TOWARDS AI-EMPOWERED WIRELESS NETWORKS: FROM EDGE TO CORE

by

Prabhu Janakaraj

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

2021

Approved by:

_____
Dr. Pu Wang

_____
Dr. Minwoo Lee

_____
Dr. Dong Dai

_____
Dr. Gabriel Terejanu

_____
Dr. Jiang Xie

ABSTRACT

PRABHU JANAKARAJ. Towards AI-Empowered Wireless Networks: From Edge to Core. (Under the direction of DR. PU WANG)

Wireless multi-hop networks have been widely exploited for deploying cost-efficient network backbones including wireless community mesh networks, high-speed urban networks, global wireless internet infrastructure, battlefield networks, and public safety/disaster rescue networks. Federated learning (FL) is a distributed machine learning technology for next-generation AI systems that can collaboratively improve a shared global model while preserving user privacy and encompassing users at a larger-scale. FL systems are designed to be used on Single-hop wireless networks which consists of edge servers that are connected to the high-speed internet core. Enabling FL over wireless multi-hop networks can democratize AI and make it accessible in a cost-effective manner. However, our preliminary study found that FL over wireless multi-hop networks possess significant challenges due to the presence of multiple noisy interference-rich wireless links which not only slows the learning process due to the underlying communication delay but also leads to nomadic model updates. The inherent bottleneck for FL over wireless multi-hop networks are (1) One-size-for-all model deployment and training, where each edge device process the same number of local iterations for model updates (2) Model-based optimization is not feasible for multi-hop FL, since FL performance metrics cannot be formulated as a closed-form function for network control parameters such as packet forwarding decision and transmission power for each router.

In this thesis, we proposed a novel Artificial Intelligence (AI) empowered wireless network systems that can guarantee stability, high accuracy, and faster convergence speed by taming communication latency, system heterogeneity, and statistical heterogeneity. Towards this goal (1) We have developed a Programmable Wireless Network

Operating System (WINOS), which allows the user to implement AI routing solutions (2) We have developed a novel Hierarchical Synchronous FL system architecture that can maximize communication efficiency in addition to tolerating stragglers for non-blocking executions (3) We have implemented a naturally federated application, Gait based user authentication and recognition mechanism using millimeter wave, which is one of the privacy-preserving biometric authentication mechanism (4) We have developed a novel domain adaptation solutions that helps in applying FL system trained in single domain to different domains that possess spatial variations. Finally, our experimentation result shows that AI enabled wireless networked systems are extremely efficient in handling heterogeneity and communication latency, surpassing the traditional network systems performance.

ACKNOWLEDGEMENTS

Throughout the journey of my PhD program, I received tremendous support and assistance. I extend my gratitude to my advisor, Dr. Pu Wang, who has been a great support with his expertise in interdisciplinary research. His perpetual push empowered me to enhance my competence as a researcher with much-broadened skills.

I am grateful to my doctoral committee members for their worthwhile suggestions in shaping my research. I thank them for their consideration to serve in my committee and their valuable time in evaluating my doctoral dissertation.

I would like to thank my fellow lab members for their support and collaboration. I am grateful to have such an incredible team with diverse backgrounds and skill-sets.

I am thankful to the College of Computing and Informatics and UNC Charlotte for supporting me by GASP scholarship throughout my PhD program. I would also like to acknowledge Idaho National Laboratory and the National Science Foundation for the invaluable support with the research fundings.

Finally, I would like to thank my thanks to my loving wife, parents, friends and family members for their patience, care and support over the past few years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Motivation

The enormous growth of wireless devices such as Internet-of-Things, smart home sensors, wireless medical devices and smart home gadgets are possessing severe demands to build efficient and intelligent wireless networks that can guarantee optimal network performance. Next generation smart factories are adopting, Industry 4.0 a new industrial revolution that heavily embraces on adopting digital technology such as Internet-of-Things, remotely controlled robots, sensor networks and vision systems to access and control manufacturing facility in real-time. Upcoming mmWave 5G mobile network, seek to enable wireless application services with high throughput and sub-millisecond latency to improve users Quality of Experience (QoE). Different from cellular systems with high deployment and operational costs, wireless multi-hop networks, consisting of a mesh of interconnected wireless routers, have been widely exploited to build cost-efficient communication backbones, including *wireless community mesh networks* [1] (e.g., NYC mesh [2]), *high-speed urban networks* (e.g., Facebook Terragraph network [3], *global wireless Internet infrastructures* (e.g., SpaceX Starlink satellite constellation [4] and Google Loon balloon network [5]), *battlefield networks* (e.g., rajant kinetic battlefield mesh networks [6]), and *public safety/disaster recuse networks* [7]. Conventional communication networks only carried user's traffic such as email, web browsing, video and voice but today's networks also transports data between machines (Machine-to-Machine) that are used for real-time control of physical elements such as actuator of a robot. Wide range of modern day environments discussed above demand for a wireless core communication network infrastructure that can provide guaranteed network performance such as low communication delay, pri-

ority based forwarding, robust against dynamic link conditions such high interference and time-varying network topology.

1.2 Problem Description

1.2.1 Wireless Core - Why we need to apply Machine Learning for Networking ?

Traffic engineering (TE) [8] is a widely applied technique for optimizing network performance by optimal measurement, real-time network traffic analysis, designing optimal forwarding and routing rules to improve the quality of service (QoS) requirements for a large volume of traffic flows. End-to-end (E2E) delay is one of the key QoS metrics TE aims to optimize. Well know TE solutions are simply a variant of shortest path routing protocol including OSPF, IEEE 802.11s [9], and [10] B.A.T.M.A.N. Besides their simplicity in easing the implementation, they cannot guarantee optimal E2E TE performance. Large body of theoretical research work on stochastic network utility maximization (NUM) [11,12] exists on optimal TE, where multi-hop TE problem is formulated as constrained maximization problem of the utility function under stochastic dynamics in user traffic and time-varying wireless channels. However, these solutions suffer from the some key issues. Firstly, there exists some strong assumptions on the network model, such as per-flow per-link queuing structure, unbounded buffer size for each queue, and bounded variance of traffic arrivals. The former two assumptions may not hold for actual routers and the latter assumption does not hold for heavy tailed traffic flows, which have been identified in real networking systems [13–15]. Second, they cannot be used to minimize E2E delay because E2E delay cannot be explicitly and mathematically related to the TE control parameters, such as traffic balancing ratio over each output link, which, however, have to be included in the utility function in the NUM formulation. Third, they are not designed to handle non-stationary conditions caused by the time-varying network dynamics, such as the dynamic traffic patterns. Lastly, some key factors such as the instantaneous link rate

are required as the input, however they cannot be accurately estimated in wireless networks. Because of above limitations, these solutions are barely implemented as the TE solutions to handle real multi-hop wireless networks.

Artificial Intelligence (AI) has made numerous break throughs in variety of applications including computer vision, gaming, robotics, self-driving cars and complex domains such as natural language processing. Recently, using AI in communication networks is gaining attraction due to growing complexity of network elements along with highly dynamic demand from applications. A specific machine learning paradigm, reinforcement learning (RL) has provided diverse solutions to complex problems such as robotics [16, 17], cloud computing [18, 19], advertisement [20], and finance [21]. Adoption of reinforcement learning has been enabling experience-based model-free TE [22–25], which has several key advantages: (1) it needs neither strong assumptions nor accurate modeling of the network, thus allowing it to achieve robust and resilient performance in complex networking systems with high-level uncertainties and randomness, (2) it is designed to handle non-stationarity, and thus it is able to automatically adapt to the time-varying network dynamics, (3) it can deal with large and sophisticated state/action spaces when it is combined with the recent advances in linear and non-linear function approximation (i.e. deep learning).

Existing research works on distributed model-free TE mainly focuses on applying Q-learning and its variants in a multi-agent setting [26–30]. However, Q-learning based TE has fundamental limitations. Q-learning is an off-policy learning method, which suffers from higher variance and slower convergence. Due to the aforementioned limitation Q-learning, leads to a sub-optimal TE performance when there is not sufficient experience to learn. Besides Q-learning is an action-value based method, which can only learn deterministic TE policy. This lead to the case that each traffic flow can only take a single routing path to reach the final destination. Many varieties of RL algorithms and their extensions that can address the limitations of Q-learning in

theory and in the general sense. However, it is still unclear (1) How these algorithms can be generalized to a multi-agent setting to enable distributed TE? (2) What is the actual performance of these algorithms when applied for TE problems? [31]. Many solutions have been proposed in literature but none of them have been shown working on the real hardware nor on a realistic network emulator due to lack of a cohesive system framework. In addition, RL based solutions require explicit control channel to exchange network states that aids the agent to make a timely control decision such as next-hop for forwarding the packet. However, employing a dedicated control channel in wireless network will lead to wastage of limited channel resource. Although, there exists a range of network measurement tools they are generally centralized solutions and do not explicitly provide key metrics required for real-time control decision at the immediate network element.

1.2.2 Wireless Edge - Why we need to optimize network for Machine Learning ?

Distributed machine learning, specially federated learning (FL), has been envisioned as a key technology for enabling next-generation AI at-scale. FL significantly reduces privacy risks and communication costs, which are critical in modern AI systems. Recently, FL systems over edge computing networks have received increasing attention. In FL, the workers, i.e., edge devices, collaboratively learn a shared global model while keeping their data locally to prevent privacy leakage. The workers only need to send their local model updates to the server, which aggregates these updates to continuously improve the shared global model. With single-hop wireless connections, edge devices can quickly reach the FL servers co-located with cellular base stations [32–34]. Different from single-hop wireless connections, wireless multi-hop networks are much more cost-effective and robust against service disruptions posed by single point of failure such as failure of edge server co-located at the cellular station. Enabling FL over wireless multi-hop networks not only can augment AI experiences

for urban mobile users, but also can democratize AI and make it accessible in a low-cost manner to everyone, including the large population of people in low-income communities and under-developed regions.

Despite the impressive features of federated learning and wireless multi-hop network, there are incumbent challenges that could inherently affect the model accuracy. The FL algorithms generally adopt a single-layer server-client architecture, where a central server collects and aggregates the model updates of all workers. The wireless routing paths towards the central server can be easily saturated in such flat FL architecture. the de-facto FL algorithm, FedAvg [35] and many its variants operate in a synchronized manner where the server has to wait and collect a minimum number of local model updates before performing model aggregation and moving to the next round. The long and random multi-hop delay dramatically increases the number of stragglers (slow devices), therefore prolonging the training time per-round.

So far, there are limited research efforts on optimizing wireless FL systems. Existing efforts all focus on single-hop FL over cellular edging computing system [32, 34, 36]. With such assumption, the impact of wireless communication control parameters (e.g., transmission power) on the FL related metrics (e.g., model update delay and loss reduction) can be formulated in an explicit closed-form mathematical model, which greatly eases the FL system optimization. Such model-based optimization is not feasible in multi-hop FL, where the FL performance metrics (e.g., FL convergence time) cannot be explicitly formulated as a closed-form function of the networking control parameters, such as transmission power and packet forwarding decision at each router.

1.2.3 FL Application on Wireless Edge

Deep Learning has been the most powerful tool for providing rich user experience on mobile devices through AI based applications such as smart keyboards for predictive text, recommender systems, advertising and user authentication using techniques such

as facial recognition. These applications generally share the small portion of the user data with the service providers infrastructure to improve the respective AI models. However, recent advocacy on privacy and data locality possess huge challenge to service providers for model training, as the data should not leave the end-users device. As a solution to user privacy and data locality there has been a spike in implementing FL applications, since the workers only need to send their local model updates instead of raw data. Majority of the studies on FL infrastructure and implementations focused on conventional infrastructures such as datacenter's and network deployments which has reliable network conditions such as guaranteed network bandwidth and end-to-end delay. The performance and complexity of FL application over multi-hop wireless network is still unexplored.

1.3 Overview of the Proposed Research

In this thesis, we would like to develop wireless specific machine learning solutions that can systematically benefit (1) Wireless Core Network (2) Wireless Edge Devices and (3) FL applications running on Wireless Edge Devices. Towards this goal, we propose three inter-dependent thrusts:

- **AI-Enabled Wireless Core Network:** To develop a system-in-loop emulator for wireless network along with distributed In-Band network telemetry system (S-INT). Our objective for the emulator is to facilitate a simple but realistic platform for implementing RL routing solutions that can improve the agent's experience in a virtual environment and then transfer the acquired knowledge to commercial off-the-shelf hardware seamlessly. In particular, we will develop an ML-oriented wireless network operating systems (WINOS) framework and dataplane which enables In-Network telemetry system to obtain realtime network condition and experience of data packets such as 1-hop delay and E2E delay. Our WiNOS framework, dataplane and S-INT telemetry system will allow users to rapidly prototype of RL algorithms that can seamlessly operate online and

adapte to realtime network conditions gathered by telemetry system.

- **AI-Assisted Wireless Edge System:** Wireless edge computing networks provide sophisticated edge devices, equipped with rich sensing, computation, and storage resources. In this scenario, we will develop novel distributed computation systems that leverage our AI-optimized wireless core networks to enable distributed ML, i.e., federated learning (FL), where edge devices collaboratively learn a shared global model while keeping their data locally to prevent privacy leakage.

- **FL on AI-Empowered Wireless Edge:** Finally, we will develop a deep radar sensing system as a representative wireless application running on the edge. On the one hand, our radar system will utilize deep neural networks to learn the salient features from high-dimension radar signals for a variety of downstream applications. On the other hand, our radar system will exploit our proposed FL system to address the domain adaptation issues and improve the model generalization in the presence of spatio-temporal radar signal variations.

CHAPTER 2: AI-Oriented Wireless Network Operating System: WiNOS

2.1 Overview

Self-driving network is evolving as an automated network orchestration design paradigm for next-generation network systems. The core of such design is based on training autonomous network systems with real-time experiences such as network state measurements using machine learning algorithms. However, existing network measurement techniques cannot gather such real-time experiences because of centralized architecture leading to considerable control overheads in wireless networks. In this chapter, we designed and implemented AI-Oriented Wireless Network Operating System (WINOS) framework and a distributed In-band network telemetry system (S-INT). Our proposed S-INT system reduces network measurement overhead by embedding telemetry into flowing data traffic with a specialized packet header. WINOS system, on the other hand, enables programmable wireless network control and programmable measurement using S-INT. Efficacy of the proposed system is validated by implementing Multi-Agent Reinforcement routing as a traffic engineering application to optimize end-to-end performance. To the best of our knowledge, our implementation is the first one in the literature that enables multi-agent reinforcement learning algorithm to run on an actual physical wireless multi-hop network. Promising networking performance in terms of delay, throughput, and packet loss observed in the initial experiments show that our distributed wireless network OS, WINOS integrated with S-INT serves as the first step towards the realization of self-driving wireless networks.

2.1.1 Challenges

Reinforcement learning algorithms are experience-driven optimization solutions. Therefore, how to efficiently and effectively collect experiences or network measurements is of significant importance. In the traditional wired SDN architecture, out-of-band centralized telemetry approach is generally exploited. In this case, by using the reliable dedicated control channel between the control plane and the data plane along with a variety of network monitoring tools, such as OpenFlow statistics, SNMP [37], sFlow [38], and NetFlow [39], key network telemetry data such as network topology, link delay, port status, queue delay, and link congestion can be obtained at the network controller, where the centralized machine learning algorithms can be performed to automate and optimize network management. However, due to the limited bandwidth and dynamic wireless conditions, wireless networks can not be optimized in a centralized manner. This demands the deployment of distributed reinforcement learning algorithms [26, 40], which in turn requires distributed In-band Network Telemetry (INT) systems. INT [41], originated from The P4 Language Consortium (P4.org), is a solution that enables collecting and reporting of network status, by the data channel (plane), without utilization and intervention of the control channel (plane). However, P4 INT requires additional hardware support and lacks OpenFlow integration, where Openflow is the de-facto SDN protocol that supports the programmable forwarding of network routers.

2.1.2 Our Contributions

Our objective in this work is to develop a framework that can be used on Commercial-off-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. Towards this goal,

- We have developed a AI-Oriented Wireless Network Operating System (WINOS)

which seamlessly integrates programmable measurement, i.e., the proposed S-INT In-band telemetry framework and the programmable wireless network control.

- We have 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.

- We have implemented a Multi-Agent Reinforcement Routing application for Self-Driving Wireless Mesh Networks using WINOS. In particular, each router, acting as an agent, learns the optimal local traffic engineering (TE) policy in such a way that the collective TE policy of all routers can achieve the optimal end-to-end (E2E) TE performance in terms of delay, throughput, and packet loss.

- We conducted extensive experiments using WINOS and S-INT systems in a wireless multi-hop network testbed. Our preliminary results show that the S-INT is a cost-effective in-band network telemetry solution and our WINOS framework can effectively utilize the telemetry data to prototype RL routing application.

2.2 Wireless Network Operating System (WiNOS) with In-band Telemetry

We propose the distributed Wireless Network Operating System (WINOS), which provides extended modules to realize the vision of self driving wireless networks. As shown in Figure 2.1, WINOS is designed to support the fast prototyping of AI algorithms for intelligent networking. It is built on the top of OpenFlow Manager based on RYU controller [42], telemetry-enabled datapath based on Ofsoftswitch13 [43] software switch, telemetry manager, network state and telemetry database based on MangoDB [44], and radio interface manager based on NetLink library. To implement

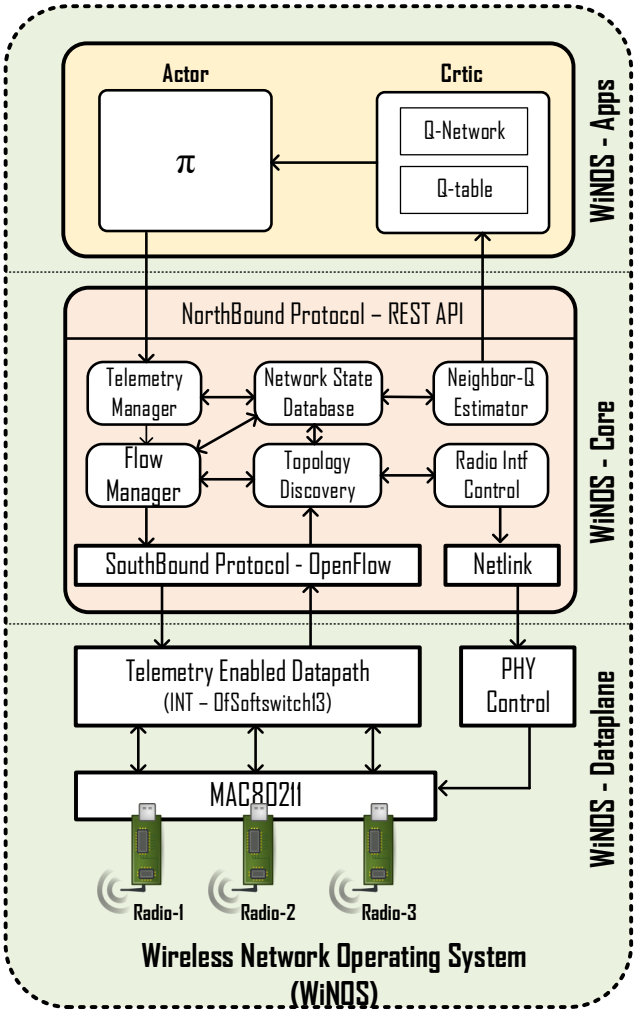


Figure 2.1: WINOS with telemetry-enabled OpenFlow datapath

S-INT telemetry system we need extensive modification on OpenFlow datapath and OpenFlow manager. In particular, OpenFlow datapath should be able to interpret telemetry data and OpenFlow manager should be able to collect and serve the data to other applications.

2.2.1 OpenFlow Manager

SDN strongly relies on OpenFlow due to its simplicity of MATCH, ACTION and STATISTICS criterion. Although adopting the same principle for self-driving networks seems dauntingly useful, it should be capable to convey the experience of the

network packets as results of followed ACTION. Hence, we propose to extend the OpenFlow STATISTICS module to incorporate telemetry data for conveying packet experience.

OpenFlow1.3 protocol defines the structure of messages for representing the datapath elements such as ports, flowtable, packet, and so forth. OpenFlow enables switch interface to communicate with the controller using OpenFlow protocol. This imposes a requirement that any new functionality implemented on the datapath certainly requires modifying the OpenFlow protocol itself. In our case, to share the telemetry data from dataplane to the control plane, we have extended OpenFlow flow statistics message structure with our telemetry template fields such as SRC DPID, DST DPID and TLV fields.

2.2.2 Telemetry Manager

Telemetry data access control and gathering process is handled by the telemetry manager. Any network application, which requires such telemetry data, sends the request to telemetry manager. After receiving the request, it checks Telemetry database and OpenFlow manager to identify if such data is already being gathered. If the application request type is new, then telemetry manager will identify the flows of interest and instructs the OpenFlow manager to disseminate OpenFlow rules to the corresponding datapath.

2.2.3 Network State and Telemetry database

Our network state database provides a RPC based interfaces for data access within within kernel layer and also provides access interface via Northbound API for network applications, such as reinforcement learning based routing algorithm. By having a database within the network controller, it is possible to enable stateful network applications as firewall and also we can develop applications for data intensive traffic engineering applications. In our work, we extensively use MangoDB to store and

serve the telemetry data. In addition, our state base also stores the network state such as link condition and topology formation in a time sequence manner.

2.2.4 Link Discovery Module and Radio Interface Manager

Network orchestration and control in SDN network highly rely on the gathered network topology information. In a fully closed/connected network such as wired networks, network topology is discovered using extended link layer discovery protocol (LLDP). However, such network discovery mechanism fails in wireless networks. The primary reason for such failure is related to how the links and nodes are perceived in OpenFlow. In paricular, direct adoption of the existing link discovery mechanism in wireless multi-hop networks will make all wireless nodes appear as if they are all 1-hop away from each other and they connect to a single port on the data plane. This complicates how we forward packets to the wireless nodes that are several hops away. Wireless channel is a broadcast medium and nodes accept packets that are transmitted with wireless interface MAC address. Hence, we propose to extend the link discovery mechanism to be integrated with MAC80211 SoftMAC module [45]. MAC80211 itself contains discovery schemes that tells which nodes are connected to each other along with the link status. This combination enables network control and visibility in wireless multi-hop networks. In addition, with our integrated system we can control inherent wireless network properties such as link interference, transmit power, channel selection, topology formation and transmit contention window size.

2.2.5 Telemetry-enabled OpenFlow Datapath

To implement our S-INT telemetry framework, we modified the key modules of Ofsoftswitch13 according to the details in section 2.3. Ofsoftswitch13 is a software switch that is designed based on the specifications of OpenFlow protocol version 1.3. This software switch runs entirely on userspace, thus making it the most suitable for prototyping new packet handling routines. The userspace switching leverages

Linux TAP/TUN interface for integration with the operating system network stack. Ofsoftswitch13 supports attaching both virtual and physical ports to its datapath bridge. Packets received on these ports are processed through four key modules including packet parser, header extractor, flow table lookup, telemetry processor, and datapath executor as shown in Figure 2.3 and 2.4.

2.3 S-INT: Distributed In-band Per-Packet Network Telemetry

Self-driving networks is only feasible with accurate and timely feedback from the network elements. In-band telemetry service can utilize data packets traversing the network ports for network metric measurement and transportation. However, existing in-band telemetry solutions are proposed for wired networks and centralized architectures [46], [47] [48], where in-band telemetry metadata increases the packet size by a significant order. Wired networks are capable of transporting fat packets of size 9000 bytes. Wireless networks are cannot handle fat packets. First, fat packets require additional wireless transmission time which will reduce the overall network utilization. Second, wireless networks can only have a maximum packet size of 2304 bytes. Thus, implementing the in-band telemetry system in wireless is challenging and every telemetry header can only have a specific metric.

