Fall 10-1-2021

A Hybrid SDN-based Architecture for Wireless Networks

Qiaofeng Qin
Yale University Graduate School of Arts and Sciences, qinqf_pku@yahoo.com

Abstract

A Hybrid SDN-based Architecture for Wireless Networks

Qiaofeng Qin
2021

With new possibilities brought by the Internet of Things (IoT) and edge computing, the traffic demand of wireless networks increases dramatically. A more sophisticated network management framework is required to handle the flow routing and resource allocation for different users and services. By separating the network control and data planes, Software-defined Networking (SDN) brings flexible and programmable network control, which is considered as an appropriate solution in this scenario.

Although SDN has been applied in traditional networks such as data centers with great successes, several unique challenges exist in the wireless environment. Compared with wired networks, wireless links have limited capacity. The high mobility of IoT and edge devices also leads to network topology changes and unstable link qualities. Such factors restrain the scalability and robustness of an SDN control plane. In addition, the coexistence of heterogeneous wireless and IoT protocols with distinct representations of network resources making it difficult to process traffic with state-of-the-art SDN standards such as OpenFlow.

In this dissertation, we design a novel architecture for the wireless network management. We propose multiple techniques to better adopt SDN to relevant scenarios. First, while maintaining the centralized control plane logically, we deploy multiple SDN controller instances to ensure their scalability and robustness. We propose algorithms to determine the controllers' locations and synchronization rates that minimize the communication costs. Then, we consider handling heterogeneous protocols in Radio Access Networks (RANs). We design a network slicing orchestrator enabling allocating resources across different RANs controlled by SDN, including LTE and Wi-Fi. Finally, we combine the centralized controller with local intelligence, including deploying another SDN control plane in edge devices locally, and offloading network functions to a programmable data plane. In all these approaches, we evaluate our solutions with both large-scale emulations and prototypes implemented in real devices, demonstrating the improvements in multiple performance metrics compared with state-of-the-art methods.

A Hybrid SDN-based Architecture for Wireless Networks

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Qiaofeng Qin

Dissertation Director: Leandros Tassiulas

December 2021

Acknowledgments

It would not be possible for me to finish this dissertation without the help and support from so many people. I would like to thank everyone who has assisted me through my PhD journey at Yale.

First, I am deeply indebted to my advisor, Prof. Leandros Tassiulas. Being so inexperienced and immature at the early years of my PhD time, I feel extremely fortunate that Prof. Tassiulas generously provided me with his guidance. His profound knowledge in the communication and networking area has always enlightened me the adequate research direction to proceed. He also created the best laboratory environment to satisfied the requirement of each group member. Besides, with his help I got opportunities to have collaborations and internships with people from other academic and industrial institutions. I think these benefits I received from him will be lifelong and impact my future career, for which I will always be grateful.

I would also like to express my deepest appreciation to Prof. Amin Karbasi, and Prof. Sekhar Tatikonda for agreeing to join my committee and Prof. Iordanis Koutsopoulos for reviewing my thesis paper as an external reader. I am very impressive of their knowledge and academic achievements, and feel lucky to receive their invaluable feedback on the research directions.

I am also grateful to all my co-authors and collaborators, Konstantinos Poularakis, George Iosifidis, Kin K. Leung, Sastry Kompella, Liang Ma, Erich M. Nahum and Miguel Rio, who have played a decisive role in finishing the works described in this dissertation, as well as Nakjung Choi, Muntasir R. Rahman and Marina Thottan, who provided me with a valuable opportunity of a summer internship and helped me to learn the state-of-the-art industrial tendencies, opening up my eyes and making it possible for me to explore deeper to the thesis topic. At the same time, as researchers more senior than me, they set up the role models of how to achieve better scientific explorations.

I would like to thank senior members of our group, Nikolaos Makris and Harris Niavis for sharing the experience and offering great help during the setup of the experiment testbed. Thanks also to Mandy Singer and Marsha Marcum for spending a great amount of time on the orders of experiment equipment. Without efforts from them, I would not get the chance to verify the theoretical conclusions with real devices and networking environments.

I also had great pleasure of working with all my other group members, Yuang Jiang, Nikolaos Papadis, Nick Nordlund, Victor Valls, Akrit Mudvari, Aosong Feng and Panagiotis Promponas. I enjoy each time discussing progresses and ideas with you, which not

only broadened my perspective but also inspired me to think of new research directions.

Outside my research group, I also received help for countless times during my studies at Yale. I am grateful to Prof. Fengnian Xia leading the Laboratory of Emerging Materials and Devices, as well as his group members Cheng Li, Bingchen Deng and Qiushi Guo. This is the place where I started my studies and researches as a PhD student. I would also like to extend my gratitude to many other peers in the department of electrical engineering, especially Zheng Gong and Lin Chen for their helpful advice and the campus life we spent together.

