REINFORCEMENT LEARNING BASED MULTI-LAYER TRAFFIC ENGINEERING FOR INTELLIGENT WIRELESS NETWORKING: FROM SYSTEM TO ALGORITHMS

by

Pinyarash Pinyoanuntapong

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2023

ABSTRACT

PINYARASH PINYOANUNTAPONG. Reinforcement Learning based Multi-layer Traffic Engineering for Intelligent Wireless Networking: From System to Algorithms. (Under the direction of DR. PU WANG)

The AI-digital era is characterized by an unprecedented surge in data usage, spanning from data centers to IoT devices. This growth has driven the evolution of AI-optimized networks, designed to fuse AI capabilities with advanced network solutions seamlessly. However, these networks grapple with challenges such as the complexity of network layer protocols, discrepancies between simulated AI models and their real-world implementations, and the need for decentralized AI training due to network distribution.

To address these challenges, we introduce the AI-oriented Network Operating System (AINOS). At the core of AINOS are two foundational sub-platforms: the "Network Gym," tailored for AI-driven network training, and "Federated Computing," designed for decentralized training methodologies. AINOS provides a comprehensive toolkit for rapid prototyping, deployment, and validation of AI-optimized networks, bridging the gap from simulation to real-world deployment.

Harnessing the powerful features of AINOS, we prototyped AI-optimized networking solutions using a safe Reinforcement Learning (RL) strategy for Traffic Engineering (TE) at both the link and network layers. At the link layer, we implemented a scalable RL-based traffic splitting mechanism that learns optimal traffic split ratios across Wi-Fi and LTE through guided exploration. For the network layer, we devised an online Multi-agent Reinforcement Learning (MA-RL) approach with domain-specific refinements to determine optimal paths in real-time for wireless multi-hop networks. In our exploration of Network Assisted AI optimization, we reduced Federated Learning training time with our MA-RL multi-routing approach and proposed a robust Decentralized Federated Learning solution that leverages single-hop connections for enhanced network performance. Our results demonstrate the strengths of AI-enhanced networks in proficiently managing heterogeneity and latency.

ACKNOWLEDGEMENTS

I would like to express my profound gratitude to the individuals and institutions that have been instrumental in my journey through the PhD program. Their unwavering support and guidance have been invaluable to me. First and foremost, I extend my heartfelt appreciation to my advisor, Dr. Pu Wang. His expertise in interdisciplinary research and his continuous encouragement have been the pillars of my academic growth. Dr. Wang's unwavering belief in my potential empowered me to enhance my skills and competence as a researcher. I am equally thankful to my doctoral committee members, whose insightful suggestions and feedback played a pivotal role in shaping the trajectory of my research. Their willingness to serve on my committee and their dedication to evaluating my doctoral dissertation are deeply appreciated. I would like to acknowledge the collaborative spirit of my fellow lab members. Their support and camaraderie have been invaluable to me, and I am fortunate to be a part of such a diverse and talented team. I am grateful to the College of Computing and Informatics at UNC Charlotte for their support, particularly through the GASP scholarship, which made my PhD program financially feasible. I would also like to extend my gratitude to Intel and the National Science Foundation for their generous research funding, which significantly contributed to the success of my research endeavors. I appreciate the opportunity to collaborate with Intel researchers, whose expertise and collaboration greatly enriched my work. Last but not least, I want to express my sincere thanks to my loving wife, parents, friends, and family members. Their unwavering patience, care, and support over the past few years have been my source of strength and motivation. This journey would not have been possible without the collective support and encouragement of these individuals and organizations. I am deeply grateful for their contributions to my academic and personal growth.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AI-NOS	AI-oriented Network Operating System
AIoT	Artificial Intelligence of Internet of Things
DNN	Deep Neural Networks
DRL	Deep Reinforcement Learning
E2E	End-to-end delay
FL	Federated Learning
MA-RL	Multi-agent Reinforcement Learning
MATM	Multi-access Traffic Management
MDP	Markov Decision Process
QoS	Quality of Service
RATS	Radio Access Technologies
RL	Reinforcement Learning

CHAPTER 1: INTRODUCTION

1.1 AI Networking

In recent decades, there is been a marked surge in data usage from data center infrastructures, mobile devices, and IoT devices. As communication networks pivot from cloud server-based architectures towards edge devices, they have become foundational to this burgeoning digital landscape. Amidst this shift, network service providers find themselves grappling with the dual responsibility of ensuring reliable connectivity while also maintaining a high quality of service (QoS) for end devices. Factors like throughput, end-to-end delay (E2E), jitter, and packet loss have gained prominence, particularly given the congestion arising from increased data traffic.

Simultaneously, breakthroughs in wireless communication, such as 5G, coupled with advancements in Artificial Intelligence (AI), have given rise to the Artificial Intelligence of Things (AIoT)[2]. This innovative convergence features AI-empowered IoT devices adept at autonomously analyzing data and making proactive, precise decisions. These AIoT applications, including smart surveillance camera networks, intelligent transportation systems, connected healthcare, smart homes, and smart grids, are laying the groundwork for future smart cities. As a consequence, the installation of a vast number of smart edge devices is imperative. However, with the profusion of AIoT devices comes a deluge of data. To manage this explosive growth, AI-driven automated processing, often centralized, becomes essential. Such centralized machine learning models demand that training data be housed at a singular server, necessitating the transfer of immense volumes of IoT device data from the network's edge to this central hub. This dynamic not only stresses our communication networks but also amplifies potential vulnerabilities concerning data privacy.

Traffic engineering (TE), as evidenced by numerous studies [3, 4, 5], stands out as a pivotal method for optimizing network performance. This is achieved through dynamic measurement and analysis of real-time network traffic and crafting optimal routing decisions to accommodate the quality of service (QoS) demands prompted by high traffic volumes. Focusing on key QoS metrics such as end-to-end (E2E) delay and E2E throughput, TE endeavors to fine-tune these parameters. Nevertheless, optimizing these E2E TE metrics poses a formidable challenge in wireless multi-hop networks, owing to substantial uncertainties and variances in traffic flow patterns, wireless link statuses, operational conditions of wireless routers, and network topology.

Concurrently, as the mobile industry propels towards 5G and beyond, it is becoming palpably clear that a singular access technology will be insufficient to cater to the diverse requirements intrinsic to both human and machine communications. Meeting the escalating performance prerequisites for current and future applications necessitates the incorporation of multi-access traffic management at the network edge, reinforcing the essentiality of comprehensive strategies to address the ever-expanding and complex communication demands.

1.2 FL in Wireless Multi-hops

Federated learning [6] is technique to build a machine learning model in a distributed approach. The training happens in decentralized edge devices. All the edge nodes train on there locally available data and local model is shared with the server to aggregate into a global model. This approach will reduce the data privacy and security problem when dealing with sharing the training data across nodes. As the edge devices or workers will share only the model updates to the servers,this enable a cost efficient communications on the links. FL can reduce the communications iterations required by increasing the number of worker nodes to increase the computations which leads to faster convergence times. The global model is continuously updated with help of the local models shared by the worker nodes or edge devices. FL has advantage which multi-hop networks can be benefit from such as reduced channel utilization and computational time, faster model convergence times. Enabling FL over multi-hop networks can not only enhance the over all experience for end clients but also increase the accessibility of AI for diverse applications. However, the majority of the studies on FL infrastructure and implementations focused on conventional infrastructures such as datacenters and network deployments, which have reliable network conditions such as guaranteed network bandwidth and end-to-end delay. The performance and complexity of FL applications over multi-hop wireless networks are still unexplored.

1.3 Problem statements

This demands the efficient and intelligent utilization of limited network resources to optimize network performance. To address the constantly increasing QoS expectations of network entities, such as mobile users, automobiles, unmanned aerial vehicles, and Internet of Things devices, and to build low-latency, ultra-reliable, and energy-efficient networks, Reinforcement Learning (RL) [7] and Deep Reinforcement Learning (DRL) [8], which combines Deep Neural Networks (DNNs) with RL, have been proposed as effective tools to provide Artificial Intelligence (AI)-enabled network solutions for key issues in the future internet such as network acceleration and energy efficiency. It allows a network entity to learn an optimal decision-making policy by interacting with dynamic and uncertain network environments, which is commonly modeled as a Markov Decision Process (MDP) [9].

Recent advancements in Reinforcement Learning (RL) have proved promising technologies to enable the experience-based model-free solution. In particular, it addresses advanced internet challenges such as routing optimization, congestion control, edge computing, and multi access traffic management (MATM). There are several key advantages.

- RL achieves robust and resilient performance in complicated network communication system uncertainties and randomness by requiring neither strong assumptions nor accurate network modeling.

- RL designs to handle non-stationary. Therefore, it automatically adapts to the time-varying network dynamics
- RL can deal with large and sophisticated state/action spaces when combined with the recent advances in linear and non-linear function approximation, which is called Deep Reinforcement Learning (DRL).

Despite these advancements, the deployment of AI networking encounters numerous challenges due to the absence of AI-enabled platform:

- **Protocol Complexity:** The layered protocols in modern networks complicate AI integration, demanding careful navigation through each layer of the protocol stack, each with their specific standards and communication formats.
- **Sim-to-Real Transition:** The transition of AI models from simulations to real-world networks presents significant obstacles, mainly due to the disparities and variabilities between simulated environments and actual operational networks.
- **Decentralized AI Training:** Networks typically operate in a distributed manner, contrasting with the centralized nature of traditional AI training models, thereby necessitating new strategies for implementing AI training that can proficiently operate within a decentralized network infrastructure.

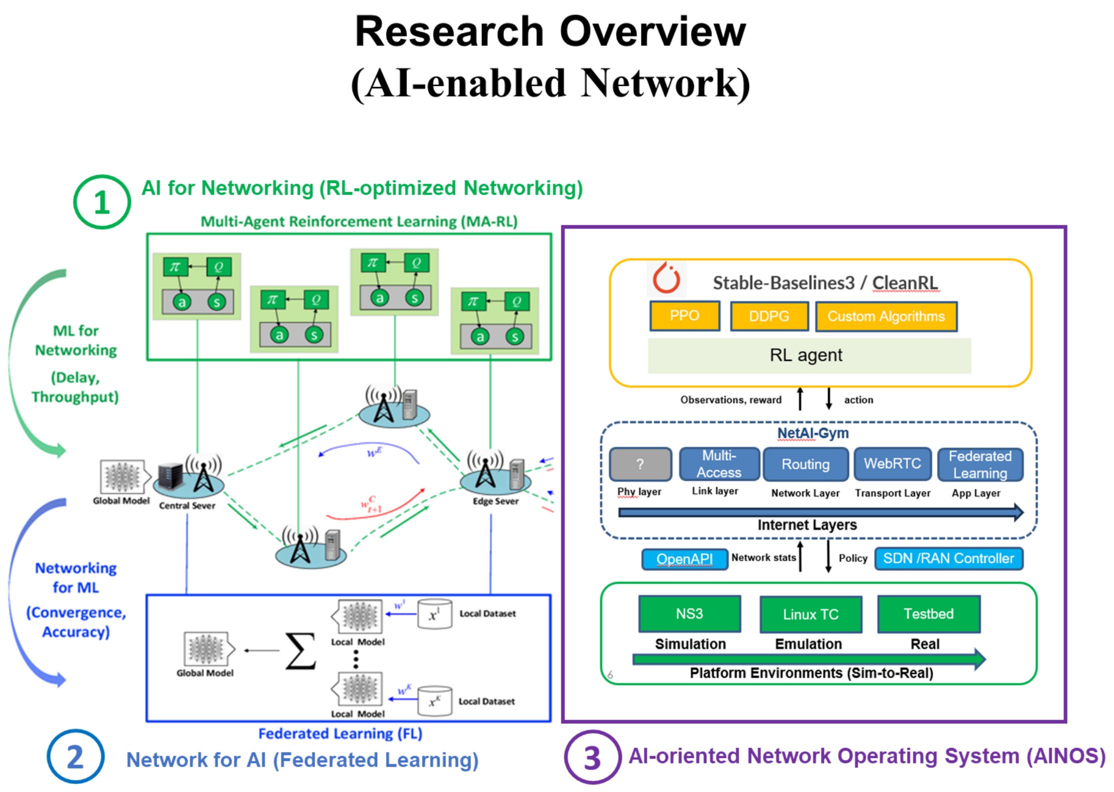


Figure 1.1: Research Overview

In this dissertation, we focus on two research directions : AI for wireless networking and Network-Assisted AI optimization, both fundamentally anchored by our AI-enabled network platform, the AI-oriented Network Operating System (AINOS). Within the core of AINOS lie two foundational sub-platforms: the "Network Gym," which is tailored for AI-driven network training, and "Federated Computing," designed to facilitate decentralized training methodologies. The first research trajectory ventures into AI for wireless networking, employing RL-based multi-layer TE to optimize diverse network performance metrics, including delay and throughput, traversing different network layers, and particularly in the scope of federated learning traffic. Significantly, the "Network Gym" serves as our developmental platform for implementing RL-based algorithms and strategies. Concurrently, the second trajectory involves an exploration of Network-Assisted AI optimization, where our aim is to enhance communication efficiency and design robust model aggregation schemes within federated learning applications. Our strategy leverages the "Federated Computing" component of AINOS, aiming to streamline and optimize decentralized training methodologies. Therefore, the contributions of this dissertation are as follows.

1.5 Contributions

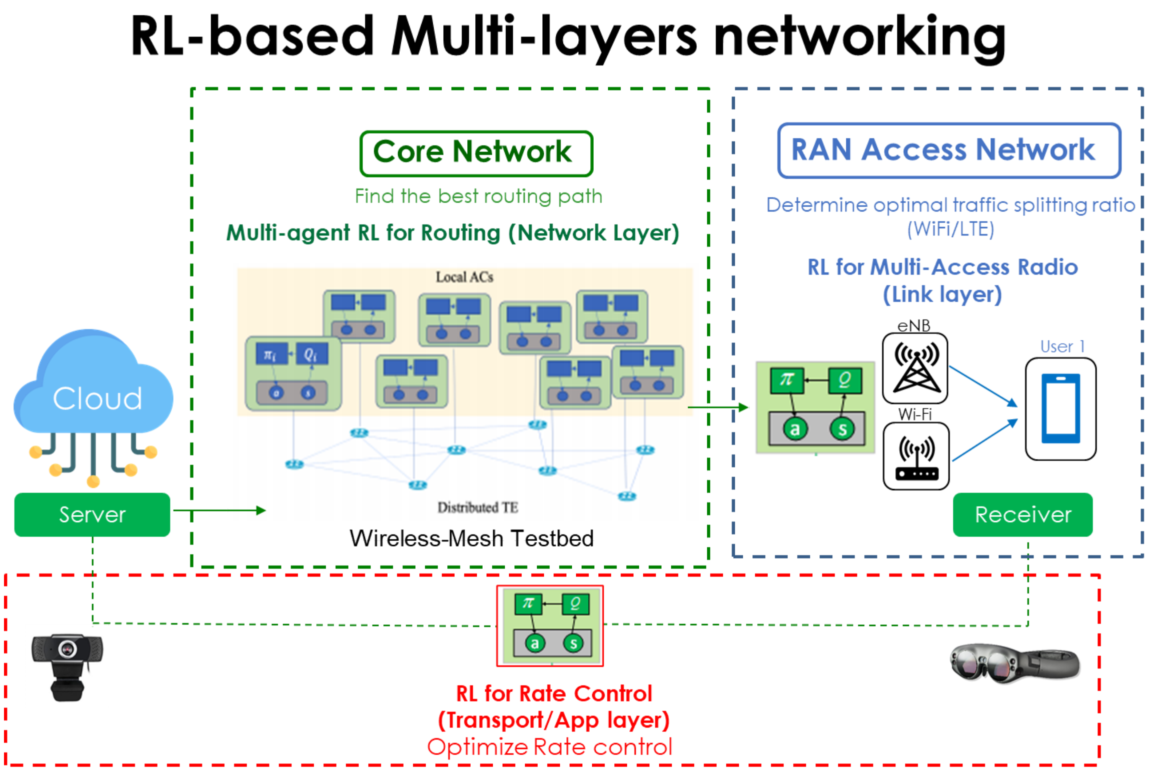

RL-based Multi-layers networking
Core Network
Find the best routing path
Multi-agent RL for Routing (Network Layer)
Local ACs
Distributed TE
Wireless-Mesh Testbed
RAN Access Network
Determine optimal traffic splitting ratio (WiFi/LTE)
RL for Multi-Access Radio (Link layer)
eNB
Wi-Fi
User 1
Receiver
Cloud
Server
RL for Rate Control (Transport/App layer)
Optimize Rate control

Figure 1.2: RL Multi-layers Traffic Engineering

- First, we introduce AINOS, incorporating two innovative sub-platforms: "Network Gym," designed for AI-driven network training, and "Federated Computing," engineered to facilitate decentralized training methodologies. Additionally, we developed a Programmable Wireless Network Operating System (WINOS) and a physical wireless mesh testbed, an AI-enabled platform specifically tailored for wireless multi-hop network physical testbeds.

- We implemented a centralized, online Reinforcement Learning (RL)-based multi-access traffic management system, utilizing guided exploration methods. This approach systematically explores and learns optimal traffic distribution across multiple Radio Access Technologies (RAT) by utilizing NetworkGym, a remote online simulation tool provided by Intel, ensuring effective management of multi-RAT environments.

- We formulated RL-based Traffic Engineering (TE) for wireless multi-hop routing as a Markov Decision Process (MA-MDP) and provided insights into the algorithmic foundations of RL-based TE in this domain [10, 11]. We developed an online multi-agent reinforcement learning (MA-RL) algorithm, complemented with domain-specific action space refining schemes, enabling on-the-fly learning of delay-minimum forwarding paths for wireless multi-hop routing problems, aimed particularly at reducing federated learning training time [12, 13]. We validated these methodologies using WINOS and our physical testbed, substantiating their efficacy and reliability in practical scenarios.

- Empowered by FedEdge [13], we propose a robust and decentralized generic Federated Learning (FL) framework, adept at operating in both synchronous (Sync-DFL) and asynchronous (Async-DFL) modes, thereby augmenting the adaptability of AIoT systems amidst various conditions and device capabilities [14, 15]. In particular, Async-DFL emerges as a pioneering approach, delivering a fully asynchronous FL framework that strategically mitigates worker waiting and represents a significant advancement in AIoT environments with heterogeneous devices and fluctuating computing and networking speeds.

1.6 Dissertation Organization

The remainder of this research is structured as follows: In Chapter 2, we introduce our AINOS system platform. Chapter 3 proposes a Scalable and Safe Deep Reinforcement Learning Design for Multi-Connectivity Traffic Management at the link layer. Subsequently, in Chapter 4, we present the Markov Decision Process (MA-MDP) for Network Routing Traffic Engineering, laying down the algorithmic foundations of the technique. Then, we demonstrate our Multi-Agent Reinforcement Learning (MA-RL) routing approach, aimed at optimizing Federated Learning traffic in a live testbed. Finally, in Chapter 5, we propose a robust and decentralized generic Federated Learning (FL) framework that mitigates stragglers by employing asynchronous and single-hop communication strategies.

CHAPTER 2: AI-ENABLED WIRELESS NETWORK OPERATING SYSTEM: AINOS

2.1 System Overview

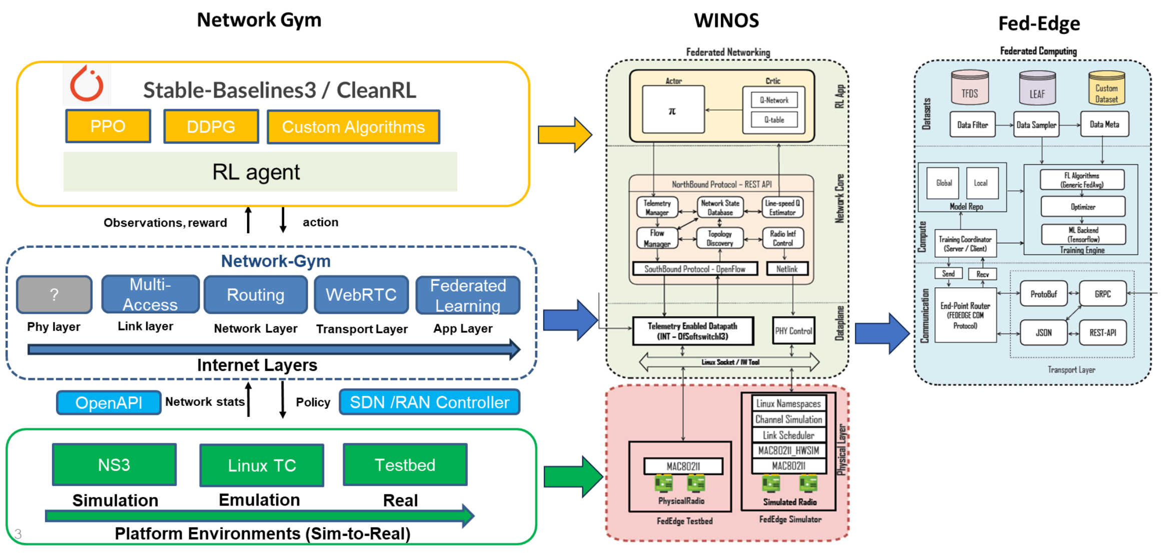


Figure 2.1: Architecture of AINOS

In this chapter, we introduce AINOS, a sophisticated AI wireless experimental platform designed with a system-in-loop framework. This platform is constructed to support a wide array of AI network applications, spanning from traffic management to network routing, and extending to federated learning. AINOS is not only robust but also versatile, being compatible with various environmental platforms including ns3 simulation, Mininet-WiFi emulation, and wireless mesh physical testbeds. The key advances of this platform include the ease of algorithm development, seamless integration with real network protocols, and efficient code and knowledge translation between simulator and testbed. Two critical sub-platforms reside at the core of AINOS: the "Network Gym" and "Federated Computing". The "Network Gym" is developed specifically to provide a fertile environment for AI-driven network training, functioning as our platform for developing and implementing Reinforcement Learning (RL)-based algorithms and strategies. On the other hand, "Federated Computing" is designed to optimize and facilitate decentralized training methodologies, thereby streamlining our approaches in these training solutions.

These platforms work cohesively, offering a formidable environment for the development of AI network applications while showcasing the innovative integration of networking and Artificial Intelligence.