Taking into account the practicability, we have designed and implemented S-INT, a distributed in-band telemetry system, where each router runs its own telemetry module, which is built on the top of OpenFlow datapath/processing pipeline. The proposed in-band telemetry system is enabled by three key components: new packet header called S-INT telemetry header, new packet matching actions: PUSH_INTL and POP_INTL, and the telemetry processor.

2.3.1 S-INT telemetry header

We have defined a new header called S-INT telemetry header of size 16 bytes with experimental EtherType that can be appended to data packets traversing the port.

Figure 2.2 shows the packet structure with our proposed header. Network application developers can utilize the fields within the header through our extended OpenFlow actions to gather the interested metrics. They can also specify sampling frequency, hop count, or even end-to-end datapath's as their constraints for measurement. In addition, we propose a template-based telemetry system where each telemetry template is unique and have their own measurement objective such as delay, bandwidth, and hop collection measurement. Telemetry header consists of three fields for representing source datapath ID (the telemetry sender), destination datapath ID (the telemetry receiver/sink), and TLV field to specify which telemetry template is used. Each packet can only carry a single template due to the limitation of MTU size. However, in scenarios requiring to obtain two or more telemetry data we suggest to use alternate templates over a sequence of packets.

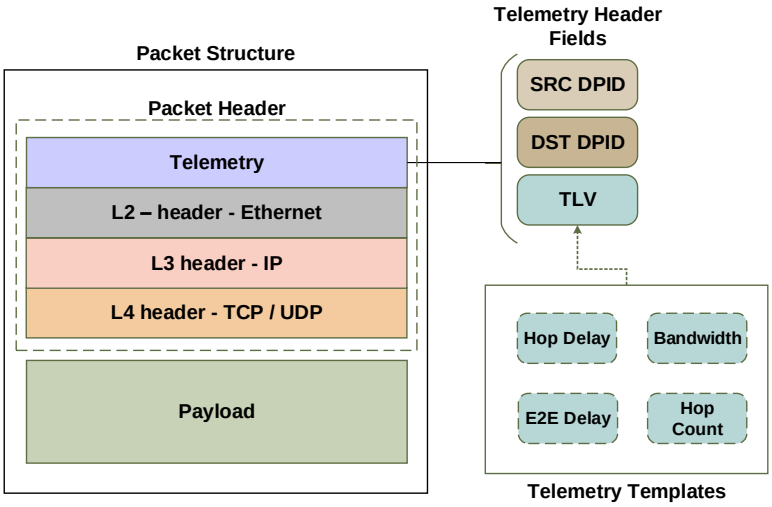


Figure 2.2: Packet stucture with S-INT Telemetry Header and Telemetry Template

2.3.2 PUSH and POP Actions

OpenFlow protocol provides functions to encapsulate and decapsulate packets with headers such as MPLS, VLAN and so forth for data transport. We leverage the same functions to add and remove telemetry header to and from data packets. Figure

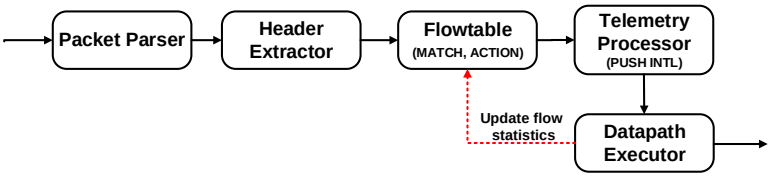


Figure 2.3: Telemetry-enabled OpenFlow Datapath/Processing Pipeline: PUSH INTL performed by the telemetry sender

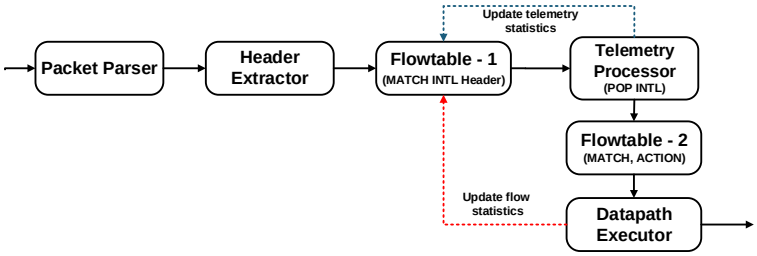


Figure 2.4: Telemetry-enabled OpenFlow Datapath/Processing Pipeline: POP INTL performed by the telemetry receiver

2.3 shows the datapath processing chain for PUSH telemetry header ACTION. At the sender's datpath, flowtable lookup is performed to identify the packet forwarding path based on MATCH and ACTION selection. If the path traversed by the packet is of interest for telemetry information, then the packet will have an additional action PUSH_INTL:template_type. This action appends the data packet with a new header and modifies the packet EtherType to local experimental EtherType. The above mentioned action is the last executed action by the datapath executor, before sending out the packet. Flow statistics of the respective flow is then updated by the datapath executor in terms of cumulative packet count and volume in bytes.

Datapath processing pipeline for POP action follows the sequence as in figure 2.4. Since local experimental EtherType is used as the identifier to know if the received packet contains telemetry header, at the receiver's datapath, flowtable MATCH is first performed to identify the EtherType. If it contains S-INT header, then the next ACTION to be performed on the packet is POP_INTL and then the EtherType is changed to IPv4 Packet. Packet is then handled in a normal dataplane processing

pipeline, where a second flowtable lookup is performed for MATCH and ACTION. Datapath executor then forwards the packet following the ACTION and updates the flow statistics.

Depending on the hop in which we implement the POP_INTL action, we can obtain 1-hop and End-to-End telemetry data. If the we are only looking for 1-hop telemetry data, then we can even send the packets without any further encapsulation. However, if we are looking for 1-hop away or end-to-end telemetry data, then we need to encapsulate the packet further with transport headers such as MPLS.

2.3.3 Telemetry processor

Telemetry processor is the core of our S-INT framework. The PUSH_INTL and POP_INTL actions determines the operations performed by telemetry processor. If the action is PUSH_INTL:template_type, then the telemetry processor appends the header with the fields for the respective template type. If the received packet contains the telemetry header, upon identifying the telemetry template from the header simple arithmetic operations are performed with the telemetry data to retrieve link delay, bandwidth, and other network state information. After retrieving data, telemetry processor updates the telemetry statistics with the new data.

In Figure 2.5, we show an illustrating example for measuring the link delay between two telemetry-enabled OpenFlow datapath using our S-INT system. Consider host STA1 sends a packet to the host STA2. With OpenFlow rule as show in figure 2.6, WB1 adds telemetry header to the packet with the template type as delay using the PUSH_INTL action. Delay adds timestamp (tx_ts) to telemetry header before sending our the packet. Once the packet is received by WB2 and telemetry header is removed with the POP_INTL action and template_type as delay. After retrieving the timestamp from the removed header, difference between current timestamp (rx_ts) on WB2 and packet header timestamp (tx_ts) is computed to get the delay experienced by the packet.

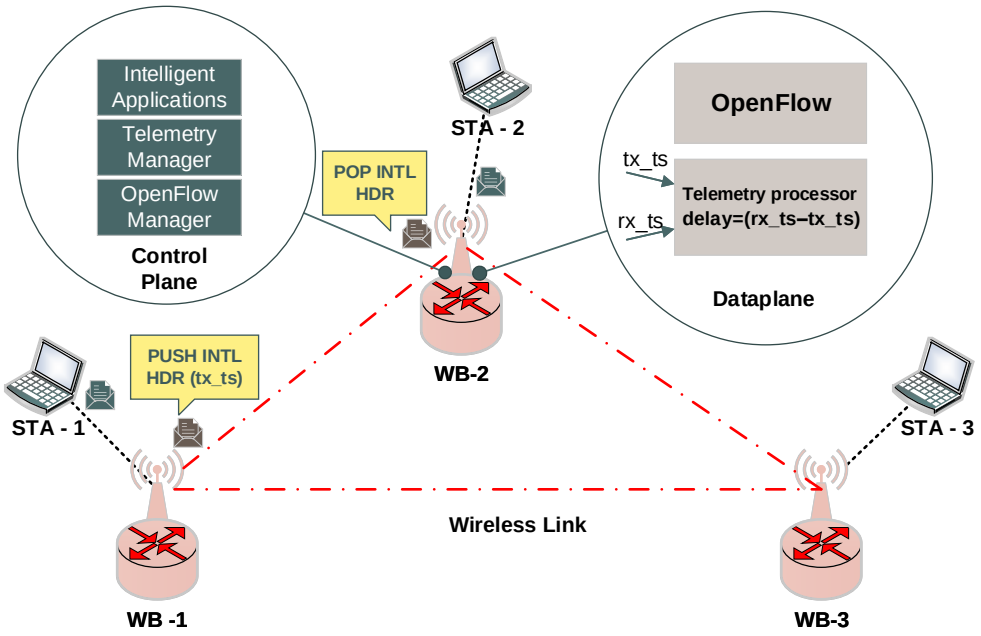


Figure 2.5: S-INT Overall Architecture. Packet leaving WB1 contains the timestamped (tx_ts) and WB2 telemetry processor computes different between local timestamp (rx_ts) and packet timestamp (tx_ts) to get the delay

In order to realize the above mentioned illustration we should also extend the below listed datapath modules:

2.3.4 Packet Parser

Packet parser typically consists of the structure of every packet that exits in today's network including IPv4, IPv4, TCP, UDP, and ICMP. Once a packet is received on the switch port, the conformity of the packet is verified by checking the structure. Non-conformed packets are automatically dropped. We extended the data structure of the packet library to include our customer header structure as in figure 2.2 to pass the verification stage.

2.3.5 Header Extractor

The header extractor identifies the fields within the received packet based on the packet type determined by packet parser. Every packet type has strictly defined

OpenFlow rule on	MATCH	ACTION	STATS
WB1	in_port = 1	apply: push_intl, delay, out_port = 2	tx_pkts, rx_pkts, intl_delay
WB2	in_port = 2	apply: pop_intl, delay, out_port = 1	tx_pkts, rx_pkts, intl_delay

Figure 2.6: S-INT: OpenFlow rules in Flowtable for delay estimation between Telemetry enabled OpenFlow datapath WB1 and WB2

header attributes with own data type. The header extractor scrutinizes the received packet and identifies the values for the specific header type attributes. For instance, if it receives a Ethernet packet then header extractor will identify the source and destination Ethernet addresses. We extended this module with our new header structure and data type shown in Fig. 2.2.

2.3.6 Flowtable

Packet forwarding in SDN is handled based on the MATCH and ACTION instructions. The flowtable is a multidimensional row and column with depth of upto 1024. Typical structure of the flowtable is grouped into MATCH, ACTION and STATISTICS columns. MATCH columns will generally comprise of fields in L2, L3 and L4 packet headers in addition to datapath port numbers. ACTION column will contain the instruction of how to handle the matched packet. Typically, it can have a simple instruction such as forward the packet to a specified port, modify a packet, encapsulate or decapsulate a packet. STATISTICS columns are built with a handful of counters that identify statistics such as count and size of packets processed by the specific rule. To implement our telemetry templates, we extended the STATISTICS counters with additional fields to include telemetry metrics such as delay, bandwidth, and hop count. Figure 2.6 shows an example of flowtable structure with telemetry flows. Based on the chosen telemetry template, the resulting statistics fields will be varying. As a result, our PUSH_INTL and POP_INTL actions should be indepen-

dent from the normal datapath processing action.

2.4 Multi-Agent Reinforcement Routing for Self-driving Wireless Mesh Networks

As one of the most important network management methods, traffic engineering (TE) aims to dynamically analyze real-time network traffic, and planning optimal routing rules to meet the quality of service (QoS) requirements for the traffic flows. Optimizing these E2E TE metrics such as E2E delay and throughput is very challenging in wireless multi-hop networks due to the profound dynamics in traffic flow patterns, wireless link status, working conditions of wireless routers, and time-varying network topology. Recent advances in reinforcement learning (RL) have provided promising technologies for enabling experience-based model-free TE [25, 26, 40].

In this section, we demonstrate a prototype of the self-driving wireless network by implementing a learning-based routing application on a WINOS-empowered wireless mesh network testbed. In particular, this routing application is based on our multi-agent reinforcement learning-based TE framework proposed in [40].

2.4.1 MDP for delay-optimal Traffic Engineering

The distributed traffic engineering (TE) can be formulated as multi-agent extension of Markov decision process (MA-MDP) for N routers [40], which is defined as a tuple of $\langle \mathcal{S}, \mathcal{O}_{1:N}, \mathcal{A}_{1:N}, \mathcal{P}, r_{1:N} \rangle$.. In this MA-MDP formulation, The environmental states $\mathcal{S}$ consist of 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. $\mathcal{O}_i$ defines the local observation of each router i. It contains the network state information, which is available at each router i. $\mathcal{A}_i$ is the set of actions that can be performed by router i. For our TE application, $\mathcal{A}_i$ contains the IDs of the next-hop neighbors of router i that router i can use as the next-hop forwarding node. P is the network state

transition probabilities, which are generally unknown. r_i is the reward function of each router i, which is the (negative) 1-hop delay from router i to its neighbor. For each packet that enters the router i, the router needs to determine the forwarding action ($a \in A_i$) based on its local observation $o \in O_i$ of the network status. After the forwarding action is performed or the packet is sent out, the router will receive a reward r_i (i.e., the (negative) delay) when the packet arrives at its next-hop router $i+1$, which has its own local observation $o' \in O_{i+1}$. The return $G_i = \sum_{i=1}^{T} r_i$ is the accumulated reward (i.e., negative E2E delay) induced by forwarding a packet from its ingress router to its egress router. Each router selects forwarding actions based on a local policy π_i, which tells how the router chooses its action based on the observation. The policy can be stochastic by choosing an action according to certain probability or deterministic by choosing a fixed action. Our objective is to find the optimal policy π_i for each router so that the expected return $J(\pi)$ of the joint policy $\pi = \pi_1, ..., \pi_N$ is maximized (2.1), i.e.,

$$J(\pi) = E[G_i|\pi] = E[\sum_{i=1}^{T} r_i|\pi] \tag{2.1}$$

2.4.2 Multi-agent Off-policy Softmax RL Algorithm

To solve the above MA-MDP problem, we adopt the multi-agent actor-critic (MA-AC) architecture [40]. In this case, each router has its own actor and critic running locally. The local critic uses exponential moving average to estimate the action-value functions $q_i^{\pi_i}(s, a)$, which criticize the action selections. Using critic's inputs, the actor improves the target policy towards the direction that can maximize the expected return,

Local Critic for Policy Evaluation: The performance of the policy π is measured by the action-value $q_i^{\pi}(s, a)$, which is a E2E TE metric. The action-value $q_i^{\pi}(s, a)$ of router i can be written as the sum of 1-hop reward of router i and the action-value

of the next-hop router $i+1$, i.e.,

$$q_i^{\pi_i}(s,a) = E\left[r_i + q_{i+1}^{\pi_{i+1}}(s',a')\right]. \tag{2.2}$$

By applying exponential weighted average, the estimate of $q_i^{\pi_i}(s,a)$, denoted by $Q_i^{\pi_i}(s,a)$, can be updated 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.,

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i + Q_{i+1}^{\pi_{i+1}}(s',a') - Q_i^{\pi_i}(s,a)] \tag{2.3}$$

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

2.4.3 Local Actor for Policy Improvement

Based on the estimated action-value, i.e., Q value, the local actor aims to improve the local policy towards the direction that can maximize the expected return $J(\pi)$ in eq. (2.1). In this paper, we adopt the off-policy softmax algorithm as the local routing policy. In this case, the *target policy* the router aims to learn and improve is the greedy policy, i.e., selecting the action with the maximum estimated action-value. The *behavior policy*, which generates the actual actions for the learning agent, i.e., router, is softmax policy, where each action a is selected with a probability $P(a)$ based on the exponential Boltzmann distribution

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

2.4.4 Learning Algorithm Implementation as an Application of WiNOS

The implementation of off-policy softmax learning algorithm is based on two north-bound APIs provided by WINOS as shown in Fig. 2.7. The first API provided by network state database will provide the Q estimation for the local critic. Based on the Q value, the actor improves the target greedy policy, while generating the ac-

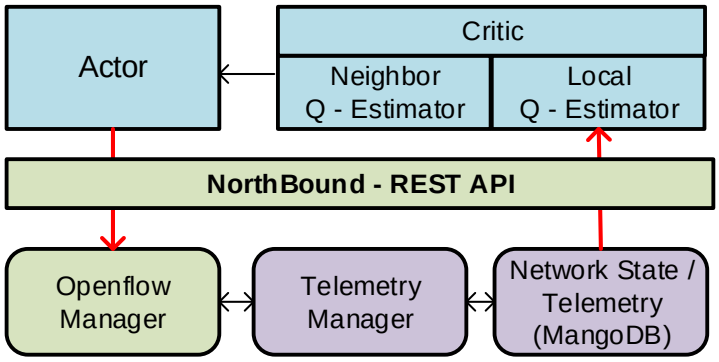


Figure 2.7: multi-agent actor-critic reinforcement routing can be quickly prototyped as an application running on the WINOS

tual actions to be performed by router based on softmax policy. The actual action, i.e., next-hop router selection, is then forwarded to the OpenFlow manager based on another northbound API, Finally, the OpenFlow manager will translate the human-readable actual actions to the underlying OpenFlow instructions.

2.4.5 Backward Neighbor Q Estimation

The key challenge to implement reinforcement routing algorithms is how to estimate the Q value without inducing so much control overhead. In particular, estimating Q value relies on the measurement of per-hop per-packet delay as shown in eq. 3.6. Directly requesting the delay information from the neighboring router could introduce overhead to the bandwidth-limited wireless channel. Therefore, it is necessary to redesign the way of exchanging information among neighbor. As illustrated in Fig 2.8, we propose the backward neighbor Q estimation for each router i, which aims to estimate the action-value of the (backward) neighbors whom the router i receives data from. The local Q estimation of router i is directly coming from its forward neighbors. The motivation of such design is based on the fact that the action-value Q_1 of predecessor router 1 is estimated based on the reward r_1 and the action-value Q_2 of current router 2. Both r_1 and Q_2 are immediately available at current router 2,

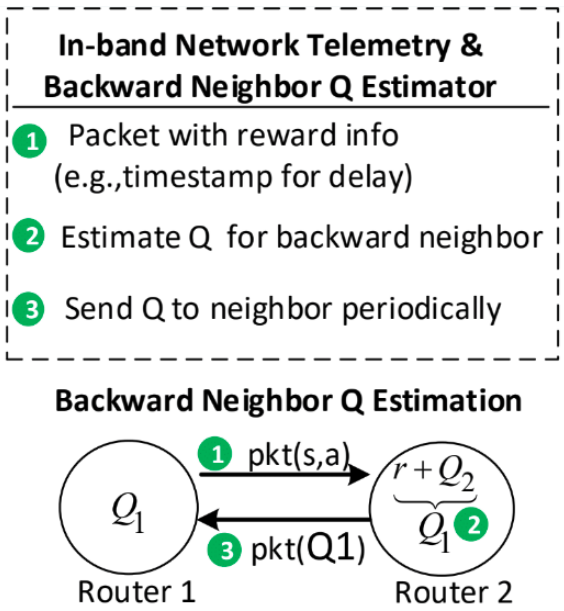


Figure 2.8: Backward Neighbor Q estimation

instead of the predecessor router 1. Therefore, it is more cost-effective to let current router estimate the action-value of its backward router. Such scheme allows the action-value to be updated at the line speed, i.e., the speed at which packets come in the router and also reduce the overhead delay in control channel.

2.4.6 Implementation details

In our implementation, the local agent periodically observes the flow table from local controller REST API, which is provided by RYU rest_ofctl application. According to Fig. 2.8, when a packet from Router 1 is sent to Router 2, if there is a MATCH entry for the packet header fields, the forwarding rule at Router 2 is executed, which is defined by ACTION fields. In particular, MATCH fields contain [source destination mac, destination mac address, destination IP], where the destination IP of the packet is used as the local state/observation at each router. Based on such local state, the ACTION `set_field` (i.e., source destination mac, destination mac address and output port) is modified according to the softmax behavior policy, which needs the the local Q estimation that can be obtained from the forward neighbor router of the router 2. To stabilize the learning process, the ACTION fields are only updated for

every N packets.

At the same time, whenever router 2 receives a packet from the router 1, router 2 will keep updating the Q value of router 1 according to $Q_1^{\pi_1}(s,a) \leftarrow Q_1^{\pi_1}(s,a) + \alpha[r + Q_2^{\pi_2}(s',a') - Q_1^{\pi_1}(s,a)]$, where the reward r is the negative one-hop delay from router 1 to router 2. r is obtained by the proposed S-INT framework. Finally, router 2 will send the updated Q_1 back to Router 1 periodically via POST request. After Router 1 receives the Q_1 from router 2, it uses the new Q_1 as its local Q estimator.