I gratefully acknowledge the assistance I acquire from various open source projects mentioned in this dissertation. Particularly, I thank Roberto Riggio of 5G-EmPOWER project and Robert Schmidt of FlexRAN project for sharing great wireless architecture implementations and maintaining vital, productive open source communities, where I have learned a lot from discussions among researchers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Challenges of Wireless Network Control

Nowadays, wireless networking is playing an increasingly important role in both industrial worlds and everyday life. By interconnecting a multitude of devices interfacing with the physical world, the Internet of Things (IoT) are carrying more and more applications such as wearable health monitoring sensors and autonomous vehicles [117]. It is predicted that the number of connected devices will reach more than 75 billion up to 2025 [164]. The proliferation of IoT devices leads to a huge amount of network traffic. Driven by this demand, the next generation of wireless technology such as 5G has begun being deployed worldwide [117]. The newly emerging wireless technologies and application scenarios requires an efficient and robust network control architecture. However, due to the essence of wireless communications, unique challenges of developing a network control architecture exist in multiple aspects compared with traditional wired networks.

Diversity of Network Services. IoT is supporting diverse use cases, which may have different requirements of network resources. For example, the low network latency is crucial for the autonomous vehicles. While based on high-resolution video streaming, the virtual/augmented reality (VR/AR) applications require a large bandwidth [43]. 5G networks have provided with specifications for these new types of services, including ultra-reliable and low-latency communications (URLLC), enhanced mobile broadband (eMBB) and massive machine-type communications (mMTC) [135]. These distinct demands must be satisfied simultaneously with the same network infrastructure. It is the responsibility of the network control architecture to allocate and schedule different types of network resources properly.

Another important trend that cannot be neglected is the popularity of Artificial intelligence (AI) and machine learning (ML) technologies. ML has been shown as an effective way to analyze and classify the massive data generated by IoT sensors [69]. However, it also results in heavy computation and communication overheads when deployed in the wireless network. The compatibility of the network control architecture is indispensable for integrating the AI technologies in wireless and IoT scenarios.

Heterogeneous Wireless Networks. Most modern mobile devices, such as laptops and smartphones, are capable to connect to multiple types of radio access networks (RANs), for example, the cellular networks and wireless local area networks (WLANs) [125]. These RANs have different communication protocols and mechanisms allocating radio resources. The wireless control architecture should have the ability to achieve resource allocation and traffic engineering across different access technologies, which can be even more difficult when the mobile device has multi-connectivity to more than one RAN simultaneously.

Besides the access network, such heterogeneity of protocols also exists in IoT networks. IoT devices have distinct communication protocols with different design purposes from the traditional network services, such as ZigBee [42] and 6LoWPAN [112] which focus on low data-rate and low energy-consumption applications. There are also multiple popular protocols coexisting in the application layer, such as MQTT [163] and CoAP [23]. It is difficult for the control architecture to handle the different communication and routing patterns brought by these protocols in a uniform and efficient way. What is more, these protocols are vulnerable to different types of network attacks, enlarging the risk of the IoT network being attacked. It is another task for the control architecture to develop corresponding network security policies.

Scalability and Robustness The wireless networks have more dynamics than wired networks. It is common for the wireless communication channels to have fluctuating signal strength, leading to wireless links with unstable connections and limited bandwidth. What is more, a large portion of end devices in the wireless environment are mobile. The movement along with the joining and leaving of network devices causes the frequent changing of network topology. The control architecture must be robust enough to maintain the network management under all these changes with limited available bandwidth of the control channel. The scalability of the solution is another difficulty because of the large amount of interconnected devices in IoT.