2.2 Network Gym

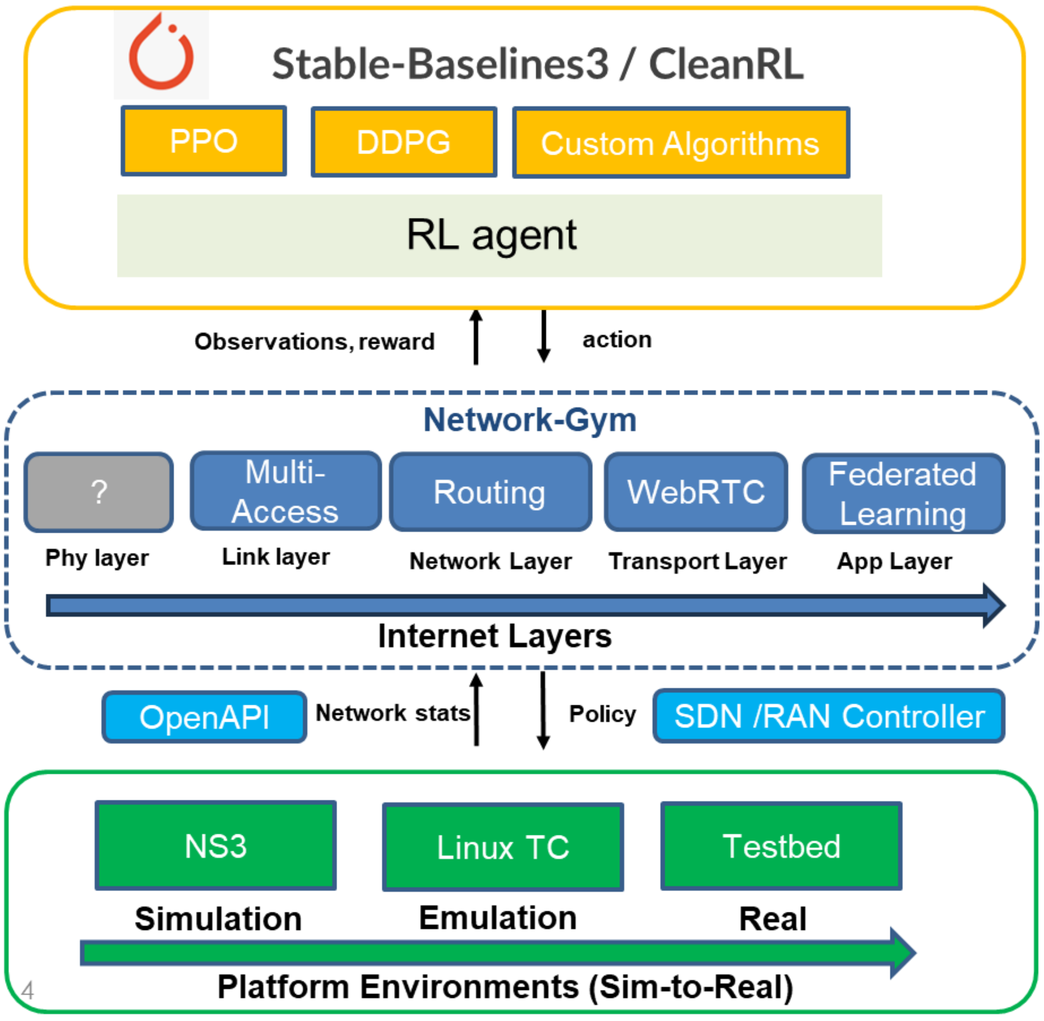


Figure 2.2: NetworkGym

The AI-enabled networking platform offers several significant contributions. The Algorithm plane, using standard and custom libraries, fast-tracks algorithm prototyping, enhancing efficiency. The Control plane, incorporating AI-oriented protocol stack abstraction and APIs, enables seamless integration of AI control mechanisms into existing network protocols [16]. The Data plane, consisting of a simulator, emulator, and testbed controlled by south-bound APIs, minimizes simulator-to-reality drift in AI model deployment, thus ensuring consistent performance. Furthermore, the platform supports distributed and federated model training, a pivotal feature for creating scalable network AI solutions. Its unique design allows for intelligent traffic splitting at the link layer, delay-optimal multiple-path routing at the network layer, and adaptive rate control for real-time video streaming at the transport/application layer.

2.2.1 Algorithm Plane

The Algorithm Plane plays a pivotal role in the platform, focusing on the rapid development and prototyping of algorithms tailored for AI-driven networking solutions. Within this component, network engineers and developers have access to a range of libraries and agents to expedite their algorithm development process [16].

2.2.1.1 Custom Algorithm Agent

For utmost flexibility, NetworkGym allows users to define their specialized agents using the Gymnasiumâs API. Detailed instructions on creating custom agents can be found in the Gymnasiumâs tutorial [16], enabling developers to craft algorithms that suit their specific networking requirements.

2.2.1.2 Stable-Baselines3 Agent

The Stable-Baselines3 Agent stands out by incorporating state-of-the-art (SOTA) Reinforcement Learning (RL) algorithms sourced from the stable-baselines3 framework. These algorithms include renowned ones such as:

- PPO (Proximal Policy Optimization)
- DDPG (Deep Deterministic Policy Gradient)
- SAC (Soft Actor-Critic)
- TD3 (Twin Delayed Deep Deterministic Policy Gradient)
- A2C (Advantage Actor-Critic)

Moreover, these algorithms have been seamlessly integrated to interact with the NetworkGym Environment, allowing network professionals to harness the power of RL for networking tasks [17].

2.2.1.3 CleanRL Agent

To cater to custom algorithm development, the platform offers the CleanRL Agent [18]. This agent serves as a valuable resource for crafting specialized algorithms tailored to unique networking challenges. Network engineers can leverage the CleanRL Agent to create and implement their custom solutions effectively.

Together, these libraries empower network professionals to efficiently design and customize AI algorithms, ultimately enhancing the efficiency of AI solution creation.

2.2.2 Control Plane

The Control Plane is the orchestrator of AI integration into networking protocols, offering a structured approach to managing control mechanisms. This component incorporates:

- **AI-Oriented Protocol Stack Abstraction:** This layer of software abstracts and interfaces with network protocols, allowing for seamless integration of AI control mechanisms. Network protocols remain intact, while AI algorithms optimize network behavior.

- **APIs (Application Programming Interfaces):** APIs serve as the communication bridge between AI applications and the network control plane. They standardize data exchange and instructions, facilitating AI's influence on network operations.

The Control Plane ensures that AI and traditional network protocols coexist harmoniously, simplifying the introduction of AI-based enhancements.

2.2.3 Data Plane

The Data Plane plays a critical role in AI model testing and deployment by offering a suite of tools for network simulation and emulation:

- **Simulator (ns-3):** We employ the ns-3 (Network Simulator 3) platform to simulate network conditions. ns-3 is known for its speed and scalability, making it capable of simulating large-scale network topologies and deployments efficiently. It supports various Radio Access Technologies (RATs) such as 4G, 5G, and Wi-Fi, allowing us to replicate different wireless network technologies and scenarios, including congestion, latency, and packet loss. However, it's important to note that ns-3 does not run real network software and protocol stacks; it simulates their behavior to provide a controlled testing environment for our AI models.

- **Emulator (tc-link and Mininet-WiFi):** For emulating real network devices and their behavior, we rely on tc-link in conjunction with Mininet-WiFi. Mininet-WiFi is a versatile emulator that enables our AI models to interact with virtual network components as if they were operating in a real-world network. Tc-link allows us to manipulate and control network traffic conditions within this emulator, providing fine-grained control for testing under various network scenarios. It bridges the gap between simulated and real-world conditions.

- **Testbed:** A testbed, whether physical or virtual, provides a controlled network infrastructure that can be monitored and manipulated through south-bound APIs. It ensures that AI models are tested in a real-world network environment.

The Data Plane's suite of tools minimizes the disparity between simulated and real-world network behavior, ensuring consistent AI model performance.

In addition to these core components, the platform supports distributed and federated model training, as well as intelligent traffic splitting, delay-optimal multiple-path routing, and adaptive rate control. These features collectively contribute to scalable and efficient AI solutions for network optimization and management.

2.3 FedEdge Design and Prototyping

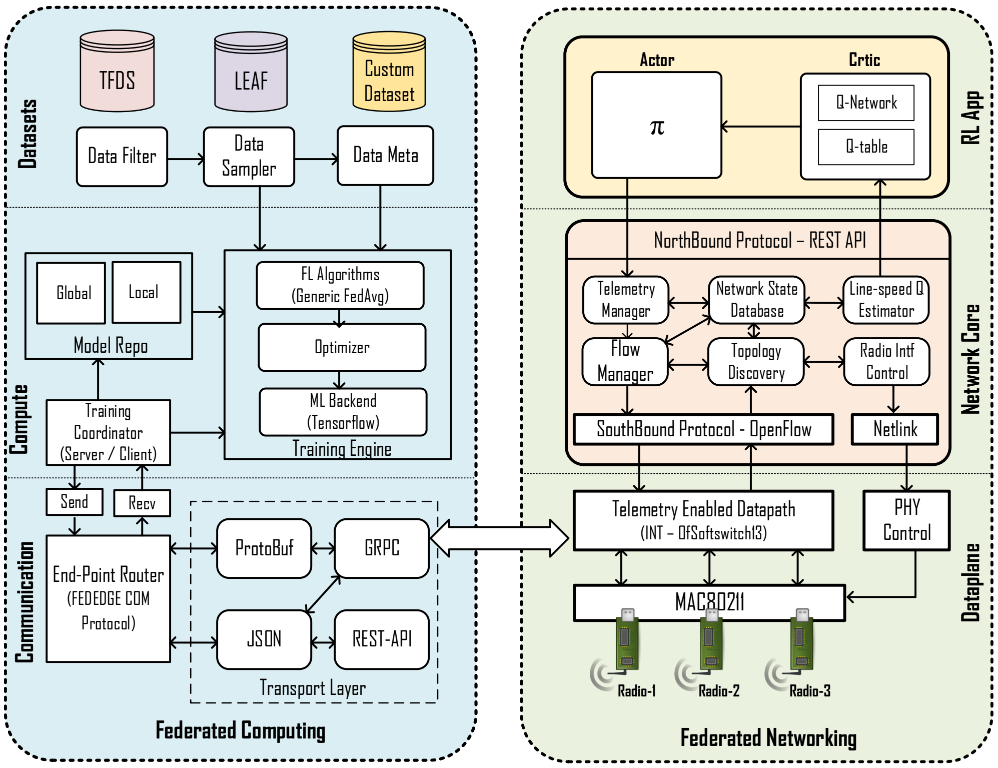


Figure 2.3: Architecture of FedEdge Framework with FedEdge simulator

2.3.1 FedEdge Overall Design

Our objective in this work is to develop a framework that can be used on Commercialoff-the-shelf hardwares and as well as within system-in-loop emulator framework to allow users to seamlessly implement and validate the effectiveness of RL routing solutions. Therefore, (1) We first developed a AI-Enabled Wireless Network Operating System (WINOS), which seamlessly integrates programmable measurement, i.e., the proposed SINT In-band telemetry framework and the programmable wireless network control. (2) We designed and implemented S-INT, a distributed in-band telemetry system, where each router runs its own telemetry module that is built on the top of OpenFlow datapath/processing pipeline.

FedEdge is the first experimental prototype framework developed to support wireless multi-hop federated learning. FedEdge provides modularity with communication functions between the nodes and easy to integrate to a real testbed or a simulation without any modification to the underlying architecture or the code-base. The architecture of FedEdge contains two components namely federated computing and federated networking. Each of the components are built in a layered approach where each layer has bidirectional communications to upper or lower layers. In general federated computing involves in customizing and configuring FL functionality and federated networking is a AI based wireless network operating system which is mainly responsible for fast and reliable wireless network links between the aggregator node and worker nodes.The main goal of this component is to optimize the the wireless network through the AI enabled algorithms to do route optimizations and provide in-band telemetry data used for online training of RL agent.Federated computing contains three layers (1) Datasets layer stores the training datasets for FL (2) Compute layer provides with core functionalities, for instance to train a model and store the trained model and finally (3) Communication layer used FedEdge COMM protocol to establish and maintain connections with aggregator and worker nodes. Similar to federated computing component federated networking component has three layers (1) Dataplane Layer will integrate software defined switch allows programmable packet switching and also allowing the in-band telemetry to cost-efficient real-time reporting of network status. (2) The Network Core Layer will do traditional network functions such as node discovery, maintaining network links counters and status database and managing the traffic flows etc. (3) RL App layer contains the actor-critic RL agent for learning delay-optimized routes in the edge nodes. The integration of simulator to FedEdge architecture is simple because of the modularity of FedEdge, which can use the topology built by FedEdge simulator and train the RL agent on the wireless multihop backbone network.

2.3.2 Federated Networking

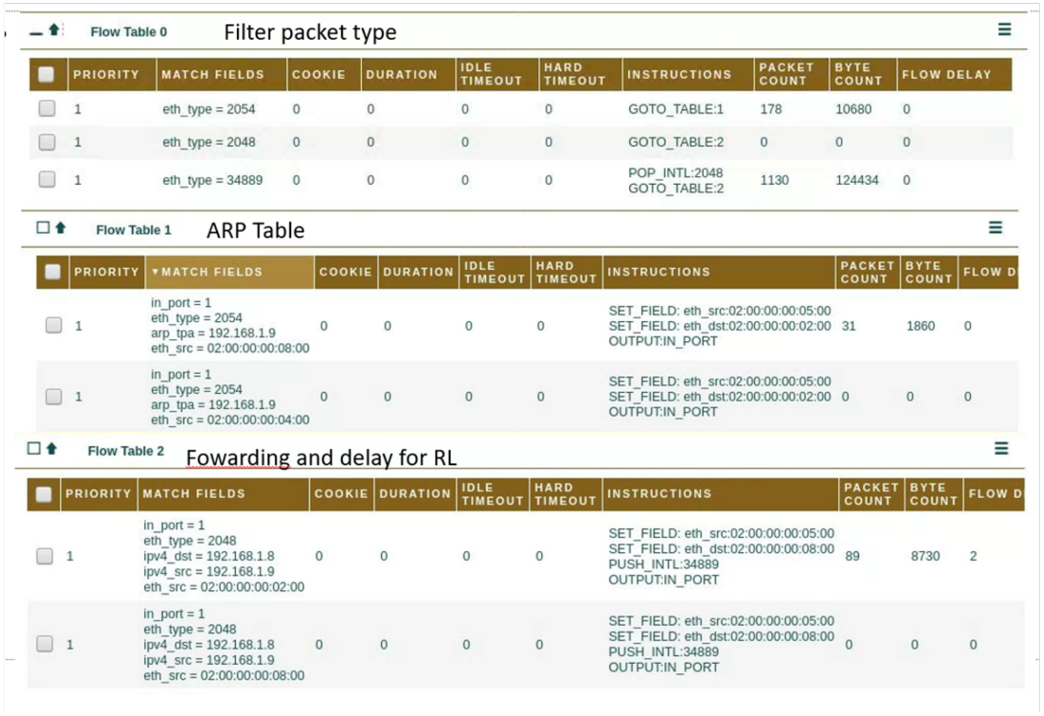
In the previous section, we detailed our design and implementation of federated computing. The key objective of our work is to improve the convergence time of federated learning systems by optimizing communication delay over multi-hop wireless networks. To tame the network latency and to implement reinforcement learning routing module, first we need a platform that enables visibility of per-packet networking statistics (such as delay) for RL training. In addition, we need to realize distributed and programmable network control so that the MA-RL policies can be learned, deployed, and executed in a real-time fashion. To satisfy the above two requirements, we developed a federated networking subsystem by enhancing and customizing WiNOS, a distributed wireless network operating system proposed in our previous work [19]. The federated networking subsystem is composed of three layers (1) Dataplane, (2) Network Core services, and (3) RL Applications. Compared with WiNOS, the new enhancements include the redesigned dataplane and upgraded core services to support multi-radio networking, customized in-band telemetry processing to support federated networking, and the new RL application with domain-specific action space refining. The details of some important components of federated networking system are introduced as below.

2.3.3 Telemetry-enabled Dataplane

The crucial function of dataplane is to forward packets in-line with the native Linux wireless MAC80211 network stack and to provide programming primitives to control packet forwarding. To enable programmable packet forwarding on wireless multi-hop networks, we leveraged OpenFlow-based Datapath to send and receive data packets. Our datapath is developed using OpenFlow Software switch, namely Ofsoftswitch13 [20]. Dataplane functionality is pivotal for realizing AI-enabled forwarding schemes. Nowadays, dataplanes are not only designed to handle packet forwarding, but they also gather vast amount of data for network monitoring using SNMP, sFLOW, Collectd, and many more. However, existing solutions do not support actionable data sampling or measurement schemes that can be rapidly exploited for routing schemes with delay minimization as the objective. In addition, they require additional channel resources and fail to capture real-time delay of packets.

With an AI-enabled platform as the core of our system design, we proposed and developed a real-time in-band network telemetry module. The primary objective of our telemetry solution is to collect real-time experience of packets, such as per-hop delay over each traversing link or end-to-end link, in a cost-efficient manner. Towards this end, we have developed and implemented a distributed in-band telemetry system [19], where each router runs its own telemetry module built on the top of OpenFlow processing pipeline. Our in-band telemetry system consists of a new telemetry packet header, two new packet matching actions (i.e., PUSHINTL and POPINTL) and the telemetry processor. To assist AI-enabled federated networking, we reconfigure the telemetry processor so that whenever a FL packet passes through the router, the router will recognize such packet and insert a timestamp (i.e., the time instance when FL packet arrives at the router) into the telemetry packet header. Then, by using the PUSHINTL action, the router performs packet encapsulation by adding the telemetry packet header into the FL packet. After receiving such FL packet, the next-hop router decapsulates the FL packet via POPINTL action and retrieves the timestamp. The difference between the timestamps of the sending router and receiving router is the one-hop packet delivery delay. The key advantage of our in-band telemetry is that the routers are able to use data packets to carry measurement data with minimum cost, where the measurement data or experience are the keys for training RL agents. In this work, the in-band telemetry system is tested and optimized so that the extra delay induced by the telemetry processing operations is negligible.

Last, wireless networks have another dimension of control on PHY, such as the channel and power, that may significantly affect its overall performance. We have extended programmability to control PHY using NetLink interface from the controller. Each RF hardware on the router is attached as a virtual port on the datapath and link layer discovery such as neighbor peering is handled by MAC80211 stack.



Flow Table 0 Filter packet type

PRIORITY	MATCH FIELDS	COOKIE	DURATION	IDLE TIMEOUT	HARD TIMEOUT	INSTRUCTIONS	PACKET COUNT	BYTE COUNT	FLOW DELAY
1	eth_type = 2054	0	0	0	0	GOTO_TABLE:1	178	10680	0
1	eth_type = 2048	0	0	0	0	GOTO_TABLE:2	0	0	0
1	eth_type = 34889	0	0	0	0	POP_INTL:2048 GOTO_TABLE:2	1130	124434	0

Flow Table 1 ARP Table

PRIORITY	MATCH FIELDS	COOKIE	DURATION	IDLE TIMEOUT	HARD TIMEOUT	INSTRUCTIONS	PACKET COUNT	BYTE COUNT	FLOW D
1	in_port = 1 eth_type = 2054 arp_tpa = 192.168.1.9 eth_src = 02:00:00:00:08:00	0	0	0	0	SET_FIELD: eth_src:02:00:00:00:05:00 SET_FIELD: eth_dst:02:00:00:00:02:00 OUTPUT:IN_PORT	31	1860	0
1	in_port = 1 eth_type = 2054 arp_tpa = 192.168.1.9 eth_src = 02:00:00:00:04:00	0	0	0	0	SET_FIELD: eth_src:02:00:00:00:05:00 SET_FIELD: eth_dst:02:00:00:00:02:00 OUTPUT:IN_PORT	0	0	0

Flow Table 2 Fowarding and delay for RL

PRIORITY	MATCH FIELDS	COOKIE	DURATION	IDLE TIMEOUT	HARD TIMEOUT	INSTRUCTIONS	PACKET COUNT	BYTE COUNT	FLOW D
1	in_port = 1 eth_type = 2048 ipv4_dst = 192.168.1.8 ipv4_src = 192.168.1.9 eth_src = 02:00:00:00:02:00	0	0	0	0	SET_FIELD: eth_src:02:00:00:00:05:00 SET_FIELD: eth_dst:02:00:00:00:08:00 PUSH_INTL:34889 OUTPUT:IN_PORT	89	8730	2
1	in_port = 1 eth_type = 2048 ipv4_dst = 192.168.1.8 ipv4_src = 192.168.1.9 eth_src = 02:00:00:00:08:00	0	0	0	0	SET_FIELD: eth_src:02:00:00:00:05:00 SET_FIELD: eth_dst:02:00:00:00:08:00 PUSH_INTL:34889 OUTPUT:IN_PORT	0	0	0

Figure 2.4: MultiFlow Table

2.3.4 Network Core Services

OpenFlow-enabled controller provides core services for interacting with the datapath and to develop network applications for orchestrating the packet handling behavior. Our core services include OpenFlow Manager based on RYU controller, telemetry manager, network state, and telemetry database based on MangoDB [21], and radio interface manager based on NetLink library. OpenFlow manager is responsible for providing the required APIs or handlers to monitor and control the datapath in realtime by adding/removing entries into OpenFlow table

Standard packet forwarding behavior, such as receiving and forwarding packets over ports, can be realized using a single flow table. However, the complexity of the flow table increases exponentially when handling telemetry packets due to the nature of sequential processing of flow table instructions. Hence, we leveraged multi-flow table-based flow instructions for handling packet forwarding, as shown in Figure 2.4.

First, table-0 instructions will identify the presence of telemetry by matching the ether_type, and then the relevant action is performed. Second, table-1 handles ARP requests/replies, and finally, table-2 flow instructions perform the actions for forwarding the packets to the output port.

The telemetry manager instructs and gathers packet flow monitoring metrics from the telemetry-enabled datapath. Our network state database provides RPC-based interfaces for data access within the kernel layer and also provides access interfaces via Northbound APIs for network applications, such as reinforcement learning-based routing algorithms.