2.5 Experimental Evaluations

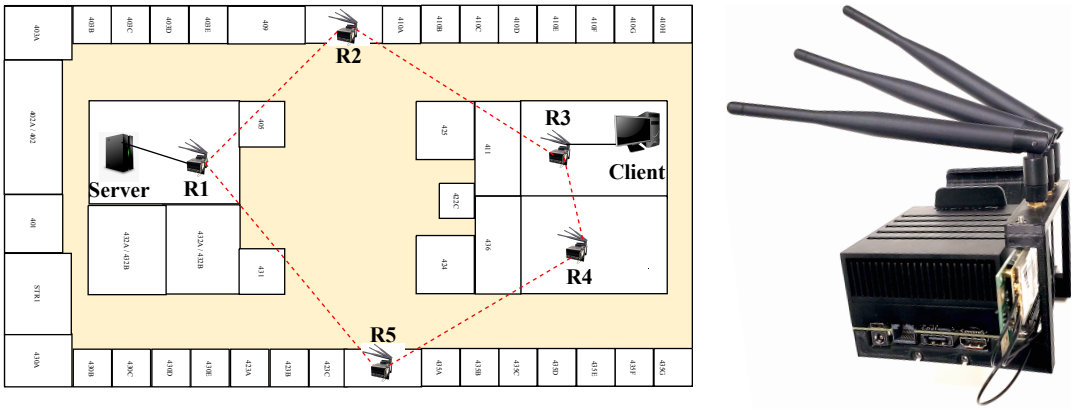
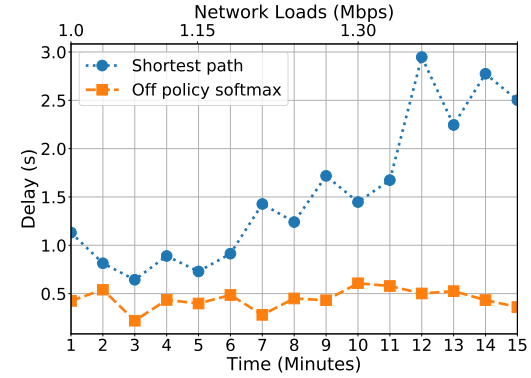


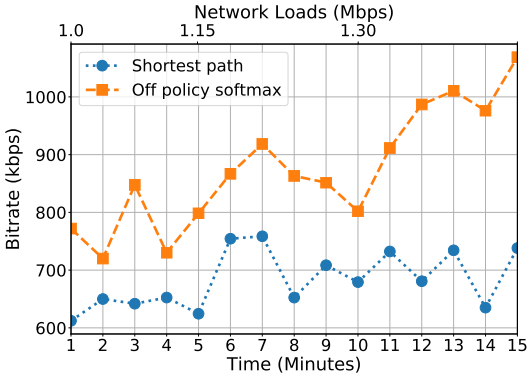
Figure 2.9: (a) Testbed Topology (b) Nvidia Jetson Xavier Wireless Router

2.5.1 Testbed Setup

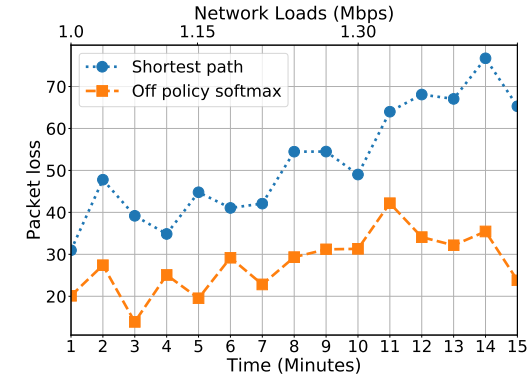
Our experimental physical testbed consisted of 5 Nvidia Jetson Xavier nodes with Compex WLE900VX wireless interface card. We deployed our WINOS system on top of Ubuntu 18.04 Linux operating system running on each Nvidia Jetson node. Each wireless router was configured to operate on fixed 5Ghz Channel 36 and 40Mhz channel width in 802.11ac operating mode. We installed 5 wireless routers at various locations covering our lab floor area and 2 client hosts were connected at locations R1 and R3 as show in Figure 2.9. After the deployment, we manually inspected the topologies formed at every router and noticed that there were two possible paths from the client to the server. First, the upper bound path goes through R3 $\rightarrow$ R2 $\rightarrow$ R1



(a) E2E Delay



(b) Throughput



(c) Packet Loss

Figure 2.10: Average of 10 runs under high network increasing load conditions, we measured the network metrics for every 1 minute

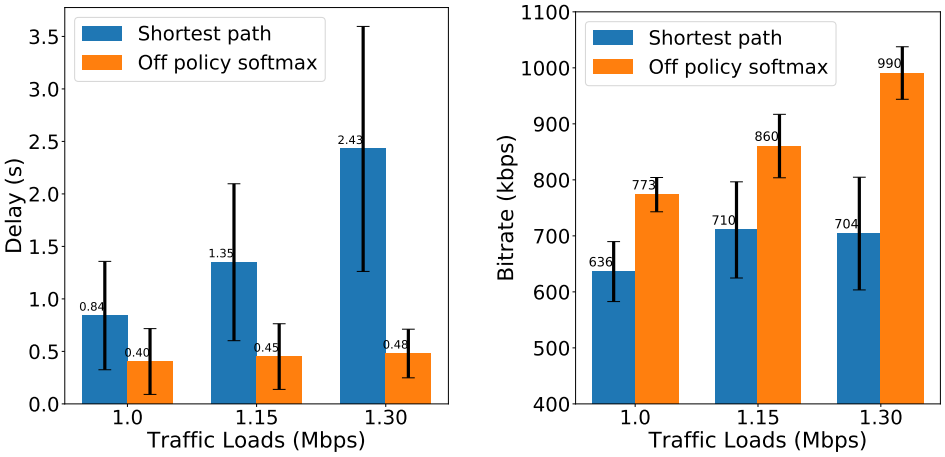
(2 hops) and lower bound path from R3 → R 4 → R5 → R1 (3 hops).

A client sends a UDP traffic flow to the server with the varying traffic intensity of 1.0, 1.15 and 1.30 Mbps respectively following a Poisson distributed packet inter-departure time per second. The traffic flow lasts for 15 minutes. Each data traffic intensity keeps unchanged for 5 minutes and then the intensity is increased to next level. We use the average end-to-end throughput,the average end-to-end packet delay, and average end-to-end packet loss as the performance metrics. Since the wireless network conditions are dynamically changing overtime, we average the experiment results of 10 runs that are performed at different times (e.g., morning and night) of two consecutive days.

2.5.2 Learning-algorithms in the fields

The objective of our experiments is to show that the proposed S-INT and WINOS enables the quick prototyping of learning-based algorithms for self-driving wireless networks. Towards this goal, the widely used shortest path (in terms of hop) was selected as a baseline to compare to off-policy algorithm with softmax action selection. As mentioned in [40], softmax action-selection, in general, helps the agent to select second-best control links with a probability, and it helps the exploration of other paths to balance the traffic in high network loads and to reduce average packet delivery time. Thus, we selected softmax policy as a behavior policy for the agent and learning rate is set 0.1. The path from R3 → R2 → R1 was used as the shortest path to send a packet from the client to the server. Figure 2.10 shows the average delay, throughput, and packet loss rate for every 1 minute and the top x-axis shows the data traffic load increasing every 5 minutes. The dynamic network environment comes from the nature of the wireless medium (link delay) and increasing traffic load (queuing delay). The overall performances show the efficient routing policies of the learning-based TE since it is able to adapt to the non-stationary network environment and learn the optimal path dynamically. In term of the end-to-end delay, it can be seen that the off-policy

softmax remains the same packet delivery delay as low as 0.5 second for the whole experiment. The shortest path routing performs poorly i.e., delay increases as traffic load increases. In Figure 2.10(b), the throughput of off-policy softmax surges up when higher data traffic (1.15 and 1.30 Mbps) is injected into the network. However, the throughput of shortest path increases only around 100 Kbps due to congestion, and the packet losses of shortest path routing continuously rise up to around 70 packets per second as shown in Figure 2.10(c).

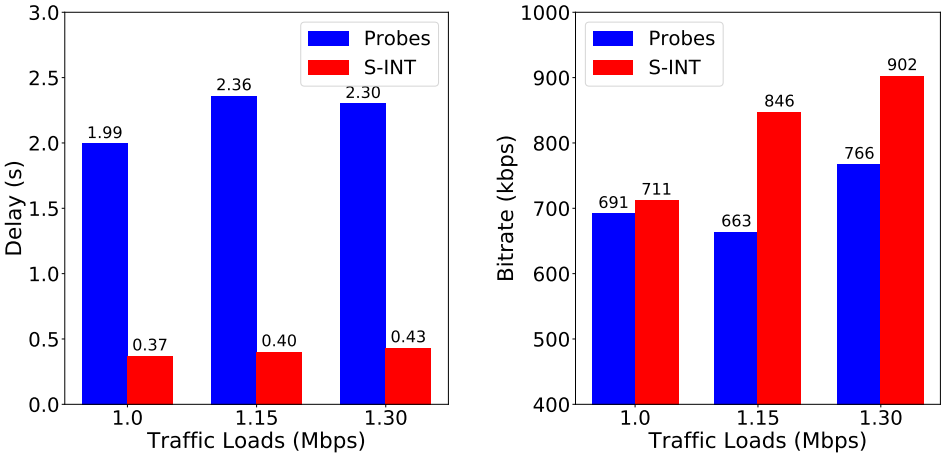


(a) E2E Delay (b) Throughputs

Figure 2.11: Stability Analysis of Off-policy softmax routing and shortest path routing

2.5.3 Stability Analysis

The performance gap we observed in the previous section becomes even more evident when we evaluate the stability analysis over 10 runs. Figure 2.11 illustrates the mean and variance of the delay and throughput for each network load condition. Although the wireless network environment is heavily dynamic, the variances of delay and throughput of learning-based TE maintain low around 0.9 sec in term of delay and 100 Kbits in term of throughput. For example, even with highest network traffic (1.3 Mbits) injected into network, the average and variance of the end-to-end delay remains relatively small as 0.48 $\pm$0.81 for learning-based TE, while the shortest path



(a) E2E Delay (b) Throughput

Figure 2.12: Performance comparison between probes and S-INT (single run)

routing experiences high average delay along with high delay variance (i.e., jitter) Similar phenomenon is also observed for the end-to-end throughput. To sum up, as the network traffic load grows, the learning-based TE algorithm leads to much stable and superior networking performance.

2.5.4 Overhead Analysis

The key feature of S-INT is to reduce the control overhead when the learning algorithms need to collect experiences for training. To observe the effectiveness of S-INT, we compare it to the probe-based measurement method, which sends extra probe packets to collect the network metrics such as link delay. The ICMP data packet is used to carry timestamp. Each router sends a timestamp probe packet after each data packet is sent out. In this way, the probe-based approach can achieve the same per-packet telemetry as S-INT.

We evaluate the network delay and throughput performance under S-INT and probe-based telemetry respectively in Figure 2.12. X-axis represents the varying traffic loads, and y-axis shows the average of end-to-end delay and throughput for 5 minutes. The figure clearly shows that S-INT approach significantly reduces the control overhead and leads to much higher throughput and lower delay. This is

because even with the small size (56 bytes) of probe packets, sending a probe packet out for every out-going data packet is very costly.

2.6 Conclusion

In this chapter, we proposed a distributed In-band telemetry system (S-INT) and a wireless network operating system (WINOS) for self-driving wireless networks. Our proposed system provides two key benefits (1) Programmable measurement using S-INT, resulting in low-overhead telemetry system and (2) Programmable wireless network control from WINOS for quick and easy implementation of AI-enabled distributed traffic engineering solutions such as Multi-Agent reinforcement routing. We implemented a traffic engineering application based on Multi-Agent Reinforcement routing on a physical wireless mesh testbed, using S-INT and WINOS systems. Our results show promising network performance in terms of delay, packet loss and throughput. We strongly believe, our proposed distributed WINOS and S-INT systems will open more research opportunities to realize Self-Driving wireless networks.

CHAPTER 3: FedEdge: Towards Network-Accelerated Federated Learning over Wireless Edge

3.1 Overview

Federated learning (FL) is a distributed machine learning technology for next-generation AI systems that allows a number of workers, i.e., edge devices, collaboratively learn a shared global model while keeping their data locally to prevent privacy leakage. Enabling FL over wireless multi-hop networks can democratize AI and make it accessible in a cost-effective manner. However, the noisy bandwidth-limited multi-hop wireless connections along with statistical and system heterogeneities can lead to delayed and nomadic model updates, which significantly slows down the FL convergence speed.

To address such challenge, in this chapter, the regularized local stochastic gradient descent (SGD) is first adopted to mitigate computation-induced convergence latency caused by statistical and system heterogeneities. Then, to combat communication-induced convergence slowdown, the multi-agent reinforcement learning (MA-RL) algorithms are developed, which find the delay-optimal routing paths to minimize the model exchange latency between the edge devices (i.e., workers) and the remote server. To validate the proposed solutions, FedEdge is developed and implemented, which is the first experimental framework in the literature for FL over multi-hop wireless edge computing networks. FedEdge allows us to fast prototype, deploy and evaluate novel FL algorithms along with RL-based system optimization methods in real wireless devices. Finally, using FedEdge, our experimentation results in a physical testbed show that the proposed network-accelerated FL system can significantly improve FL convergence speed. The content of this chapter is partly reprinted with permission

from P. Pinyoanuntapong, P. Janakaraj, P. Wang, M. Lee and C. Chen, "FedAir: Towards Multi-hop Federated Learning Over-the-Air,"

3.1.1 Challenges in Multi-hop Federated Learning

Despite its great potential advantages to democratize AI, *multi-hop FL*, short for FL over multi-hop wireless edge computing networks, is still an unexploited area. The classic FL systems use single-hop wireless communications to directly connect to the edge servers or connect to edge routers that then reaches the remote cloud servers via high-speed Internet core. In multi-hop FL networks, the end-to-end (E2E) model updates between the server and workers need to go through multiple noisy and bandwidth-limited wireless links. This results in much slower and nomadic model updates due to much longer and more random E2E delay. Such profound communication constraints fundamentally challenge the efficiency and effectiveness of classic FL systems as detailed below:

- **Degraded scalability of FL over wireless multi-hop networks**: The FL algorithms generally adopt a server-client architecture, where a central server collects and aggregates the model updates of all workers. The routing paths towards the central server can be easily saturated in wireless multi-hop networks due to the limited network bandwidth. In addition, different from classic distributed model training in data centers, FL exploits production networks to carry on model training traffic between workers and the server. Therefore, FL traffic has to compete with the background production network traffic (e.g., Internet traffic) for limited network bandwidth. As a result, when the number of workers or the background traffic volume increases, network congestion will deteriorate progressively, which critically degrades the benefits of computation parallelization and slows down convergence speed.

- **Difficulties of model-based optimization for multi-hop FL system**:

so far, there are limited research efforts on optimizing wireless FL systems. Existing efforts all focus on single-hop FL over cellular edge computing systems [32, 34, 36]. With such assumption, the impact of wireless communication control parameters (e.g., transmission power) on the FL related metrics (e.g., model update delay and loss reduction) can be formulated in an explicit closed-form mathematical model, which greatly eases the FL system optimization. Such model-based optimization is not feasible in multi-hop FL, where the FL performance metrics (e.g., FL convergence time) cannot be explicitly formulated as a closed-form function of the networking control parameters, such as packet forwarding decision at each router.

3.1.2 Our Contributions:

The objective of this work is to develop a novel multi-hop FL system that can guarantee high accuracy and faster convergence by systematically taming algorithm and networking-induced delay.

- To our knowledge, this is the first work in the literature to reveal, formulate, and experiment on the inherent interplay between multi-hop wireless networking and federated learning.
- To minimize the FL convergence time, we exploit multi-agent reinforcement learning for FL system optimization, which minimizes the networked-induced convergence latency by learning the delay-minimum routing paths for FL traffic flows.
- We develop and prototype FedEdge, which is the first experimental framework in the literature for FL over multi-hop wireless edge computing networks. FedEdge allows us to fast prototype, deploy and evaluate novel FL algorithms along with machine learning-based FL system optimization methods in real-life wireless devices.

- We demonstrate via extensive experiments that the proposed FedEdge system and reinforcement learning-based FL system optimization have the great potential to effectively improve the convergence performance of FL in wireless multi-hop networks.

3.2 Runtime convergence of Federated Learning

3.2.1 Federated Learning via Regularized Local SGD

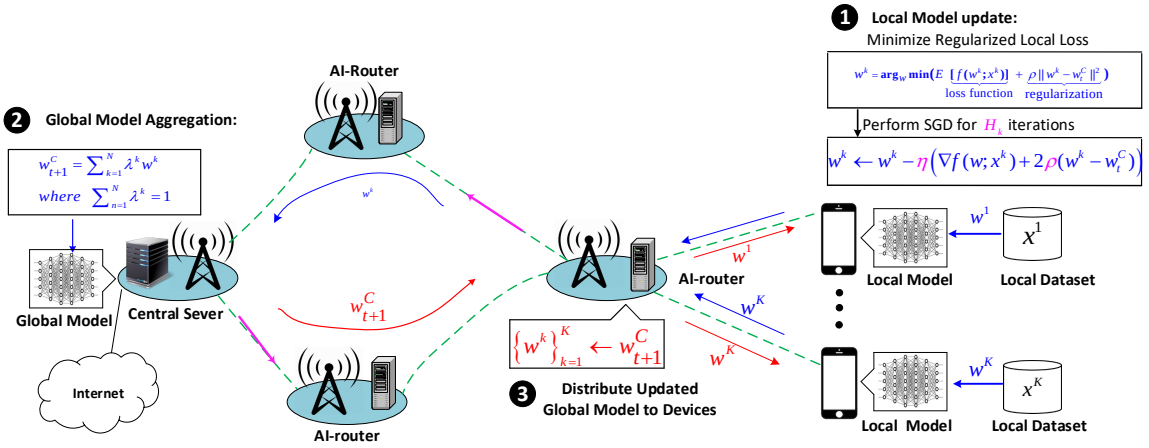


Figure 3.1: Federated learning via local SGD

Federated learning methods are designed to handle distributed training of neural networks over multiple devices, where the devices have their local training data and aim to find a common model that yields the minimum training loss. Such a scenario can be modeled as the following distributed parallel non-convex optimization

$$\min_{w} F(w) = \sum_{k=1}^{N} \lambda^k F^k(w), \quad F^k(w) = E\left[f(w^k; x^k)\right] \tag{3.1}$$

where $F(w)$ is the global loss, $F^k(w)$ is the local loss of device k, N is the number of devices, $\lambda^k = \frac{n^k}{n}$ and $\sum_{k=1}^{N} \lambda^k = 1$, where n^k is the number of training samples on device k and $n = \sum_k n^k$ is the total number of training samples in network. The local loss $F^k(w)$ is a non-convex function over data distribution $x^k \sim \mathcal{D}^k$, which is possibly different for different device k. The optimization problem in eq. (3.1) can be

generalized by adding a quadric regularization term in the objective function [?, 49], i.e.,

$$F^k(w) = E\underbrace{[f(w^k; x^k)]}_{\text{loss}} + \underbrace{\rho||w^k - w_t^C||^2}_{\text{regularization}} \tag{3.2}$$

where w_t^C is the global model and ρ is the penalty parameter that determines how much deviations from the global model the local model is allowed.

To solve above optimization problem, FL methods are following a common stochastic optimization technique, called local SGD, which alternates between local SGD iterating and global model averaging for multiple (server-worker communication) rounds, where the worker is the device that participates in the collaborative model training. As shown in Fig. 3.1, during each round, the worker tries to reduce its local loss $F^k(w)$ by performing H_k mini-batch SGD iterations with each iteration updating the model weights, following:

$$\text{Local SGD Iterating: } w^k \leftarrow w^k - \eta\frac{1}{B}\sum\nolimits_{x^k \in \mathcal{I}^k} \left(\nabla f(w^k; x^k) + 2\rho(w^k - w_t^C)\right) \tag{3.3}$$

where $\mathcal{I}^k$ is a subset (mini-batch) of the training samples on worker k and $B = |\mathcal{I}^k|$ is the size of the mini-batch. After finishing H_k local SGD iterations, the workers send their local models $\{w^k\}_{k \leq K}$ to the central server, which averages them and updates the global model accordingly

$$\text{Global Model Averaging: } w^c = \sum_{k=1}^{K} \lambda^k w^k \tag{3.4}$$

where K is the number of devices selected to be the workers. The new global model is sent to the workers and the above procedure is repeated.

It is worthy to notice that minimizing regularized loss ensures that the local workers will not fall into the model update trajectories that are far away from the current

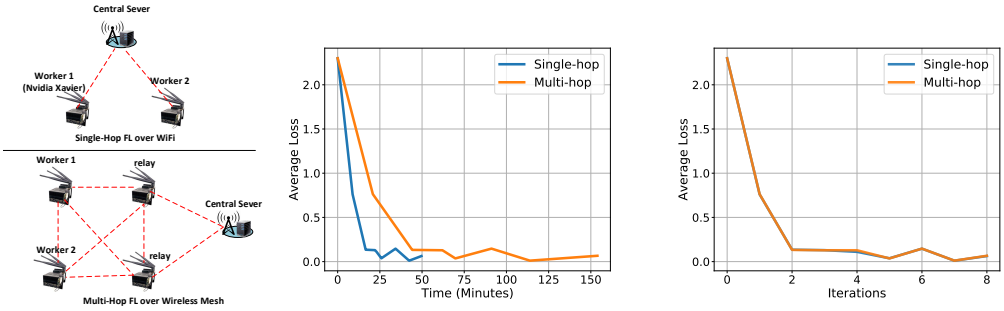
global model. Such practice can effectively prevent the potential divergence caused by statistical heterogeneity [50] and system heterogeneity [51]. On the one hand, the workers involved in FL training tend to possess significantly diverse data samples so that they follow unbalanced and non-IID data distribution, thereby introducing statistical heterogeneity. On the other hand, the workers generally possess diverse computation resources (e.g., CPU, GPU and RAM). To mitigate the blocking effects of stragglers (slow workers) and reduce computation-induced latency, each worker can perform different number of local iterations H_k according to its computation constraint, which leads to system heterogeneity. When the penalty parameter equals to zero, i.e., $\rho = 0$ and all workers adopts the uniform local updates, i.e., $H_k = H,\ \forall k \leq N$, then the local SGD algorithm in eq. (3.3) and (3.4) becomes the classic FedAvg algorithm [35].

3.2.2 Convergence of Local SGD

3.2.2.1 Iteration Convergence

Before local SGD is applied in FL settings, it already showed very promising performances for distributed optimization in data center environments. The key advantage of local SGD is its low communication overhead along with high convergence speed. Recent research shows that for non-convex optimization with both IID and non-IID data, local SGD can achieve fast $\mathcal{O}(1/\sqrt{KT})$ convergence [52,53], i.e., achieving linear speedup w.r.t. the number of workers K, where T is the total number of iterations performed by each worker. This is the optimal convergence performance achieved by the celebrated parallel mini-batch SGD methods [54, 55], where each worker sends its model or gradient to the server after each local SGD iteration is done. Therefore, parallel mini-batch SGD achieves the optimal $\mathcal{O}(1/\sqrt{KT})$ convergence at the cost of T communication rounds. However, to achieve the same convergence performance, local SGD only needs $\mathcal{O}(T^{3/4}K^{3/4})$ communication rounds [52, 53]. In other words, local SGD can preserve the fast convergence with significant less communication costs

by putting more computation loads on the workers, i.e., by letting workers perform $\mathcal{O}(T^{1/4}/K^{3/4})$ local SGD iterations instead of one.



(a) (b) (c)

Figure 3.2: We use Nvidia Xavier nodes to implement two workers and one server to train MNIST dataset for digit recognition task. To test the impact of wireless networking, we use the exactly same parameters (e.g., initial model weights, number of rounds, number of local iterations, batch size, and learning rate) for FL over single-hop and multi-hop wireless networks, respectively. (**a. Network topology**) for single-hop network, we use IEEE 802.11ac for wireless connections. For multi-hop mesh network, we use IEEE 802.11s for multi-hop routing, which still uses IEEE 802.11ac for MAC/PHY functions. The testbeds are deployed in the first floor of UNCC CS department with interferences from co-existing campus WiFi networks (**b. Runtime Convergence**) The wireless multi-hop FL converges much slower with respect to the true training runtime (wallclock time) (**c. Iteration Convergence**) The single-hop and multi-hop FL systems have the same iteration convergence performance.

3.2.2.2 Runtime Convergence of Local SGD

Local SGD method (e.g., de-factor FL algorithm FedAvg) is generally implemented in a synchronous manner, where the SGD update sequences on the workers are synchronized (by model averaging). In other words, the server needs to wait for the model updates from all workers and then it can perform model aggregation, after which the workers can resume their local SGD updates for the next round. As a result, if the actual training runtime (wallclock time) t is used instead of iteration index T, the convergence of local SGD could be as worst as $\mathcal{O}(\sqrt{\tau_{max}}/\sqrt{Kt})$(where each worker only performs one local iteration). τ_{max} is the delay of the slowest worker (straggler) to deliver its local model to the server, which could be very small in high-speed data

center networks and wireless single-hop networks (e.g., WiFi or cellular). In wireless multi-hop networks, τ_{max} becomes a more dominant factor affecting the true runtime convergence due to the large, random and heterogeneous E2E communication delays experienced by the workers. As a result, the theoretically fast convergence of local SGD can be practically slowed down in wireless multi-hop networks. Such projection is also verified through a simple experiment (Fig. 3.2). Moreover, the linear convergence speedup by increasing the number of workers K could also be accompanied with the increased delay τ_{max} due to escalated network congestion, which leads to convergence slowdown.