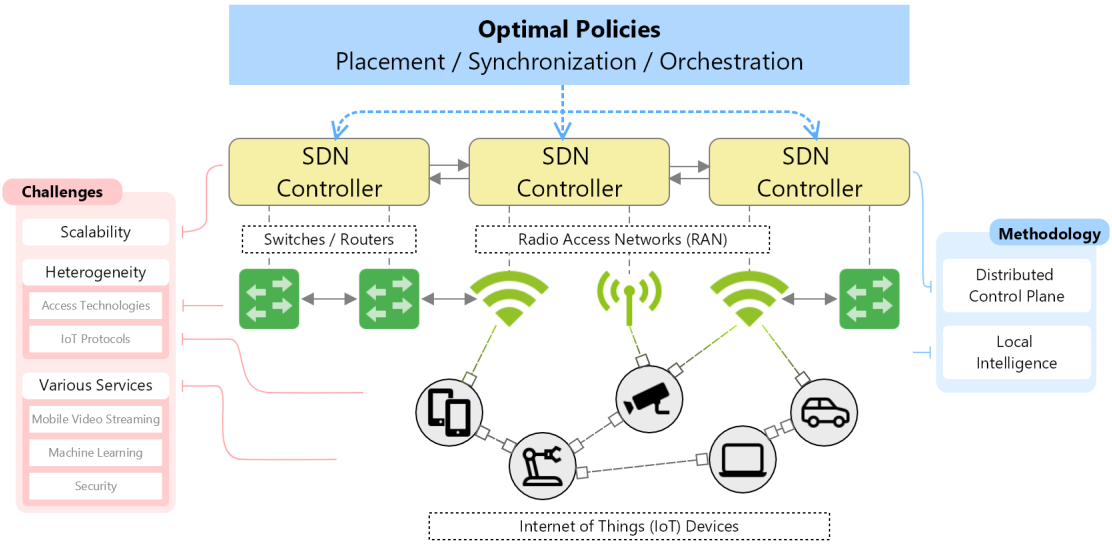


Figure 1.1: An overview of the proposed softwarized architecture.

1.2 A Softwarized and Hybrid Control Architecture

Driven by the challenges listed above, this thesis adopts Software-defined Networking (SDN), which is a promising paradigm that revolutionizes the way for network management [77]. The key concept of SDN is to decouple the control plane and data plane of the network to simplify the network control. In this way, a centralized network controller is set up, having the global view of the whole network. Then, the network control policies can be made accordingly in a programmable manner through protocols such as OpenFlow [103].

Having achieved great successes in wired environments such as data center networks [68], approaches are also being made to adopt SDN in wireless scenarios. The programmability makes SDN an enabler of network function virtualization (NFV) technology. Multiple virtual network functions can be deployed using the same set of network infrastructure. Therefore, SDN and NFV provide a solution to satisfy the diverse demands of various wireless network services [185]. However, the remaining two challenges of the wireless control architecture cannot be overcome straightforwardly. Popular SDN standards such as OpenFlow do not have enough support for different wireless protocols and radio resources. The centralization of the SDN controller makes the control link a bottleneck of traffic overheads, and even becomes a single point of failure under unstable wireless connections, weakening the robustness and scalability.

To leverage SDN for wireless network control, this thesis considers making improve-

ments to the basic SDN architecture in the following two aspects:

- **Distributed Network Control Plane**. To strengthen the robustness of the control architecture, instead of only one controller, multiple controller instances can be deployed at different locations of the network. Although controllers are no longer physically centralized in this setting, the centralization can remain at the logic level, as long as the controllers synchronize with each other to have the same information of network states. In this way, the single point of failure problem can be solved.

  Maintaining multiple controllers and keeping them synchronized bring additional costs. Some further concerns are required. One of them is the controller placement problem. The amount and locations of controllers much be deliberately considered in order to minimize the costs. Another concern is the synchronization mechanism. It is non-trivial to determine how frequently a controller shares the information to its peers, so that all controller instances can remain consistent under the changing network environment with reasonable communication costs. These issues should also be considered in the context of the heterogeneous network cases, where the controller instances belong to different RANs.

- **Local Intelligence in the Network Data Plane**. Despite the huge advantages brought by the centralized network control, it is also necessary to consider the merits of keeping certain control logic distributed. On the one hand, though lacking the flexibility and programmability, traditional distributed routing protocols such as AODV [132] and OLSR [34] for wireless networks have shown a better robustness towards network topology changes and failures. On the other hand, edge computing has been developed as a solution of the growing IoT traffic [187], where data are processed at the local gateway to reduce the service latency and the bandwidth costs to the centralized cloud. With networking devices becoming smarter and programmable, it is even possible to deploy some local functions in the network data plane, i.e. executing the function logic totally in the forwarding devices without reporting to the controller and waiting for commands. This further reduces the packet processing latency, leading to higher efficiency.

  A trade-off exist between the centralized and distributed control architectures. For this approach, the main task is to find out how many and what services should be moved local from the centralized SDN controller, as well as moved from the control plane to the data plane, so that the efficiency and scalability can be maximized by combining the merits of both control schemes.

This thesis will propose a novel control architecture applying above improvements, namely the hybrid SDN-based control architecture, as depicted in Figure 1.1, to overcome the challenges of today's wireless networks and IoT application scenarios. All the additional concerns brought by the distributed control plane and local intelligence will also be discussed.