CHAPTER 3: A SCALABLE AND SAFE DEEP REINFORCEMENT LEARNING DESIGN FOR MULTI-CONNECTIVITY TRAFFIC MANAGEMENT

3.1 Overview

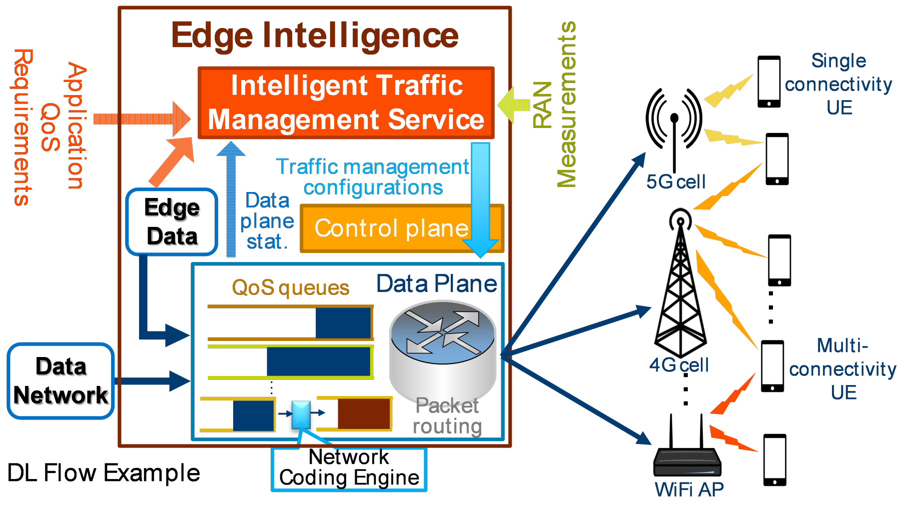


Figure 3.1: Edge-based Multi-Access Traffic Management Framework .

The desire to create multiple concurrent connections via multiple access technologies, such as LTE+5G+WiFi, to achieve higher bandwidth and stability has grown as more client devices are equipped with multiple radio interfaces. A new multi-access traffic convergence model is motivated by the emergence of edge computing where the convergence point at the network can be combined with intelligent traffic management that distribute packets across multi-path to improve quality of service (QoS), as illustrated in Figure 3.1. Multi-access traffic distribution techniques can be created using cutting-edge data-driven machine learning (ML) approaches. We apply the reinforcement learning (RL) method, which develops a multi-access traffic distribution strategy by interacting with the environment. However, poor policy throughout the exploration during initial training stages for online RL may seriously degrade performance. In live-network deployment, failure to meet QoS can harm user experience, such as connection outages and interruptions. In this chapter, we propose two approaches to ensure QoS performance with guided exploration for an online-RL-based multi-access traffic management

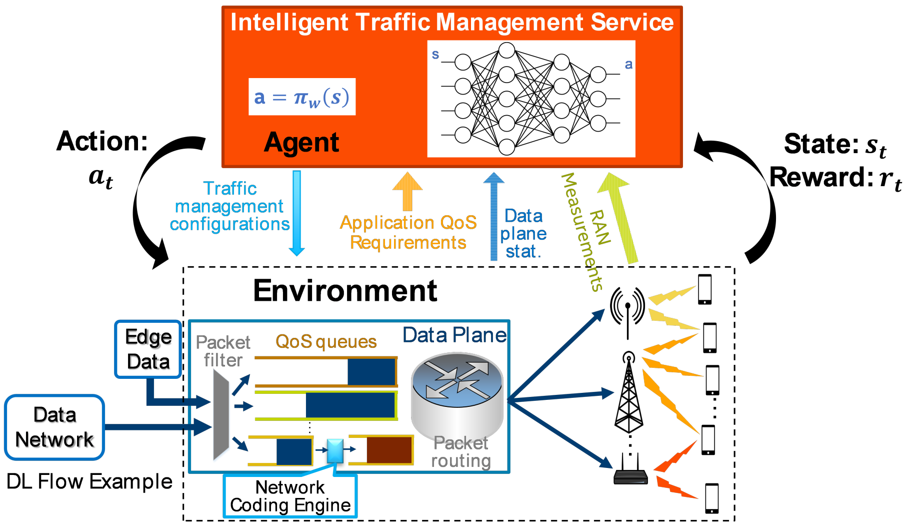


Figure 3.2: Reinforcement Learning based design of intelligent Multi-Access Traffic Management

The goal for edge-based intelligent multi-access traffic management is to determine the best traffic steering (packet routing) strategy for multi-access users based on radio conditions and quality of service (QoS) targets. We developed reinforcement learning (RL) based design of multi-access traffic management. As summarized in Figure 3.2, the idea is to design an intelligent traffic management RL-agent that learns through interaction with the environment. The arrows from âEnvironmentâ to âAgentâ indicate the candidate metrics to be collected that can be used as input states to the agent and/or for reward calculation. The arrow from âAgentâ to âEnvironmentâ indicates the control or policy (output action) to be configured by the Intelligent Traffic Management agent.

We consider the case where the output action for the Intelligent Traffic management RL-agent is the traffic split ratio: $\overline{\boldsymbol{a}}^{(u)} = \left[a^{(u,1)}, \ldots, a^{(u,l)}, \ldots, a^{(u,L)}\right]$, where $a^{(u,l)}$ denotes the portion of traffic for user u to be routed towards the L_{th} access links among all L available access links. The action space for $a^{(u,l)}$ is continuous in [0,1]^L, *i.e.*,$0 \leq a^{(u,l)} \leq 1, \forall l$, with a constraint of $\sum_{l=1}^{L} a^{(u,l)} = 1$ Since $\sum_{l=1}^{L} a^{(u,l)} = 1$, we can substitute $a^{(u,L)} = 1 - \sum_{l=1}^{L-1} a^{(u,l)}$. For the simple case where there are only two available access links, RL-agent output action for user u is simply a^u, where a^u is the ratio of traffic steer to the 1st link and $1 - a^u$ is the ratio of traffic steered to the 2nd link. In the remaining of the disclosure, we will drop the indexing for user link for $a^{(u,l)}$ for simplicity while explaining our design. All concepts we described for a_t are applicable to $a^{(u,1)}, \ldots, a^{(u,L-1)}$, where t is the time index.

3.2 Challenges

Typically, for continuous action space, deep RL trains a policy neural network that chooses a suitable action $_t$ based on the learned experience. Then, an additive Gaussian noise $n_t \sim \mathrm{N}(0, \sigma)$, where 0 is the mean and σ represents the standard deviation (std), will be added to $_t$ to produce the final output action a_t to explore unknown action space:

$$a_t = \mu_t + n_t, \text{ where } n_t \sim \mathrm{N}(0, \sigma)$$

This approach of selecting action with noise is categorized as a stochastic policy according to Gaussian distribution. The likelihood probability of taking current policy can be calculated by the probability distribution function of Gaussian distribution with $mean = \mu_t$ and $std = \sigma$.

The level of randomness of current policy, i.e., how aggressively current policy explores unknown space, will rely on the value of the Gaussian noise. On one hand, the higher Ï tends to give more exploration toward unknown actions. This may lead to poor training performance and unstable policy issues. On the other hand, if a low Ï number was chosen, the policy may approach local optimum as exploration knowledge is limited. Therefore, σ should be chosen as a hyper-parameter with impact to the training performance during the training process. In online-RL training, an agent takes random action to explore and learn new information in order to achieve a good long-term reward. However, poor exploration decision could lead to a severe performance degradation. In traffic splitting distribution case, if the exploration random policy decides to send more traffic to a congested link, it can accumulate a large queue resulting in long-term degradation of latency performance.

3.3 Proposed Solution

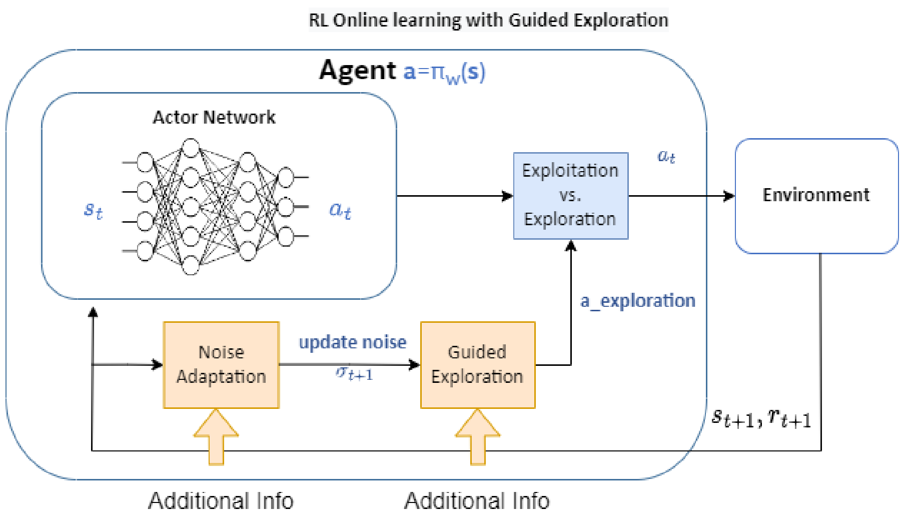


Figure 3.3: Summary of the proposed guided exploration for online RL-based multi-access traffic management.

In this invention, we propose mechanisms to ensure that the Intelligent Traffic management RL-agent selects output action in better exploration direction during online RL training. Figure 3.3 summarizes our proposal: based on the state and reward observation, we developed 1) noise adaptation and 2) guided exploration strategies to properly adjust the final output a_t to ensure the exploration direction and amount are properly controlled.

3.3.1 Adaptive Noise Exploration

We propose to adaptively increases or decreases the noise level based on recently observed rewards. The basic idea is to scale up the noise level when recent reward is low and scale down the noise level when recent reward is high.

An example of additive noise level adjustment strategy is the following:

$$\begin{aligned} \sigma_0 &= \sigma_{\text{init}} \\ \sigma_{t+1} &= \begin{cases} \sigma_t - \delta_{\text{down}} & \text{if } r_t > TH_{\text{high}} \\ \sigma_t & \text{if } TH_{\text{low}} < r_t \leq TH_{\text{high}} \\ \sigma_t + \delta_{\text{up}} & \text{if } r_t \leq TH_{\text{low}} \end{cases} \end{aligned} \tag{3.1}$$

where $TH_{\text{low}}, TH_{\text{high}}, \epsilon_{\text{up}} > 0$, and $\epsilon_{\text{down}} > 0$ are the design parameters selected by model developers for âreward lower threshold for noise adjustmentâ, âreward upper threshold for noise adjustmentâ, ânoise increment step sizeâ, and ânoise decrement step sizeâ, respectively.

Another example of multiplicative noise level adjustment strategy is the following:

$$\sigma_0 = \sigma_{\text{init}}$$
$$\sigma_{t+1} = \begin{cases} (1 - \epsilon_{\text{down}}) \cdot \sigma_t, & \text{if } r_t > TH_{\text{high}} \\ \sigma_t, & \text{if } TH_{\text{low}} < r_t \leq TH_{\text{high}} \\ (1 + \epsilon_{\text{up}}) \cdot \sigma_t, & \text{if } r_t \leq TH_{\text{low}} \end{cases} \tag{3.2}$$

where $\epsilon_{up} > 0$ and $\epsilon_{up} > 1$ are the design parameters selected by model developers for ânoise upward scaling ratioâ and ânoise downward scaling ratioâ, respectively.

For the multi-access traffic management use case, we can set the reward function as the absolute value of one-way delay difference between Wi-Fi and LTE links. However, the average one-way delay varies largely in different network conditions, leading to different upper limits. Therefore, as an example, we re-scale the delay difference by the following normalization function

$$r_t = r_{\text{diff}} = -\frac{d_t}{d_{\max}}$$

, where d_t denotes the absolute value of delay difference between WiFi and LTE links, t is the time index, d_{max} is the historical maximum value of delay difference between WiFi and LTE links observed by the user. Then, we can carefully select upper and lower reward thresholds for noise adjustment and apply either the additive noise level adjustment or multiplicative noise level adjustment strategy with proper parameter setting for step sizes or scaling ratios during the training process.

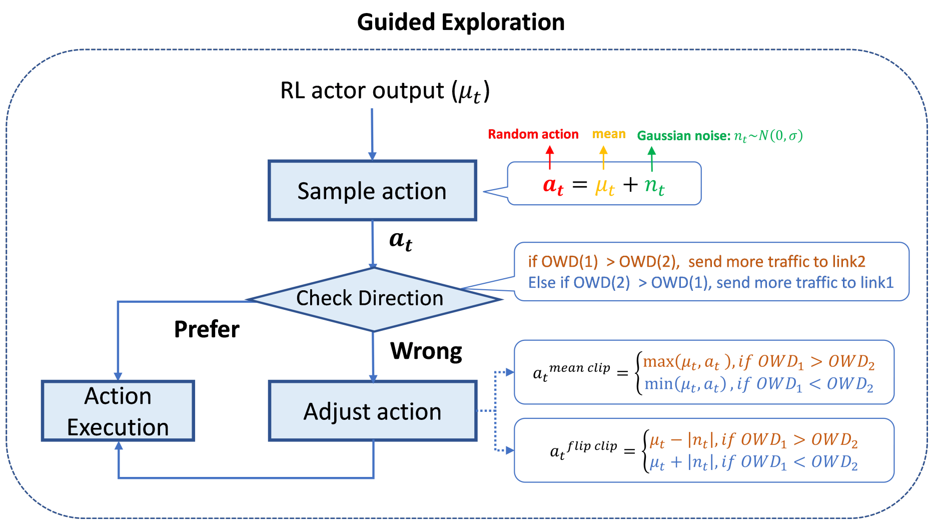

Guided Exploration
RL actor output (μ_t)
Random action
mean
Gaussian noise: $n_t \sim N(0,\sigma)$
Sample action
$a_t = \mu_t + n_t$
a_t
Check Direction
if OWD(1) > OWD(2), send more traffic to link2
Else if OWD(2) > OWD(1), send more traffic to link1
Prefer
Wrong
Action Execution
Adjust action
$a_t^{mean\ clip} = \begin{cases} \max(\mu_t, a_t), if\ OWD_1 > OWD_2 \\ \min(\mu_t, a_t), if\ OWD_1 < OWD_2 \end{cases}$
$a_t^{flip\ clip} = \begin{cases} \mu_t - |n_t|, if\ OWD_1 > OWD_2 \\ \mu_t + |n_t|, if\ OWD_1 < OWD_2 \end{cases}$

Figure 3.4: Guided Exploration Workflow

3.3.2 Clip Random Exploration

We can leverage knowledge from previously developed policies to guide the exploration process during RL training. The idea is that previously developed policies can provide insight on which exploration direction is preferred. We propose a guided exploration with clipped action space method that uses prior knowledge to control the direction of agent to only explore unknown actions in safe action space region. Thus, the clip function prevent agent to explore toward wrong direction. For example, for the multi-access traffic management use case, the preferred direction of exploration can be determined by the congestion level. When a link is more congested, the preferred exploration direction is to reduce load from the link; for a less congested link, the preferred exploration direction is to steer more traffic toward the link. That is, agent is encouraged to explore toward the direction of sending more traffic to less congested link.

The workflow of our proposed guided exploration algorithm is summarized in Figure 4. After sampling an action based on the policy network output μ_t and additive Gaussian noise n_t, we leverage prior knowledge to perform a check on âexploration direction preference.â If the exploration direction is not preferred, we apply adjustment to the action, such as ignoring the noise or applying noise in the opposite direction.

Checking exploration direction preference:

For multi-access traffic management, prior knowledge metrics for determining the preferred the direction of exploration can be as follow,

- One-way delay (OWD) of the link, OWD_i
- Queue accumulation trend, q_i
- Wi-Fi MCS-index vs LTE CQI trend, r_i

For example, by comparing the One-Way Delay (OWD) of an access link to the average OWDs across all available paths, we can assess how congested the access link is compared to other links. A simple rule can be encouraging to explore towards increasing a_t if $OWD_i < \text{Avg}_j OWD_j$, and to explore towards decreasing a_t if $OWD_i > \text{Avg}_j OWD_j$. An example of the above rule for the 2-links case is shown in Figure 3.4.

Similarly, we can create rules based on queue size of each link, or the link quality conditions, to check the preference of exploration direction. A threshold can also be added to the checking logic, so that action adjustment occurs only when the preference is significant enough.

We can also compare the prior knowledge metrics from all users to determine which set of users should apply guided exploration and let the remaining users to explore freely.

Adjust action when exploring towards unpreferred direction

When the random action is in the wrong direction region and the checking logic determines that an action adjustment is needed, we can modify the action towards the preferred exploration direction. Below are some examples of clipping rules for multi-access traffic management with only 2 links, where a_t is the ratio of traffic steered to link 1, and $1 - a_t$ is the ratio of traffic steered to link 2:

Example clipping rule 1:

$$a_t^{\text{clip}} = \begin{cases} \max(\mu_t, a_t), & \text{if } OWD_1 > OWD_2 \\ \min(\mu_t, a_t), & \text{if } OWD_1 < OWD_2 \end{cases} \tag{3.3}$$

The idea is that when the exploration is in the wrong direction, we can simply ignore the random noise, forcing $a_t = \mu_t$ to prevent random exploration. Otherwise, the random action is in the right direction, and the agent is allowed to explore freely in the right direction.

Example clipping rule 2:

$$a_t^{\text{clip}} = \begin{cases} (\mu_t - |n_t|), & \text{if } OWD_1 > OWD_2 \\ (\mu_t + |n_t|), & \text{if } OWD_1 < OWD_2 \end{cases} \tag{3.4}$$

The idea is that when the exploration is in the wrong direction, we can simply apply the random noise to the opposite direction, i.e., forcing to explore in the preferred direction. Otherwise, the random action is towards in the right direction, and the agent is allowed to explore freely toward the right direction.

Another example clipping strategy is that we can scale down the noise or limit the maximum value of the noise when exploring towards the unpreferred direction.

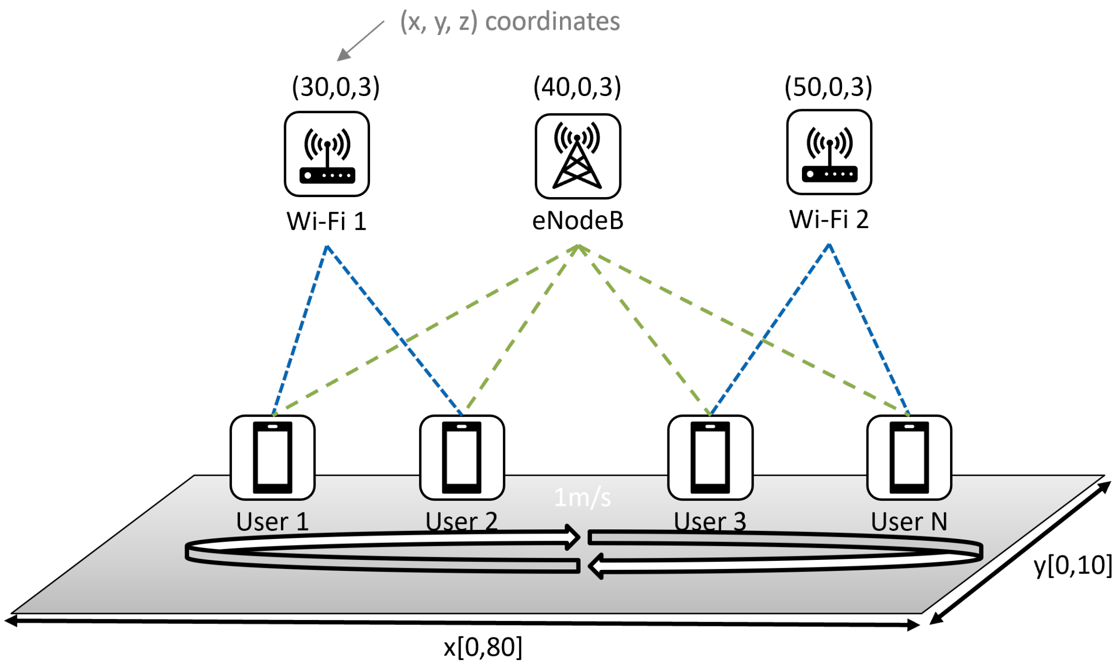


Figure 3.5: Deployment topology

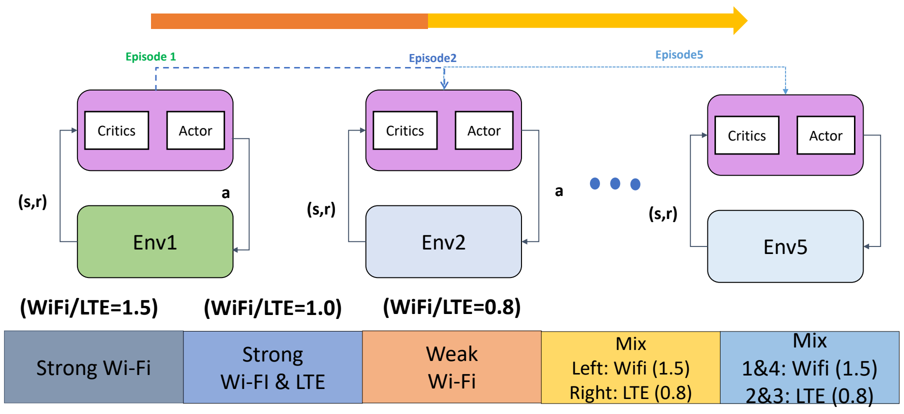


Figure 3.6: Sequential Training Workflow

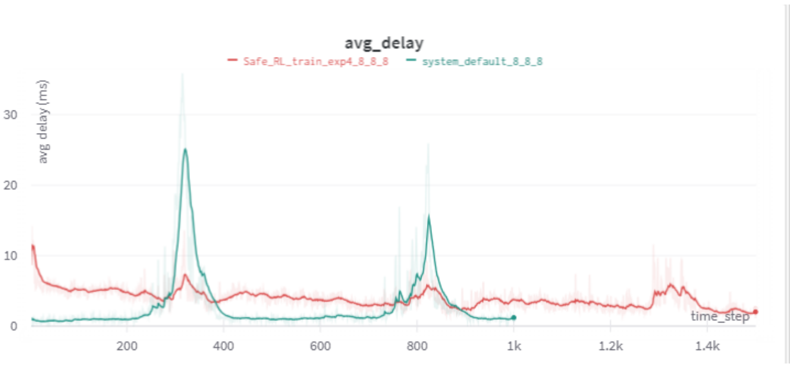


Figure 3.7: Comparison of safe-RL and GMA-baseline during the training

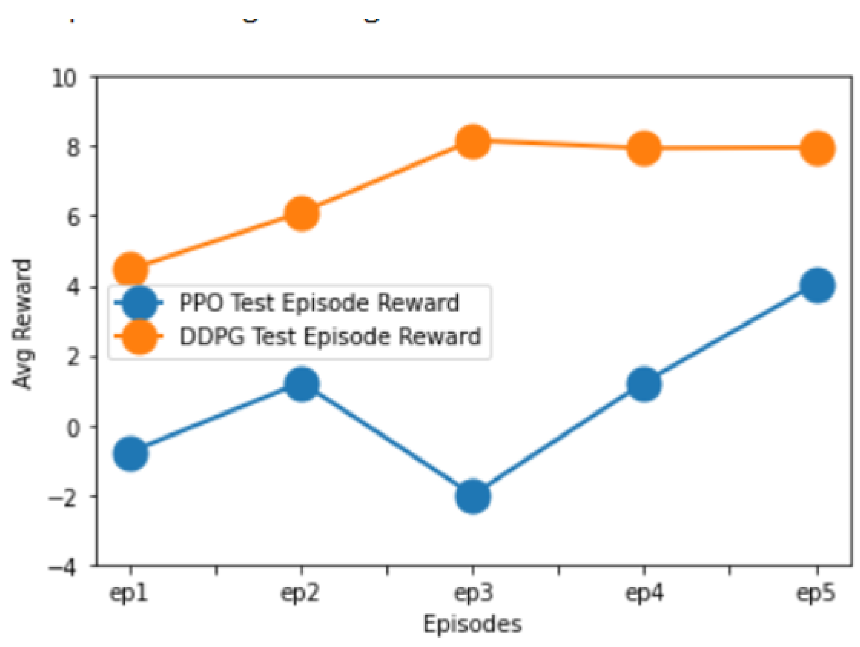


Figure 3.8: Comparison of PPO and DDPG Test Evaluation on all environment

3.4 Results and Conclusion

In Figure 3.7, we observe latency spikes around 30 and 25 ms in the GMA baseline at approximately 300 and 800 time steps. These spikes can be primarily attributed to the challenges inherent in the mobility-oriented setup, particularly when users move farther away from the base station and access point. As users distance themselves, weaker connection signals become a prevalent issue, significantly impacting the network's performance and resulting in intermittent latency spikes.