3.2.3 Hierarchical Synchronous FL system

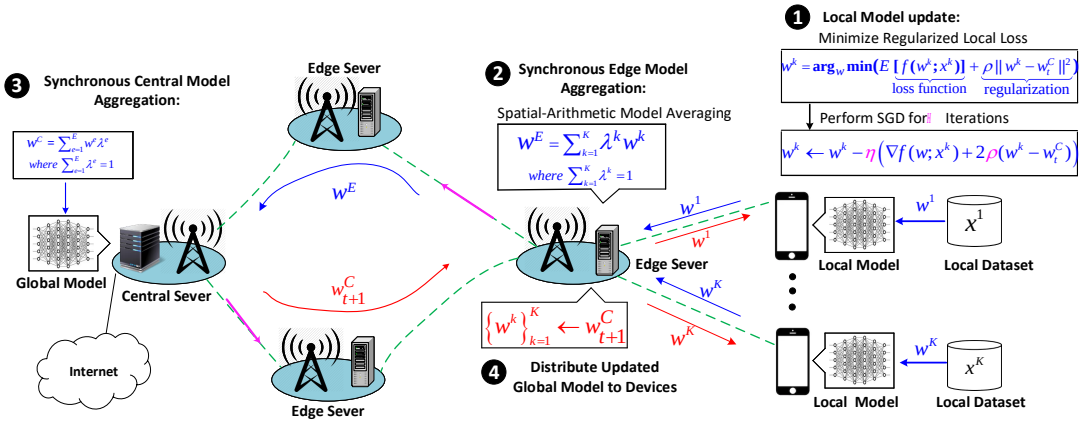


Figure 3.3: Hierarchical Synchronous Federated Learning System (HS-FL)

The standalone synchronous or asynchronous local SGD cannot achieve good accuracy-runtime trade off due to the adverse impacts of straggler and staleness. In addition, existing local SGD methods generally adopt single-layer server-worker architecture, where the central server is responsible for receiving and aggregating the model updates from all workers. This is not a scalable solution because the central server and its neighboring network links become the bottleneck of the whole FL system. We proposed Hierarchical Synchronous FL system (HS-FL), which is a variant of local SGD with **(1)** layered computation to improve system scalability.

Step 1. Local model update. As shown in Fig. 3.3, each worker tries to minimize **regularized loss function**, which contains a quadratic penalty term,

$$w^k = \arg_w \min(E \underbrace{[f(w^k; x^k)]}_{\text{lossfunction}} + \underbrace{\rho||w^k - w_t^C||^2}_{\text{regularization}})$$

where w_t^C is the current global model and ρ is the penalty parameter that determines how much deviations from the global model the local model is allowed to have. Minimizing regularized loss ensures that the local workers will not fall into the model update trajectories that are far away from the current global model, thus preventing the potential divergence of local SGD methods under IID and non-IID data distributions [50]. In this case, the model update via SGD at worker k follows

$$w^k \leftarrow w^k - \eta \left(\nabla f(w; x^k) + 2\rho(w^k - w_t^C) \right).$$

The above update rule can be substituted by momentum SGD [53], which is shown recently to have good stability and convergence improvements under non-IID. With two-layer aggregation, each edge server acts as the local aggregator to aggregate the model updates from its locally served edge devices, while the central server acts as the global aggregator to aggregate the synthesized updates from edge servers to update the global model.

Step 2. Synchronous edge model averaging. The edge server waits for the model updates from all the workers in its served area and then aggregates them into a single edge model w^E to maximize communication efficiency of the whole system, i.e.,

$$w^E = \sum\nolimits_{k=1}^{K} \lambda^k w^k$$

where $\sum_{k=1}^{K} \lambda^k = 1$. Besides communication efficiency, applying synchronous model averaging within the small service area of each edge server will not induce much

adverse impact of the stragglers because the single-hop wireless communication generally has much less delay compared with multi-hop communications. Moreover, synchronous model averaging will not create stale updates, thus helping us to improve the model accuracy.

Step 3. Synchronous global model averaging. Whenever the edge models arrive, the central server applies synchronous model averaging methods to continuously aggregate the edge models into the global model w_{t+1}^C following:

$$w^C = \sum_{e=1}^{E} \lambda^e w^e$$

Step 4. Global model distribution. The new global model w_{t+1}^C is sent back to the edge devices to replace their local models. As we only need to send global models to edge workers, the global traffic is significantly reduced for model exchange.

3.3 Optimizing FL Convergence via Reinforcement Learning

Our overall objective is to minimize the run-time convergence time to achieve a desired FL accuracy. Towards this goal, the optimal strategy is to minimize the worker-server delay of the slowest straggler, which experiences the maximum delay among all workers. However, in the highly dynamic wireless environments, the role of straggler can be randomly switched among different workers as time proceeds. In this paper, we sought a sub-optimal solution, where we minimize the average 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. As a result, the 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-optimal 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 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 *effective delay* between router i and the selected next-hop router. The effective delay, calculated by $d_{i,i+1}/pdr_{i,i+1}$, captures the joint effects of packet delivery delay $d_{i,i+1}$ and packet loss rate or packet delivery ratio ($pdr_{i,i+1}$) due to noisy wireless channel between router i and $i+1$. The packet delivery delay $d_{i,i+1}$ is the time interval between the time when packet arriving at router i and the time when the packet arriving at the next-hop router $i+1$. The packet delivery delay $d_{i,i+1}$, which contains the queueing delay, processing delay and transmission delay, is a random value real-timely measured 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 goal is to find the optimal policy π_i for router i so that the expected return $J(\boldsymbol{\pi})$ from the initial state (i.e.,E2E server-worker delay) is optimal, where $J(\boldsymbol{\pi}) = E[G_1|\boldsymbol{\pi}] = E[\sum_{i=1}^{T} r_i|\boldsymbol{\pi}]$ where $\boldsymbol{\pi} = \pi_1, ..., \pi_N$.

3.3.1 Delay-optimal Model Update 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 $\boldsymbol{\pi}$ to minimize the average server-worker delay. To implement the multi-agent reinforcement learning algorithm, we utilize the actor-critic-executor architecture pro-

posed in our previous work [40], where each router individually runs a local critic, a local actor, and a local executor.

Critic for Policy Evaluation

The performance of the policy π is measured by the action-value $q_i^{\pi}(s,a)$, which is a E2E TE metric. The action-value $q_i^{\pi}(s,a)$ of router i can be written as the sum of 1-hop reward of router i and the action-value of the next-hop router $i+1$, i.e.,

$$q_i^{\pi_i}(s,a) = E\left[r_i + q_{i+1}^{\pi_{i+1}}(s',a')\right]. \tag{3.5}$$

By applying exponential weighted average, the estimate of $q_i^{\pi_i}(s,a)$, denoted by $Q_i^{\pi_i}(s,a)$, can be updated 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.,

$$Q_i^{\pi_i}(s,a) \leftarrow Q_i^{\pi_i}(s,a) + \alpha[r_i + Q_{i+1}^{\pi_{i+1}}(s',a') - Q_i^{\pi_i}(s,a)] \tag{3.6}$$

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

Local Actor for Policy Improvement

Based on criticâs inputs, the local actor improves the local policy, which aims to maximize the cumulative sum of reward $J(\boldsymbol{\pi})$. This can be done by applying greedy policy, where each router i greedily improve its current policy π_i, i.e., select the action with the maximum estimated action-value,

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

Besides greedy policy, we also exploit two near-greedy policies to encourage exploration. The first one is $\epsilon-$greedy policy with exponential decay. With such policy, the router selects the greedy action defined in eq. (3.6) with with probability $1-\epsilon(t)$

and select other actions with probability $\epsilon(t)$. The ϵ decays exponentially as time proceeds, i.e., $\epsilon(t) = \epsilon_0\beta^t$, where $0 < \epsilon_0 < 1$ and $0 < \beta < 1$. The second near-greedy policy is softmax-greedy policy, where each action a is selected with a probability $P(a)$ according to the exponential Boltzmann distribution,

$$P(a) = \frac{\exp(Q_i^{\pi_i}(s,a)/\tau)}{\sum_{b\in\mathcal{A}_i}\exp(Q_i^{\pi_i}(s,b)/\tau)} \tag{3.7}$$

Local Executor

The local executor performs the actions according to the behavior policy, which is either same as the target policy π_i (on-policy learning) or similar to the target policy but more exploratory (off-policy learning). For off-policy learning, the the target policy is generally the greedy policy and the behavior policy is generally near-greedy to enable explorations.

3.4 FedEdge Design and Prototyping

3.4.1 FedEdge Overall Design

Existing FL experimental frameworks (e.g., TensorFlow Federated (TFF) [56] and PySyft [57]) only support simulations of federated learning without taking into account any networking impacts. In addition, the frameworks mentioned above were highly dependent on websockets for their communication, which leads to a reactive disconnection between the worker and server when their links experience a short-lived delay. Moreover, these frameworks can not be readily deployed on the physical edge devices without cumbersome modifications. To address above challenges, we develop and prototype FedEdge, which is the first experimental framework for wireless multi-hop FL. FedEdge allows us to fast prototype, deploy and evaluate novel FL algorithms along with RL-based system optimization methods in real-life wireless devices.

FedEdge has two unique features. The first feature is its modularity for both com-

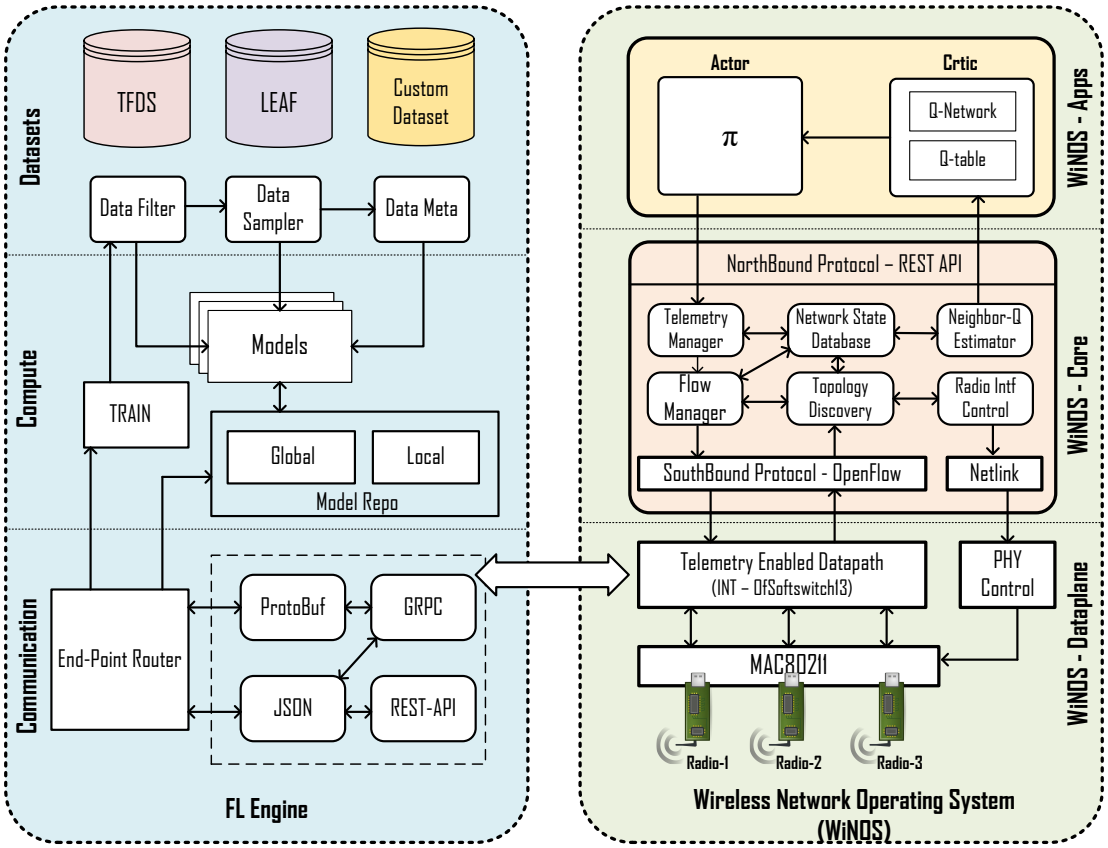


Figure 3.4: Architecture of FedEdge Framework

munication and computation functions. This allows programmers to from both worlds to simultaneously evolve and evaluate the complex FL system under diverse network and computation conditions. Second, FedEdge is a highly programmable experimentation platform that can be easily deployed on real wireless systems. With these two features, we can design and deploy new FL algorithms along with customized datasets, preprocessing schemes and training pipelines. Simultaneously, we can innovate on various networking mechanisms and test their impact on the performance on federated learning systems. Meanwhile, we can also specify different communication protocols used for the training execution, such as TCP/UDP, SYNC, or ASYNC HTTP connections

As shown in Figure 3.4, FedEdge consists of two key components: FL engine, which customizes and configures FL-related functions (e.g., FL training algorithm setup, model selection, and dataset preparation) and WiNOS, which is a distributed AI-oriented wireless network operating system. WiNOS is responsible for providing fast wireless networking connections between the aggregator and workers. What is more important, WiNOS is designed in nature to facilitate AI-empowered networking optimization, including customizable actor-critic RL agent for instantiating a variety of AI-enabled routing algorithms, in-band telemetry that enables cost-efficient data collections for online RL training, and programmable routing table (datapath) for real-time RL policy executions.

Each FedEdge component follows a layered design. In particular, FL engine consists of three layers. (1) FL Datasets layer stores the datasets for federated training (2) FL compute layer provides vital functions to train the model and save the models and (3) FL communicate layer establishes logic and reliable connections between workers and aggregator by using the proposed FedEdge communication protocol (FedEdge COMM). AI-WiNOS is composed of 3 layers. (1) WiNOS dataplane incorporates SDN software switch with our proposed in-band telemetry scheme to enable programmable

packet forwarding, while simultaneously providing low-cost real-time collection and reporting of network state measurements. (2) WiNOS Core provides essential networking services and functions. such as topology discovery, network state database, and traffic flow management. (3) WiNOS application layer hosts the actor-critic RL agent to enable delay-optimal routing between the aggregator and the workers.

3.4.2 FL Engine

3.4.2.1 FL Datasets

Federated learning is based on the well-established machine learning technique with a key significant difference in how the data is used to train the models. One of the FL techniqueâs primary objectives is to protect the userâs privacy by avoiding sharing data from the origin device. Firstly, without reinventing the wheels, we leverage the existing datasets hosted by Tensorflow TFDS [58] and LEAF [59], both of which are well known in the FL community. We also provide extended APIs to incorporate custom datasets specific to the experimenter to integrate within our FL framework.

Dataset-Setup

In our FL training pipeline, the first step is to distribute the datasets to different workers in the training process. To do so, the experimenter has to specify three primary parameters, which are (1) Number of workers involved and (2) the Data distribution type - I.I.D or non-I.I.D and (3) Dataset name and the repository of the dataset (TFDS, LEAF, or CUSTOM). Next, our FedEdge Datasets module will split the dataset respectively and transfer them over to the worker/clients.

Pipeline

In the FL training process, the experimenter can choose the data for training in an arbitrary fashion for each global round. For each global round, the aggregator may specify the class and number of samples used for the current epoch. With such flexibility in the training process, we provide a highly customizable pipeline for consuming

data during the training process with two sub-modules including data filtering and sampling. At each round of global training step, the aggregator can pass parameters such as the data class to filter and sampling technique for the training process. In this case, the sampling procedure will determine either the entire filtered data should be used or have to be sub-sampled to limit the number of samples consumed for the training step. To simplify the above-mentioned process, we consider all datasets will contain a META which provides the key statistics about the dataset, such as the number of classes and number of samples for each class and in-total.

3.4.2.2 FL Compute

Our motivation for designing FedEdge compute module is to provide an extensible processing stack such that it is easy to customize the training pipeline without being limited to the communication protocol. Compute module is comprised of 3 stages of processing: (1) Data Staging (2) Model Repo and (3) Model Trainer. In the following section, we will briefly describe the functionality of each stages.

Data Staging

Data for the worker is selected at the beginning of the FL training procedure. In this data staging process, the user can further preprocess the data using FedEdge's in-built functions to encode, normalize and standardize the data. Such synthesized data is then gathered into batches by the batch size specified during the initial training epoch. Furthermore, batches are then converted to a Tensorflow accelerated data pipeline to avoid the IO bottleneck, thereby accelerating each epoch runtime.

Model Repo

Federated training procedure mandates frequently exchanging model under training between worker and the aggregator. Some FL algorithms require the model of current round to be updated using the models from the previous training rounds. To facilitate such procedure, we implemented a model repo that stores the global and local models

for a specified time duration. Each model is timestamped before writing to the repo so that it is easy to distinguish the current model and historical models.

FL train

It involves two types of nodes that perform different operations. To further simplify the context of FedEdge compute, we describe the functions of FedEdge compute based on the nodeâs role in the training process and the sequence of communication among the nodes are shown as the sequential flow in 3.5. The detailed operations of FL training are shown in Algorithm 1 and 2.

FedEdge aggregator node is the central node that controls the life cycle of FL training. The aggregator has a crucial role in initiating, coordinating, and monitoring the FL training cycle. Before beginning the FL training cycle, each worker should register their IDs, a combination of IP and port numbers to the aggregator's worker registry built into the FedEdge Communicate End-point router module. This registry is constructed using a hash map that stores worker ID as the key and their communication HTTP/GRPC end-point as the value. Only registered workers can participate in the training cycle. The aggregator performs a two-stage process to launch the FL training sequence: (1) Construct the model and upload it to the workers (2) Launch the training with the user-supplied training configuration. In the following section, we briefly discuss the two-stage process:

- *Stage-1:* Our FedEdge compute module provides in-built models for image classification tasks using convolutional neural network (CNN) and multi-class logistic regression (MCLR). Besides, users can easily modify the in-built models and integrate their customized loss functions. Each model will then be trained with the user's choice of optimization algorithms. Currently, FedEdge compute module provides in-built support for the regularized local SDG algorithm by default, which can be considered as the generalized FedAvg. This newly created model is then shared with the workers for training. To send the model, the

Algorithm 1 Local SGD - Aggregator

Input: maxround r_m, worker epoch H_k, $k \leq N$, batch size B
Output: Global Model W^{c}

AGGREGATOR PROCESS

Initialize Worker Registry: R
Initialize Current Model Queue: Q_{fresh}
Initialize Worker State Queue: Q_s
WORKER NODE REGISTRATION
On request for register
$K \leftarrow workerid$
for round $r \leq r_{\mathrm{m}}$ **do**
  if r = 1 **then**
    Initialize : w^{c}
    for k in R **in parallel do**
      $updateWorker \leftarrow w^{\mathrm{c}}$
      $Q_{\mathrm{s}} \leftarrow GLOBAL\_MODEL\_RECV(k)$
    end for
    for k in R **in parallel do**
      Initiate training sequence : w_{t}
      $Q_{fresh} \leftarrow w^k \leftarrow train(k, H_{\mathrm{k}}, B)$
      $Q_{\mathrm{s}} \leftarrow LOCAL\_MODEL\_RECV(k)$
    end for
  else
    Perform Model Aggregation
    $w_t^c \leftarrow \sum_{e=1}^{E} \lambda^k w^e$
    for k in R **in parallel do**
      $updateWorker \leftarrow w^c$
      $Q_{\mathrm{s}} \leftarrow GLOBAL\_MODEL\_RECV(k)$
    end for
    for k in R **in parallel do**
      Initiate training sequence : w_{t}
      $Q_{fresh} \leftarrow w^k \leftarrow train(k, H_{\mathrm{t}}, B)$
      $Q_{\mathrm{s}} \leftarrow LOCAL\_MODEL\_RECV(k)$
    end for
  end if
end for
return W^{c}

Algorithm 2 Local SGD - Worker

Input: workerid k, worker epoch H_k, batch size B
Output: Local Model w^{k}

Initialize Status Queue: Q_w
Initialize Dataset Store: D_S
REGISTER(workerid)
FUNCTION *train*(k, H_{k}, B)
$Q_w \leftarrow TRAINING\_STARTED$
while i $< H_k$ **do**
  for bs in $\mathrm{D_s}$ **do**
    $w^k \leftarrow w^k - \eta \frac{1}{B} \sum_{x^k \in \mathcal{I}^k} \left(\nabla f(w^k; x^k) + 2\rho(w^k - w_t^C) \right)$
  end for
end while
$Q_w \leftarrow TRAINING\_FINISHED$
return w^k to AGGREGATOR

aggregator revisits the worker registry to obtain each workerâs HTTP/GRPC end-points. If the worker successfully received the model, then the corresponding worker state will be updated based on the message from the worker with the context $GLOBAL\_MODEL\_RECV$.

- *Stage-2:* To begin the FL training cycle, user needs to supply the following parameters at the minimum: global rounds, local rounds per node, dataset repo and dataset, model to train, and data partition type. The global round controls the total number of rounds workers will use to train the shared model, and local rounds define the number of epochs each worker will use to update it local model. Since our FedEdge framework hosts datasets from a variety of repo's, users should specifically mention the dataset and the repo used for the experiment. With the user-supplied parameters, a training cycle is constructed where a cycle is defined as (1) launching a global round (2) wait to receive models from all workers and updates the state of the corresponding worker to $LOCAL\_MODEL\_RECV$. (3) perform model aggregation, and (4) finally share the aggregated model with the workers. This cycle will terminate once the required number of global rounds are reached or when the target accuracy

is achieved.

FedEdge worker node operates on the request of the aggregator node. FedEdge aggregator interacts with the worker node to initiate the training sequence by sharing the global model and initial training parameters. The model received from the aggreagator node will be stored as a global model into the model repo. Worker node loads the global model and creates its copy of the local model by cloning. This local model is then used to train over multiple epochs with the supplied optimization algorithm to minimize the local loss function. Finally, the updated model is then shared with the aggregator.

3.4.2.3 FL Communication

The reliability of the FL system is heavily dependent on the robust communication stack for collaborative learning. As more research efforts are focused on improving the overall FL system performance by improvising computation, little to no effort has been targeted towards optimizing the communication layer. We believe the lack of a flexible platform is one of the inherent challenges to pursue research in this venue. Extendability and ease of programmability as the objective, we adopted modular design which consists (1) End-point router and (2) FL-Transport Layer as the submodules. In the following section, we will briefly discuss the functionality of the modules as mentioned earlier:

End-point router

FedEdge platform exposes functions for FedEdge compute and dataset layers through End-Point router (EPR). This routing layer routes messages from external nodes to the respective processing function to handle data pipeline, model training, and model exchange. Since each FedEdge node provides some service in-addition to consuming a service, EPR is designed to function in a dual role such as a server and client. In the client role, the FedEdge node can send registration requests, upload local models, and

reply to the aggregatorâs status query. On the other hand, it accepts requests from the aggregator to preprocess the dataset, launch training, and receive a global model in the server role. To adopt a synergy in FedEdge nodes, we proposed a communication protocol. The general idea of the FedEdge communication protocol (FedEdge COMM) is to define the messages that should be exchanged between the aggregator and worker nodes. Besides, the protocol also defines the status flags that should be set by each node based on the role type and its current status. The communication flow of our FedEdge COMM is shown in figure 3.5. Following the FedEdge COMM, the aggregator node first initializes the port for accepting connections from the worker. Besides, the connection state tracker is initialized to keep track of the nodes communicating with the aggregator. After this phase, worker nodes register their IP address and device ID to the worker registry within the connection state tracker module. Once the aggregator has received the training resources, it will initialize the global model and share it with all the workers within the worker registry. If the workers successfully received the model, a notification will be sent to the aggregator. If the aggregator determines it has the required number of workers for the training phase, then the training request is dispatched to all workers with the batch size and number of rounds. On receiving training requests, the worker initiates a training round and updates the local model to the aggregator at the end of the training.