1.3 Summary of Contributions

This dissertation aims at developing a hybrid architecture based on Software-defined Networking for wireless network control. More specifically, the proposed architecture will focus on solving the *heterogeneity* and *scalability* problems in the context of various cutting-edge IoT applications scenarios. The outcome of this dissertation includes the architecture design, theoretical optimization and prototype implementation based on state-of-the-art open-source SDN software. To summarize, this work makes the following novel contributions:

- *Heterogeneity support with the distributed control plane.* We investigate the application of SDN in heterogeneous radio access networks, including WLAN and LTE. We propose an architecture allocating radio resources to different network services across both LTE base stations and WLAN access points. To handle the competition among network and service providers, we design an auction-based algorithm performing negotiation among different participants towards the optimal allocation policy. We deploy our system in real networking devices and verify its performance with multiple typical network applications such as the web service and mobile video streaming.

- *Scalability support with the distributed control plane.* We focus on two crucial issues during the deployment of the distributed SDN control plane, the *controller placement problem* and the *controller synchronization problem*. By measuring the costs of several production-ready SDN controllers on real mobile devices, we figure out the pattern of the traffic overhead and delay during the intercommunication among different controllers. We then propose an optimization algorithm determining the locations to deploy controllers that minimizes theses costs, as well as a novel synchronization mechanism that reacts to the feedback from network applications to reach a balance between performance and costs. We evaluate our methods with both routing and load balancing services.

- *Heterogeneity support with the local intelligence.* We leverage the data plane programmable model, P4 [24], to deploy gateways in the IoT networks capable of supporting heterogeneous protocols. Taking IoT security as a representative application, we propose a machine-learning-based algorithm classifying incoming packets into multiple categories with a high accuracy. Packets of unknown protocols can be processed without preknowledge, generating protocol definitions and flow tables which are directly executed in the gateway data plane.

- *Scalability support with the local intelligence.* We then extend the IoT machine learning framework to networks of larger scales with multiple edge domains. The scalability problem is addressed regarding two aspects. First, we reduce the network latency of executing computation-intensive machine learning algorithms by adopting binarization techniques to make the learning models lightweight, so that they can be offloaded from the central server to the local data plane. Moreover, we propose a federated learning [104] scheme able to train algorithms with local samples at the edge without uploading the whole dataset to the central cloud. We investigate additional techniques to minimize the communication overhead during this process for better scalability.

1.4 Organization

The remainder of this dissertation is organized as follows:

Chapter 2 background knowledge and literature reviews of the topics relevant to this dissertation.

Based on our previous work [139], Chapter 3 presents the optimal scheme of distributed controller placement.

Chapter 4 extends our previous work [137] and presents the mechanism design and optimization algorithm of distributed controller synchronization.

Chapter 5 is derived from our previous work [138], proposing the architecture of an orchestrator that achieves network slicing over SDN controllers of heterogeneous RANs.

Based on our previous work [141], Chapter 6 describes enhancing the SDN control architecture with local intelligence enabled by the programmable data plane.

This approach is further discussed in Chapter 7 about how to improve the efficiency and scalability, which is based on our previous work [140].

Chapter 8 summarizes the conclusions and provides with an outlook for future work.

Chapter 2

Background

In this chapter, we describe the necessary context of our work. We review the concept and development of the Software-Defined Networking (SDN) movement. Specifically, we focus on approaches which increase the robustness and scalability of the system or extend SDN into wireless networking scenarios. Such approaches are made in both the network control plane and data plane.

2.1 Software-Defined Networking

Software-defined Networking (SDN) is a successful approach to make the computer networks programmable [44]. In tradditional networking devices such as switches and routers, packet forwarding behaviors are configured through the devices' interfaces, which have fixed functions depending on specific vendors, with which network innovations are difficult to be made to satisfy the increasing traffic volumes and demands. SDN solves this problem by separating the network control and data planes [77]. A centralized SDN controller is deployed at a remote server which defines the packet forwarding behaviors with a high-level abstraction of network elements. The controller manages multiple forwarding devices through standardized protocols. OpenFlow [103] is one of the most widely-used SDN standard. An OpenFlow switch processes packets according to flow tables consisting of multiple match-action rules. It keeps a TCP connection to the SDN controller to report packets that cannot be matched in the flow table. The controller will make decisions and install new flow table entries in the switch data plane. Originated in the campus network, OpenFlow has extended its application to more wired networking scenarios. The standard has been supported by hardware switches from multiple equipment vendors, as well as software, virtualized switches such as the Open vSwitch [133] project and Intel Data

Plane Development Kit (DPDK) [65].

Through protocols like OpenFlow, the SDN controller acquires a global view of the network, facilitating the network operators to make decisions with great flexibility. Multiple controller platforms have been developed after the creation of OpenFlow, such as NOX [55], Floodlight [1], POX [74] and Ryu [2]. Network functions, such as routing and traffic engineering can be designed with popular programming languages including C++, Java and Python. Then, such functions can be deployed as a module in the controller platforms without any modification on the network infrastructure.