On the other hand, our experimental results effectively demonstrate the efficacy of our proposed safe-RL mechanism in addressing these latency surges. This mechanism effectively mitigates the sudden packet latency spikes, a phenomenon observed in both the GMA-based baseline and RL training without the safe-RL mechanism. Our safe-RL approach empowers our agent to make informed decisions that prioritize maintaining a safe latency range, ensuring consistently low latency around 5 ms throughout the training process.

More experiments were conducted for our DRL-based multi-access traffic management solution, In addition to showing robustness under varying traffic loads, we further tested our DRL model with different user densities. We evaluated our designs of RL-based Multi-Access Traffic Management for five deployment environments with a mix of strong, medium, and weak WiFi and/or LTE links and three different loading conditions for UDP traffic by varying the location of four User Equipments (UEs), as shown in Figure 3.5. In Figure 3.6, we sequentially performed RL training under different conditions, and models trained from different stages were tested across all deployments. As shown in Figure 3.8, we observed that RL with the Proximal Policy Optimization (PPO) method can quickly learn and adapt to new environments during the training phase. However, PPO fails to memorize previous experiences and is unable to train a model that performs well in all environments. Therefore, we implemented the Deep Deterministic Policy Gradient (DDPG) method with an experience replay buffer to overcome the forgetting issue and designed scalable models with Long Short-Term Memory (LSTM) to support various numbers of users. Results show that DDPG with an experience replay buffer, models trained at later stages continue to perform well in previous training environments, and we can train an ML model that performs well in all evaluated conditions.

CHAPTER 4: MULTIAGENT MARKOV DECISION PROCESSES (MA-MDP) FOR NETWORK ROUTING TRAFFIC ENGINEERING (ALGORITHMS FOUNDATIONS)

4.1 Overview

The overall objective of this chapter is to provide the insight of algorithms foundations of RL-based TE for wireless multi-hop routing. Thus, we develop a modular and composable multi-agent learning system, which provides modules and module extensions that can be selected and assembled in various combinations to generate a specific multi-agent reinforcement learning algorithm that can automate the E2E TE in multi-hop computer networks. Towards this goal, (1) we formulate distributed TE as a multi-agent extension of Markov decision process (MA-MDP); (2) to solve this MA-MDP problem, we propose a modular and composable learning framework consisting of three interleaving modules, each of which can be implemented using different algorithms along with different algorithm extensions. These implementations can be selected and assembled in various combinations to generate a specific reinforcement learning algorithm; (3) we propose a distributed multi-agent actor-critic-executor (MA-ACE) architecture to simplify the interleaving operations between framework modules, thus facilitating fast learning algorithm prototyping and instantiation; (4) we present preliminary results through simulations in a discrete-time network environment.

4.2 Proposed Research

We formulate the traffic engineering problem as a multi-agent extension of Markov decision process [22, 23, 24], where a set $\mathcal{N} = (1, ..., N)$ of agents (i.e., routers) interacts in an environment with an objective to learn the award-maximizing behavior. An MA-MDP learns which next-hop each router should send its packets to in order to move the packets to their destinations with the optimal end-to-end traffic engineering (E2E-TE) performance metrics including E2E delay, E2E throughput, and hybrid E2E TE metric that jointly considers delay and throughput. An MA-MDP for N routers is defined by a tuple $< \mathcal{S}, \mathcal{O}_1, ..., \mathcal{O}_N, \mathcal{A}_1, ..., \mathcal{A}_N, \mathcal{P}, r_1, ...r_N >$, where

- $\mathcal{S}$ is a set of environment states, which include the network topology, the source and destination (i.e., source and destination IP addresses) of each packet in each router, the number of packets (queue size) of each router, and the status of links of each router, e.g., signal-to-interference-plus-noise ratio (SINR).

- $\mathcal{O}_i, i = 1, ..., N$ is a set of observations for each router i, which include local network states available at router i. For example, the observation can be simply the destination of the incoming packet.

- $\mathcal{A}_i, i = 1, ..., N$ is a set of actions for each router i, which include the next-hop routers the current router can forward the packets to.

- $\mathcal{P} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \times \mathcal{S} \mapsto [0, 1]$ is the state transition probability function, which models the environment dynamics. The environment dynamics are driven by the unknown stochastic packet arrival processes of traffic sources, and the packet departure processes that are determined by the action and link status of each router.

- $r_i, i = 1, ..., N : \mathcal{A}_i \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function of each router i, which is defined based on the E2E-TE metrics we want to optimize. In this paper, we aim to optimize E2E delay. Therefore, the reward $r_i = d_i$ is defined as the (negative-signed) 1-hop delay d_i a packet experiences when it is forwarded from current router i to next-hop router $i+1$. d_i includes processing delay, queueing delay, transmission delay, and propagation delay.

When a packet enters a router i, the router obtains its local observation $o \in \mathcal{O}_i$ of the network states and takes an action $a \in \mathcal{A}_i$ to determine where to send this packet to. As a result, the router receives a reward r_i (e.g., (negative) one-hop delay) when the packet arrives at its next-hop router $i+1$, which has its own local observation $o' \in \mathcal{O}_{i+1}$. Each router selects actions based on a policy π_i, which specifies how the router chooses its action given the observation. The policy can be stochastic $\pi_i(a|o) : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0,1]$, where given current observation $o \in \mathcal{O}_i$, the router sends a packet to the next-hop router $a \in \mathcal{A}_i$ according to the probability $\pi_i(a|o)$ with $\sum_{a \in \mathcal{A}_i} \pi_i(a|o) = 1$. The policy can be also deterministic $\pi_i(o) : \mathcal{O}_i \mapsto \mathcal{A}_i$, where given current observation $o \in \mathcal{O}_i$, the router sends a packet to a fixed next-hop router $a \in \mathcal{A}_i$. The return $G_i = \sum_{k=i}^{T} r_k$ is the total reward from intermediate state s_i to final state s_T, where s_i and s_T are the states when a packet arrives at the intermediate router i and destination router T, respectively. Let s_1 be the initial state when a packet enters the network from its source router. The goal is to find the optimal policy π_i for each router i so that the expected return $J(\pi)$ from the initial state (E2E TE metric) is maximized,

$$J(\pi) = E[G_i|\pi] = E[\sum_{i=1}^{T} r_i|\pi] \tag{4.1}$$

where $\pi = \pi_1, ..., \pi_N$. Using different reward function (e.g., 1-hop delay d_i), $J(\pi)$ can characterize different individual E2E-TE metric (e.g., expected E2E delay $E[G_i^{(d)}|\pi] = E[\sum_{i=1}^{T} d_i|\pi]$).

In TE problems, the states are fully observable. That is, the state is uniquely defined by the observations of all routers (i.e., $P(o|s,a) > 0 \implies P(s'|o) = 1$). Thus, in the following sections, for simplicity of notation, we represent an observation o as a state s. In the scope of this paper, we examine only the E2E delay as a reward for delay-optimal traffic engineering. Moreover, we assume that the environmental dynamics, which are characterized by the state transition probability and the distribution of the rewards, are unknown for practical real-world applications. This leads us to propose the following model-free multi-agent reinforcement learning framework.

4.3 Distributed TE Learning Framework

The proposed modular framework architecture includes a generic composing procedure, which assembles a variety of different algorithm and extension options to enable fast prototyping and evaluation. It consists of three key modules, each of which can be implemented using different algorithms along with using different algorithm extensions. These implementations can be selected and assembled in various combinations to generate specific reinforcement learning algorithms. This framework will be developed based on the generalized policy iteration (GPI) strategy [25]. GPI was initially developed to generalize single-agent value-based reinforcement learning algorithms. We will extend it for solving generic TE problems, which needs to exploit emerging policy gradient based learning along with function approximation in a multi-agent setting.

In particular, our framework consists of three interleaving modules for each router/agent: policy evaluation, policy improvement, and policy execution. Let us consider a particular router i and its policy π_i to be learned. Policy evaluation estimates the action-value functions,

$$q_i^{\pi_i}(s, a) = E^{\pi_i}\left[G_i = \sum\nolimits_{k=i}^{T} r_k | s_i = s, a_i = a\right] \tag{4.2}$$

that measure the expected return (expected E2E TE metric) if the router i performs a given action in a given state. Next, policy improvement utilizes the estimated action-values $q_i^{\pi_i}(s, a)$ to adjust current policy π_i in the direction of greater expected return. After that, the agent executes a behavior policy b_i to generate new action-reward experiences for next-round policy evaluation and improvement. As illustrated

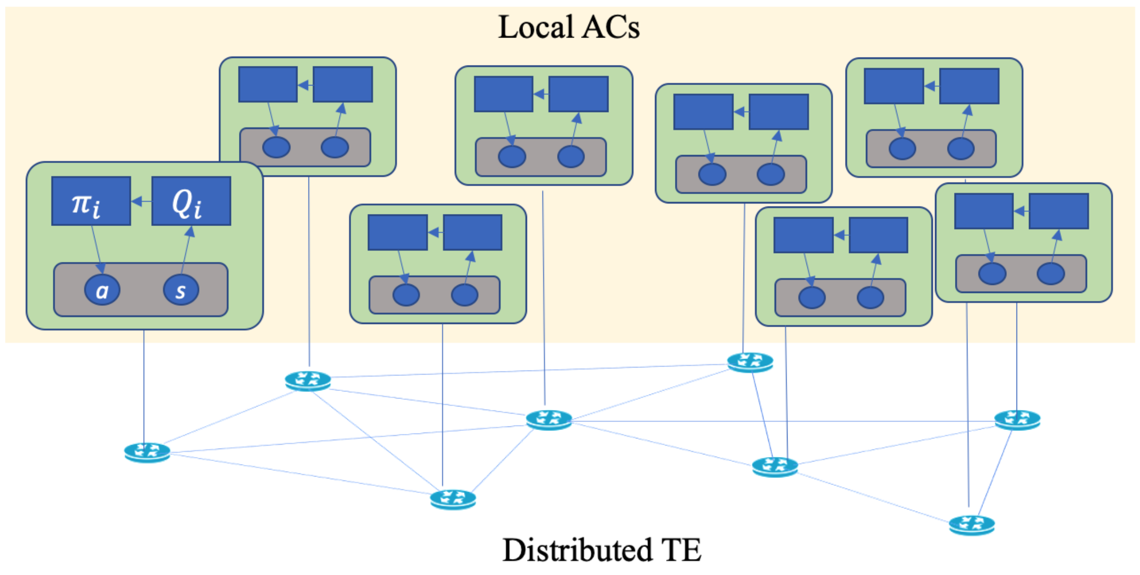


Figure 4.1: Distributed TE Framework which adopts multi-agent, asynchronous actor-critic (AC) architecture. Each router's actor updates its policy function while critic updates the function approximation of state-action Q values.

in Fig. 4.1, we adopt a distributed actor-critic-executor architecture similar to asynchronous advantage actor-critic (A3C) [26] to simplify and implement the interleaving operations between policy evaluation, improvement, and execution. Each router has its own actor, critic and executor running locally and in parallel. The local critic uses a variety of methods to estimate the action-value functions $Q_i^{\pi_i}(s,a)$, which criticize the action selections. Based on critic's inputs, the actor improves the target policy that we want to learn and optimize. Then, the executor executes the actions according to the behavior policy, which is either equal to the target policy or similar to the target policy but more exploratory.

4.3.1 Local Critic for Policy Evaluation

4.3.1.1 1-hop Action-value Estimation

As shown in eq. (4.2), the performance of the policy π is measured by the action-value $q_i^\pi(s,a)$, which is a E2E TE metric. Thus, there will be no direct training sample for policy evaluation until a packet forwarded by this router arrives at its destination.

Inspired by temporal-difference prediction [25], we can apply *spatial-difference (SD) prediction* to quickly update the estimation of $q_i^{\pi}(s,a)$ only using local information exchanged between adjacent routers. In particular, the action-value $q_i^{\pi}(s,a)$ of router i can be recursively rewritten as the sum of 1-hop reward of router i and the action-value of the next-hop router $i+1$, i.e.,

$$q_i^{\pi_i}(s,a) = E\left[r_i + q_{i+1}^{\pi_{i+1}}(s',a')\right]. \tag{4.3}$$

This equation (4.3) indicates $q_i^{\pi_i}(s,a)$ can be estimated by averaging the samples of $r_i + q_{i+1}^{\pi_{i+1}}(s',a')$. This leads to a simple SD predication method based on *exponential weighted average (EWA)*, which iteratively updates the estimate of $q_i^{\pi_i}(s,a)$, denoted by $Q_i^{\pi_i}(s,a)$, based on 1-hop experience tuples (s,a,r_i,s',a') and the estimate of $q_{i+1}^{\pi_{i+1}}(s',a')$ of next-hop router, denoted by $Q_{i+1}^{\pi_{i+1}}(s',a')$, i.e.,

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i + Q_{i+1}^{\pi_{i+1}}(s',a') - Q_i^{\pi_i}(s,a)] \tag{4.4}$$

where $\alpha \in (0,1]$ is the learning rate.

4.3.1.2 n-hop Action-value Estimation

1-hop action-value estimation can be generalized to n-hop one by using n-hop return r_i^n instead of 1-hop return r_i to update the action-value estimate. The n-hop return r_i^n is the accumulated reward when a packet arrives at the n-hop router, i.e., $r_i^n = \sum_{k=i}^{i+n-1} r_k$. For example, if the reward is 1-hop delay, r_i^n represents n-hop delay. With n-hop return, the action-value estimation process in eq. (4.4) can be generalized to

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i^n + Q_{i+n}^{\pi_{i+n}}(s',a') - Q_i^{\pi_i}(s,a)]$$

for all $n \geq 1$ where $Q_{i+n}^{\pi_{i+n}}(s',a')$ is the action-value of router $i+n$. It is shown in previous research [27, 28, 29] (e.g., in video gaming settings) that n-hop/n-step estimation may lead to better policy with higher expected return (e.g., better E2E TE metric in our case) because the n-hop estimate has smaller bias compared with the 1-hop one. However, n-hop estimate may not work well in non-stationary cases and may slow down policy learning process because of the higher variance in the n-step estimation and the longer waiting time to obtain n-hop return from the router n-hop away.

4.3.1.3 Expected n-hop Action-value Estimation

Variance in the action-value estimates is unavoidable because the environment can introduce stochasticity through stochastic rewards r_i^n and stochastic environment state transitions $\mathcal{P}$. There is little we can do to reduce the variance caused by environmental stochasticity, except using a suitably small learning rate α. Besides environmental stochasticity, the action change of next-hop or nth-hop router introduces additional variance. To mitigate such variance, instead of $Q_{i+n}^{\pi_{i+n}}(s', a')$, the expected value $E[Q_{i+n}^{\pi_{i+n}}(s', a')] = \sum_{a' \in A_{i+n}} \pi_{i+n}(s', a') Q_{i+n}^{\pi_{i+n}}(s', a')$ is used to update $Q_i^{\pi_i}(s, a)$, $\forall n \geq 1$ [30]:

$$Q_i^{\pi_i}(s, a) \leftarrow Q_i^{\pi_i}(s, a) + \alpha[r_i^n + E[Q_{i+n}^{\pi_{i+n}}(s', a')] - Q_i^{\pi_i}(s, a)].$$

4.3.2 Local Actor for Policy Improvement

Policy evaluation process drives the action-value function to accurately predict the true returns (E2E TE metrics) for current policy. Policy improvement process improves the policy with respect to current action-value function. Policy improvement can be done through action-value methods or policy-gradient methods. In this paper, we focus on action-value methods.

4.3.2.1 Action-value methods

Action-value control algorithms aim to learn an deterministic target policy, which maximizes the performance objective $J(\pi)$ in eq. (4.1), i.e., the expected return from start state, by selecting a fixed greedy action with respect to the expected return from any state. This can be done by letting each router i greedily improve its current policy π_i, i.e., select the action with the maximum estimated action-value,

$$\pi_i(s) \leftarrow \arg\max_a Q_i^{\pi_i}(s, a).$$

Since value based methods can only learn deterministic policies, this naturally leads to single-path TE solutions, where a single routing path is learned between a source source-destination pair. The single-path TE solutions are simple and easy to implement. However, to improve E2E delay at high traffic load cases, multi-path TE solutions are generally preferred, where each source-destination pair is connected with multiple routing paths to better distribute traffic load and reduce E2E delay. To address this problem, two generic near-greedy action selections can be exploited: (1) $\epsilon-$greedy policy, where with probability $1-\epsilon$, select the best action and with probability ϵ, other actions are selected, and (2) softmax-greedy policy, where each action a is selected with a probability $P(a)$ according to the exponential Boltzmann distribution

$$P(a) = \frac{\exp(Q_i^{\pi_i}(s, a)/\tau)}{\sum_{b \in \mathcal{A}_i} \exp(Q_i^{\pi_i}(s, b)/\tau)}$$

where τ is a positive parameter called the temperature. High temperatures (as $\tau \to \infty$) cause the actions to be almost equally probable (close to random action selection). Low temperatures (as $\tau \to 0$) get close to the deterministic action selection. The near-greedy methods force each router to select next-hop forwarding nodes stochastically, which create multiple routing paths connecting source and destination nodes.

4.3.2.2 Double learning

All action-value algorithms above involve maximization during the construction procedure for the target policies. More specifically, the target policies follow the greedy or near-greedy action selection methods given the current action values, which are defined with a maximization operation. In this case, the maximum of the estimated action values is used as an estimate of the maximum of the true action value. This can lead to a significant overestimation bias and thus degrade the gain of the learning algorithms. To address this overestimation bias, double learning can be adopted [31], which decouples action selection from its evaluation using a pair of estimators. In particular, we divide the time steps (or equivalently the experiences) into two sets and use them to learn two independent estimates, namely $Q^{\pi_i}_{i,1}(s,a)$ and $Q^{\pi_i}_{i,2}(s,a)$. One estimate, e.g., $Q^{\pi_i}_{i,1}(s,a)$, can be used to determine the improved action denoted by $a^*(Q^{\pi_i}_{i,1})$, according to greedy or near-greedy strategies $\pi_i \in \{\text{greedy, softmax, } \epsilon - \text{greedy}\}$. Then, we use the other estimate $Q^{\pi_i}_{i,2}(s,a)$ to provide the estimation of the value of the improved action, i.e., $Q^{\pi_i}_{i,2}(s,a^*) = Q^{\pi_i}_{i,2}(s,a^*(Q^{\pi_i}_{i,1}))$, Then, the role of the two estimates can be reversed to yield a second unbiased estimate $Q^{\pi_i}_{i,1}(s,a^*(Q^{\pi_i}_{i,2}))$.

Double learning strategy can be combined with any policy evaluation and policy improvement methods to create new learning algorithms. For example, we can design the double expected softmax learning by combining double learning with expected action-value estimation and softmax policy improvement. In this case, the action value update rule for the first estimate $Q^{\pi_i}_{i,1}(s,a)$ is given as follows

$$Q^{\pi_i}_{i,1}(s,a) \leftarrow Q^{\pi_i}_{i,1}(s,a) + \alpha[r_i + \Psi_{i+1,2}(s',a') - Q^{\pi_i}_{i,1}(s,a)].$$

Here, $\Psi_{i+1,2}(s',a') = \sum_{a\in\mathcal{A}} P(a,1) Q^{\pi_{i+1}}_{i+1,2}(s',a')$ where $P(a,1) = \frac{\exp(Q^{\pi_i}_{i,1}(s,a)/\tau)}{\sum_{b\in\mathcal{A}_i}\exp(Q^{\pi_i}_{i,1}(s,b)/\tau)}$. By the same way, we can obtain the update rule for the second estimate $Q^{\pi_i}_{i,2}(s,a)$. Similarly, we can also combine double learning with greedy policy improvement, where $\Psi_{i+1,2}(s',a') = Q^{\pi_{i+1}}_{i+1,2}(s', \arg\max_a Q^{\pi_i}_{i+1,1}(s',a))$.