FL-Transport Layer

FedEdge nodes on Wireless multi-hop networks are prone to experience dynamic network conditions that might cause unreliable transport layer operation. With modularity at the core of the FedEdge framework design, we let programmers freely define the underlying transport layer mechanism for the FedEdge COMM. While EPR submodule exposes FedEdge compute and dataset layersâ functions, these APIs are accessible based on the transport layer of FL (FL-Transport). With service-based architecture as the design principle behind FL-transportâs transport layer, we have

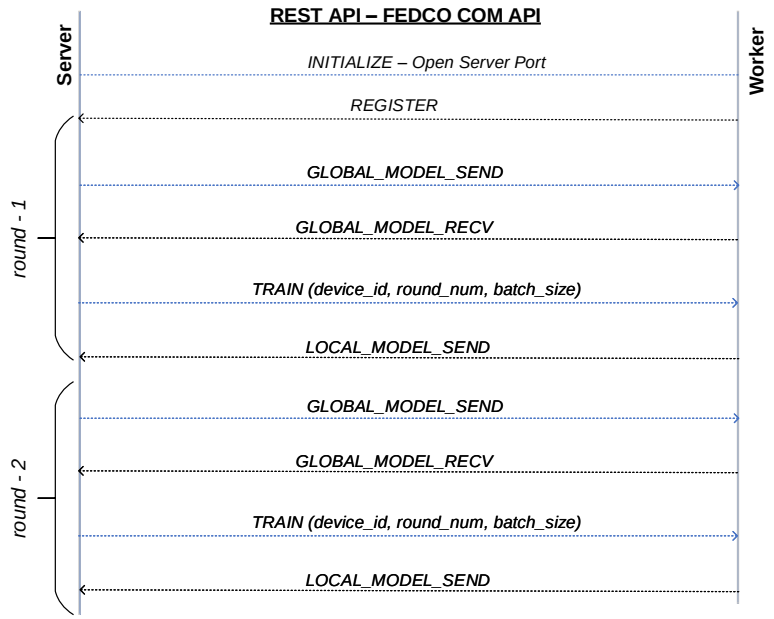


Figure 3.5: FedEdge Communication Protocol

adopted two communication mechanisms (1) HTTP- REST API and (2) GRPC.

HTTP-REST API: The widely used service based infrastructure protocol, HTTP-REST API operates on the principle of standard request/response format. Our HTTP- TCP REST API is constructed on top of the TCP protocol stack. Therefore, we extend REST APIs programmability to configure the number of TCP streams, session control, and keep alive. To simplify the payload transport, we utilized JSON message format to transport the FL model. In addition to the FL model, the user can easily extend the API to add additional attributes such as the time stamp of the model and the total time take to complete training by defining new key/value pair before JSON encoding. To simplify the implementation efforts, we adopted a range of frameworks such as FastAPI [60], Asyncio [61] and HTTPx [62] for building the communication protocol. FastAPI framework facilitates application developers to implement REST API for functions that can be invoked by remote nodes using HTTP as the transport protocol. Asyncio and HTTPx enable the developers to im-

plement asynchronous HTTP clients so that the aggregator can communicate with all workers within the cluster concurrently. While server function within the aggregator and worker is built around FastAPI and HTTPx, client functions are built on top of AiOHTTP and Asyncio.

GRPC: Google Remote Procedure Call [63] is a high performance framework for calling remote methods by passing parameters alike local functions. The core idea of the GRPC framework is to define a service that will implement the interfaces which can be called by the remote entities to execute an operation. In FedEdge framework, we leveraged GRPC as the communication channel between the server and all workers. Besides, dual message format is supported for over the channel such using either JSON or Protocol buffers. While JSON encodes messages as texts, protocol buffers uses compiler to transcode structured data into serialized byte streams. In addition, there exists native support for asynchronous execution, HTTP2 and data compression, which significantly reduce the overall traffic volume in wireless multi-hop FL.

3.4.3 WiNOS: AI-oriented Wireless Network Operating System

In the previous section, we detailed our design and implementation of FL engine. 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 above two requirement, we adopts WiNOS, which is a distributed AI-oriented wireless network operating system proposed in our previous work [64]. WiNOS is built by exploiting software-defined networking technologies and in-band network telemetry. In particular, WiNOS is composed of three layers (1) WiNOS dataplane (2) WiNOS Core & (3) WiNOS Apps - Generic RL

framework.

3.5 Experimental Evaluation

Our overall objective in this evaluation study is to show that RL-based networking can efficiently improve the convergence speed of FL algorithms in a physical testbed with our proposed system design. Towards this objective, first we show that FL algorithms convergence time can be improved presence of stragglers if the local model updates are regularized. Then, we show that FL algorithms convergence can be accelerated further when the underlying network routing paths are optimized using RL routing algorithms. Finally, we study the convergence time of our proposed FL system under single-layer and hierarchical synchronous architecture with RL routing.

Models and Dataset: To evaluate the performance of Federate learning, we used FedAvg, which trains a deep learning model on the MNIST [65] digit recognition task. The training experiments are performed over two separate models whose weights get updated using federated learning.

> **MNIST CNN:** we used a CNN with 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. **MNIST LSTM:** we used two stacked LSTM layers each with 128 hidden units. We added a dropout layer after each LSTM layer. The LSTMs were followed by a fully connected layer with 32 units and Relu activation, and a fully connected layer with softmax activation was used as the model output layer. The model has a size of 0.8 Mbytes.

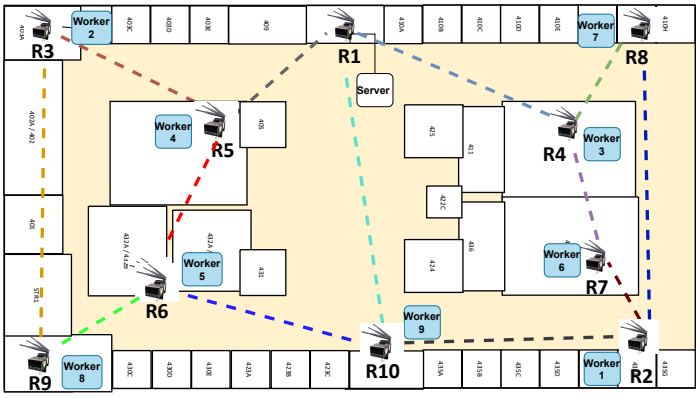


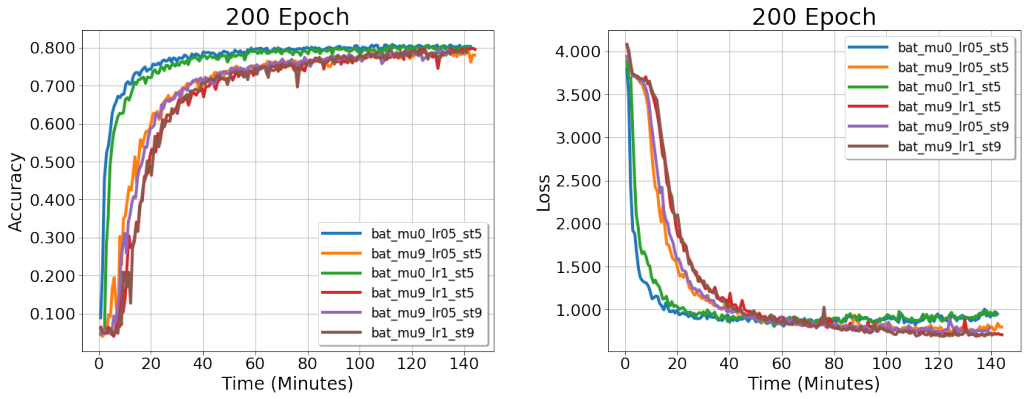
Figure 3.6: Testbed - Topology

3.5.0.1 Regularized local SGD:

In this evaluation, we show that by performing regularized model updates, we can improve the convergence in presence of heterogeneous workers. We study the effect of stragglers by letting each worker to process less number of local rounds, which will lead to model divergence because of varying number of local updates by workers. Then, we show that the straggler effect is evident from the noisy updates when the model is unregularized (i.e., $\mu = 0$). And then we show that, when the local model updates are regularized (i.e., $\mu > 0$), the convergence is less noisy and prevents the model divergence.

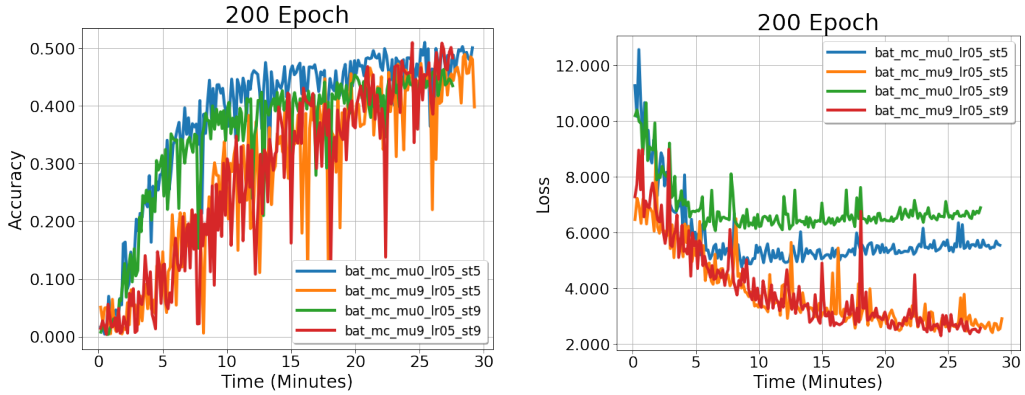
Testbed Setup: As shown in Fig.3.6, a software-defined wireless mesh network testbed was deployed on the 4th floor of Woodward Hall at UNCC. This testbed consists of 10 Nvidia Jetson Xavier nodes connected to Gateway 5400 multi-radio wireless nodes as shown in figure 9. Each Nvidia Jetson node serves as FedEdge node with FL engine for FL training. On the other hand, Gateworks routers serves as FedEdge-node with the programmable wireless network operating system on top of Ubuntu 20.04 Linux.

Analysis: We trained the FL model under different straggler constraints to 50% and 90%, which limits the number of local SGD updates in the workers. This variation



(a) FEMNIST CNN - Accuracy (b) FEMNIST CNN - Loss

Figure 3.7



(a) FEMNIST MCLR - Accuracy (b) FEMNIST MCLR - Loss

Figure 3.8: Convergence of CNN and MCLR model with regularized local SGD

in stragglers will affect the global model update noise, potentially leading to degraded convergence. In addition, we used different learning rate combinations to see the stragglers effects. First, fig:**??**, we concluded that when CNN models are trained with higher learning rate $lr = 0.1$, the global model tends to diverge much quickly when compared to much lower learning rates. However, when local SGD updates are regularized (i.e., $\mu = 0.9$), we can prevent the model divergence. Though the

Table 3.1: FL Hyperarameters

Parameter	FEMNIST CNN	FEMNIST LSTM
Number of global rounds	50	50
Batch size	32	32
learning rate	0.05	0.05
Model size	5.8 Mbytes	0.8 Mbytes

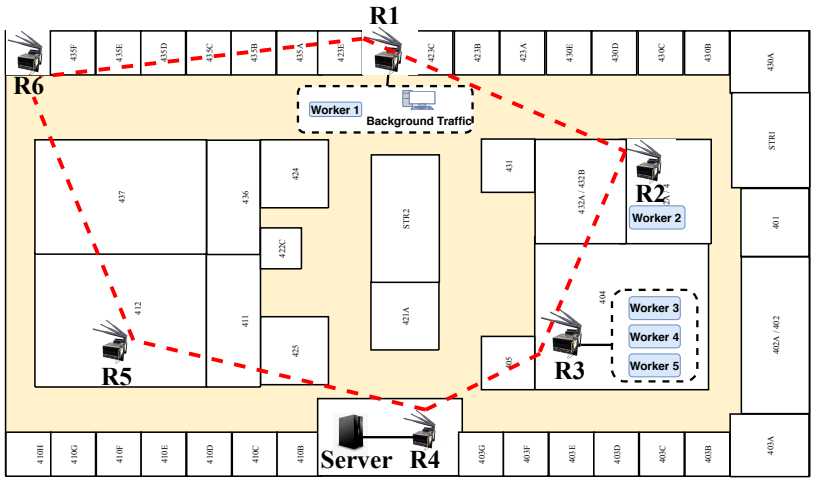


Figure 3.9: Testbed - Topology

quantifiable loss is only marginal in CNN, for simpler models like MCLR model divergence is accelerated as we can see from fig: 3.13. In this analysis, we concluded that stragglers divergence is more evident for simple model such as MCLR and it can be prevented by regularized local SGD updates.

3.5.0.2 Delay Optimal Model Update with RL routing:

Testbed Setup: As shown in Fig. 3.9, a software-defined wireless mesh network testbed was deployed on the 4th floor of Woodward Hall at UNCC. This testbed consists of six Nvidia Jetson Xavier nodes connected with WLE900VX wireless interface card and WINOS system on top of Ubuntu 18.04 Linux operating system running on each Nvidia Jetson node. Each mesh router was configured to operate in Mesh point (MP) mode, with fixed 5 Ghz channel, 40 Mhz channel width in 802.11ac operating mode, and 30 dBm transmission power. The Nvidia Jetson not only serves as a wireless mesh node to form a wireless multi-hop backbone, but also host local workers to train the federated learning model. We deployed five local workers on three routers (R1, R2, and R3), where each local worker has its own IP address. The server is connected to R4 to run the global model updates as shown in Fig. 2.9. Worker 1 and a background traffic client were attached to R1. Worker 2 was deployed on R2. We

Table 3.2: FL Hyperarameters

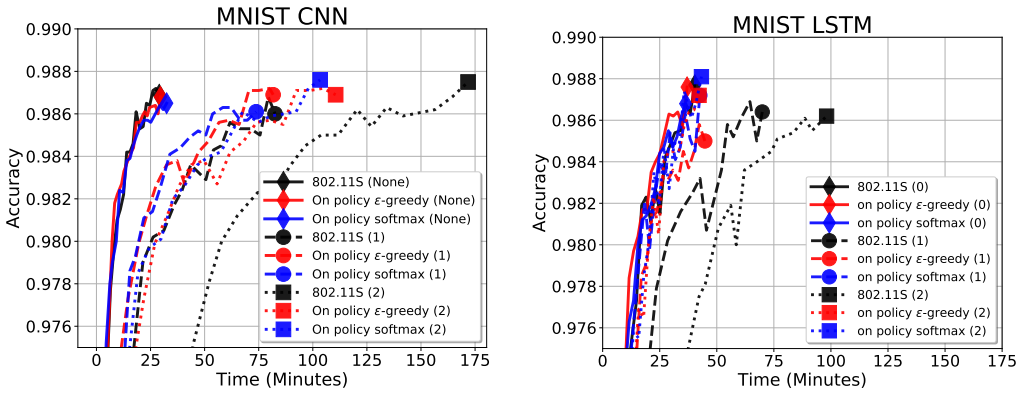
Parameter	MNIST CNN	MNIST LSTM
Number of global rounds	20	20
Number of local iterator	1	1
Batch size	32	32
learning rate	0.01	0.01
Model size	5.8 Mbytes	0.8 Mbytes

deployed the other workers on R3.

We study the FL convergence time under different network congestion conditions by varying the *background traffic intensity* from none to 1 Mbps and 2 Mbps respectively from a client at R1 and with *different model complexities*. The background traffic is generated by the client at R1 and sent to server following a fixed routing path of (R2 $\rightarrow$ R3 $\rightarrow$ R4). To emphasize the impact of networking, we use the exactly same parameters shown in Table 3.2 for FL across different experiments.

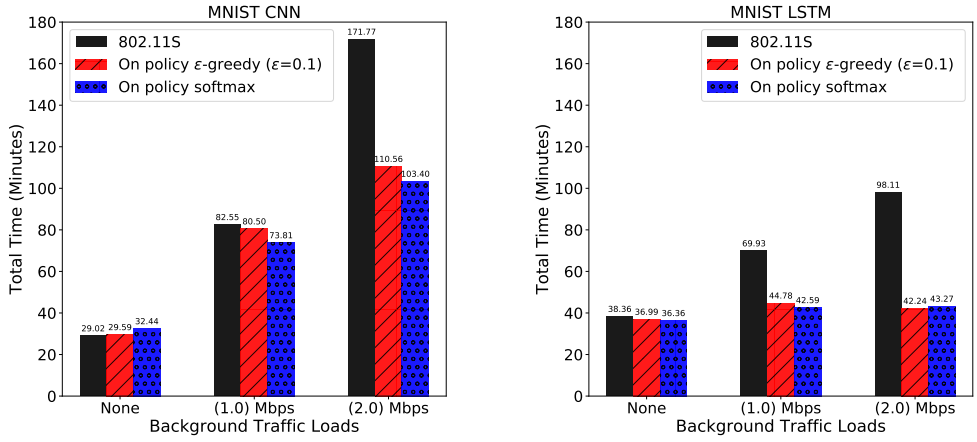
Analysis: Fig. 3.10 and Fig. 3.11 depict the FL performance in accuracy and convergence time when varying the network traffic load and model complexity. When there is no injected background traffic (solid line with diamond marker), FL traffic can fully utilize all the network resources without any interference from other application traffic. As the straggler (worker 1) can freely send the FL packets to the right path straight away, which is faster than the left one, we can observe that all algorithms achieved a similar performance in terms of accuracy with (98.7%).

In the case of 1 Mbps background traffic load, both on-policy softmax and $\epsilon-$greedy RL-based routing algorithms performed slightly better than the baseline (802.11s) in terms of the total convergence time. Both RL-based routing algorithms converged to select the left path and learned to avoid sending the FL traffic of worker 1 at R1 to the right path, which is congested by a continuous background traffic flow and FL traffic from the workers (2-5). However, the benefit of selecting the left path is not that evident in this case because the end-to-end throughput of the left path was less



(a) MNIST CNN (b) MNIST LSTM

Figure 3.10: Comparison results after 20 epochs of accuracy over time of 802.11s routing (black), On-policy $\epsilon-$greedy (red), and On-policy softmax (blue) RL-based routing, respectively, by varying the load of background traffic from None, (1) Mbps, and (2) Mbps (solid, dashed, dotted)-lines.



(a) MNIST CNN (b) MNIST LSTM

Figure 3.11: Total convergence time comparison of 802.11s routing, On-policy $\epsilon-$greedy, and On-policy softmax.

than the right path and the model size of MNIST CNN (5.8 Mbytes) is relatively large,

The performance gain significantly increased when the background traffic increased to 2 Mbps (dotted-line with square marker). Since 802.11s is a layer 2 unicast routing protocol, only the destination MAC address is used to route the packet without taking into account the traffic source information. As a result, 802.11s was not able to distinguish the background traffic flow from FL traffic from worker 1. Therefore, it forwarded both background and FL traffic flows to the same link/path, which increased the communication time between straggler (worker 1) and the server. As shown in Fig. 3.11(a), 802.11s took almost up to 3 hours (171 Minutes) to finish the 20 rounds of training whereas both RL-based routing algorithms took less than 2 hours (110 Minutes) because they learned to optimally distribute different flows among different routing paths.

By varying the model complexity, we investigate how computational and communication overhead affect the FL performance. For both CNN and LSTM, when there is no communication overhead, CNN enjoys the fast local computation. Thus, we observe about 30 minutes of convergence time for CNN and a bit longer 37 minutes for LSTM. However, when the background traffic is injected, the simplicity of the model benefits as it lowers the communication overhead. We can observe that the overall time is greatly reduced with LSTM (even with 802.11s).

Our experiments showed that RL-based networking can efficiently improve the convergence performance of FL algorithms especially when model is complex and the network traffic load is high. Interestingly, the results also show that the choice of model complexity can be another factor to affect the performance of FL.

3.5.1 Hierarchical Synchronous FL

Testbed Setup: As shown in Fig.3.12, a software-defined wireless mesh network testbed was deployed on the 4th floor of Woodward Hall at UNCC. This testbed

consists of 10 Nvidia Jetson Xavier nodes connected to Gateway 5400 multi-radio wireless nodes as shown in figure 9. Each Nvidia Jetson node serves as FedEdge node with FL engine for FL training. On the other hand, Gateworks routers serves as FedEdge-node with the programmable wireless network operating system on top of Ubuntu 20.04 Linux.

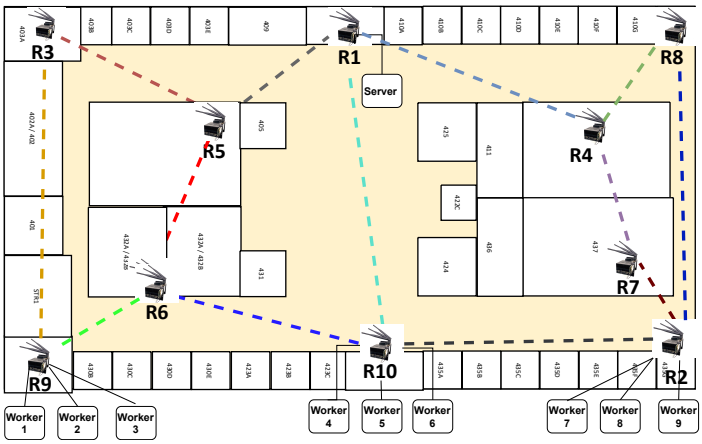
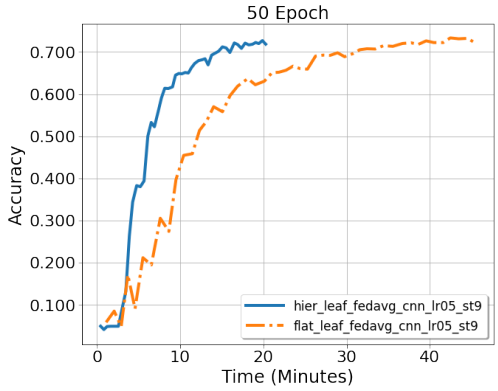
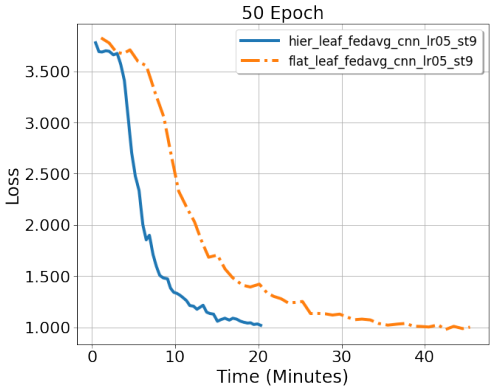


Figure 3.12: Hierarchical - Topology

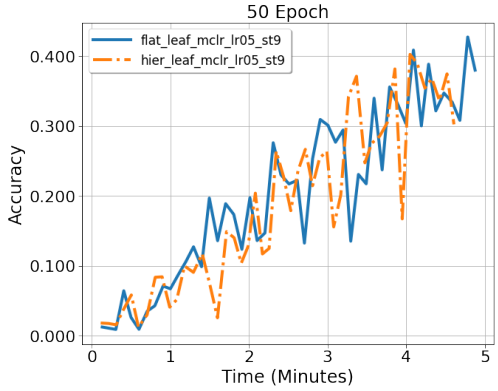
Analysis: In this study, we analyzed the FL convergence and wall clock time under hierarchical and single layer architecture. For flat architecture, we used the same topology as in subsection 3.5.0.1. For hierarchical architecture, we used the topology as shown in fig: 3.12. In single layer architecture central server have to communicate with all 9 workers independently, whereas in hierarchical architecture central server only communicates with the edge server and edge server will communicate with the edge worker for the training. In addition, to study the effect of communication bounds in both the architectures, we evenly distributed the workers for each edge server. From **??**, we can see that our proposed hierarchical architecture reduces the convergence time and overall training time for 50 global rounds. On the other hand, flat architecture takes twice the time to reach the same accuracy.