2.2 Control Plane Approaches

In spite of the successful use cases of SDN and OpenFlow in campus [103] and data center [68] networks, extra efforts are required to bring SDN to the wireless environment. On the one hand, the centralized architecture design is not scalable enough with the high dynamics and the large amount of interconnected devices in wireless networks, especially IoT applications, because of the single point of failure problem. Distributed deployment of controllers is proposed to tackle with this problem. On the other hand, some properties of wireless networking are not considered in the design of SDN protocols such as OpenFlow. Several extensions and new protocols are proposed to adopt the SDN concept in wireless scenarios.

2.2.1 Distributed SDN Controllers

ElastiCon [40] is an early work proposing a distributed SDN architecture by deploying a cluster of multiple controllers in the network. While being scalable, in order not to lose the advantages of the global network view and programmability, the control plane should keep centralized logically. ElastiCon deploys a database sharing its access to all controllers to achieve this. Multiple other approaches [13] exist, keeping the consistency of the whole controller cluster either by deploying a hierarchical architecture [81], or making each controller synchronize with all peers. Among them, ONOS [16] and OpenDaylight [105] are two representative controller platforms that provide with built-in solutions for the distributed deployment and achieve production-grade performance. However, the synchronization among different controller instances leads to large communication overheads, as measured in [113] and [167]. Therefore, the main usage of above approaches is still limited in the data center networks rather than wireless scenarios.

2.2.2 Software-Defined Wireless Networks (SDWN)

Software-defined Wireless Networks (SDWN) [183] are approaches that apply the core SDN idea of separating the network control and data planes to wireless scenarios. Due to the variety of heterogeneous wireless protocols, these approaches differ a lot from each other and focus on different scenarios. For instance, SoftCell [71] is an architecture for LTE core networks which is aware of network dynamics such as topology changes. There are other approaches for cellular core networks including [33] and [131].

Another research direction is Software-defined Radio Access Networks (SD-RAN), focusing on applying the SDN centralized control over Radio Access Networks (RAN). CAPWAP [17] is a protocol for the centralized management of a collection of WLAN access points. Odin [168] proposes the light virtual access point (LVAP) abstract. Such virtualization technologies make it possible to allocate radio resources among multiple users or network services, which is an important application of SD-RAN. Similar approaches also exist for LTE access networks. FlexRAN [48] and Orion [47] achieves network slicing among multiple LTE eNodeBs through virtualization. Fewer works consider the heterogeneous cases, e.g., coexisting LTE and WLAN. SoftMobile [32] and EmPOWER [146] are architectures for this purpose. However, different problems remain to be solved such as the scalability of the system as well as the competition among multiple RANs and service providers.

Software-defined Wireless Sensor Networks (SDWSN) [79] approahces such as Sensor OpenFlow [100] and SDN-WISE [51] extend OpenFlow to support IoT protocols. They also aim at limiting the traffic overheads between the controller and IoT devices due to the low-rate essence of IoT communications.

2.3 Data Plane Approaches

The approaches introduced so far focus on enhancing the capability of the network control plane, while the functions of the data plane remain relatively simple. In both the original and modified OpenFlow protocols of above approaches, the packet header fields matched by flow rules are predefined. It takes great efforts to extend these definitions so that a new protocol with unique packet headers can be supported, as indicated in [100]. Besides, the OpenFlow switches are stateless. Without storing the states of different network connections, any complex packet processing behaviors can hardly be achieved at the switch without the participation of the SDN controller, which may hamper the scalability of the

system. To solve this problem, innovations also happen at the data plane design, making the forwarding devices stateful and even programmable [21]. These novel architectures enable in-network computing, which realizes network functions with low costs.

2.3.1 Programmable Data Planes

OpenState [20] and FAST [109] achieve stateful data plane packet processing by replacing the stateless match-action table with a state machine, where packet can trigger state transitions. These works have shown that such switch architectures can realize more advanced network services locally, such as TCP connection tracking and stateful firewalls. P4 [24] is a more comprehensive high-level language enabling network engineers to dynamically program the packet processing pipeline. It has two additional features. P4 is protocol independent that permits users to define new packet header structure and parsers. It is also target indepdent, which can be deployed in various devices including high-performance hardware switches [25], FPGA [191] and software switches [155, 35]. These features make it possible to be deployed in IoT networks serving packets in heterogeneous protocols. For example, [172] propose a P4-based architecture for the service automation of several popular IoT protocols.