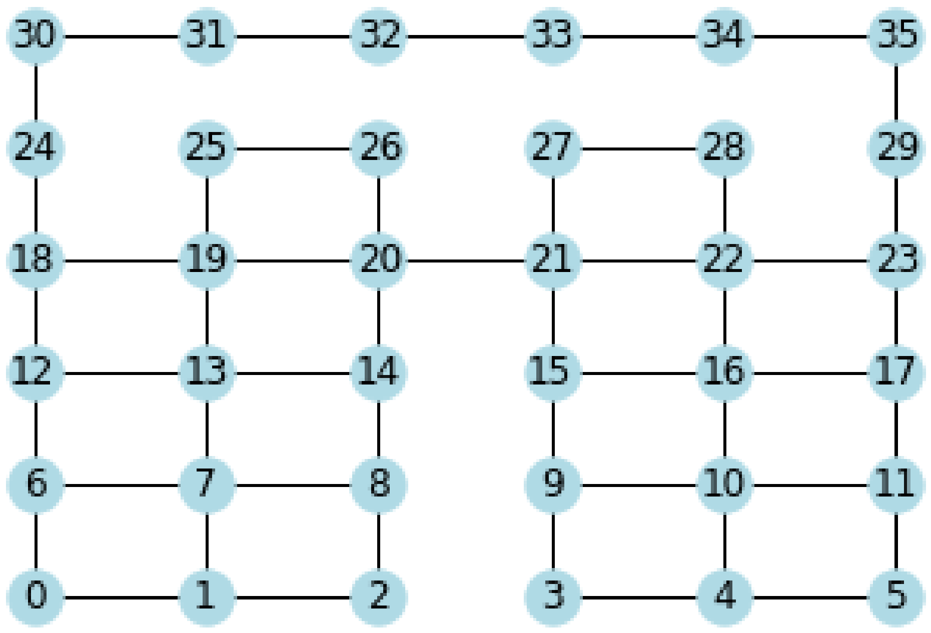


Figure 4.2: Network topology in the simulations [1]

4.3.3 Local Executor for Policy Execution

In reinforcement learning, the behavior policy b_i is the policy that generates the actual actions of the learning agent, which yields the actual experiences for improving target policy π_i. For on-policy learning, the behavior policy b_i is same as the policy being improved (target policy π_i). For off-policy learning, the behavior policy b_i is different from the target policy, where the target policy is generally the greedy policy and the behavior policy is near-greedy to encourage explorations. The policy π_i will be learned when the evaluation process and the improvement process stabilize, that is, no longer produce changes.

4.4 Experiments

4.4.1 Experiment Setup

We evaluate the performance of the learning-based TE algorithms in a discrete-time network simulator [1], which is widely used to investigate the performance of

Q-learning based routing algorithm and its variants [1, 32, 33, 34, 35]. The network topology is shown in Fig. 4.2, where the nodes represent routers and edges represent network links. In this discrete-time simulation, the whole network is driven by three major factors: the packets arrival pattern, the average packet arrival rate, and the queuing and link delay. First, packets are periodically generated with a randomly selected source router and destination router for each packet. Second, the network loads are based on the packet arrival rate and it is driven by Poisson distribution with parameter λ, the average number of packets injected into network per time step. Third, when a packet arrives at a router, it has to wait in a FIFO queue (queue length = 1000 packets) and thus experiences queuing delay. The packet will be transmitted over the communication link if it becomes the head-of-line packet. The transmission delay of the communication link needs to be constant in this simulator and accordingly, we use the unit transmission delay for the simulations (i. e. , the link delay is set to 1.0).

Local states at each router include the source and destination IPs of the incoming packet and local queue length, i.e., $s = (srcIP, dstIP, queue)$. The action is the next-hop forwarding node, i.e., $nexthop\ ID$. The critic sets 1-hop value estimation ($n = 1$). The 2-hop value estimation ($n = 2$) is included to examine the comparative efficacy of n-hop learning.

In this experiment, we investigated deterministic and near-deterministic policies with ϵ-greedy ($\epsilon = 0.1$) and softmax ($\tau = 1$).

Off-policy, on-policy, expected value estimation (Expected), double learning (DBL), and n-hop learning ($n = 2$) are employed in the framework to examine the efficacy of learning-based adaptation. Two baseline algorithms are also included: the shortest-path routing and the off-policy greedy algorithm (i.e., Q-routing [1]). For Q-routing, both behavior and target policies are greedy, and the explorations are implicitly driven by the changes in action value (Q-value) estimations.

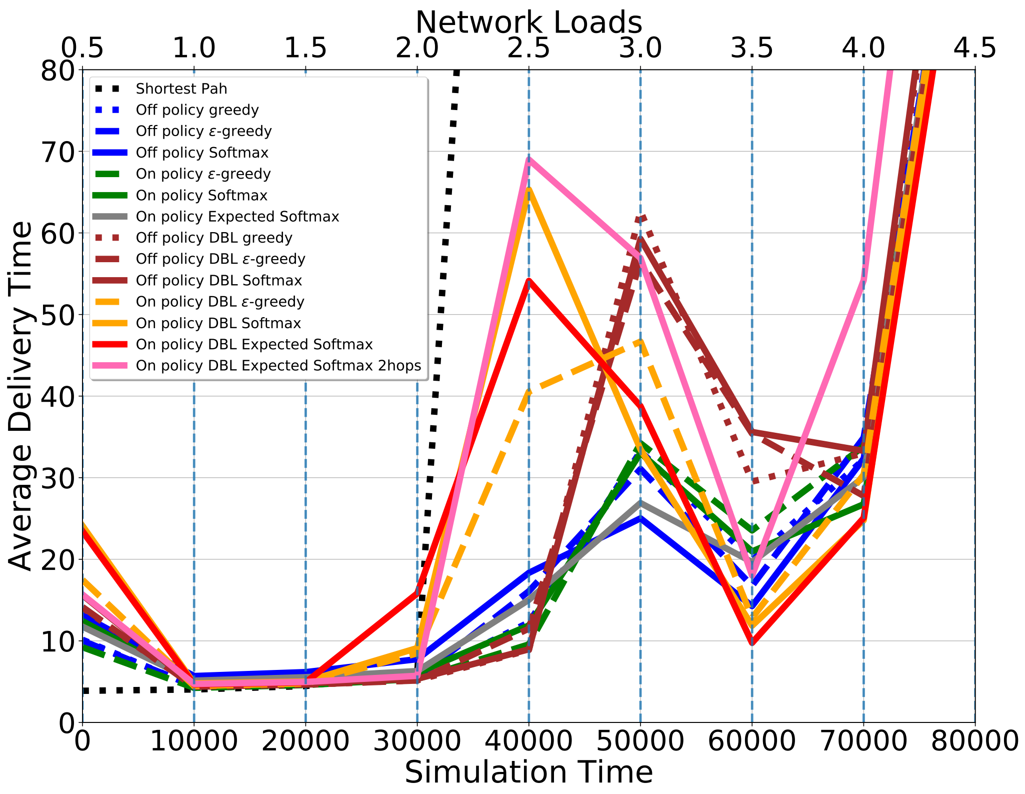


Figure 4.3: Average of 50 runs of E2E delay patterns on low to high network loads

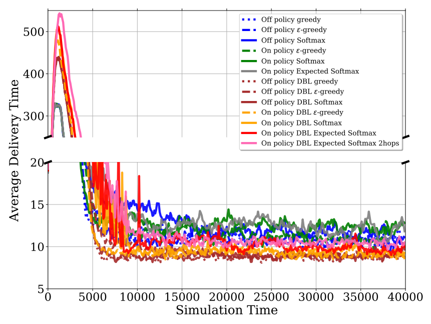
4.4.2 Varying Traffic Loads: Low to High

Unlike previous research that has tested algorithms in single traffic load case, we consider the low to high load condition to study the overall performance of all adaptive learning algorithms in non-stationary network environment, where the state transition probability function keeps changing as traffic load increases. In this experiment, the following changes to the network parameters were made according to time. We initialized Q-table values with 0. The first 10k time steps were given for the initial exploration at a low load $\lambda = 0.5$. λ is increased by 0.5 for every 10k time steps. From pilot tests, we select the best performing learning rate $\alpha = 0.9$ for all algorithms except 2-hop learning algorithm, whose optimal learning rate is $\alpha = 0.5$.

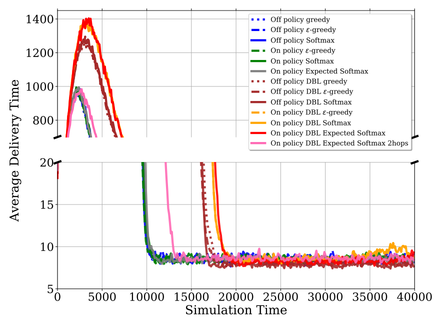
4.4.3 Adaptivity under High Traffic Load Region

The results in Fig. 4.3 show that all of reinforcement learning approaches learn efficient routing policies while the shortest path routing scheme failed to tolerate the increased packet loads (when $\lambda > 2.0$). During the medium traffic loads ($2.0 \leq \lambda \leq 3.0$), off-policy learning algorithms outperform on-policy algorithms, where off-policy softmax with double learning is the best one during the load ($2.0 \leq \lambda \leq 2.5$) and off-policy softmax is the best during the load ($\lambda = 3.0$). During high traffic loads ($3.5 \leq \lambda \leq 4.5$), on-policy algorithms outperform off-policy approaches. Moreover, for on-policy algorithms, expected value evaluation helps to improve the delay performance due to the smaller variance in the return estimation. Double learning, for most of the cases except loads ($\lambda = 3.0$), is beneficial by reducing maximization bias. Softmax action-selection, in general, helps the agent to select second-best control links with a probability, and it helps exploration of other paths to balance the traffic in high network loads and to reduce average packet delivery time. 2-hop learning algorithm does not bring much performance gain.

In this experiment, we further investigate the performance of the learning algorithms in high-traffic load cases. We increase the network loads from $\lambda = 3.5$ to $\lambda = 4.4$ by 0.1 on every 10k time steps. As illustrated in Fig. 4.4.2, when network loads are high, again double learning algorithms with softmax action selection (orange, red, and brown solid lines) shows highest adaptivity and best performance. As network loads further increases, off-policy double learning adapts poorly, comparing to on-policy double learning algorithms after $\lambda = 3.9$. These results show additional exploration, leaving from deterministic policy, helps balancing high network traffic loads. Both softmax and on-policy learning add efficient exploration for this, and avoiding over-estimation with double-learning improves overall E2E delay performance. Fig. 4.4.2 shows the delay performance with higher time resolution, where the average packet delay is computed for every 1,000 time steps. We observe that all algorithms take some time to learn and eventually adapt to the traffic load changes well.



(a) High learning rate ($\alpha = 0.9$)



(b) Low learning rate ($\alpha = 0.1$)

Figure 4.5: Average of 50 runs for convergence of learning algorithms in fixed high network loads ($\lambda = 3.5$). For each run, we measured the average delivery time for every 100 time steps

4.4.4 Convergence Analysis under a Stationary Case

In Fig. 4.5, with fixed high network loads ($\lambda = 3.5$) , we test the convergence of learning modules using different learning rates (0.1 and 0.9, respectively). The the fixed traffic load case represents stationary network condition, where state-transition probability function converges as time proceeds. As shown Fig. 4.5, all learning algorithms converge eventually. As expected, high learning rate = 0.1 leads to better delay performance for all algorithms because of low bias in action-values (i.e., Q values) estimation. However, the learning time is much higher with the low learning rate. High learning rate (0.9) shortens the convergence time at least by half to reach fairly good (through not optimal) performance. Moreover, we observe the double learning models stably converges the best results for both low and high learning rates. With high learning rate, double learning algorithms only one third of convergence time, compared with low learning rate case.

4.5 Conclusions

In this work, we formulate distributed TE problems in reinforcement learning problems and suggest a composable framework for diverse learning algorithm application. Through extensive comparative experiments, we demonstrated high adaptiveness of learning-based distributed TE approaches, which lead robust load-balancing network systems that minimize the E2E delay. Empirically, we were also able to observe that reducing over-estimation bias with double-learning along with non-deterministic action selection with softwax improves adaptivity and sustainability of network systems.

CHAPTER 5: NETWORK ASSISTED AI OPTIMIZATION THROUGH MA-RL ROUTING

5.1 Overview

In this chapter, we aim to present our novel system design and practical use case of RL-based TE wireless multi-hop routing in the real physical system over FL application. Therefore, our objective in this work is to develop a framework that can be used on Commercialoff-the-shelf hardwares and as well as within system-in-loop emulator framework to allow users to seamlessly implement and validate the effectiveness of RL routing solutions. Therefore, (1) We first developed a AI-Enabled Wireless Network Operating System (WINOS), which seamlessly integrates programmable measurement, i.e., the proposed SINT In-band telemetry framework and the programmable wireless network control. (2) We designed and implemented S-INT, a distributed in-band telemetry system, where each router runs its own telemetry module that is built on the top of OpenFlow datapath/processing pipeline. (3) We adopt and applied a Multi-Agent Reinforcement Routing application for FL wireless multi-hop application using WINOS. In this case, each router is an agent, which observes the network states (e.g., the source IP and destination IP of the incoming FL packet) and learns the optimal networking policy (e.g., the forwarding action at each router) based on the reward signals (i.e., negative per-hop delay) with an objective to maximize the expected total return (i.e., end-to-end delay) from the initial state (i.e., when the FL packet entering the network) to the terminal state (i.e., when the FL packet leaving the network) as a result it can minimize the wall-clock convergence time to achieve the desired FL accuracy

Line-speed Action-state Value Estimation: The key challenge to implement reinforcement routing algorithms is how to estimate the action-state values (i.e., Q values) without inducing so much control overhead. In particular, estimating Q values relies on the measurement of per-hop per-packet delay as shown in eq. (4.4). Directly requesting the delay information from the neighboring router could introduce significant overhead to the bandwidth-limited wireless channel. Therefore, it is necessary to redesign the way of exchanging information among neighbors. We design the line-speed Q value estimation for each router i, which aims to realize Q estimation at the line speed, i.e., the speed at which packets come in the router. The local Q_i estimation of router i is directly coming from its next-hop neighbor. The motivation of such design is based on the fact that the action-state value Q_i of router i is estimated based on the per-packet reward (per-packet delay) r_i and the action-state value Q_{i+1} of the next-hop router $i+1$. Both Q_{i+1} and per-packet reward r_i is immediately available at next-hop router $i+1$ where r_i is obtained via the in-band telemetry introduced above. Therefore, it is more cost-effective to let next-hop router $i+1$ estimate the action-state value Q_i of the current router i via exponential moving average. The estimated action-state value $E(r_i + Q_{i+1})$ is sent back by the next-hop router $i+1$ to current router i periodically (e.g., every five seconds). Such a scheme allows the action-state value to be updated at the line speed, while orderly reducing the control overhead.

5.2 Optimizing FL Convergence via Reinforcement Learning

5.2.1 Problem Formulation

Our overall objective is to minimize the run-time convergence time to achieve the desired FL accuracy. Towards this goal, the optimal strategy is to minimize the worker-server delay of the slowest worker, which experiences the maximum delay among all workers. However, in highly dynamic wireless environments, the role of the slowest one can be randomly switched among different workers as time proceeds. In this paper, we sought a sub-optimal solution, where we minimize the average

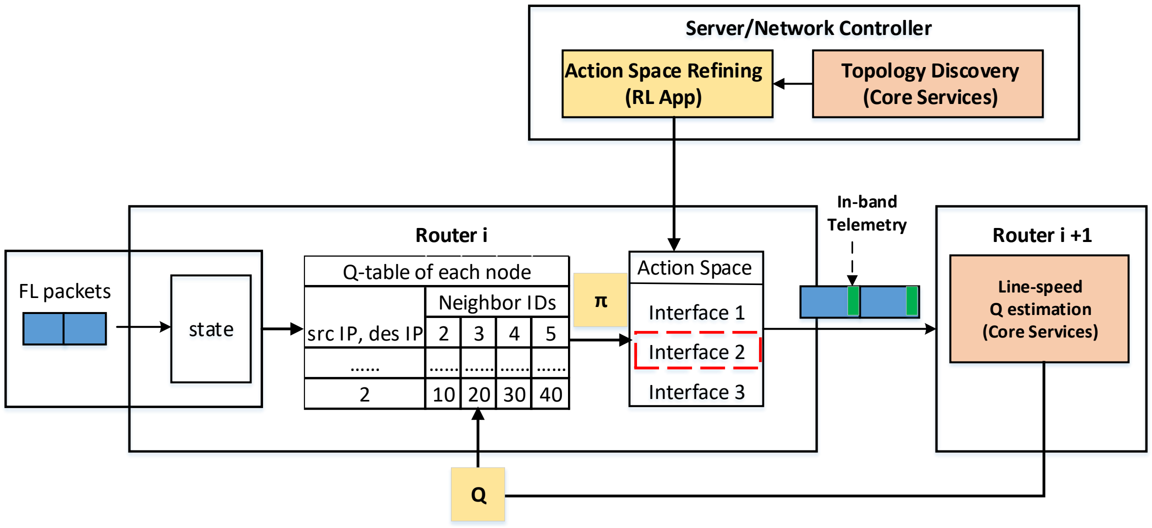


Figure 5.1: RL Application with tabular Q estimation

end-to-end (E2E) delay between all workers and the server. However, even for such sub-optimal solution, we cannot apply the classic model-based optimization because the server-worker E2E delay cannot be explicitly formulated as a closed-form function of the routing/forwarding decisions [36]. As a result, a model-free optimization strategy based on multi-agent reinforcement learning is much more desirable, where each wireless router exploits its instantaneous local experiences to collaboratively learn the delay-minimum routing paths between the workers and the server.

In particular, this problem can be formulated as the multi-agent Markov decision processes (MA-MDP), which can be solved by multi-agent reinforcement learning algorithms. Given the local *observation* o_i, which is the source IP and destination IP of the incoming FL packet, each router or *agent* i selects an *action* a, i.e., the next-hop router, to forward this packet, according to a local forwarding *policy* π_i. After this packet is forwarded, the router i receives a *reward* r_i, which is the negative one-hop *delay* between router i and the selected next-hop router. The packet delivery delay $d_{i,i+1}$ is the time interval between the time when packet arrives at router i and the time when the packet arrives at the next-hop router $i+1$. The packet delivery delay $d_{i,i+1}$, which includes the queuing delay, processing delay and transmission delay, is

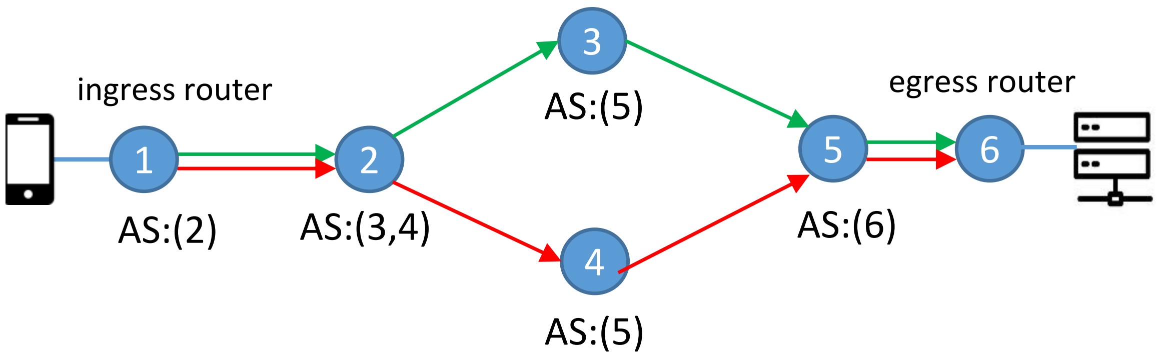


Figure 5.2: Loop-free action space (AS) refining. There exist two loop-free paths between the ingress router and egress router. For each router, we refine the action space (AS) for each router in the network that two routing paths traverse through.The RL algorithm will explore the actions in the refined action space to learn loop-free routing paths.

a random value measured in real-time by in-network telemetry module introduced in the next section. The *return* $G_i = \sum_{k=i}^{T} r_k$ is the total reward from intermediate state s_i to final state s_T, where s_i and s_T are the states when a FL packet arrives at the relay router i and destination router T, respectively. Let s_1 be the initial state when a FL packet enters the network from its source router. The source/destination router is the router that a worker or the server is attached to. The *objective* is to find the optimal policy π_i for router i so that the expected return $J(\pi)$ from the initial state (i.e.,E2E server-worker delay) is optimal, where $J(\pi) = E[G_1|\pi] = E[\sum_{i=1}^{T} r_i|\pi]$ where $\pi = \pi_1, ..., \pi_N$.

5.2.2 Convergence Optimization via Multi-agent Reinforcement Learning

To solve the above MA-MDP problem, we exploit the multi-agent reinforcement learning, where the routers (agents) distributively learn the optimal target forwarding policy π to minimize the average server-worker delay. To implement the multi-agent reinforcement learning algorithm, we adopt a distributed actor-critic architecture similar to asynchronous advantage actor-critic (A3C) [37, 26], where each router individually runs a local critic and a local actor,

5.2.3 Loop-free Action Space Refining

The MA-RL routing is one kind of the distributed routing algorithms. However, the key idea of RL is to improve the policy by learning from experiences including failures. Therefore, an agent can freely explore and learn all possible routing paths including the ones with loops, where the data packets continue to be routed within the network in an circle. Our experiments show that the routing loops can have a catastrophic impact on a RL-based networking, such as the slowly converged routing policy and TCP disconnections between the server and workers. To address this problem, we propose a action space refining algorithm, which aims to construct the loop-free action spaces for each router in such a way that the routers can independently and distributively explore any forwarding action (i.e., the next-hop router) from such refined action space, while avoiding generating routing paths with loops. The refined action space is defined with respect to each pair of ingress and egress routers. The ingress router is the router from which the FL traffic flow enters the network and the egress router is the router from which the FL traffic leaves the network. Therefore, the maximum number of action spaces constructed on each router is equal to $2N$, where N is the total number of routers in the network. The action space refining algorithm works as shown in the Fig. 5.2. First, build the global network topology. Then, find all the loop-free paths between the ingress router and egress router by applying iterative depth-search-first (DSF) traversal or K-shortest path finding algorithms (with sufficiently large K). Next, for each router, there may exist multiple paths traversing it and its action space is a set of the next-hop nodes of all the traversing paths. It is easy to prove that employing such refined and loop-free action spaces, our MA-RL forwarding scheme will surely learn the routing paths without loops.

It is worth to note that the action space refining algorithm is only performed by a network controller which has the global network topology. The implementation details are shown in the next section.