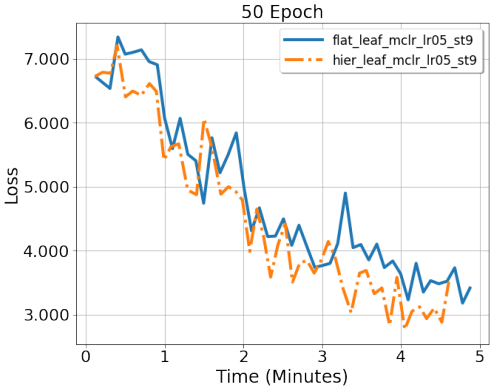
(a) FEMNIST CNN - Accuracy



(b) FEMNIST CNN - Loss



(c) FEMNIST MCLR - Accuracy



(d) FEMNIST MCLR - Loss

Figure 3.13: Convergence of CNN and MCLR model with regularized local SGD

3.6 Conclusion

FL over wireless multi-hop networks is challenging due to dynamic network performance resulting in non-optimal routing paths and high communication delay. To maximize the FL accuracy with minimum convergence time, we proposed MARL methods as model-free optimization approaches, where the distributed routers exploit their instantaneous local experiences to collaboratively tune networking parameters on-the-fly. To analyze the convergence of FL system with MARL routing solution, we developed FedEdge, a modular wireless edge system for federated learning with programmable network control. Our experimental results show that the RL-routing algorithms have a great potential to accelerate the convergence of FL in the wireless multi-hop networks, compared with the widely-adopted standardized IEEE 802.11s protocol. To the best of our knowledge, this is the first work to prototype, optimize

and demonstrate the wireless multi-hop FL system.

CHAPTER 4: Domain-invariant Gait Recognition via Millimeter Wave Radar

4.1 Overview

Radar-based biometric identification is an emerging user identification platform that exploits radar return signals to capture human biometrics (such as gait, gesture, lip motion, and cardiac motion), which can be used to predict a user's identity. Despite its unique advantages (such as privacy-preserving and resilience to weather/lighting conditions), the generalization performance of this technology is still unknown and greatly hinders its practical deployment. To address this challenge, we collect and investigate a non-synthetic dataset, which revealed the existence of distinct spatial and temporal domain shifts in radar-based gait biometric data. We show that spatio-temporal domain shifts, when not addressed jointly, can significantly degrade identification accuracy. Moreover, we propose a data-efficient yet straightforward domain shift mitigation approach for tuning deep learning models over their entire life cycle. Our approach exploits an unsupervised domain shift detector to measure the malignancy of domain shifts. Such metrics allow us to determine the domains that maximize the net contributions upon adapting to, after which an appropriate domain adaption method is utilized to improve both spatial and temporal generalization. We show that our approach improves data efficiency by reducing the number of domains that necessitate adaptation while maintaining the generalization performance of a blind approach that uses data from all domains. The content of this chapter is partly reprinted with permission from P. Janakaraj, K. Jakkala, A. Bhuyan, Z. Sun, P. Wang and M. Lee, "STAR: Simultaneous Tracking and Recognition through Millimeter Waves and Deep Learning," 2019 12th IFIP Wireless and Mobile Networking

4.1.1 Challenges

The root cause of this generalization issue is dataset shift [66]. Simply put, it occurs when the testing data distribution differs from that of the training data, i.e., the i.i.d. assumption is violated. Depending on the type of difference in the test data distribution, it can be categorized as a covariate, posterior probability, or concept shift. Mitigating dataset shift, also known as dataset drift or domain shift, is known to be one of the hardest problems in machine learning and is prevalent in numerous application domains. This leads us to the ethos of Domain Adaptation, addressed by myriads of methods in literature [67].

After an in-depth analysis of dataset shifts and domain adaptation, we found the current taxonomy of domain shifts is not adequate to explain the full extent of our problem. Apart from the manifestations of domain shifts mentioned above, we seek to distinguish them further. Concretely, we acknowledge the presence of spatial and temporal domain shifts (SDS/TDS). We treat TDS as shifts that arise temporally as a consequence of the inherent dynamics of a domain while, treating SDS as shifts induced from introducing new spatial locations. This distinction allows us to understand and explain the behavior of the shifts that manifest in radar-based gait data.

Before addressing a shift, we need to be aware of its presence. An overwhelming amount of domain adaptation methods only address shifts that behave similar to SDS [67], i.e., they only consider explicitly introduced domains. So, the problem of detecting the presence of a shift, which is a sine qua non for TDS, has not received the attention it deserves. Moreover, among the work available on drift detection [68], a majority of them make assumptions about the behavior of softmax classifiers, which have long been invalidated [69, 70]. We draw attention to a simple yet effective unsupervised domain shift detector based on metric learning [71].

4.1.2 Our Solution

In this chapter, We draw attention to a simple yet effective unsupervised domain shift detector based on metric learning [71]. Our shift detector enables a data-efficient domain shift mitigation approach, where the shift malignancy of each domain is first measured, and domains that maximize the net contributions upon adapting are selected. An appropriate domain adaption method is then applied only to the selected domains to improve model generalization. Our prominent contributions and findings are as follows:

- We curated a non-synthetic dataset consisting of radar-based gait biometric data. This dataset, for the first time, allows one to study and improve the spatio-temporal generalization performance of a radar-based biometric identification system.
- We introduced the distinction of TDS and SDS, which enables us to explain the system performance degradation.
- We uncovered a correlation between TDS and SDS, which unveiled significant ramifications for data collection and domain adaptation. In particular, we found SDS can be mitigated to a certain extent by using the TD data and vice-versa, but both SDS and TDS have to be addressed explicitly for consistent generalization performance.
- We revealed that if multiple sources of domain shifts appear, each source of domain shifts yields a different net contribution to generalization upon adapting to that domain.
- We elucidated the effectiveness of metric learning based shift detectors and reaffirm the limitations of softmax thresholding for out of domain detection.

- Finally, we exploit our shift detector to develop a data-efficient domain shift mitigation approach.

4.2 System Design

4.2.1 Preliminaries

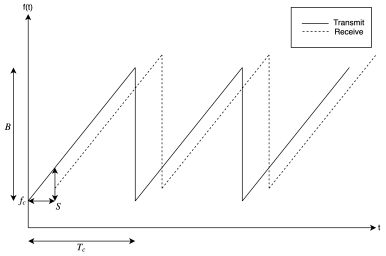


Figure 4.1: FMCW signal with linear ramp

The fundamental concept in radar systems is the transmission of a signal, which is reflected by the objects in its propagation path. The key advantage of Frequency Modulated Continuous Wave (FMCW) radar system is its capability of measuring the range/location and the velocity of the moving target simultaneously. In particular, the signal used in FMCW radars is called chirp, whose frequency increases linearly with time as shown in Fig. 4.1. The chirp is characterized by a start frequency (i.e., carrier frequency) f_c, bandwidth B and duration T_c. The slope of the chirp $S = B/T_c$ characterizes the frequency changing rate. The transmitter of the radar sends a chirp signal and the receiver captures the reflected chirp generated by the object. A frequency mixer combines the transmitted and received chirps to produce the beat signal. For a target at distance d, the beat signal can be described by

$$A\exp(j2\pi(f_0 t + 2d/\lambda_c) = A\exp(j2\pi(2d/\lambda_c))\exp(j2\pi f_0 t) \tag{4.1}$$

where A is the signal attenuation gain. $f_0 = 2dS/C$ is called beat frequency. $\lambda_c = C/f_c$ is the wavelength of the carrier frequency where C is the speed of light. Exploiting the beat frequency f_0, the distance of the target can be easily obtained by

$$d = \frac{f_0 C}{2S}. \tag{4.2}$$

When the target is moving to a new location, the frequency of beat signal will be changed if the distance between the previous location and the new location is larger than the radar range resolution, which is directly related to the bandwidth B, i.e.,

$$d_{res} = \frac{C}{2B}. \tag{4.3}$$

Due to extremely high carrier frequency of mmwave FMCW radar, very large bandwidth B can be used, which leads to fine-grained range resolution.

While the frequency of the beat signal is used to measure target distance, the phase of the beat signal $\exp(j2\pi(2d/\lambda_c))$ defined in eq. (4.1) can be exploited to measure the target velocity even if the radar can not detect the location change of the target due to the range resolution constraint. Assume the target moves over an very small distance after N chirps are sent and received, which generates N beat signals. In this case, all N beat signals will have the same beat frequency but with different phases. In particular, the phase of the beat signal $n \in (1, 2, ..., N)$ is equal to

$$\exp(j2\pi(2v/\lambda_c)nT_c) \tag{4.4}$$

where $2v/\lambda_c$ is called Doppler frequency. The eq. (4.4) indicates the phases of beat signals for each distinguishable distance constitute a new signal whose carrier frequency is exactly the Doppler frequency. When a human subject is walking, the different human parts (e.g., arms, legs, foots, and torso) move at different velocities, which lead to different Doppler frequencies. The radar can distinguish the movements of two parts only if their speed difference is larger than the velocity resolution, which is determined by the observation duration NT_c and the carrier wavelength λ_c, i.e.,

$$v_{res} = \frac{\lambda_c}{2NT_c}. \tag{4.5}$$

Since mmWave radar has a wavelength in the scale of millimeters, it characterizes fine-grained motion dynamics of human gait. For example, compared with sub-6GHz radar, the 77GHz mmWave radar can achieve over 12 times higher velocity resolution.

Why DNN for mmWave Gait recognition?

The key challenges to exploit mmWave micro-Doppler signatures for gait recognition are two-fold. First, the micro-Doppler signatures come from the reflections that directly bounce off from human bodies. However, mmWave signals have very short wavelengths and thus can be easily reflected back by surrounding obstructions. These reflections carry the environment-dependent information, which is harmful for the recognition task and has to be removed. In particular, these environment reflections include static ones, which are directly induced by the stationary obstructions (e.g., walls) and dynamics ones, which indirectly bounce from other stationary obstructions and then bounce from human bodies. The harmful environment reflections carry delayed and distorted micro-Doppler information. Second, mmWave gait biometric is of high dimensional data. Therefore, heuristically selecting features from such data is suboptimal, which may fail to characterize the salient and discriminative patterns to distinguish a large number of people from each other. This naturally requires us to implement automatic feature extraction by exploiting deep neural networks.

4.2.2 Radar Data Processing

Our system consists of three subsystems including human target detection, localization & tracking and human target recognition.

Firstly, the human target detection subsystem is responsible for detecting the human subject. Secondly, the detected human target's location is tracked by estimating the velocity of the walking. The traveling locations and traveling time are used to

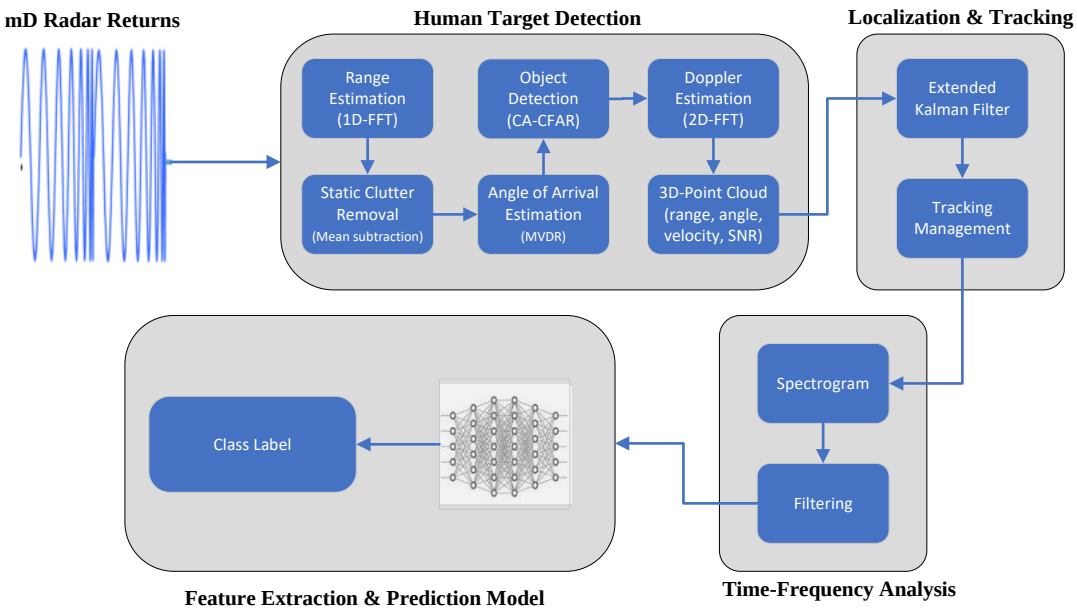


Figure 4.2: Overview of Radar Data Processing

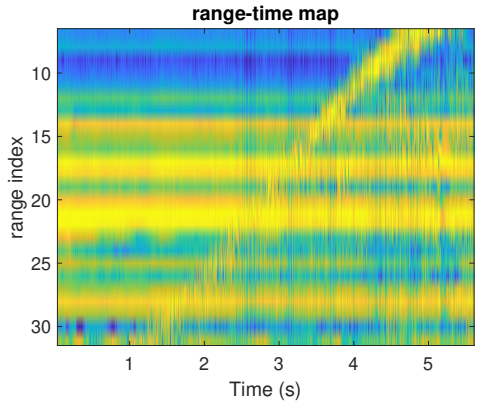
determine the dimension of mmWave data sample (i.e., K and N). The collected data samples from a group of human subjects are then converted into mmWave gait biometric samples, which are fed into the recognition subsystem. The recognition subsystem exploits these samples to train a deep neural network that automatically learns the salient features of the mmWave gait biometrics. The trained network is used to identify the human subjects. To make sure the system works well in the multipath-rich environments (e.g., indoor scenes), dedicated data-preprocessing solutions, such as high-pass filter, angular localization using minimum variance distortion loss(MVDR), cell averaging constant false alarm rate detector (CA-CFAR), extended kalman filter (EKF) and spectorgram enhancement scheme, are also integrated into the system to mitigate the distortions in the mmWave biometric samples caused by environment-induced static/dynamic reflections and background noise. The system architecture is shown in fig:4.2 and the details of each system block are presented in the following sections.

4.2.2.1 Human Target Detection

Instantaneously varying walking velocity of human subject with irregular surface contour intensifies the challenge for mmWave application to indoor environments. As mmWave radar identifies targets based on the energy level from reflected RF waves, prevailing objects within the environment (eg. walls and furnitures) may have tremendous ramification in human target detection. To suppress, the environment induced ramifications and narrow the target detection to human body we utilized a multi stage approach. Radar return signals (i.e., Chirps) from human subject and environment are processed by applying 1D-FFT to resolve the range of the reflector from the radar. Since the objective is to obtain the range of the subject, FFT is applied over the range dimension resulting in a Range-Time map as show in fig:4.3.

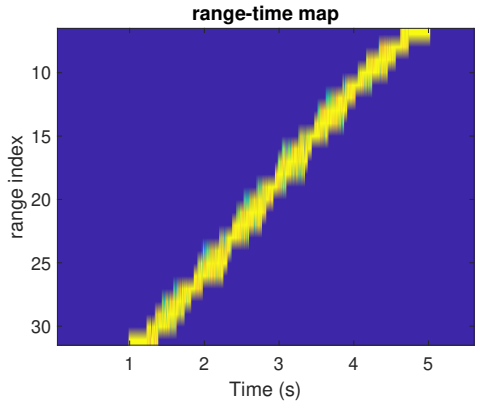
As we recall that the radar reflections are heavily impacted by the environments, the Range-time map is further processed to remove the reflections contributed by the static objects (eg: walls, chairs) as shown in fig: 4.6. Since static objects are stationary with zero velocity, the corresponding reflection signals can be simply removed by subtracting the mean of 1D-FFT data from all chirps for each antenna. Considering the fact that human subjects tends to have dynamic motion the relative velocity can be exploited to distill the radar reflection from ambiances (i.e., dynamic). In addition, since mmWave radar has antenna array with angular variations, we can exploit the azimuth angle to further synthesize the radar return signals. Since our mmWave radar consists of antenna array with 4 antenna elements spaced at equally separable distance, we applied the digital beamforming for RF processing technique that can virtually separate the radio returns received at different angle and elevations.

In particular, we utilized CAPON beamformer [72] to improve the angular resolution which resulted in rich range-azimuth heat map. The objective of CAPON beamformer is to eliminate the interference by negating the noise and improvising the signal of interest (SOI). By exploiting the SNR of SOI at various Rx antenna and



0.5
(a)
0

Figure 4.3: Range-Time Map



0.5
(a)
1

Figure 4.4: Pre-processed Range-Time Map

Figure 4.5: (a) Range-Time Map (b) Pre-processed Range-Time Map

dynamically adjusting the weights for each antenna's SNR, we can adaptively enhance the radar reflections from each azimuth angle of the antenna. However, the dynamic reflections still exists due to the co-existence and multi-path propagation effects as shown in fig:4.7. In order to remove such effects, we then applied object detection technique based on thresholding. In particular, we adopted Two pass CA-CFAR [73], which is one of most commonly used target detectors for noisy and interference-rich environments. In this case, the threshold characterizes the expected false-alarm probability and the average noise level encompassing the cell-under-test. The resulting

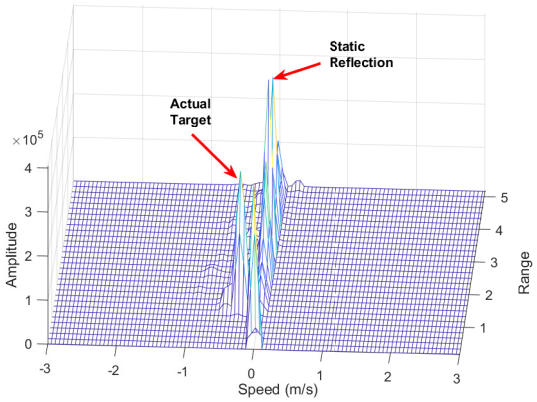


Figure 4.6: Range-Doppler Map with Static Reflections

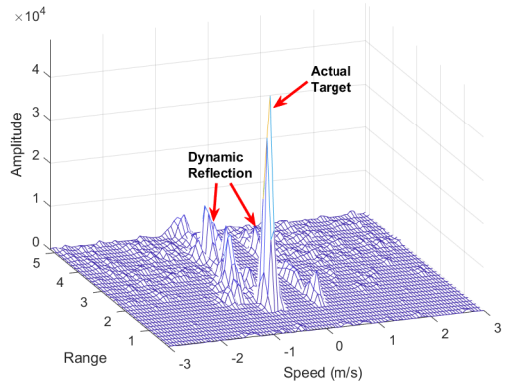


Figure 4.7: Highpass-filtered Range-Doppler Map with Dynamic Reflections

CA-CFAR output will be a set of points that describes the target elevation, range, azimuth and doppler.

4.2.2.2 Localization & Tracking

Contrary to computer vision based tracking solutions, realizing target tracking for mmWave radar is quite challenging, particularly for indoor case. To counter such a challenge, we have applied extended Kalman filter (EKF) to predict and estimate the target state (location, velocity, and direction) so that the virtual targets, which have large state deviations from the real tracking target, can be eliminated. Firstly, multiple points are clustered together based on the distance metric and then the centroid of the target is set as the tracking vector for that point cloud. Then, for

each frame or time-interval we keep track of the centroid of the point cloud which will then be projected as the trajectory of the target. In other words, EKF acts a moving spatial passband filter that can only allow to pass the point clouds/reflections orientated from the human moving trajectory.

4.2.2.3 mmWave Biometric Generation and Enhancement

The pre-processed range-time map is then converted into a mmWave biometric sample, which is an aggregated spectrogram. To improve recognition accuracy, we further enhance the spectrogram. First, the spectrogram is normalized by dividing the amplitude of each point in the spectrogram with the total energy of the spectrogram, which is the sum of the amplitudes of all points in the spectrogram. Second, mean filtering is applied by calculating the mean of the spectrogram, which is treated as the estimated noise floor. Then, the noise floor is subtracted from the spectrogram. Third, a 2D-Gaussian filter is convolved over the spectrogram to further reduce noise.

4.2.3 Feature Extraction and Classification

Spectrograms contains unique deep features that are representative biometric signatures for each human subject. Indeed it is impossible to different the signature of two human subjects with naked eye and formal detectors. Owing to the success of neural networks in extracting the deep features of images, we adopted deep convolutional neural network (DCNN) to learn high-level salient features from mmWave biometric data samples.

4.3 Spatial and Temporal Detection

4.3.1 Dataset

Our experimental study comprehended a dataset collected gait data from 10 volunteers between the ages of 18-35. Our dataset was collected using a TI IWR1642 mmWave sensor. The radar chirp configuration 10 used in was utilized to collect data from 4 different locations. The primary location, which was 11 used as the source do-

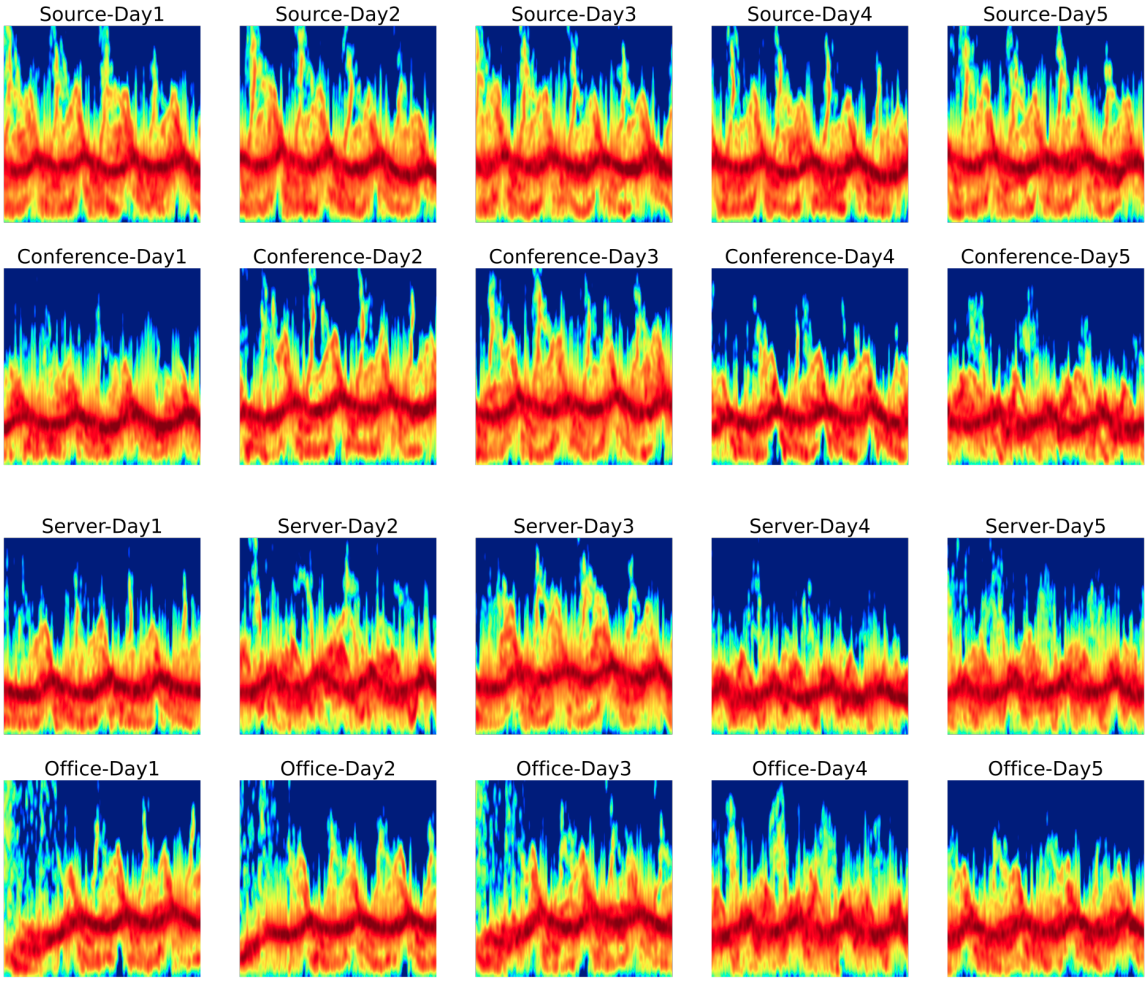


Figure 4.8: Gait spectrograms of the same person for 5 days at different locations. First row: Source (our lab), second row: Conference Room, third row: Server space and fourth row: Office room

main, was a research lab with cubicles. Each of the ten subjects was asked to 12 walk back and forth in the room with a radar pointed at them. The subjectâs path was pre-determined 13 and was always perpendicular to the radar signalâs propagation direction. Moreover, each day of a 14 userâs data contains 100 data samples. 4-seconds of walking data was considered as an individual 15 data sample. Each subject's data was collected in four different locations. The source location was a research space with cubicles, and the other three areas consisted of a server, conference, and an office room. As naturally originating phenomenon, diverse locations induce significant

shifts in radar returns for each subject due to environments. By maintaining four distinct locations, we introduce Spatial Domain Shift (SDS). In the source location, data was collected on ten different days for each subject. Five separate days of data was acquired for each of the three other locations, which are used as target domains. Fig:4.8 shows the spectrogram images for the same person at 4 different locations on 5 days.