2.3.2 In-Network Computing

The increasingly capable and programmable data plane devices motivate people to develop in-network computing [153] technologies. By offloading some computation tasks to the data plane, the network throughput and latency will be improved because packets are no longer forwarded to a remote server or host. Though having more constraints of memory and computation power compared with genral CPUs, the in-network computing schemes are evaluated in [170] through case studies of different applications including key-value storing and consensus algorithms to show advantages. [180] and [161] claim that the machine learning algorithms which are regarded computation-intensive can also be assisted or performed by the data plane. There are also IoT-related use cases, such as the architecture proposed in [98], which aggregate data from multiple sensors in P4 switches.

2.4 Summary

We introduced the SDN architecture in this chapter. We figured out that the distributed deployment of the SDN control plane and the in-network computations with the pro-

grammable data plane are two potential approaches to tackle with the scalability and heterogeneity issues of adopting SDN in wireless and IoT scenarios. Describing the related works in both areas, in the next chapters, we will present a series of our works that solve the open problems in either approach, as well as combining both approaches for a better control architecture.

Chapter 3

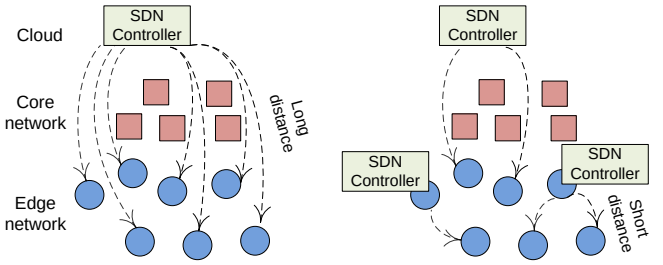
Distributed Control Plane: Controller Placement Problem

Driven by the trends of edge and fog computing in IoT applications which provide elastic resources and services to end users, the network control plane should be deployed at the network edge, since the processing capacity resides at the network periphery as opposed to traditional data-centers. Despite their momentum, the deployment of such control architecture is a complex and challenging problem.

In this chapter, we propose to adopt SDN control for such scenarios. We provide a proof-of-concept implementation of a multi-controller edge system and concentrate on the deployment strategy of controllers. Guided by our measurement of traffic delay and overheads, we model the problem of determining the placement of controllers in the edge network. Using linearization and supermodular function techniques, we present approximation solutions which perform close to optimal and substantially better than state-of-the-art methods. Finally, we analyze the interplay between various performance and reliability objectives.

3.1 Introduction

Emerging architectures, such as fog and mobile edge computing, distribute substantial amounts of data storage, processing and communication resources at the extremes of the network, in proximity to end-users, thereby allowing to bypass fundamental delay issues of traditional centralized cloud platforms [108]. While still at an infancy stage, these architectures are considered to be a key enabler for next-generation wireless (5G) and Internet of Things (IoT) systems [130] for supporting both computation-intensive and delay-sensitive



(a) Single controller. (b) Distribution of controllers.

Figure 3.1: A remote controller vs many controllers placed at the edge.

services.

Despite their momentum, resource management in fog/edge architectures remains a very complex task, especially when a diverse set of services with different computation/storage/communication requirements need to be supported [157]. To facilitate resource management, we can benefit from novel softwarization technologies such as SDN. The main principle of SDN is to shift all the control functions from the data plane nodes to a programmable network entity, the controller. However, *this is a centralized approach, while edge architectures emphasize the distribution of resources and their management*. Therefore, it is challenging to apply SDN ideas at the edge part of the network.

To exemplify, in order for SDN protocol to work properly, the state of the data plane nodes, e.g., the traffic statistics, link metrics and other protocol-specific parameters [181], should be reported to the controller in a *timely manner* so as to make efficient resource management decisions. This condition is easier to meet in wired networks where the communication between the controller and the nodes is much more stable and faster than in the wireless counterpart.

A method that can be used to solve the above issue is the *placement of many controller instances in proximity to edge nodes*, as it is depicted in Figure 3.1. The placed controllers, physically distributed but logically centralized, cooperate to manage the edge nodes which can reduce delay due to the shorter distance to them. Such placement strategies are possible today via commercial software controller implementations which support a cluster mode with built-in synchronization mechanisms [105], [16].

The controller placement problem has been extensively studied over the past decade, especially in the context of data center and wired ISP networks (e.g., see the pioneer work in [60] and [176] for a survey). However, this problem obtains an interesting new twist in the context of edge architectures for the following reasons:

- *Delay of network management*: Certain links between the edge nodes may be wire-

less in nature, unstable and of low rate. Moreover, it may happen that many of these links separate a node from a placed controller resulting in slow statistic collection and node re-configuration through a multi-hop path. Hence, the controller placement strategy can drastically affect the delay of network management.