5.3 RL Application

RL routing solution is developed as a network application that can access and pass messages using core services REST API. The first step to implement an Actor-Critic RL algorithm (Section 5.2) is to build the Q-table based on the local topology information retrieved from the network state database as shown in Figure 5.1. Following this, we can initiate the Q-table using the estimated Q-values $E(r_i + Q_{i+1})$ stored in the database. Actor disseminates the routing control decisions following the critics and policy π to the OpenFlow manager. OpenFlow manager translates the RL decisions into actionable OpenFlow datapath instructions for orchestrating the packet forwarding behavior. To accelerate the RL policy convergence, the loop-free action space is calculated by the action space refining application (Section 5.2.3). This application is running on the network controller, which has the global network topology readily available via the topology discovery module. Our topology discovery module can operate in either a centralized or distributed manner. The centralized approach directly uses the link layer discovery protocol (LLDP) that is initialized and coordinated by the network controller. For the distributed approach, each router discovers its one-hop neighbors via IEEE 802.11 local topology discovery scheme and the local typologies from all the routers are then aggregated by the network controller to form the global topology. It is worthy to note that both the network controller and wireless routers employ the same federated network subsystem shown in Figure **??**, except that the network controller runs an additional action space refining application. This implementation enables online reinforcement learning which updates Q table in real-time. Therefore, it does not require to train in a simulated environment before deployment to real physical wireless routers.

Besides the Q table, we can also adopt different function approximations such as neural networks for non-linear Q value approximations especially when more network states are utilized, such as traffic matrix and queue lengths. However, this can raise additional concerns related to computation requirements for deployment. Adoption and investigation of different nonlinear function approximation is out-of-scope of this paper, leaving them as a future work.

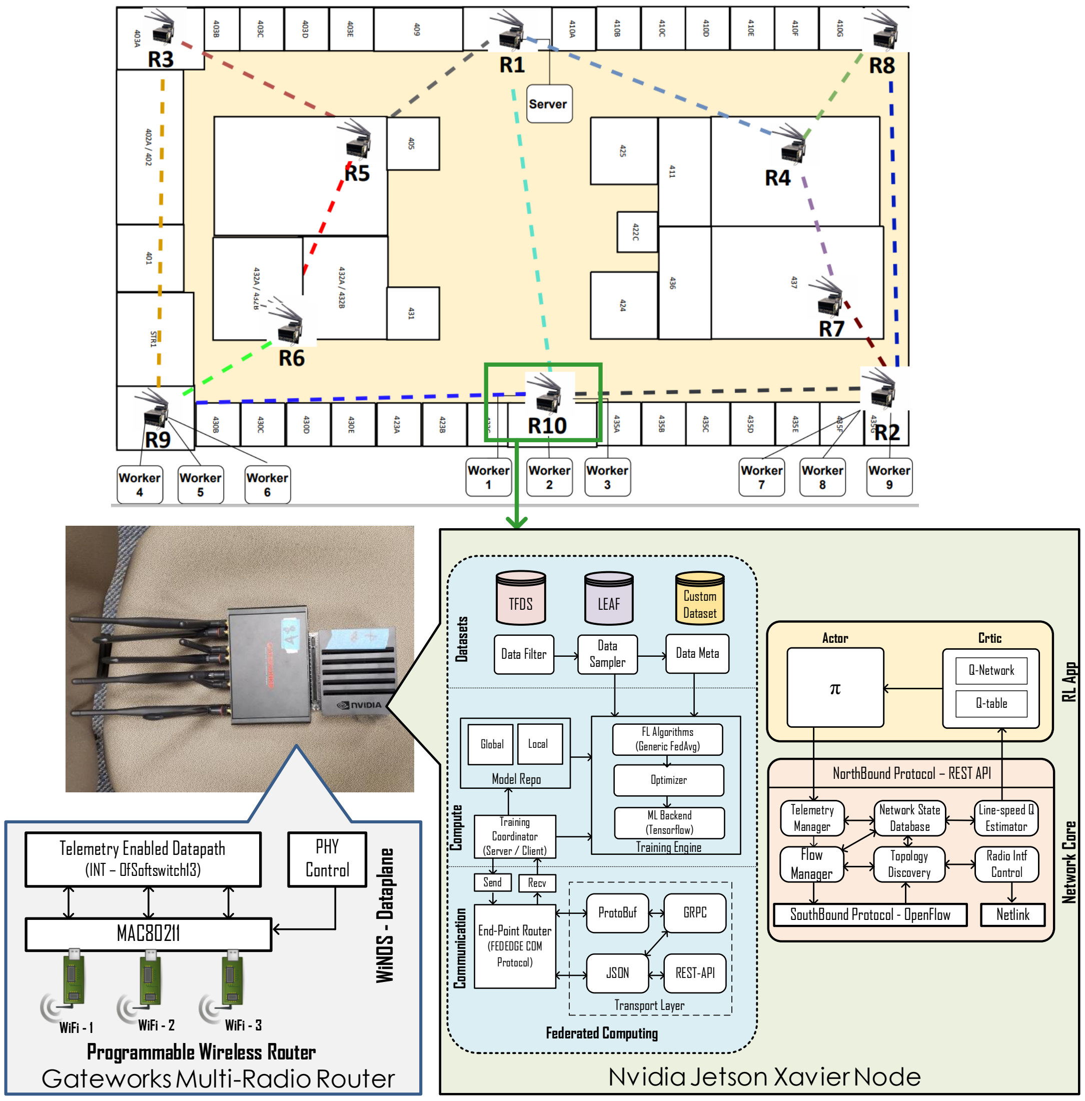


Figure 5.3: EdgeML - Testbed Topology and Node View

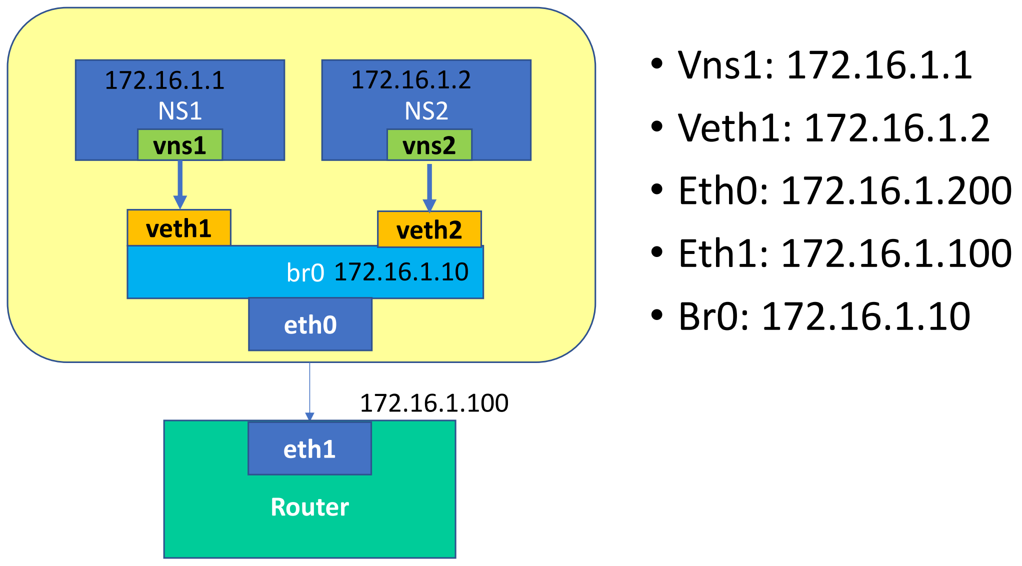
5.4 System Implementation

5.4.1 Overall Implementation

We prototype our EdgeML system with a mesh topology as shown in Figure 5.3. This testbed consists of 10 Nvidia Jetson Xavier nodes, each of which is connected to one Gateway 5400 multi-radio wireless router. Each Nvidia Jetson node serves as federated computing node which handles the federated learning training. In addition, the network core and RL-app of federated networking subsystem are also hosted on Nvidia Jetson node. On the other hand, Gateworks routers serve as federated networking node that only hosts a dataplane submodule. The dataplane is orchestrated using OpenFlow protocol by the network core services hosted on Nvidia jetson node. Such decoupled system design allows resource-rich Nvidia nodes to handle the computation-intensive FL operations and RL-based networking intelligence while keeping the operations of resource-limited routers simple and fast.

5.4.2 Federated Networking Subsystem Implementation

Our multi-radio wireless federating networking nodes (Gateway routers) are off-the-shelf small-factor single-board computers which support Linux operating systems with multiple PCIe slots for adding wireless radio cards. In our testbed, we deployed Ubuntu 20.04 as the operating system and 3 x Compex WLE900VX-I wireless cards to enable multi-radio wireless nodes. Each wireless radio is set to operate on 5 Ghz channels and 20 Mhz channel width in 802.11ac operating mode with 15 dBm transmission power. As a result, each wireless router in our testbed can reach roughly 40 Mbps aggregated data rate from three radio cards. On top of the node operating system, we deploy a dataplane submodule that facilitates a software bridge with a programmable packet handling routine (i.e OpenFlow Flowtable). All three wireless cards were configured to operate on disjoint channels, and then they were added to OpenFlow bridge as OpenFlow ports. Since our router is embedded hardware with limited computation power and cpu cores, the timely availability of CPU processing cycle is essential for seamless performance. Hence, in our testbed, we explicitly define CPU cores for the OpenFlow process by using Linux *Taskset* functionality. It is worth to note that our proposed experiential framework is relying on OpenFlow/SDN programmable routing table to implement the RL routing policy. Therefore, there will be additional computation costs with the trade-off for fully programmable network stacks.



- Vns1: 172.16.1.1
- Veth1: 172.16.1.2
- Eth0: 172.16.1.200
- Eth1: 172.16.1.100
- Br0: 172.16.1.10

Figure 5.4: Federated Computation - Namespace Isolation

5.4.3 Federated Computing Subsystem Implementation

A federated computing system is deployed in Nvidia Jetson Xavier node with has 16GB combined RAM for GPU and CPU. Jetson nodes come with Ubuntu 20.04 operating systems and TensorFlow packages as part of the hardware. In our testbed, each jetson node can host different number of FL worker nodes. Since the primary goal of our experiment is to study the impact of wireless networking on FL, we enabled isolation only at the network level using network namespace. The main

Table 5.1: FL Hyperarameters

Parameter	FEMNIST CNN	CIFAR-10 MobileNet
Number of global rounds	30	70-80
Number of local iterator	10	10
Batch size	32	100
learning rate	0.01	0.1
Model size	5.8 Mbytes	7 Mbytes

advantage of such approach is that, within the single hardware we can deploy multiple workers with isolated TCP/IP layer. Figure 5.4 shows the schematic of the namespace virtualizated network for worker on each jetson node. On each jetson node, we create a virtual bridge using Linux brigde-utils and then we create namespaces for each worker. Following that, the interfaces from the namespace is added to the newly created bridge using virtual ethernet pairing. At this point, all workers within each jetson node can communicate independently. To facilitate external connectivity, nodes master ethernet interface is also added to the same bridge. Finally, each worker can be launched from their respective namespace by executing the API scripts.

5.5 Experimental Evaluation

We conduct extensive experiments to evaluate the effectiveness of our proposed networked-accelerated FL system on our physical testbed. We will compare the performance of our proposed RL-based federated networking with widely-adopted production-grade wireless networking protocol BATMAN-ADV [38] under a variety of settings by varying the number of workers, the percentage of stragglers, and worker location distributions.

5.5.1 Experiment Setup

Model and Dataset: Our experiments consider image classification tasks on FEMINIST and CIFAR-10 datasets. Two different models are used in the training experimentsâa shallow two layers of CNN model and MobileNet, whose weights are updated using federated learning.

- FEMNIST CNN: We first use FEMNIST, the federated version of MNIST [39] on the LEAF[40] character recognition task, where LEAF is a benchmarking framework for federated learning. FEMNIST consists of handwritten digits (10), uppercase (26), and lowercase (26) letters leading to a total of 62 classes with each image having 28×28 pixels. The whole dataset is partitioned into 3550 data portions/users with Non-IID data distribution. In our experiments, we sub-sampled the dataset by 0.02%, yielding 71 users.

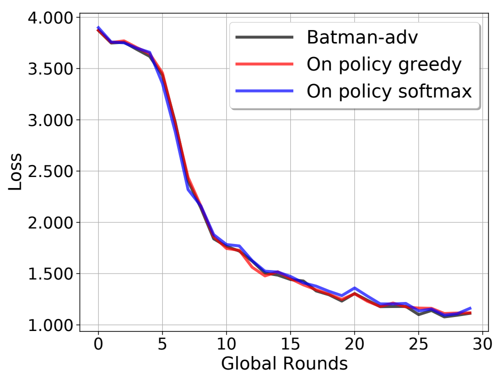
  Initially, we evenly distribute these users among three edge routers R9, R10, and R2 as shown in Figure 5.3. For each router, we assign three active workers with each active worker consisting of 7–8 users, and one of the 7–8 users will become the active worker at each global round/epoch of federated training. We employ a convolutional neural Network (CNN) model during testing. The CNN model has two convolution layers, with 32 and 64 filters respectively. Each convolutional layer was followed by a 2x2 max pooling layer. The convolutions were followed by a fully connected layer with 128 units with ReLU activation. A final fully connected layer with softmax activation was used as the final output layer. The model has a size of 5.8 Mbytes.

- CIFAR-10 MobileNet: To demonstrate the feasibility of a real-world scenario, we introduce the CIFAR-10 dataset [41] and the MobileNet model [42]. CIFAR-10 has 10 classes, 50,000 training samples, and 10,000 testing samples. We used the Dirichlet distribution Dir(β) to build Non-IID heterogeneous partitions for all workers. The value of beta is 0.5, determines the degree of heterogeneity. To train the CIFAR-10 dataset, we use a MobileNet model, a class of efficient network architectures for low power computing devices such as the Nvidia Jetson Xavier platform. To deploy multiple models in resource-constrained hardware, we reduced the width size of the model to be thinner with a width multiplier (α) of 0.5, and set the input resolution of the network to 224. Our model has a size of 7 Mbytes.

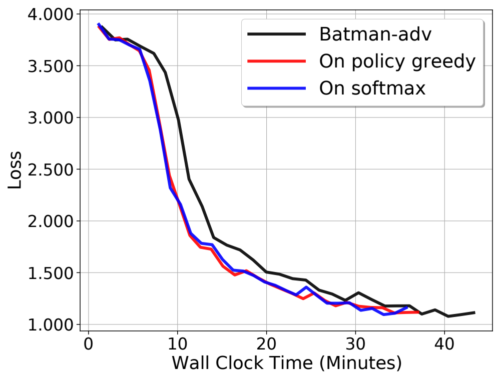
Baseline Federated Networking Protocol: To compare the performance of our proposed RL based routing for federated learning, we chose the state of the art mesh routing protocol, BATMAN-Adv [38] as the baseline. BATMAN-adv is implemented as a layer 2 proactive routing protocol based on distance vector and radio link based reliability as the routing metric. In addition, each node only maintains route information to the next node by which the final destination can be reached. Since each node only requires next hop information towards the destination node, global exchange of routing information is not necessary. From the aforementioned operation of BATMAN-adv, we identified it as the best candidate for comparison as the operation of our RL routing scheme also utilizes only next hop nodes information. Moreover, to the best our knowledge, BATMAN-Adv is the only multi-radio mesh routing protocol that works out of the box on Linux systems as it is embedded within the Linux kernel for optimized operation. During the experiments, we directly use the production-grade BATMAN-Adv protocol provided by Linux system.

RL-based Federated Networking Protocols: We study two online learning RL-based networking algorithms including on-policy greedy algorithm and on-policy softmax algorithm. As introduced in Section 5.2, on-policy greedy algorithm uses greedy policy for both target policy and behavior policy. For on-policy softmax, both target policy and behavior policy use softmax-greedy policy defined in eq. (**??**).

Hyperparameters: For FL, we use the batch size of 100 and learning rate of 0.1. For MA-RL, we use the learning rate of 0.7 for both RL approaches and temperature τ is set to be 2 for on-policy softmax. We did hyperparameter search for τ and it shows that different values of τ do not lead to significantly different performance.

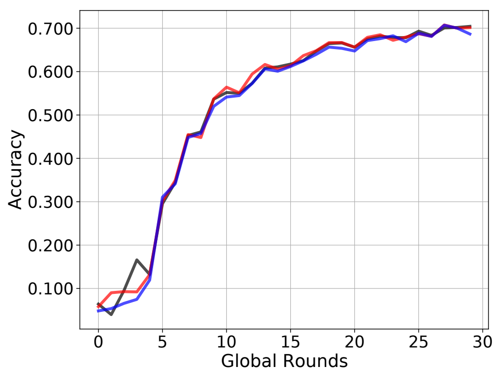


(a) Iteration loss convergence

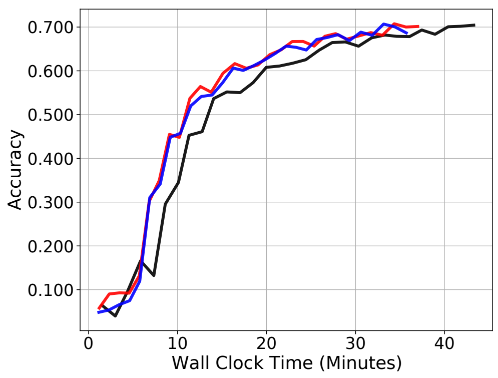


(b) Wall-clock loss convergence

Figure 5.5: Loss convergence comparison of BATMAN-Adv, on-policy greedy, and on-policy softmax with 9 workers



(a) Iteration accuracy convergence



(b) Wall-clock accuracy convergence

Figure 5.6: LEAF and 2-CNN: Validation Accuracy convergence comparison of BATMAN-Adv, On-policy greedy, and On-policy softmax with 9 workers

5.5.2 Main Results

5.5.3 FL iteration and wall-clock convergence

As shown in Figure 5.5 and 5.6, all three federated networking protocols lead to the same iteration convergence performance in the sense that they achieve the same loss or validation accuracy after running the same number of epochs. This is as expected because they use the same underlying federated training algorithm. However, RL-based federated networking protocols can achieve much better wall-clock convergence performance, compared with the baseline protocol. This is because RL algorithms can minimize the per-epoch duration by learning the delay-minimum forwarding paths for

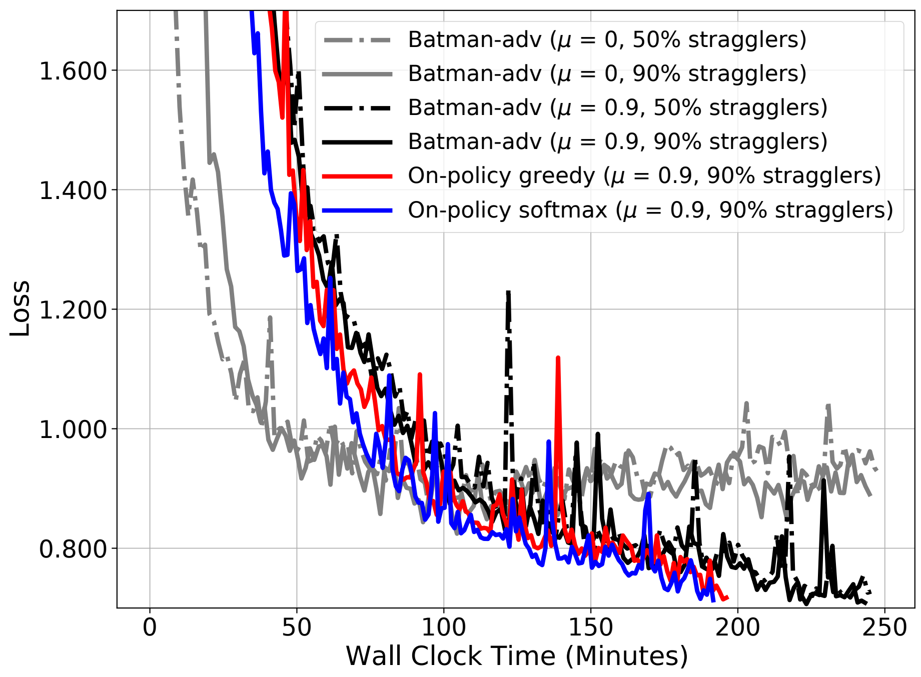


Figure 5.7: LEAF and 2-CNN: Loss Convergence Time after 170 global rounds under different routing protocols

model exchange between the server and workers.

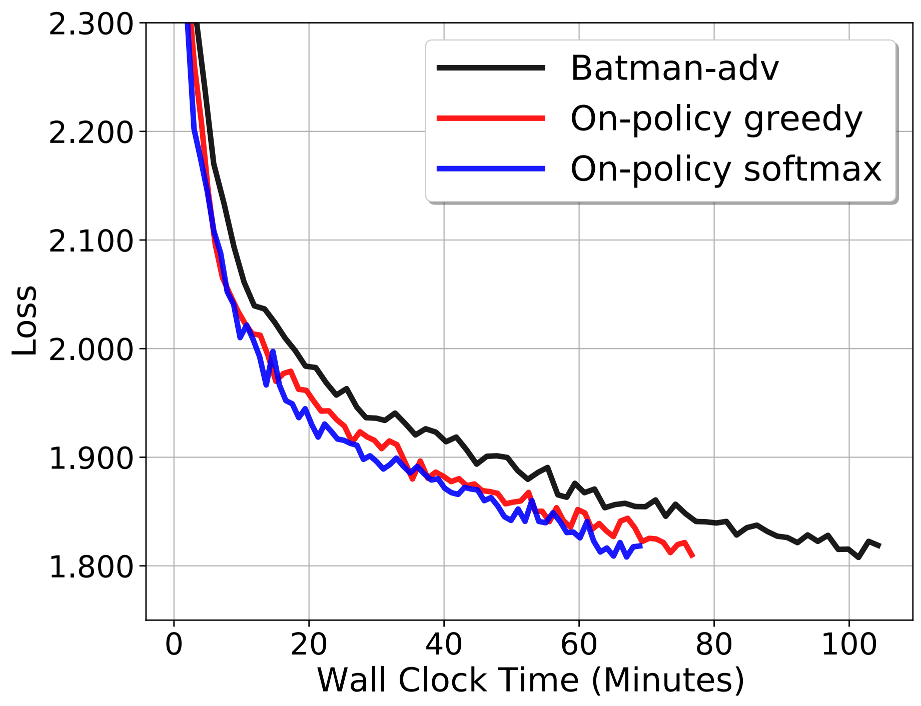


Figure 5.8: CIFAR-10 and MobileNet: Loss Convergence Time after 70 global rounds under different routing protocols

5.5.4 Results of Loss Convergence on CIFAR-10 and MobileNet

Figure 5.8 presents the performance comparison of CIFAR-10 and MobileNet with different routing algorithms in terms of loss convergence and wall clock convergence time. The results become more promising with a larger model size, which lead to higher FL traffic in the network. Both RL routing solutions reach the same loss convergence by around 70 and 79 minutes, respectively, approximately 35 minutes faster than the BATMAN-Adv baseline routing, which takes almost 110 minutes to achieve the same loss convergence.