The data collection was limited to 100 data samples in the source location and 50 data samples in each of the target locations on any given day. Unlike cameras, radars collect data actively by broadcasting signals into the environments. Unique movement patterns (e.g., gait) can induce different micro-Doppler frequency shifts in radar return signals, which can be represented as a spectrogram. Thus, a radar spectrogram can serve as the biometric print for user identification [74–83]. Apart from biometric traits, spectrogram data could also contain spatio-temporally varying noises from signal reflections and interference, which cannot be completely removed. This leads to environment-induced Temporal Domain Shift (TDS) and SDS. To add on, human gait, although unique to each individual, could contain minute variations, potentially as a consequence of a subject's mood, clothing, footwear, or some other similar aspect, which combined with the environment-induced shifts, further aggravates TDS. Changing the number of days in the train and test sets changes the amount of TDS. Similarly, using a subset of the available locations, SDS can be controlled, which makes it possible to study how the two variants of shifts interact with each other.

4.3.2 Presence of TDS and SDS:

As we can see from Fig. 4.8 the spectrograms from all locations look very similar. This is because they are spectrograms representing the same action (gait). Most of the environmental data is filtered out by the preprocessing methods, but the minuscule amounts of environmental information that seeps through was enough to induce

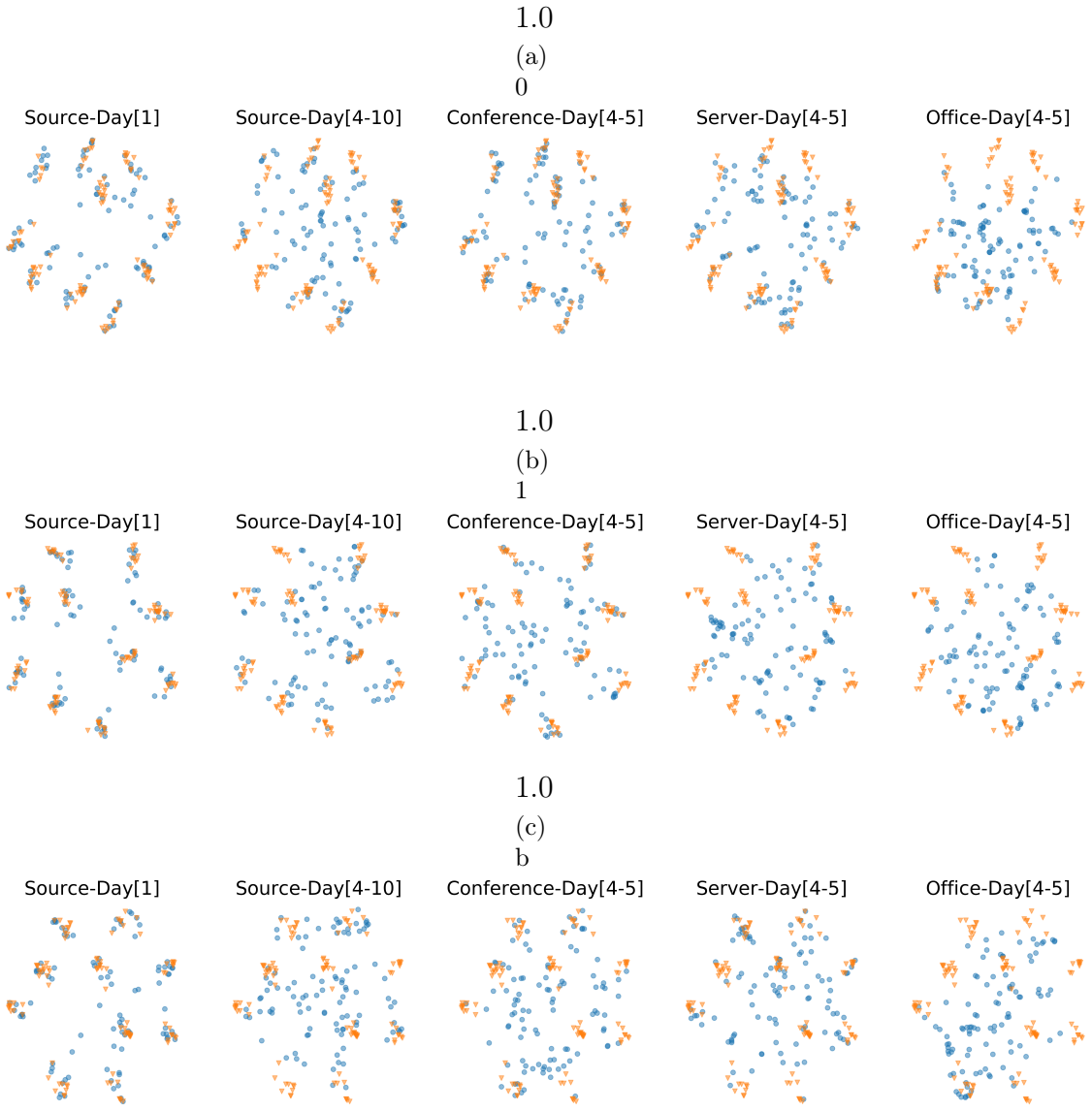


Figure 4.9: Embeddings from model trained on source and target domain data. (Top) model trained one day of source, server and conference data, (Middle) model trained via one day of source and conference data, (Bottom) model trained via one day of source and server data

domain shifts. We show the t-SNE [84] embeddings for each domain in our dataset alongside the training data embeddings. The model used to generate the embeddings was trained on a single day's data from the source location. The embeddings in Fig:4.9 show a domain shift in each new domain. However, this might not be very evident in the absence of the knowledge of the data collection procedure.

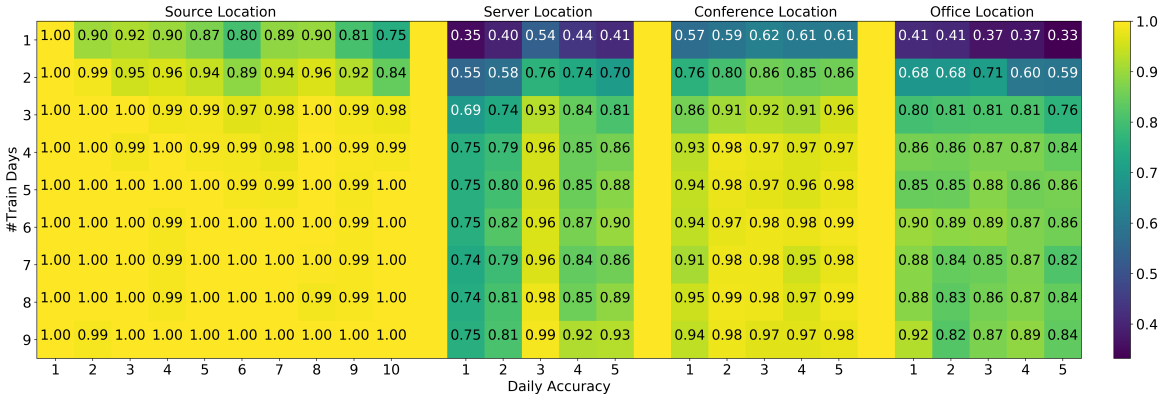


Figure 4.10: Results obtained from training only on source location data for varying number of days.

Our first experiment establishes the presence of TDS and SDS in our dataset. We trained a model on data only from the first day of our source location. Test accuracies of data from the remaining nine days of the source location and all target locations are reported in Fig. 4.10. The temporal degradation in the source and target domains is evident in the *first row* of Fig. 4.10.

The target domains also suffer from SDS apart from TDS, which is evident from the overall performance degradation in the target domains. This unequivocally establishes the presence of both TDS and SDS in our dataset. We also found a rather unexpected outcome from this experiment. There is a temporal degradation both in the source and target locations, but the deterioration does not strictly correlate with time progression. We conjecture that this behavior is a consequence of the malignancy of shifts. That is, not every shift is equally harmful. We further elaborate on this in Section 4.5.3.

Shift and Malignancy Detection: Upon further investigation, we found metric learning-based models are well suited for detecting domain shifts as well. Domain shift detection is the process of identifying shifts in the testing data. Unlike SDS, the presence of TDS is not always known. A few methods have been proposed for domain shift detection, but most either fail to detect shifts in an online fashion [68,85] or base their predictions on softmax thresholding. However, it is well established that [69,70]

softmax classifiers are not suitable for detecting domain shifts, especially for out of domain data.

To address aforementioned challenge, we rely on thresholding embeddings generated by metric learning models. Softmax classifiers are optimized to make closed-set predictions [86], i.e., pick one of the classes present in the training dataset as the prediction. Such an assumption is too strong and grossly underestimates the likelihood of seeing unknown classes or out of distribution data during testing. Metric learning methods do not make such assumptions or, at the very least, not to the extent softmax based solutions do. Metric learning optimizes a metric on embeddings generated by a base classifier, which potentially allows the classifier to ignore the concept of closed set predictions, thereby overcoming overconfident predictions and calibration issues.

Metric Learning Based Shift and Malignancy Detection Algorithm: We utilize Algorithm 1 to compute thresholds used for detecting outliers. In our experiments, L2 distance was used along with a margin m of 10. Upon computing class thresholds $Threshold$ and trained model M, any new data samples that were observed, were subject to class-wise thresholding $DistanceMetric$ between the known class centers C^{Train} and the generated embedding for the new data point.

Algorithm 1 Algorithm to Compute Class-Wise Thresholds

Input: Training data X_{Train}, validation data X_{Val}, number of classes n, $DistanceMetric$ (L2 or cosine distance), hyperparameter safety margin m.

Output: Class-wise threshold $\{Threshold_j | j = 1, 2, ..., n\}$, M, C^{Train}

1: Train model M on X_{Train} with metric learning

2: Generate class-wise embeddings $\{C_j^{Train} | j = 1, 2, ..., n\}$ from X_{Train} and M

3: Generate class-wise embeddings $\{C_j^{Val} | j = 1, 2, ..., n\}$ from X_{Val} and M

4: Compute $\{Threshold_j | j = 1, 2, ..., n\}$ = $DistanceMetric$ $(C_j^{Val} - C_j^{Train} | j = 1, 2, ..., n)$ + m

Figure 4.11

We exploit this premise to detect domain shifts. By maintaining dictionaries of class-wise mean embeddings and thresholding any new data point's distance from known embeddings, we can recognize shifts and their magnitude. Furthermore, such an approach does not require any specific offline training and can be deployed on online data sources. Not to mention, the detector will also be unsupervised, as no labeled data is required to detect a shift. The notion of detecting outliers or novel classes from embeddings is certainly not unheard of. We re-purpose it for detecting the presence and extent of domain shifts. Our experiments show that the approach agrees with labeled shift detectors.

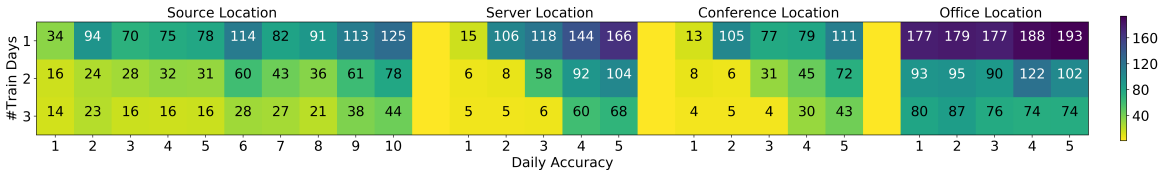


Figure 4.12: Mean distance from known class centers obtained from the source, server, and conference data for a varying number of days.

As mentioned in Section 4.4, before addressing domain shifts, we must be aware of the existence of a shift in their data. So far, we used an offline approach for shift detection, where the accuracy of the model trained using ground truth labels is used to uncover the presence of domain shifts. We now propose to sidestep this issue. By thresholding the distance to known training data embeddings, we acquire results similar to our labeled, accuracy based observations. Not only does this need no ground truth labels, but it can also be deployed online without any specific training for the problem. We note that a separate holdout dataset will be required to tune the thresholds. An added advantage of such an approach is its capability of detecting the malignancy of a shift. The accuracy-based approach does not necessarily convey the amount of shift introduced [69, 70]. We amend this issue by interpreting the magnitudes of the distance from class centers as an indicator of its malignancy. In Fig. 4.12, we present the results of Fig. 4.14, but with metric learning based shift

metrics. We find that the results strongly correlate with our hypothesis along with insights into the malignancy of each domain.

4.4 Unsupervised Domain Adaptation

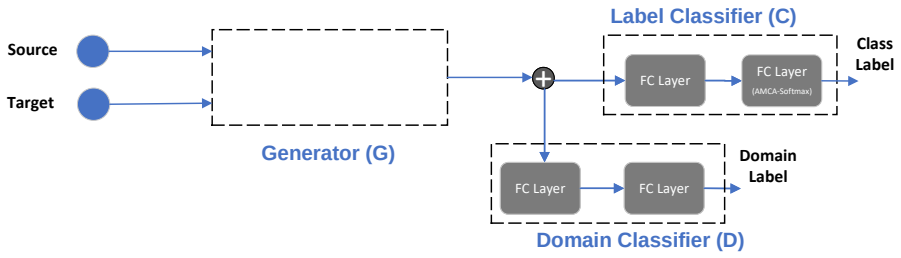


Figure 4.13: Adversarial domain adaptation

Gait recognition suffers from the inherent challenges that arises in applying SOTA DNN for real world applications. In section 4.3.2, we have shown that spectrogram data show distribution drifts both temporaly and spatially. Besides, we would like to consider the cases when we only have access to partially labelled data in source domain and extensive unlabelled data in the target domains. Towards this goal, we exploit unsupervised domain adaptation techniques for domain-invariant gait recognition. In specially, we adopt adversarial domain adaption methods to learn the domain-invariant gait representation, which is also discriminative enough for accurate user identification. As shown in Fig. 4.13, adversarial domain adaptation follows the adversarial learning strategy in a two-player minmax game similarly to Generative Adversarial Networks (GANs). A domain discriminator is trained to distinguish the source from the target domains, while a generator or encoder learns transferable embeddings that are indistinguishable by the domain discriminator.

The general problem of unsupervised domain adaptation may deal with two domains D_s and D_t corresponding to labeled samples in source domain and unlabeled samples from target domain that can represented as:

$$D_s = \{(x_i^s, y_i^s)\}_{i=1}^{ns} \tag{4.6}$$

$$D_t = \{x_j^t\}_{i=1}^{nt} \tag{4.7}$$

where ns and nt are the number of labeled and unlabeled samples for the domains. Besides, the samples are from the join distributions $P(x^s, y^s)$ and $Q(x^t, y^t)$ leading to commonly known i.i.d assumption. Our goal is to design a deep neural network $G : x \rightarrow y$ to minimize the data shift distribution across domains so that the prediction error of target domain $L_{Gt} = \mathbb{E}_{(x^s,y^s)\sim P)}[G(x^t) \neq y^t]$ is bounded by the prediction error of source domain $L_{Gs} = \mathbb{E}_{(x^s,y^s)\sim Q)}[G(x^s) \neq y^s]$.

The adversarial domain adaptation problem can be formulated as a minimax optimization problem as follows

$$\min_G L_{cls}(X_s, Y_s, G) \tag{4.8}$$

$$\min_D \max_G L_{adv_D}(X_s, X_t, G) - L_{adv_G}(X_s, X_t, D) \tag{4.9}$$

where $L_{cls}(X_s, Y_s, G)$ is the supervised classification loss for the source domain, $L_{adv_D}(X_s, X_t, G)$ is the domain discrimination loss and $L_{adv_G}(X_s, X_t, D)$ is the domain confusion lose. The above optimization problem can be reformulated as following format for avoiding training the generator twice, i.e.,

$$\min_G L_{cls}(X_s, Y_s, G) + \lambda L_{adv_G}(X_s, X_t, D) \tag{4.10}$$

$$\min_D L_{adv_D}(X_s, X_t, G) \tag{4.11}$$

where λ is the hyper-parameter that controls the tradeoff between source classification loss and domain confusion loss.

To learn highly discriminative representations, we adopt metric learning loss func-

tions for $L_{cls}(X_s, Y_s, G)$. Training networks with metric learning allows one to introduce new classes after training a base model, without having to train the model again, which translates to adding new user identities without the need for retraining. Another benefit of using metric learning is its theoretical underpinning, which is the cluster assumption. The cluster assumption states that the decision margins lie in low density or relatively unoccupied regions of the classification manifold. This assumption has previously been exploited in numerous domain adaptation techniques and is known to promote adaptation under certain constraints. Metric learning reaffirms this assumption by maximizing inter-class distances and minimizing intra-class distances. In this work, we integrate the additive margin softmax loss function [87] with two additional regularization terms [88], which, as shown in our experiments, can help to learn highly discriminative feature. First, the AM-softmax loss is given as follows.

$$L_{AM} = -log\frac{e^{s.(cos\theta_{y_i}-m)}}{e^{s.(cos\theta_{y_i}-m)} + \sum_{j=1,j\neq y_i}^{c} e^{s.cos\theta_j}} \tag{4.12}$$

Then, a constrictive regularizer R_C scheme is introduced that ensures for each class weight vector $||W_j||_2^2$ will be equal to the average norm and it defined as:

$$R_c = \frac{1}{4N}\sum_{j}^{N}(||W_j||_2^2 - \mu(||W||_2^2))^2 \tag{4.13}$$

where N is the total number of all classes. And $\mu(||W||_2^2))^2$ can be computed as:

$$\mu(||W||_2^2))^2 = \frac{1}{N}\sum_{n}^{N}||W_n||_2^2 \tag{4.14}$$

Moreover, annular regularizer scheme is applied to ensure that norm of each feature equals is scaled and it defined as:

$$R_A = \frac{1}{4M}\sum_{i}^{M}(||x_i||_2^2 - \mu(||x||_2^2))^2 \tag{4.15}$$

where M is the batch size and x_i corresponds to the feature of i-th training sample. And $\mu(||x||_2^2))^2$ can be computed as:

$$\mu(||x||_2^2))^2 = \frac{1}{M}\sum_{n}^{M}||x_n||_2^2 \tag{4.16}$$

Then, combing AM-softmax with above two regularizers leads to final loss function $L_{cls}(X_s, Y_s, G)$

$$L_{cls}(X_s, Y_s, G) = L_{AM} + \lambda R_c + \beta R_A \tag{4.17}$$

While $L_{cls}(X_s, Y_s, G)$ is used for train the generator to learn discriminative representations, then generator is also updated to fool the discriminator using domain confusion loss (or the inverted label loss function which is given by:

$$L_{adv_G}(X_s, X_t, D) = -\mathbb{E}_{x_t \sim X_t}[log(D(G(x_s)] \tag{4.18}$$

Since domain discriminator aims to classify whether a sample is coming from source domain or target domain, the domain discrimination loss is a standard binary supervised loss

$$L_{adv_D}(X_s, X_t, G) = -\mathbb{E}_{x_s \sim X_s}[log(D(G(x_s)] - \mathbb{E}_{x_t \sim X_t}[log(1 - D(G(x_t)] \tag{4.19}$$

4.5 Experiments

Model: All our experiments used an 18-layer Resnet as defined in [89]. Residual layers are stacks of convolutional, activation, batch normalization and addition layers with a skip connection. The skip connection allows the features to propagate

forward via both the main stack of layers and the skip connection bypassing the primary stack. This promotes better feature extraction and gradient flow during back propagation. We utilize 2 types of residual layers, identity and convolutional residual layers. The identity residual layers do not have any operations on the skip connection and maintain the feature map's height and width. The convolutional residual layers have convolutional and batch normalization operations in the skip connections, and the operation reduces the dimension of the output feature map. Our discriminator is constructed with a multi-layer perceptron with 128 hidden units and final layer with hidden units equal to the number of domains. And the label classifer is constructed using a Multi-Layered Perceptron (MLP) with 128 hidden units and a final layer with hidden units equal to the number of classes (10). The 128 dimensional fully connected layer's output was used as an embedding layer. Moreover, a Constrictive Regularizer [90] was utilized as the activity regularizer. Moreover, the last fully connected layer also used a Constrictive Regularizer for the kernel weights instead of the activations. Additionally, given the definition of Additive Margin Loss [91], we do not have any bias variables in the last layer. Finally, the logits were activated following the definition in [91].

Optimizer: At first step, the generator function is trained using cross entropy loss. The training step is optimized using the ADAM optimizer with initial learning rate $lr = 0.0001$ and the scheduler is utilized to decay the learning rate over epochs. Besides, we used SELU as the activation function for all the layers. The discriminator on the other hand is trained using the same configuration as the generator and all our experiments were using the same set of parameters. In addition, we used $s = 10$, $m = 0.1$ and $\lambda = 0.0004$ as the hyper parameters for the softmax loss functions.

Data Sampling: In all our supervised experiments, data from each available domain was shuffled together and sampled in batches. This was done irrespective of the domain each sample belonged to. For our unsupervised experiments, data from

each available domain was sampled separately, which allowed us to generate embeddings for data from each domain. We generate domain predictions with an MLP with 128 hidden units followed by another layer with hidden units equal to the number of domains being used for training. The domain classifier MLP was either prepended by a gradient reversal layer (GRL) [92] or treated as a discriminator in a Generative Adversarial Network (GAN) [93].