- *Overhead of control messages*: Multi-controller implementations require the periodic exchange of messages between the controllers and nodes for statistic collection and resource management [181] as well as between the controllers themselves for synchronization purposes [113]. The overheads of these two types typically increase with the number of placed controllers, their distance to the nodes and to each other. Hence, if the controllers are not properly placed, the overheads will be significant, considering the scarcity of edge network resources.

- *Heterogeneous synchronization strategies:* There are no standard protocols for synchronization among controllers. Therefore, behaviors may vary with different types of controllers. This variety should be considered when deciding the controller placement.

Given the above issues, the key open questions are: *How many controllers to place in the network and where exactly? Should we place many controllers close to the edge nodes to reduce delay of resource management or place fewer controllers close to each other to keep synchronization overheads as low as possible?*

In this chapter, we follow a systematic methodology in order to answer the above questions. We focus on networks where edge devices are SDN-compatible and can support SDN data paths and controllers, such as IoT systems. The contributions of this work can be summarized as follows:

- *SDN Controller Placement at the Edge*. We study the placement of controllers in SDN-enabled edge networks. We consider several practical features of these systems such as the different delay values of the wireless links and the impact of control overheads.

- *Experimentation and Emulation*. We analyze the operation and synchronization strategies of two state-of-art SDN controller implementations, namely ONOS and OpenDaylight (ODL). We perform experiments on a testbed of a multi-controller edge network to show that the average delay of managing a device can significantly change for different controller placement solutions. We also perform large-scale

emulations to identify two types of overheads (inter-controller and controller-node traffic) and their dependence on the network topology.

- *Optimization Algorithms*. We build upon the emulation findings to formulate the controller placement problem for two different objectives; minimization of delay and overheads. We propose exact solutions of Mixed Integer Programming (MIP), as well as scalable and fast approximate solutions (running in less than 0.1 secs) using linearization and supermodular techniques.

- *Evaluation Results*. We evaluate the proposed controller placement algorithms using two real network topologies. We find that our approach performs close to optimal and better than state-of-the-art methods. We also analyze the interplay between various performance and reliability objectives; minimizing delay can favor the reliability of controller - data plane node communication, but affect the reliability of inter-controller communication.

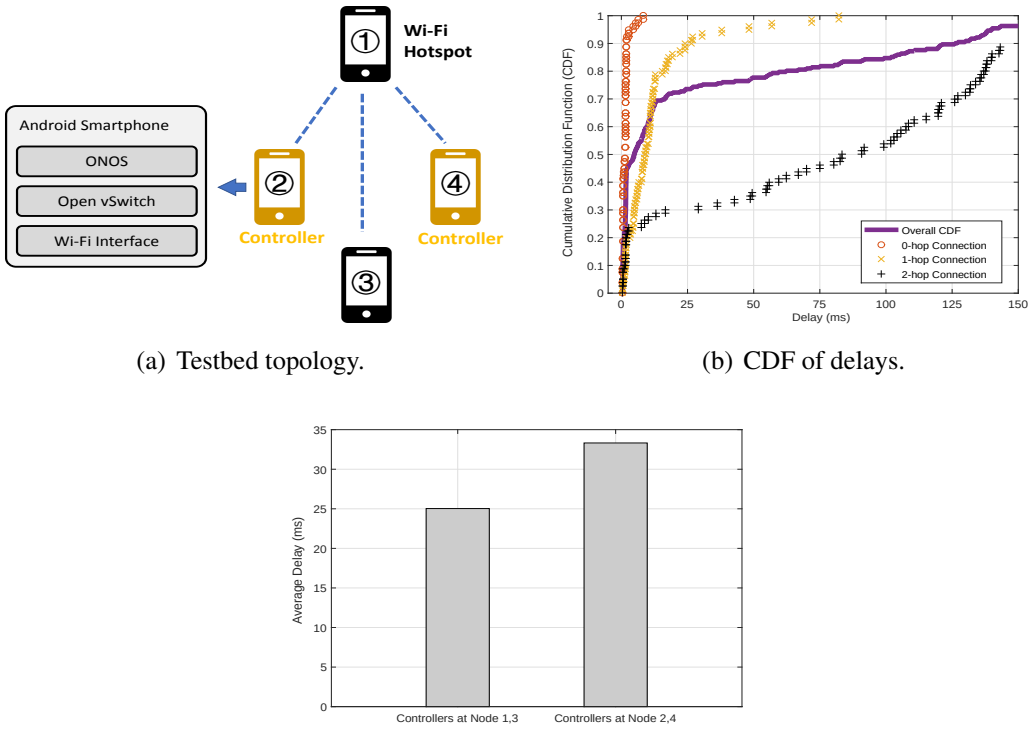
The rest of this chapter is organized as follows. Section 3.2 presents our experimentation and emulation results. Guided by these results, we model the controller placement problem in edge networks in Section 3.3. In Section 3.4, we present optimal and approximate solution algorithms for small- and large-scale problem instances respectively. Section 3.5 presents the evaluation of our proposed algorithms, while Section 3.6 reviews our contribution compared to related works. We conclude our work in Section 3.7.