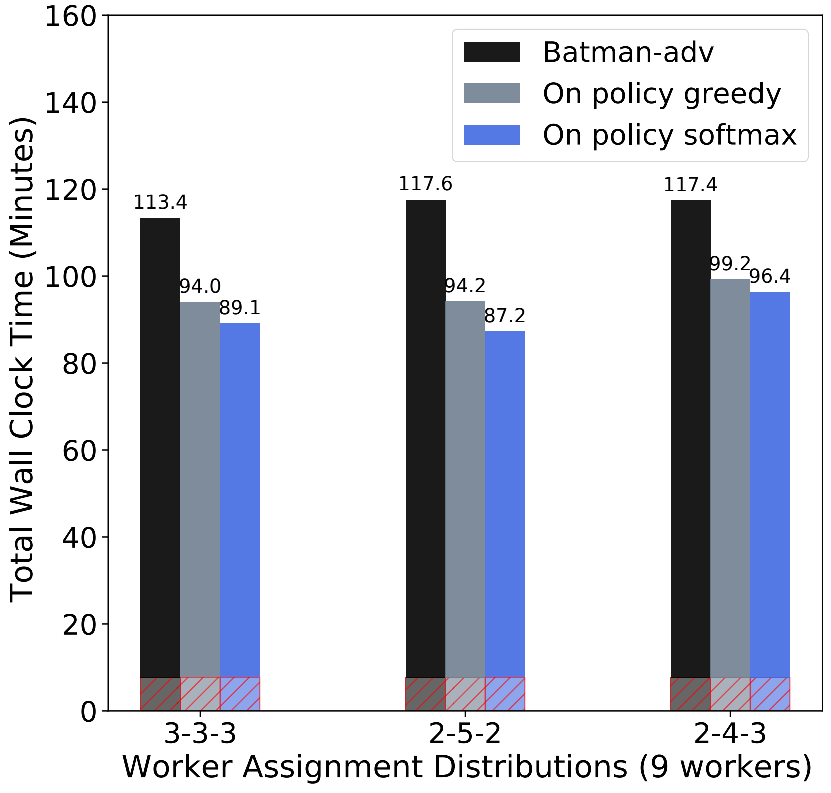


Figure 5.9: LEAF and 2-CNN: Total convergence time comparison of Batman-adv routing (black), On-policy greedy (grey), On-policy softmax (light blue), and Computation time (red hatched) under different worker location distributions after 80 global rounds.

5.5.5 Impact of worker location distribution

In Fig 5.9, we investigate how worker location distribution affects the FL convergence performance. We study the total FL convergence time, FL computing time, and FL networking time by varying the number of workers that are connected to the three edge routers (R9, R10, R2). We study three node distributions (3-3-3, 2-5-2, 2-4-3) with a total number of 9 workers. It is evident that the RL-based federated networking can consistently outperform the baseline networking protocol under different node distributions and achieve up to 25% convergence speedup, compared with the baseline. Moreover, when the network becomes congested, RL-based federated networking protocols lead to a higher performance gain because they can learn to maximize the network resource utilization to better distribute FL flows among all available forwarding paths. This advantage is shown under 2-5-2 worker distribution, where the router R10 needs to serve 5 workers, which induces higher FL traffic volume and a higher level of network congestion around router R10. In this case, on-policy softmax policy leads to the 25% speedup, which is the maximum one among the three node distributions. Regarding RL-based approaches, on-policy softmax outperforms on-policy greedy for all three cases. This is due to the fact that on-policy softmax can proportionally distribute the traffic flows among the available forwarding paths according to the E2E delay of each path. Such approach could be more effective to distribute the traffic loads. In addition, it is observed that the majority of total run time came from communication time while the computation time (around 8 minutes) only contributes to a small portion of the total training time. Therefore, optimizing the federated networking performance is very beneficial to accelerate FL convergence in multi-hop edge computing networks.

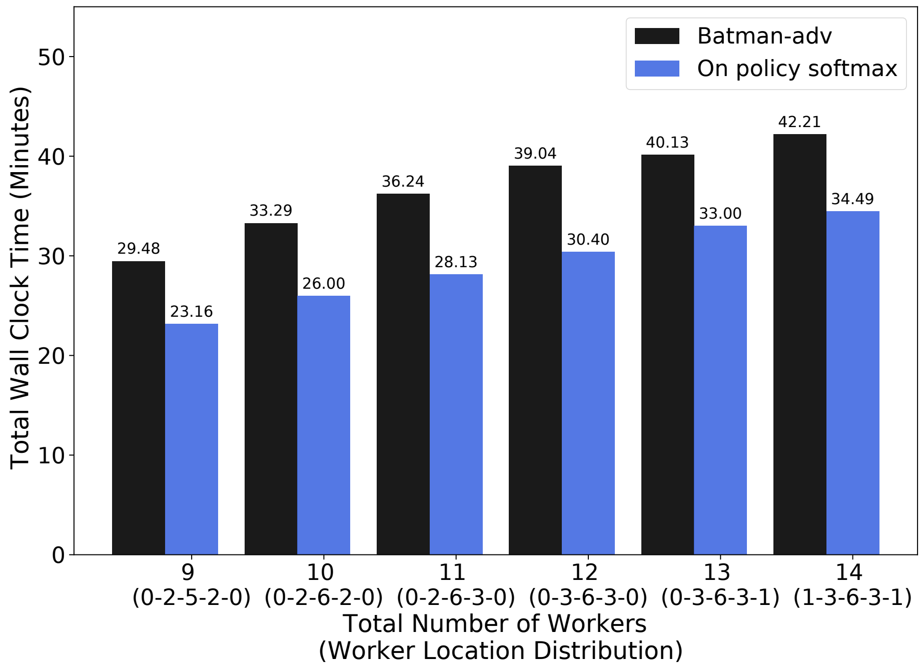


Figure 5.10: LEAF and 2-CNN: Total convergence time comparison of Batman-adv routing (black), On-policy softmax (light blue), by varying total number of workers and location after 20 global rounds.

5.5.6 Scalability Analysis

In Fig. 5.10, we evaluate the FL convergence time by varying the number of workers attached to five edge routers (R9, R10, R2, R3, R8) with a total number of workers (9 , 10, 11, 12, 13, 14). As the number of workers increases, the convergence time of both RL-based approach and baseline protocol increases. This is because as more workers participate, the total FL traffic volume injected into the network increases and gradually approach the maximum network capacity. This leads to prolonged E2E delay and model training time. However, as shown in Fig. 5.10, RL-based approach keeps outperforming the baseline scheme consistently and reduce the total time by 23%. That is, it is able to learn the delay-minimum forwarding paths even if the network becomes congested.

We only experimented with 6 and 9 workers for CIFAR10 and MobileNet because of resource-constrained hardware, as the Nividia Jetson node can only support up

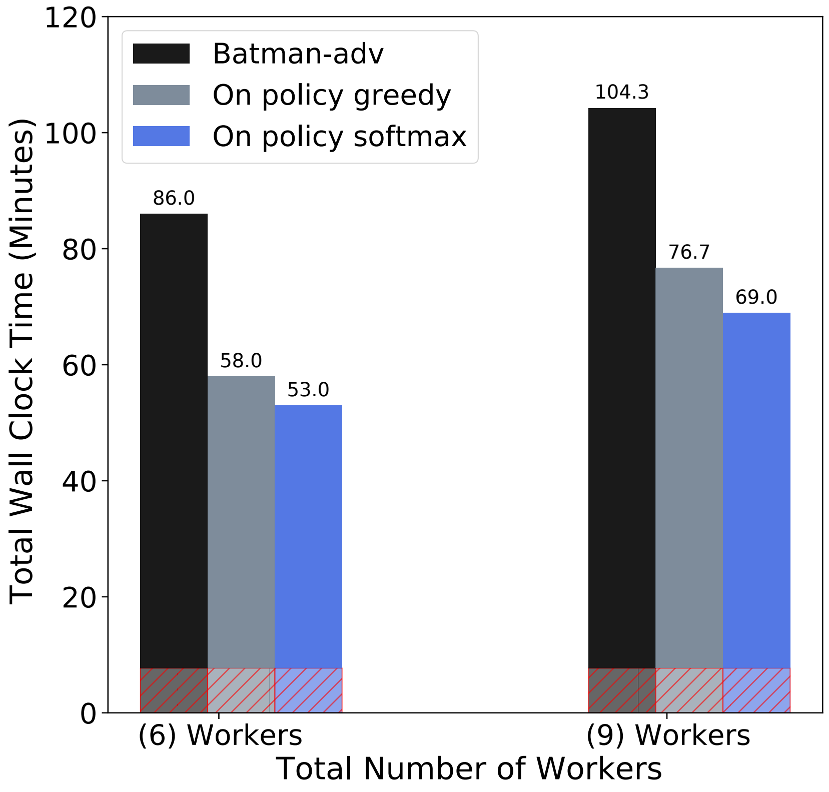


Figure 5.11: CIFAR-10 and MobileNet: Total convergence time comparison of Batman-adv routing (black), On-policy softmax (light blue), by varying total number of workers after 70 global rounds.

to 3 workers per device. As shown in Fig. 5.11, we observed the similar trend as the number of workers increases, the convergence time of all routing algorithms also increases. However, the RL-based method continues to outpace the baseline scheme, which resulted in a 30% reduction in overall time, while achieving the same level of loss and a final accuracy of 68%.

5.5.7 Conclusion

In this chapter, we present network-accelerated FL over wireless edge by optimizing the multi-hop federated networking performance. We first formulate the FL convergence optimization problem as a Markov decision process (MDP). To solve such

MDP, we propose the multi-agent reinforcement learning (MA-RL) algorithm along with loop-free action space refining schemes so that the delay-minimum forwarding paths are learned to minimize the model exchange latency between edge workers and the aggregator. To fast prototype, deploy, and evaluate our proposed FL solutions, we develop EdgeML, which is the first experimental framework in the literature for FL over multi-hop wireless edge computing networks. Moreover, we deploy and implement a physical experimental testbed on the top of the widely adopted Linux wireless routers and ML computing nodes. Such testbed can provide valuable insights into the practical performance of FL in the field. Finally, our experimentation results show that our RL-empowered network-accelerated FL system can significantly improve FL convergence speed, compared to the FL systems enabled by the production-grade commercially-available wireless networking protocol, BATMAN-Adv

CHAPTER 6: SCALABLE AND ROBUST AIOT VIA DECENTRALIZED FEDERATED LEARNING

6.1 Overview

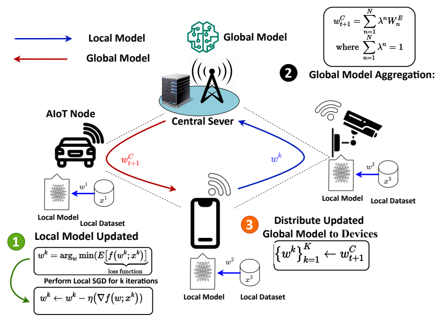
The overall objective of this chapter is to present a robust Decentralized Federated Learning (DFL) solution that leverages single-hop connections to significantly enhance network performance within the context of Artificial Intelligence of Things (AIoT). In this chapter, we delve into the challenges posed by existing Federated Learning (FL) approaches and introduce a novel DFL framework designed to overcome these challenges.

Our primary focus is to revolutionize the landscape of FL within AIoT by addressing the limitations of centralized approaches. To achieve this, we develop a DFL framework that capitalizes on single-hop connections, aiming to optimize network efficiency and scalability while preserving data privacy.

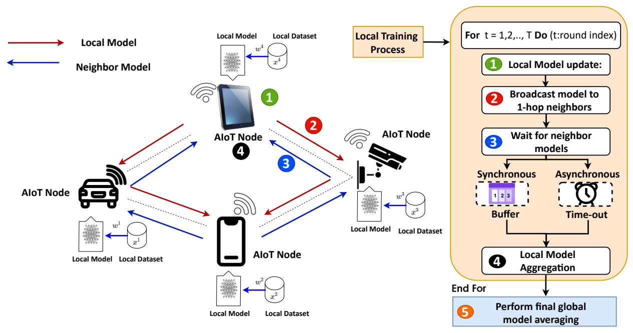
Recognizing that the central model aggregation server in traditional FL setups can lead to communication bottlenecks, we propose a solution that is both robust and decentralized. This DFL framework offers the flexibility of operating in both synchronous (Sync-DFL) and asynchronous (Async-DFL) modes, empowering AIoT systems to adapt to varying conditions and device capabilities.

Async-DFL, in particular, emerges as a pioneering approach within the literature. It serves as a fully asynchronous FL framework that eliminates worker waiting, a critical advancement for AIoT environments marked by heterogeneous devices with varying computing and networking speeds.

Throughout this chapter, we not only present the theoretical underpinnings of our DFL framework but also provide insights into its practical implementation, deployment, and experimentation. Our findings from simulations and real-world testbeds underscore the transformative potential of Async-DFL. It stands out by accelerating model convergence twice as fast as the conventional centralized FL, all while maintaining convergence accuracy and effectively mitigating the impact of stragglers.


Local Model
Global Model
Global Model
$w_{t+1}^{C} = \sum_{n=1}^{N} \lambda^n W_n^E$ where $\sum_{n=1}^{N} \lambda^n = 1$
2 Global Model Aggregation:
AIoT Node
Central Sever
w_{t+1}^{C}
w^k
w^1 x^1
Local Model
Local Dataset
w^3 x^3
Local Model
Local Dataset
1 Local Model Updated
$w^k = \arg_w \min(E[\underbrace{f(w^k; x^k)}_{\text{loss function}}])$
Perform Local SGD for k iterations
$w^k \leftarrow w^k - \eta(\nabla f(w; x^k))$
3 Distribute Updated Global Model to Devices
$\{w^k\}_{k=1}^{K} \leftarrow w_{t+1}^{C}$
w^2 x^2
Local Model
Local Dataset

(a) Central-server Federated Learning



(b) Decentralized Federated Learning

Figure 6.1: Federated Learning (FL) in IoT Network. (a) Classical centralized FL. (b) Decentralized FL

6.2 Centralized Federated Learning

Centralized Federated Learning (CFL) addresses a fundamental challenge in machine learning: how to efficiently harness data and computational power for rapid model training. Unlike traditional centralized approaches that heavily burden a single powerful machine to collect data from IoT edge devices, CFL introduces a more efficient and privacy-conscious paradigm.

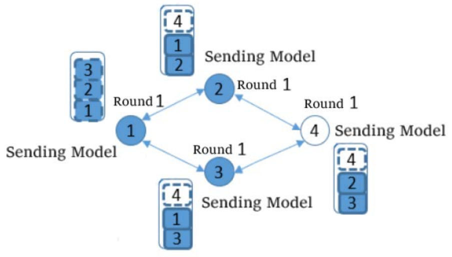
In CFL, data on IoT devices remains untouched by external sources. Instead, it employs two crucial components: workers and aggregators. Workers represent the edge devices responsible for training local models using their data, while aggregators, often central servers, manage the coordination.

In the training process, a common neural network model is initially distributed to all worker devices. These workers, each starting with an identical untrained model, iteratively update their local models through a combination of local Stochastic Gradient Descent (SGD) iterations and global model averaging. Local SGD iterations aim to minimize the training loss by conducting mini-batch SGD updates, while global model averaging occurs when worker devices send their local models to aggregators for consolidation. The updated global model is then broadcast back to the workers, and the training cycle continues.

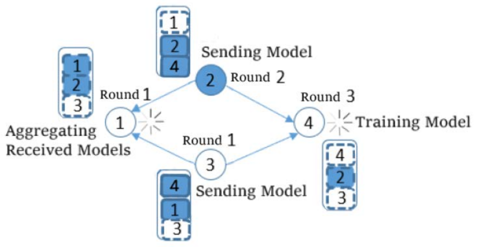
This approach significantly reduces the computational burden on a central server and mitigates privacy concerns associated with data exposure. Moreover, centralized federated learning algorithms, like FedAvg and its variants, are specifically designed to tackle distributed training challenges in non-convex optimization problems with training loss minimization as the primary objective. This paradigm offers a powerful solution for collaborative machine learning without compromising data privacy or computational efficiency.

6.2.1 Challenges of CFL

While CFL represents a significant departure from the traditional centralized paradigm towards distributed machine learning, it's important to recognize that it still faces certain issues due to its centralized model aggregation approach. Similar to centralized methods, CFL encounters an inherent communication bottleneck within the system. Despite distributing the computational load for training, network traffic remains centralized at a single point, placing significant strain on the central server's throughput and underutilizing available network bandwidth. Additionally, CFL often grapples with multi-hop communication, which further hampers its performance. Consequently, CFL tends to favor network topologies like star networks, where all workers can send their models in a single hop. However, this preference for specific topologies limits flexibility and scalability, especially when compared to the more practical IoT mesh networks commonly found in real-world applications. These challenges can hinder CFL's development and adoption in various scenarios.



(a) Synchronous DFL



(b) Asynchronous DFL

Figure 6.2: In Synchronous DFL, workers keep time by waiting for the slowest workers to fill their buffers before aggregating. In Asynchronous DFL, the faster workers aggregate immediately no matter how full the buffer is, which allows them to avoid waiting but necessitates program robustness to handle communication at any time.

6.3 Decentralized Federated Learning

6.3.1 Generic Decentralized FL Framework

In our Decentralized Federated Learning (DFL) approach, each worker communicates with its one-hop neighbors and shares the aggregation responsibility, as illustrated in Fig. 6.2. Instead of sending their local model to a central server, workers exchange copies of their models with nearby peers. This means that each worker is an aggregator for their local model and those of their neighbors, without handling the complete aggregation burden at once.

Our DFL leverages fast single-hop wireless connections, unlike the multi-hop nature of centralized federated learning. This allows workers to operate independently, without requiring knowledge of the broader network or central server supervision. This independence grants our framework flexibility, robustness, and scalability.

Fig. 6.2 depicts our versatile DFL framework, which can be configured for either synchronous DFL (Sync-DFL) or asynchronous DFL (Async-DFL) modes. Both modes share a similar local training procedure involving local model updates, model broadcasting among one-hop neighbors, model reception, and local model aggregation. This process repeats until reaching a predefined number of training epochs. After local training, each worker sends its local model to the global aggregation node, which performs the final global model aggregation, yielding the inference model for IoT devices. The global aggregation node can be a gateway device connecting IoT networks to the Internet or a randomly selected IoT device.

Sync-DFL and Async-DFL differ in their local model aggregation triggers and last-round global aggregation strategies. In Sync-DFL, aggregation begins when the model buffer is full, synchronizing all workers' training rounds. In Async-DFL, aggregation is time-triggered, with workers maintaining their own training round counters, allowing for asynchrony.

A key feature of our framework is the model buffer, serving dual purposes. First, it facilitates implicit inter-device synchronization in Sync-DFL. Second, it offers a flexible tradeoff between training convergence speed and model quality. Each model represents a unique data subset, and aggregating more models improves overall training. Sync-DFL maximizes quality but requires some workers to wait, while Async-DFL adjusts the tradeoff based on buffer arrivals before aggregation.

Synchronous DFL (Sync-DFL): With synchronous DFL, the workers follow two converse rules. First, they only aggregate once they have a full buffer, and second, they will always wait for their stragglers before beginning their own next round of training. Essentially, the worker's buffer functions as a clock to keep time with the rest of the network, as illustrated in Fig. 6.2. Each worker works in the same global round and generally begins their model broadcasting at a similar time. Once each worker finishes their broadcasting, they begin checking to see whether their buffer is full. If the buffer is not full, the worker knows that it is outpacing the network and will wait until the buffer is full to proceed. Therefore, the workers aggregate within a similar time, emptying their respective buffers and then moving on to the next round's training.

Asynchronous DFL (Async-DFL): Async-DFL takes a divergent approach compared to Sync-DFL. Instead of having workers wait to ensure an ideal or near-ideal aggregation, it allows workers to proceed regardless of the buffer's status. Workers will always have at least their own model to aggregate, even if the buffer is otherwise empty. Optionally, they may include an adjustable timer to allow other workers to catch up. Over time, faster workers gradually outpace their slower counterparts, resulting in workers sharing models with their neighbors regardless of their current training epoch step, as shown in Fig. 6.2. This leads to workers training on different epochs, with the stragglers eventually completing their training independently, without further broadcasts from their faster peers.

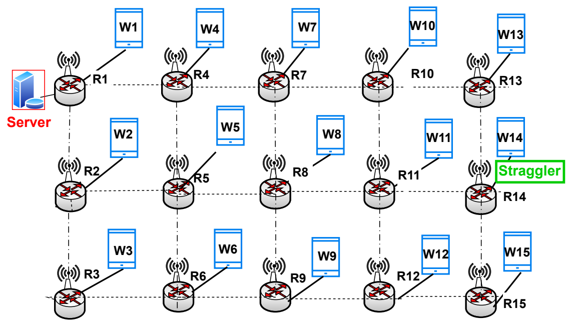
The Async-DFL approach enables faster workers to progress without waiting for stragglers. Moreover, this asynchronous design allows concurrent operation of computation and networking, with model aggregation occurring alongside model transmission. Although reduced model sharing may make Async-DFL more susceptible to overfitting, especially with smaller buffer sizes, practical scenarios often involve workers aggregating with partially filled buffers, countering overfitting as convergence progresses. Given that worker communication tends to be the limiting factor for epoch speed, avoiding unnecessary delays or redundant communication is crucial for optimizing convergence speed.

6.4 Simulation and Physical Testbed Evaluation of FL Management Modes

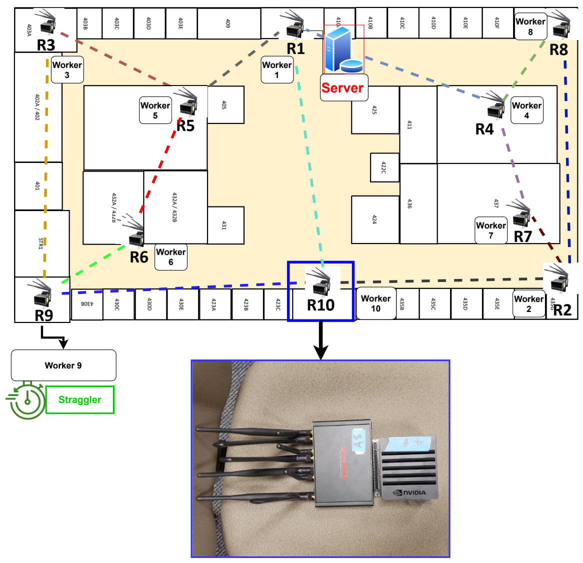
In our set of experiments, we investigate and study how Sync-DFL and Async-DFL efficiently improve the convergence speed and performance of FL in both a simulated environment and a live network testbed. Our results are based on a set of experiments in which we first ran a model training on the network in the centralized as well as decentralized, synchronous and asynchronous cases.

