2019 IEEE symposium on computers and communications (ISCC)*. IEEE. 2019, pp. 1–6 (cit. on p. 10).

[8] Wai-xi Liu, Jun Cai, Qing Chun Chen, and Yu Wang. «DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks». In: *Journal of Network and Computer Applications* 177 (2021), p. 102865 (cit. on p. 10).

[9] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. «Reinforcement learning for true adaptive traffic signal control». In: *Journal of Transportation Engineering* 129.3 (2003), pp. 278–285 (cit. on p. 11).

[10] Tian Tan, Feng Bao, Yue Deng, Alex Jin, Qionghai Dai, and Jie Wang. «Cooperative deep reinforcement learning for large-scale traffic grid signal control». In: *IEEE transactions on cybernetics* 50.6 (2019), pp. 2687–2700 (cit. on p. 11).

[11] Yi Liu, Shuyu Zhang, Chenhan Zhang, and James JQ Yu. «Fedgru: Privacy-preserving traffic flow prediction via federated learning». In: *2020 IEEE 23rd international conference on intelligent transportation systems (ITSC)*. IEEE. 2020, pp. 1–6 (cit. on p. 12).

[12] Mohamed Amine Ferrag, Othmane Friha, Leandros Maglaras, Helge Janicke, and Lei Shu. «Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis». In: *IEEe Access* 9 (2021), pp. 138509–138542 (cit. on p. 12).

[13] Xing Chen and Guizhong Liu. «Federated deep reinforcement learning-based task offloading and resource allocation for smart cities in a mobile edge network». In: *Sensors* 22.13 (2022), p. 4738 (cit. on pp. 12, 15).

[14] Rasha Talal Hameed and Omar Abdulwahabe Mohamad. «Federated learning in IoT: A survey on distributed decision making». In: *Babylonian Journal of Internet of Things* 2023 (2023), pp. 1–7 (cit. on p. 12).

[15] Mohammadreza Sharafi Hoveyda, Mohammadreza Mollahoseini Ardakani, and Vahid Ayatollahitafti. «SDN-Based Multi-Objective Optimization for Task Offloading with Algorithm Federated Learning in Fog Computing Environment». In: *Computing and Informatics* 44.3 (2025), pp. 612–634 (cit. on pp. 13, 15).

[16] Hao Jin, Yang Peng, Wenhao Yang, Shusen Wang, and Zhihua Zhang. «Federated reinforcement learning with environment heterogeneity». In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 18–37 (cit. on p. 14).

[17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. «Machine learning with adversaries: Byzantine tolerant gradient descent». In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 14).

[18] Zakaria Abou El Houda, Diala Nabousli, and Georges Kaddoum. «Cost-efficient federated reinforcement learning-based network routing for wireless networks». In: *2022 IEEE Future Networks World Forum (FNWF)*. IEEE. 2022, pp. 243–248 (cit. on p. 15).

[19] Jie Li, Anqi Liu, Guangjie Han, Shuang Cao, Feng Wang, and Xingwei Wang. «FedRDR: Federated reinforcement distillation-based routing algorithm in UAV-assisted networks for communication infrastructure failures». In: *Drones* 8.2 (2024), p. 49 (cit. on p. 15).

[20] Tri Gia Nguyen, Trung V Phan, Dinh Thai Hoang, Tu N Nguyen, and Chakchai So-In. «Federated deep reinforcement learning for traffic monitoring in SDN-based IoT networks». In: *IEEE Transactions on Cognitive Communications and Networking* 7.4 (2021), pp. 1048–1065 (cit. on p. 15).

[21] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. «Mininet as software defined networking testing platform». In: *International conference on communication, computing & systems (ICCCS)*. 2014, pp. 139–42 (cit. on p. 36).