4.5.1 Interplay between TDS and SDS

Mitigating SDS via TD data: We now show that it is not possible to entirely mitigate SDS by introducing source domain's temporal data alone and vice-versa. Limitations of improving SDS generalization by using TD data are evident in Fig. 4.10. We trained models from two, all the way up to nine days of labeled source location data, which is followed by an evaluation of daily accuracies. By introducing more temporal data, the source location generalization improves as time progresses. But as one might expect, we start to see diminishing returns in terms of source location accuracy as we increase the number of training days. Moreover, after introducing more source domain's temporal data, the target location performance also has diminishing returns so that each target location's performance is seldom on par with the source.

Mitigating TDS via SD data: We move on to establish the limitation of introducing more target/spatial domain data to alleviate TDS. We trained a model on a single day of labeled data from the source, conference, and server locations. From the *first row* of Fig. 4.14, it is clear that even when data from target domains is introduced, individual location's temporal shifts are not completely mitigated.

However, we do find that the model performance in the office location improves when compared to introducing only temporal data. Since no data from the office was used in training, we interpret it as the true SDS generalization of the model. This implies that the primary benefits of training on data from a particular type of shift is confined to easing similar kinds of shifts.

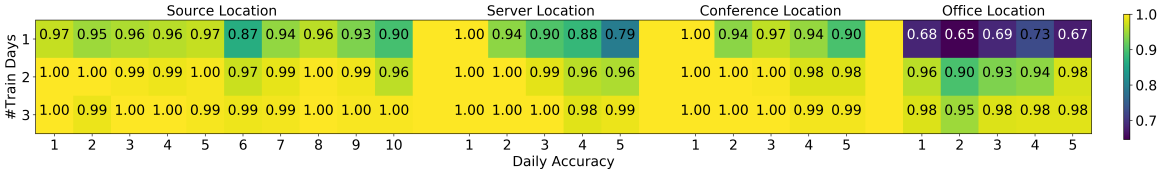


Figure 4.14: Results obtained from training on source, server, and conference location data for varying number of days. The data from office location is not used for training and only for testing

Correlation of TDS and SDS: We now draw attention to the correlation between TDS and SDS. It is evident from our prior experiments that TD or SD data individually cannot mitigate both. Nevertheless, from Fig. 4.10 and Fig. 4.14, and the findings mentioned above, a correlation between the two is evident. It might be the reason no clear distinction has been established so far, but the implication of this is rather profound. Collecting TD and SD data might incur varying costs. *When the cost disparity is substantial, it is possible to trade one for the other to a certain extent.* This brings us to one of the most exciting findings of our paper. That is, *TDS and SDS have to be addressed jointly to bring substantial generalization improvement.* We trained models with both TD and SD data along with the initial source data. We find in Fig. 4.14 that when both shifts are addressed together, we attain the best generalization in both TD and SD. By introducing multiple (2 - 3) days of data from every location, we address TDS individually in each location. Introducing data from multiple target locations (server and conference rooms) addresses SDS. Moreover, such jointly trained model generalizes well to the unseen target domain (office).

4.5.2 Adversarial Domain Adaptation Performance

Table 4.1: Model Accuracies

Method	Source->Temporal	Source->Server	Source->Conference	Source->Office	AVG
Our (GAN)	**0.99**	**0.97**	**0.97**	0.96	**0.98**
Our (GRL)	**0.99**	0.93	0.95	**0.97**	0.96
DDAN	0.97	0.85	0.92	0.81	0.89
ADDA	0.97	0.85	0.95	0.89	0.92
ST	0.97	0.86	0.95	0.85	0.90

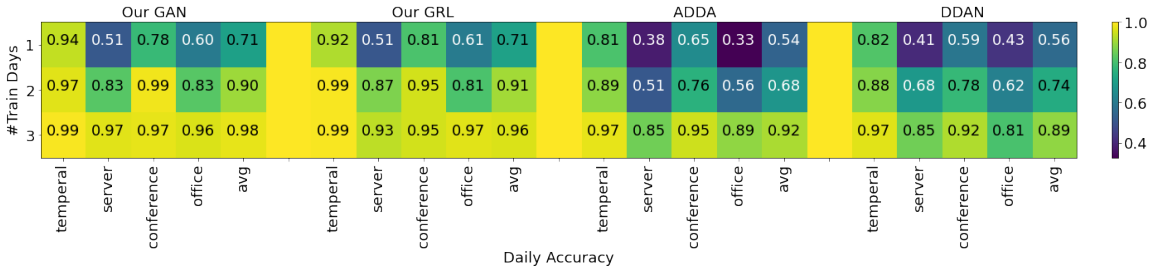


Figure 4.15: Results obtained by training models between 1 - 3 days on all locations and the respective daily test accuracies

In table: **??**, we reported the performance of our domain adaptation technique and the three baselines (ADDA [94], DDAN [95] and ST [96]). During the study, we trained our model and baselines models on source domain and then tested against the target domains separately. For example, first we train the model on 3-days' source domain labeled data and 3-days source domain unlabeled data, then test on the data of the remaining 4 days in source domain. For each target locations, we trained our model with 3-days' source domain labeled data and 3-days' target domain unlabeled data. Then, we test the remaining 2-days' target domain data. Our results conclude that our model achieve high identification accuracy and outperforms the baselines. In addition, we also studies the impact of increasing the training days from source domain on the target domain identification accuracy. From Fig.4.15, we can see that overall the adaptation performance improved when more days of data is included for the training. Moreover, our proposed model also outperforms other baseline methods.

4.5.3 Domain Importance

As mentioned in Section 4.3.2, we notice a disparity in the malignancy of each domain shift. From the plethora of research conducted on adversarial learning [97], it is evident that not every kind of data noise is equally harmful. Networks can overcome small amounts of noise in the data. This is true even when the models have not been explicitly trained with adversarial methods. Similarly, adversarial training with arbitrary data perturbations is not always a precursor for domain generalization.

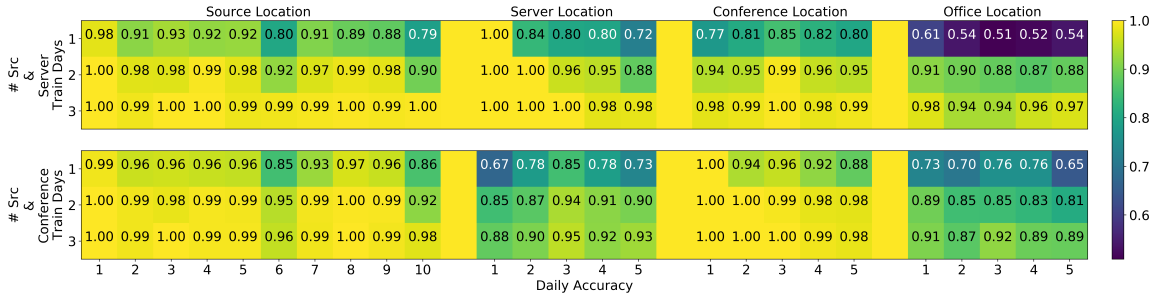


Figure 4.16: (Top) Results obtained from training on the source and server data for a varying number of days. (Bottom) Results obtained from training on the source and conference data for a varying number of days. Data from office location is for testing only

We find this to be the case for domain adaptation as well. Not all available domains are worth adapting to. In our dataset, the conference location exhibits a domain shift, but it is not equally worth adapting to when compared to server room data. We found that by adapting to the server location alone, we can generalize to all other locations shown in Fig. 4.16. The server room can be considered more adversarial than the conference room. When we train a model on the source and server data, the information discrepancy amongst the two domains is substantial compared to the source and conference room data. This results in spatial generalization with only half of the spatial target domain data. Admittedly, the model trained with source and server data is not as good as the model trained with source and conference data in every aspect. However, the overall generalization is better. Depending on the cost of data collection, it might be well worth the marginal loss in accuracy.

4.5.4 Data-efficient Domain Adaptation

Exploiting the proposed domain shift detector enables us to develop a data-efficient domain shift mitigation approach. Life cycle of the proposed approach is shown in fig: 4.17. Instead of blindly collecting a large amount of labeled or unlabeled data from all spatial and temporal domains, we use the shift detector to measure the shift malignancy of each domain. Next, domains with insignificant shifts are discarded.

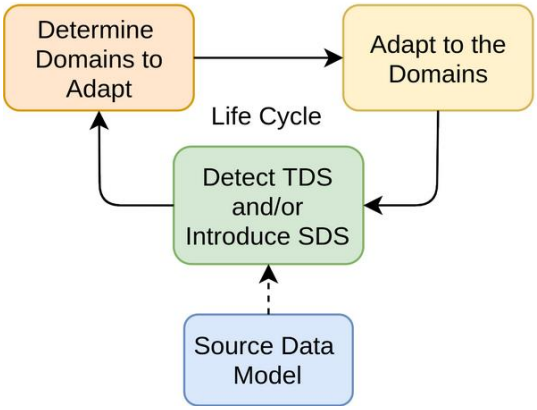


Figure 4.17: Overview of data efficient domain adaptation

Based on the cost of collecting labeled/unlabelled TD/SD data in different domains, a heuristic approach is utilized to determine and collect an appropriate type and amount of data. After this, the available domains are adapted to using an appropriate domain adaption strategy. Utilizing the domain shift detector, the number of domains to adapt was reduced, i.e., we only use the server location data with the highest domain shifts, while ignoring data from conference and office locations. In this case, as shown in Table 4.2, our unsupervised domain adaption method can achieve the same performance as the supervised method, where the model is trained with labeled data from source domain and the server location. Furthermore, we conjecture that by incorporating few-shot learning methods, the source location labeled data could also be reduced. A single day of source location data currently consists of only 1000 labeled data samples, which hinders the performance as it is not sufficient to train a robust model. We addressed this issue by using labeled data from three days of the source location. However, using other techniques such as initialization with pre-trained model weights might also work well, thereby reducing the source location's labeled data.

Table 4.2: Model Accuracies of Unsupervised and Supervised Domain Adaption

	Source-Temporal	Server-Test	Conference-Test	Office-Test
Labeled	0.99	0.98	0.99	0.97
Our (GAN)	0.99	0.98	0.99	0.97

4.6 Conclusion

Our work has established the need to differentiate spatial and temporal shifts. Contrary to some beliefs, we show that these drifts can be particularly harmful, which is especially true for radar-based gait recognition, in which the presence of the issue was not well known. The dataset we introduced has unveiled the previously unbeknown relation between TDS and SDS. We further uncovered the impact of adapting to one domain could have on other domains and introduced the promising yet straightforward avenue of methods that use metric learning to detect and estimate a shift's potential impact. Finally, we show our proposed life-cycle to tune and decide which domains to adapt to substantially optimized our adaptation efforts. Our overall approach to handling shifts establishes a promising layout for improving the generalization performance of radar-based biometric identification systems in a data-efficient manner.

CHAPTER 5: Summary and Future Directions

The rapid growth of internet-enabled devices and applications demands cost-effective network backbones with stringent quality of service requirements. While single-hop access point based networks can guarantee service requirements, they are expensive to deploy and manage. Wireless mesh networks is a multi-hop wireless network which is cost-efficient and have been deployed for wireless community mesh networks, high-speed urban networks, global wireless internet infrastructure, battlefield networks, and public safety/disaster rescue networks. Vast amount of data in todayâs networks are generally privacy sensitive and feature-rich data points which are then used to improve centralized machine learning model of an AI-based application. Recently, a distributed machine learning technology known to be Federated Learning (FL) is exploited to collaboratively improve a shared global model using edge servers. Our preliminary study shows that FL systems perform much worse when deployed over multi-hop wireless networks due to noisy interference-rich wireless links. In addition, the classic single layer FL system architecture converges slows down global model convergence due to unpredictable network link delay.

In this dissertation, we have developed a novel Artificial Intelligence-based wireless network system which provides guaranteed network stability, thereby accelerating convergence speed and accuracy of FL systems. In particular, our approach significantly reduces the communication latency by introducing AI-based routing solutions, which is the first ever experimentation on the actual physical testbed. Firstly, we have developed and implemented distributed in-band telemetry system and wireless network operating system. This system simplifies implementing AI-based traffic engineering solutions such as multi-agent reinforcement learning routing. Our exper-

imental validation shows promising network performance in terms of delay, packet loss, and throughput. Secondly, we have developed FedEdge, which is the first ever multi-hop FL system framework for multi-hop wireless networks. FedEdge frameworks allow developers to evaluate and optimize FL algorithms that are deployed over multi-hop networks. In addition, developers can optimize FedEdge network devices routing paths using reinforcement learning-based routing algorithms to improve the convergence time. In this work, we validated that FL system accuracy and convergence time can be improved by optimizing the routing paths between the edge servers and the remote server using Multi-agent reinforcement learning algorithms. Finally, we considered mmWave radar based bio-metric identification as one of the potential applications for FL systems and studied the generalization issues in practical deployments. Our study shows that the existence of spatial and temporal shifts hinders model generalization due to the underlying domain shift issues. To address the case, we have developed a data-efficient domain shift mitigation technique that can be used to tune deep learning models. We exploited unsupervised domain adaptation to improve temporal and spatial generalization issues.

5.1 Future Directions:

Our work on AI-based wireless network systems is carried out on physical hardware's which are time consuming to setup and tune for multiple experiments. We had to rely on the hardware setup because of the fact that none of the current network simulators are capable of emulating a distributed multi-radio multi-hop wireless network with real-time performance. Existing network emulators such as mininet-wifi, ns-3 and mininet are widely used for networking research, however they cannot be used for validating a distributed AI-based network systems. The main reason for such shortfall is because the emulators tend to share the CPU time thereby leading to sequential processing. However, in real-world processing and packet transmission happens in parallel. Hence, I believe designing a hardware accelerated network emu-

lator with parallel processing capability is one of the promising directions to bridge the gap between realistic network deployment and emulated network scenarios.

Federated learning on multi-hop wireless networks is more challenging because of random network delay. While we try to solve the convergence time by optimizing the network, we could also improve the convergence by employing various computation techniques such as importance sampling, quantized model updates and asynchronous FL training. By optimizing FL systems at all layers, we may be able to reduce the wall-clock time significantly with high accuracy.

REFERENCES

[1] "wireless community networks," available: https://en.wikipedia.org/wiki/List_of_wireless_community_networks_by_region.

[2] "New york city (nyc) mesh network," available: https://www.nycmesh.net/.

[3] "Facebook terragraph network," available: https://terragraph.com/.

[4] "Spacex satellite constellation wireless internet," available: https://en.m.wikipedia.org/wiki/Starlink.

[5] "Google balloon powered global wireless internet," available: https://loon.com/technology/.

[6] "rajant kinetic mesh networks for battlefield communication," available: https://rajant.com/markets/federal-military-civilian/.

[7] M. Portmann and A. Pirzada, "Wireless mesh networks for public safety and crisis management applications," *IEEE Internet Computing*, vol. 12, no. 1, pp. 18–25, 2008.

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

[9] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "Ieee 802.11 s: the wlan mesh standard," *IEEE Wireless Communications*, vol. 17, no. 1, 2010.

[10] The open mesh networks consortium. [Online]. Available: http://www.open-mesh.org

[11] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.

[12] 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.

[13] V. Ramaswami, K. Jain, R. Jana, and V. Aggarwal, "Modeling heavy tails in traffic sources for network performance evaluation," *Computational Intelligence, Cyber Security and Computational Models*, vol. 246, pp. 23–44, 2014.

[14] A. Ghosh, R. Jana, V. Ramaswami, J. Rowland, and N. K. Shankaranarayanan, "Modeling and characterization of large-scale WI-FI traffic in public hot-spots," in *IEEE INFOCOM*. IEEE, 2011, pp. 2921–2929.

[15] P. Wang and I. F. Akyildiz, "Spatial correlation and mobility aware traffic modeling for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1860–1873, 2011.

[16] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially Aware Motion Planning with Deep Reinforcement Learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.

[17] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 5048–5058.

[18] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 372–382.

[19] D. Yang, W. Rang, and D. Cheng, "Joint optimization of mapreduce scheduling and network policy in hierarchical clouds," in *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 2018, p. 66.

[20] J. Zhao, G. Qiu, Z. Guan, W. Zhao, and X. He, "Deep Reinforcement Learning for Sponsored Search Real-time Bidding." *arXiv preprint arXiv:1803.00259*, 2018.

[21] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading." *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 653–664, 2017.

[22] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach," in *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 25–33.

[23] 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.

[24] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[25] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. Liu, and D. Yang, "Experience-driven networking: a deep reinforcement learning based approach," in *Proc. of IEEE Infocom*, 2018.

[26] 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)*, 1994, pp. 671–678.

[27] 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. IEEE, 2002, pp. 1825–1830.

[28] 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),*. IEEE, 2016, pp. 341–344.

[29] 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)*. IEEE, 2017, pp. 138–145.

[30] 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),*. IEEE, 2017, pp. 155–159.

[31] P. Pinyoanuntapong, M. Lee, and P. Wang, "Delay-optimal traffic engineering through multi-agent reinforcement learning," in *2019 IEEE INFOCOM Workshop: NI 2019: Network Intelligence: Machine Learning for Networking*, 2019.

[32] M. Chen, Z. Yang, W. Saad, C. Yin, and S. C. H. Poor, "A joint learning and communications framework for federated learning over wireless networks," 2019, available: https://arxiv.org/abs/1909.07972.

[33] M. Amiri and D. Gunduz, "Over-the-air machine learning at the wireless edge," in *Proc. IEEE SPAWC*, 2019.

[34] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning based on over-the-air computation," in *Proc. IEEE ICC*, 2019.

[35] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, 2016.

[36] M. Amiri and D. Gunduz, "Federated learning over wireless fading channels." Available: https://arxiv.org/abs/1907.09769, 2019.

[37] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple network management protocol," STD 15, RFC 1157, SNMP Research, Performance Systems International, MIT âŠ, Tech. Rep., 1990.

[38] S. U. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in of@ tein sdn testbed using sflow," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–6.

[39] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[40] 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.

[41] J. Hyun and J. W.-K. Hong, "Knowledge-defined networking using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2017, pp. 54–57.

[42] Ryu: Sdn controller. [Online]. Available: https://osrg.github.io/ryu/

[43] "Ofsoftswitch13," available: https://github.com/CPqD/ofsoftswitch13.

[44] "Mangodb," available: https://www.mongodb.com/.

[45] "Softmac," available: http://bit.ly/2Rq6vwG.

[46] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[47] N. Van Tu, J. Hyun, and J. W.-K. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2017, pp. 76–81.

[48] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. Chen, and G. Mirsky, "Am-pm: Efficient network telemetry using alternate marking," *IEEE Network, accepted*, 2019.

[49] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," *arXiv e-prints*, p. arXiv:1903.03934, Mar. 2019.

[50] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019, available:https://arxiv.org/abs/1903.03934.

[51] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2020.

[52] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averagingworks for deep learning," in *Proc. AAAI 2019*, 2019.

[53] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization," in *Proc. PMLR 2019*, 2019.

[54] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *Journal of Machine Learning Research*, vol. 13, pp. 165–202, 2012.

[55] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. NIPS*, 2014.

[56] "Tennsorflow federated," available: https://www.tensorflow.org/federated.

[57] "Pysyft websocket mnist digit recognition task example," available: https://github.com/OpenMined/PySyft/tree/master/examples/tutorials/advanced/websockets-example-MNIST.

[58] "Tensorflow federated datasets," available: https://www.tensorflow.org/datasets.

[59] 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.

[60] "Fastapi web framework," available: https://fastapi.tiangolo.com/.

[61] "Asyncio," available: https://asyncio.readthedocs.io/en/latest/.

[62] "Httpx," available: https://www.python-httpx.org.

[63] "Grpc," available: https://grpc.io.

[64] 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)*, 2020, pp. 766–773.

[65] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[66] J. G. Moreno-Torres, T. Raeder, R. Alaiz-RodríGuez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[67] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[68] S. Rabanser, S. Günnemann, and Z. Lipton, "Failing loudly: an empirical study of methods for detecting dataset shift," in *Advances in Neural Information Processing Systems*, 2019, pp. 1394–1406.

[69] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[70] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. Dillon, J. Ren, and Z. Nado, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 969–13 980.

[71] M. Kaya and H. Ş. Bilge, "Deep metric learning: a survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.

[72] J. Li and P. Stoica, *Robust adaptive beamforming*. Wiley Online Library, 2006.

[73] M. A. Richards, *Fundamentals of radar signal processing*. Tata McGraw-Hill Education, 2005.

[74] W. Wang, A. X. Liu, and M. Shahzad, "Gait recognition using wifi signals," in *2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*. ACM, 2016, pp. 363 – 373.

[75] A. Pokkunuru, K. Jakkala, A. Bhuyan, P. Wang, and Z. Sun, "Neuralwave: Gait-based user identification through commodity wifi and deep learning," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 758–765.

[76] K. Jakkala, A. Bhuya, Z. Sun, P. Wang, and Z. Cheng, "Deep csi learning for gait biometric sensing and recognition," *arXiv preprint arXiv:1902.02300*, 2019.

[77] P. Janakaraj, K. Jakkala, A. Bhuyan, Z. Sun, P. Wang, and M. Lee, "Star: Simultaneous tracking and recognition through millimeter waves and deep learning," in *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2019, pp. 211–218.

[78] J. Pegoraro, F. Meneghello, and M. Rossi, "Multi-person continuous tracking and identification from mm-wave micro-doppler signatures," *arXiv preprint arXiv:2003.03571*, 2020.

[79] X. Huang, J. Ding, D. Liang, and L. Wen, "Multi-person recognition using separated micro-doppler signatures," *IEEE Sensors Journal*, 2020.

[80] Y. Lang, Q. Wang, Y. Yang, C. Hou, Y. He, and J. Xu, "Person identification with limited training data using radar micro-doppler signatures," *Microwave and Optical Technology Letters*, 2019.

[81] S. Abdulatif, F. Aziz, K. Armanious, B. Kleiner, B. Yang, and U. Schneider, "A study of human body characteristics effect on micro-doppler-based person identification using deep learning," *arXiv preprint arXiv:1811.07173*, 2018.

[82] X. Qiao, T. Shan, and R. Tao, "Human identification based on radar micro-doppler signatures separation," *Electronics Letters*, vol. 56, no. 4, pp. 195–196, 2020.

[83] Y. Lang, Q. Wang, Y. Yang, C. Hou, H. Liu, and Y. He, "Joint motion classification and person identification via multitask learning for smart homes," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9596–9605, 2019.

[84] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[85] Z. C. Lipton, Y.-X. Wang, and A. Smola, "Detecting and correcting for label shift with black box predictors," *arXiv preprint arXiv:1802.03916*, 2018.

[86] C. Geng, S.-j. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[87] F. Wang, J. Cheng, W. Liu, and H. Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018.

[88] J.-B. Liu, Y.-P. Huang, Q. Zou, and S.-C. Wang, "Learning representative features via constrictive annular loss for image classification," *Applied Intelligence*, vol. 49, no. 8, p. 3082â3092, Aug. 2019. [Online]. Available: https://doi.org/10.1007/s10489-019-01434-3

[89] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[90] J.-B. Liu, Y.-P. Huang, Q. Zou, and S.-C. Wang, "Learning representative features via constrictive annular loss for image classification," *Applied Intelligence*, vol. 49, no. 8, pp. 3082–3092, 2019.

[91] F. Wang, J. Cheng, W. Liu, and H. Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018.

[92] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[93] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[94] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," 2017.

[95] H. Zhao, J. Hu, Z. Zhu, A. Coates, and G. Gordon, "Deep generative and discriminative domain adaptation," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '19. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2019, p. 2315â2317.

[96] S. Abbasi, M. Hajabdollahi, N. Karimi, and S. Samavi, "Modeling teacher-student techniques in deep neural networks for knowledge distillation," 2019.

[97] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.