6.4.1 Experiment Setup

Heterogeneous Data Settings and Models: We utilized two benchmark datasets for federated learning: FEMNIST, which consists of 62 classes, and CIFAR-10, com-



(a) Simulated Network Topology



(b) Testbed topology

Figure 6.3: IoT Multi-hop Network topologies

prising 10 classes. Our data partition strategy for both datasets followed the non-IID (Non-Independently and Identically Distributed) settings, utilizing a realistic partitioning method from LEAF. In the case of CIFAR-10, we introduced data heterogeneity by employing the Dirichlet distribution Dir(β) to create uneven data partitions for all worker nodes. The level of data heterogeneity was controlled by the value of beta, which was set to 0.5 in our experiments. To simulate computation and communication heterogeneity, we introduced a training delay of 40 seconds for straggler nodes.

Our convolutional neural network (CNN) architecture consists of two convolutional layers, followed by a fully connected layer, and utilizes local Stochastic Gradient

Descent (SGD). The first convolutional layer has 32 filters, while the second has 64 filters, both connected via a 2x2 max-pooling layer. The fully connected layer comprises 128 units with Rectified Linear Unit (ReLU) activation and feeds into the final layer, which is fully connected with softmax activation. The size of the final layer's model is approximately 5.8 megabytes, representing a low communication overhead. Additionally, we conducted evaluations using the CIFAR-10 dataset with a deep neural network model called MobileNet [42], which has a model size of around 15 megabytes, resulting in significantly higher communication traffic.

s**CFL Baseline and DFL Implementations:** We employed our custom-designed FedEdge experimental framework [43] to implement the CFL, Sync-DFL, and Async-DFL solutions. As a CFL baseline, we utilized the widely-adopted FedAvg [44] algorithm. FedEdge [12, 45, 43] serves as a software-defined experimental platform, encompassing a federated computing module powered by TensorFlow and a software-defined wireless multi-hop networking module based on Mininet-wifi [46].

FedEdge stands out as the pioneering experimental framework in the literature tailored for Federated Learning (FL) over multi-hop wireless edge computing networks, such as IoT networks. This framework enabled us to rapidly prototype, deploy, and evaluate novel FL algorithms, alongside machine learning-based system optimization techniques, both in simulated environments and with real wireless devices. While the experimental framework FedML [44] has been employed to explore Sync-DFL possibilities, it primarily serves as a benchmarking tool rather than a platform for live-running experiments. It lacks the setup necessary for results in real-world applications and does not support the versatile Decentralized FedML framework, which encompasses both Sync-DFL and Async-DFL capabilities.

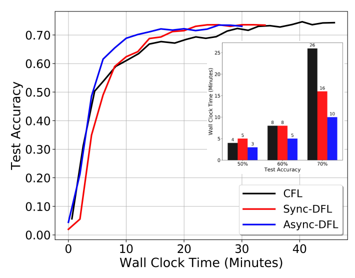
Physical Testbed Setup: We utilized a physical multi-hop wireless edge computing network as our IoT network testbed [43]. This testbed comprised ten wireless edge computing nodes, each equipped with a wireless embedded router for communication and an Nvidia Xavier node for computation. The wireless routers featured three wireless interface cards to enable multi-radio functionality. Each mesh router operated in Mesh Point (MP) mode, with fixed 2.4 and 5 GHz channels, a 20 MHz channel width in 802.11ac mode, and a transmit power of 15 dBm. To establish distributed multi-hop routing, we employed the state-of-the-art Batman-adv protocol, which facilitated server-to-worker communication in the Sync-DFL case. Specifically, we connected a server to R1, and worker 9 was designated as a straggler with a 40-second delay.

Each experiment spanned 30 global epochs and comprised 5 local rounds. We used a batch size of 10 and a learning rate of 0.002 for both the CNN (2Conv + 2FC) and MobileNet models.

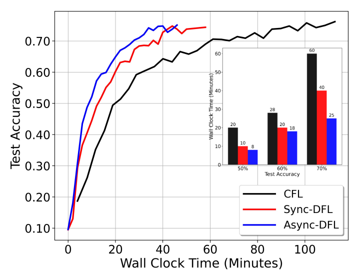
Network Simulation Setup:

In the initial phase of our evaluation, we assessed the performance of our FL solutions within a FedEdge simulator [45]. To validate and analyze the generic decentralized FL framework's performance, we established a wireless multi-hop IoT network with a 5x3 grid topology, featuring 15 workers, as depicted in Fig 4.2. The link bandwidth was set at 24 Mbps, with each worker's neighbors being the workers located one hop away in cardinal directions. In the context of centralized FL, worker 1 served as the network's server node while simultaneously participating as a worker.

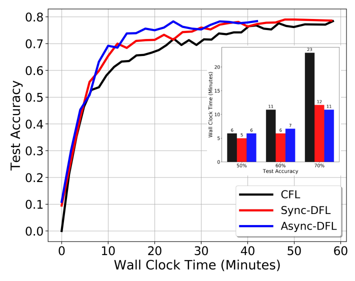
For our straggler scenario experiments involving the CIFAR10 dataset, we designated worker 14 (five network hops away from worker 1) as the straggler by introducing a 40-second delay for each training epoch. This delay simulated either the limited computing power of IoT devices or a drop in CPU performance. Each run encompassed 5 local rounds per global epoch, with a batch size of 10, a learning rate of 0.002, and 50 global rounds.



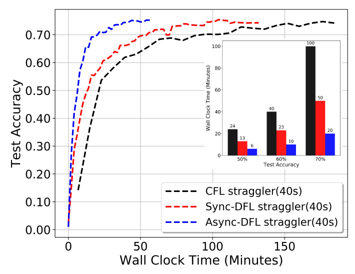
(a) Testbed: LEAF (CNN) (No straggler)



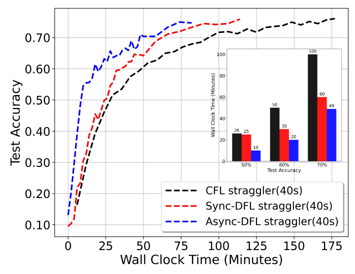
(b) Testbed: CIFAR-10 (MobileNet)
(No straggler)



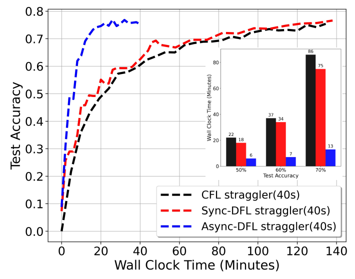
(c) Simulation: CIFAR-10 (MobileNet)
(No straggler)



(d) Testbed: LEAF (CNN) (with straggler)



(e) Testbed: CIFAR-10 (MobileNet)
(with straggler)



(f) Simulation: CIFAR-10 (MobileNet)
(with straggler)

Figure 6.4: Comparison of Centralized and Decentralized Federate Learning performance in Simulation and Testbed environments (LEAF and CIFAR-10), each with cases for having no straggler or having a straggler with a 40s per local training round.

6.4.2 Performance and Communication Comparisons

To assess model convergence, we examined the learning curves and wall-clock time required to reach specific testing accuracy thresholds (0.5, 0.6, 0.7, and 0.75) for all methods, where all methods ultimately achieve the same maximum accuracy of 0.75 (Fig. 6.4).

Model Convergence without Stragglers: Fig. 6.4 (a), 6.4 (b), and 6.4 (c) depict the performance comparison of FL in terms of accuracy and wall-clock convergence time for CFL, Sync-DFL, and Async-DFL in scenarios without stragglers. We conducted two sets of experiments in the testbed:

For the LEAF dataset with a lightweight CNN model, both Sync-DFL and CFL achieved similar accuracy levels within comparable time frames (Fig. 6.4 (a)). However, when using the MobileNet model, which is three times larger than the 4-layer CNN, CFL exhibited a longer convergence time.

In the simulated scenario with 15 workers using the CIFAR-10 dataset and the MobileNet model (Fig. 6.4 (c)), both Sync-DFL and Async-DFL reached nearly 0.7 test accuracy in approximately 11 minutes, surpassing CFL, which took around 23 minutes to achieve the same accuracy. All methods ultimately reached a maximum accuracy of approximately 0.75.

CFL's extended convergence time can be attributed to the need for each worker to transmit their updated local model to the central server through lengthy and random multi-hop communication links within the IoT network. The server also has to wait for all worker models before proceeding to the next training round. This congestion of network traffic around the server at router 1 contributes to the delay.

Sync-DFL and Async-DFL, in contrast, only require workers to send their updated models to their single-hop neighbors, making more efficient use of network bandwidth and distributing traffic load more evenly. This alleviates congestion around bottleneck links and significantly reduces model training time without compromising convergence accuracy.

Model Convergence with Stragglers: When introducing a straggler into the network, all FL solutions experience increased model convergence times, with Async-DFL being the least affected, as shown in Fig. 6.4 (d), 6.4 (e), and 6.4 (f). Sync-DFL's results more closely resemble those of CFL. This is because both Sync-DFL and CFL rely on synchronized local model training across all workers in the network, and thus their convergence times are directly impacted by the presence of stragglers. However, Sync-DFL still achieves faster convergence compared to CFL due to its confinement of model exchanges to 1-hop neighbors, while CFL continues to face communication bottlenecks around the central server. This effect is expected to persist as the network scales.

In the simulated test (Fig. 6.4 (f)), Async-DFL achieves a 0.70 accuracy in about 13 minutes, while both CFL and Sync-DFL reach the same accuracy in approximately 86 and 75 minutes, respectively. This represents a roughly six-fold increase in convergence time compared to Async-DFL. The significant difference in tolerance is visually evident in Fig. 6.4 (f).

In the testbed, LEAF was employed to illustrate relatively light job loads in a live environment, while CIFAR-10 posed a more demanding training task. In LEAF with a straggler, Async-DFL achieves over a 2x and 3x convergence speedup compared to Sync-DFL and CFL, respectively, when all methods reach the same 70% and 75% accuracies. In the CIFAR-10 testbed scenario, Async-DFL requires 1.5x less convergence time than CFL and 3x less time than Sync-DFL to reach the same maximum testing accuracy (75%). In a larger simulated network, Async-DFL achieves a 7x speedup compared to both CFL and Sync-DFL.

6.4.3 Conclusion

Navigating the escalating demands of AIoT systems, the chapter delves into and assesses the effectiveness of two distinct decentralized federated learning models, constructing a versatile and composable decentralized FL framework. The experimental outcomes underscore the DFL's eminent convergence performance in multi-hop IoT networks as compared to the conventional CFL. Furthermore, initial results indicate that Async-DFL not only accelerates model convergence speed but also exhibits notable resilience and robustness in heterogeneous IoT environments, even amidst the inevitable presence of stragglers. Thus, Async-DFL unfolds as a promising contender, illustrating substantial potential to achieve an optimal balance between model convergence speed and model quality in subsequent large-scale AIoT networks, meriting additional investigation.

CHAPTER 7: CONCLUSIONS

7.1 Summary

In reflection, the wave of the AI-digital era has bestowed upon us a cascade of data usage extending from bustling data centers to intricate IoT devices, necessitating the emergence of networks that can adeptly marry AI capabilities with advanced network solutions. The challenges that sprouted alongside, including the convoluted nature of network layer protocols, the palpable discrepancies noticed between simulated AI models and their real-world counterparts, and the burgeoning need for decentralized AI training owing to network distribution, presented notable roadblocks in fully realizing the potential of AI-optimized networks.

In our endeavor to navigate through these challenges, we introduced the AI-oriented Network Operating System (AINOS), embedding within it two pivotal sub-platforms: the "Network Gym" and "Federated Computing," each delicately crafted to cater to AI-driven network training and decentralized training methodologies, respectively. AINOS has emerged as a consummate toolkit, enabling rapid prototyping, deployment, and validation of AI-optimized networks, and, crucially, forging a bridge from simulation to tangible real-world deployment.

Through the potent functionalities of AINOS, we prototyped AI-optimized networking solutions, incorporating a secure Reinforcement Learning (RL) strategy for Traffic Engineering (TE) across both the link and network layers. Our implementations, spanning from a scalable RL-based traffic splitting mechanism at the link layer to an online Multi-agent Reinforcement Learning (MA-RL) approach at the network layer, not only discerned optimal paths in real-time for wireless multi-hop networks but also carved out innovations in Network Assisted AI optimization. This reduced Federated Learning training times and spawned a sturdy Decentralized Federated Learning solution, capitalizing on single-hop connections to boost network performance. The outcomes of our research emphatically underscore the potency of AI-enhanced networks, especially in adeptly navigating through network heterogeneity and latency.

As we look forward, the innovations and findings derived from our work with AINOS may pave the way for further exploration and development in the realm of AI-integrated networking, opening up new vistas for research and application in a world that continues to digitally evolve.

Furthermore, as we approach the advent of 6G technology in the near future, one prominent direction involves the integration of computation and communication for the purpose of co-optimization [47, 48, 49, 50, 51].

7.1.1 Future work

Recently, data-driven, reinforcement-learning (RL)-based approaches have shown great potential in other problems of networking research. In RTC-CC area, Pensieve [52] showed that RL-based bitrate adaptation for VoD can reduce frame stalling while improving bandwidth utilization. Moreover, OpenNetLab [53] is a first open platform for offline training of RL-based CC algorithms for RTC. In particular, the simulator in OpenNetLab is tailored for offline training RL-based CC algorithms for RTC by connecting a customized gym and WebRTC with ns-3. The simulator creates real WebRTC sender and receiver instances and performs event driven simulation of network environment with ns-3. With network traces obtained by probing the real network environment with measurement, the simulator schedules ns-3 events to simulate changing network environment (e.g. bandwidth, loss and jitter) as recorded in the trace. However, using simulated network environments only for learning patterns in network conditions after being trained with real network traces has limitations because it can only indirectly optimize QoE by simulating frame statistics for real video traffic applications, which is not realistic in terms of real implementation.

In order to achieve this, we plan to propose the following contributions:

- (1) High fidelity Emulator RL-based CC on FedEdge Emulator [sim-to-real], which can directly optimize QoE by learning from real frame statistics from real video traffic in addition to training with varied network conditions. We expect that this high-fidelity will bridge the gap between the simulated environment, actual RTC, and the video environment.

- (2) To avoid performance fluctuation due to unsafe policies resulting from the trial-and-error nature of the exploration in RL, we plan to apply an imitation learning strategy. In particular, the learning models can be trained in the proposed high-fidelity emulator, and then the trained models will be directly deployed, fine-tuned, and tested in real applications.

- (3) To accelerate the training convergence and generalization from various network environments, we aim to adopt Federated Learning (FL) to learn from massive concurrent video telephony sessions. In this way, the RL model can explore diverse environments to enrich its experience and converge to a universal model with swarm intelligence learned from all users.

REFERENCES

[1] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in neural information processing systems (NIPS)*, pp. 671–678, 1994.

[2] J. Zhang and D. Tao, "Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things," *CoRR*, vol. abs/2011.08612, 2020. [Online]. Available: `https://arxiv.org/abs/2011.08612`.

[3] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over mpls," tech. rep., 1999.

[4] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional ip routing protocols," *IEEE communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.

[5] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings of IEEE INFOCOM*, pp. 2211–2219, IEEE, 2013.

[6] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. KoneÄnÃœ, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019. cite arxiv:1902.01046.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[8] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[9] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

[10] P. Pinyoanuntapong, M. Lee, and P. Wang, "Delay-optimal traffic engineering through multi-agent reinforcement learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 435–442, IEEE, 2019.

[11] P. Pinyoanuntapong, M. Lee, and P. Wang, "Distributed multi-hop traffic engineering via stochastic policy gradient reinforcement learning," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[12] P. Pinyoanuntapong, P. Janakaraj, P. Wang, M. Lee, and C. Chen, "Fedair: Towards multi-hop federated learning over-the-air," in *Proceedings of IEEE SPAWC 2020*, 2020.

[13] P. Pinyoanuntapong, P. Janakaraj, R. Balakrishnan, M. Lee, C. Chen, and P. Wang, "Edgeml: Towards network-accelerated federated learning over wireless edge," *Computer Networks*, vol. 219, p. 109396, 2022.

[14] P. Pinyoanuntapong, W. H. Huff, M. Lee, C. Chen, and P. Wang, "Toward scalable and robust aiot via decentralized federated learning," *IEEE Internet of Things Magazine*, vol. 5, no. 1, pp. 30–35, 2022.

[15] W. H. Huff, pinyarash pinyoanuntapong, R. Balakrishnan, H. Feng, M. Lee, P. Wang, and C. Chen, "DHA-FL: Enabling efficient and effective AIot via decentralized hierarchical asynchronous federated learning," in *MLSys 2023 Workshop on Resource-Constrained Learning in Wireless Networks*, 2023.

[16] Intellabs, "Training agents in networkgym," Year.

[17] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and R. Voutilainen, "Stable-Baselines3: Reinforcement Learning in PyTorch," 2020.

[18] B. I. A. M. Lab, "Cleanrl: Reinforcement Learning for Robust and Reproducible Experiments," 2021.

[19] P. Janakaraj, P. Pinyoanuntapong, P. Wang, and M. Lee, "Towards in-band telemetry for self driving wireless networks," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 766–773, 2020.

[20] "Ofsoftswitch13." Available: `https://github.com/CPqD/ofsoftswitch13`.

[21] "Mangodb." Available: `https://www.mongodb.com/`.

[22] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation: Model and experiments," in *Proceedings of the fifth international conference on Autonomous agents*, pp. 616–623, ACM, 2001.

[23] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[24] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling, "Learning to cooperate via policy search," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 489–496, Morgan Kaufmann Publishers Inc., 2000.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[27] H. van Seijen, "Effective multi-step temporal-difference learning for non-linear function approximation," *arXiv preprint arXiv:1608.05151*, 2016.

[28] A. R. Mahmood, H. Yu, and R. S. Sutton, "Multi-step off-policy learning without importance sampling ratios," *arXiv preprint arXiv:1702.03006*, 2017.

[29] K. De Asis and R. S. Sutton, "Per-decision multi-step temporal difference learning with control variates," *Proceedings of the 2018 Conference on Uncertainty in Artificial Intelligence*, 2018.

[30] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected sarsa," in *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pp. 177–184, IEEE, 2009.

[31] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 23, pp. 2613–2621, 2010.

[32] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN'02)*, vol. 2, pp. 1825–1830, IEEE, 2002.

[33] Y. Shilova, M. Kavalerov, and I. Bezukladnikov, "Full echo q-routing with adaptive learning rates: a reinforcement learning approach to network routing," in *2016 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus),*, pp. 341–344, IEEE, 2016.

[34] M. V. Kavalerov, Y. A. Shilova, and I. I. Bezukladnikov, "Preventing instability in full echo q-routing with adaptive learning rates," in *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus),*, pp. 155–159, IEEE, 2017.

[35] M. Kavalerov, Y. Shilova, and Y. Likhacheva, "Adaptive q-routing with random echo and route memory," in *2017 20th Conference of Open Innovations Association (FRUCT)*, pp. 138–145, IEEE, 2017.

[36] S. Lin, P. Wang, I. F. Akyildiz, and L. Min, "Utility-optimal wireless routing in the presence of heavy tails," *IEEE Transactions on Vehicular Technology*, 2018.

[37] P. Pinyoanuntapong, M. Lee, and P. Wang, "Delay-optimal traffic engineering through multi-agent reinforcement learning," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2019.

[38] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using batman," 2008.

[39] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[40] S. Caldas, S. Meher Karthik Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A Benchmark for Federated Settings," *arXiv e-prints*, p. arXiv:1812.01097, Dec. 2018.

[41] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[42] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[43] P. Pinyoanuntapong, P. Janakaraj, R. Balakrishnan, M. Lee, C. Chen, and P. Wang, "Edgeml:towards network-accelerated federated learning over wireless edge," *CoRR*, vol. abs/2111.09410, 2021. [Online]. Available: `http://arxiv.org/abs/2111.09410`.

[44] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, "Fedml: A research library and benchmark for federated machine learning," in *Proceedings of NeurIPS 2020*, 2020.

[45] P. Pinyoanuntapong, T. Pothuneedi, R. Balakrishnan, M. Lee, C. Chen, and P. Wang, "Sim-to-real transfer in multi-agent reinforcement networking for federated edge computing," *CoRR*, vol. abs/2110.08952, 2021.

[46] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and Service Management (CNSM)*, pp. 384–389, 2015.

[47] D. Yang and D. Cheng, "Efficient gpu memory management for nonlinear dnns," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, (New York, NY, USA), p. 185â196, Association for Computing Machinery, 2020.

[48] D. Yang, W. Rang, and D. Cheng, "Mitigating stragglers in the decentralized training on heterogeneous clusters," in *Proceedings of the 21st International Middleware Conference*, Middleware '20, (New York, NY, USA), p. 386â399, Association for Computing Machinery, 2020.

[49] Z. Zhang, D. Yang, Y. Xia, L. Ding, D. Tao, X. Zhou, and D. Cheng, "Mpipemoe: Memory efficient moe for pre-trained models with adaptive pipeline parallelism," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 167–177, 2023.

[50] Y. Xia, Z. Zhang, H. Wang, D. Yang, X. Zhou, and D. Cheng, "Redundancy-free high-performance dynamic gnn training with hierarchical pipeline parallelism,"

in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '23, (New York, NY, USA), p. 17â30, Association for Computing Machinery, 2023.

[51] D. Yang, D. Cheng, W. Rang, and Y. Wang, "Joint optimization of mapreduce scheduling and network policy in hierarchical data centers," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 461–473, 2022.

[52] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 197â210, Association for Computing Machinery, 2017.

[53] J. Eo, Z. Niu, W. Cheng, F. Y. Yan, R. Gao, J. Kardhashi, S. Inglis, M. Revow, B.-G. Chun, P. Cheng, and Y. Xiong, "Opennetlab: Open platform for rl-based congestion control for real-time communications," in *APNet 2022*, July 2022.