3.2 Cost Analysis of Controller Clusters

In this section, we present experimentation and emulation results using commercial SDN controller and data plane implementations. The results provide insights about the delay and overheads of multi-controller edge systems which will be used in modeling the controller placement problem in the next section.

3.2.1 Control Delay Measurement

Testbed Set-up. In this subsection, we set up a testbed of a multi-controller edge system using off-the-shelf network devices. Specifically, we deploy four Nexus 4 Android smartphones to form a wireless network as it is depicted in Figure 3.2(a). The first smartphone works as an access point (hotspot) to provide the remaining three smartphones with Wi-Fi connections. This represents a common edge network setting, where a node can either



(a) Testbed topology.

(b) CDF of delays.

(c) Comparison of two placement schemes.

Figure 3.2: (a) Testbed of a multi-controller edge system built from smartphones that enable Open vSwitch and ONOS. (b) CDF of controller-node communication delays. (c) Comparison of average delays under different controller placements.

establish multihop connections to backbone networks, or exchange data with its peer in a D2D fashion. Besides, smartphones are representative devices widely used in edge networks. Due to their constraints of calculating and storage, our testbed shows a challenging scenario that is worthy of investigation.

We take several steps to make the network SDN-enabled. In each smartphone, we create a *chroot* environment to install the Ubuntu system running with its original Android system at the same time. By this, we are able to install popular SDN-related software in the smartphone. First, we make all devices working as data plane nodes by installing Open vSwitch [133]. This creates a virtual switch that supports SDN in each smartphone. Second, we deploy SDN controllers in Node 2 and Node 4. Though constrained in resources, the smartphone is still capable enough to run a controller instance, such as ONOS. ONOS is designed particularly for scalability and permits multiple controllers working together in the form of a *controller cluster*. Then, we establish a connection between the two controllers and assign each smartphone to its nearest controller with respect to the hop count

length distance metric.

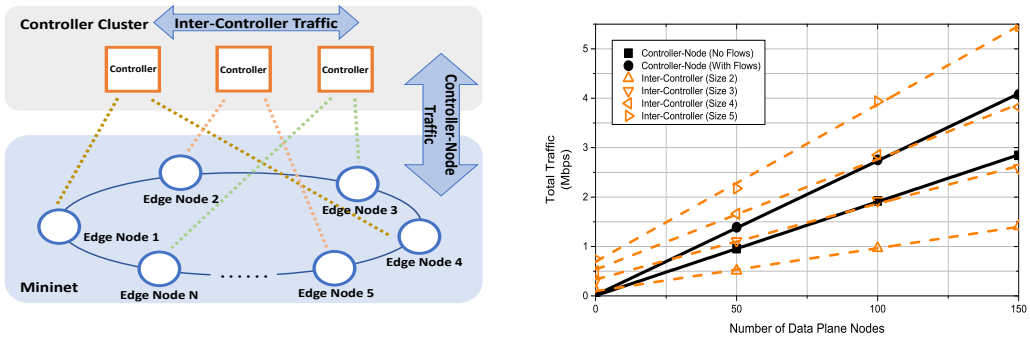
Measurement Methodology. We mainly take measurements on the network delay between the controller and data plane nodes. The devices are placed within an empty lab room, and distance over each wireless link is 2 meters. One frequent and important interaction between a controller and a data plane node is the request and reply of flow statistics. Therefore, we measure the delay at controller nodes, by analyzing the interval between sending such an OpenFlow request message and receiving its corresponding reply. We keep capturing messages since the cluster reaches the steady state and collect 250 measurements. Figure 3.2(b) shows the cumulative distribution function of the delays we measured. The average value is tens of milliseconds which is comparable or higher than the delay reported in typical wired networks [60]. From the CDF plot, we notice that the variance is large, corresponding to the relatively unstable wireless links. It is common for the delay to go even beyond 100 milliseconds.

We also notice that the placement of controllers is important, because the delay relies highly on the distance between the data plane node and its controller. For example, Node 2 and Node 4 contain controllers locally, while Node 1 and Node 3 have one-hop and two-hop connections to the controller, respectively. As a result, drawn separately in Figure 3.2(b), local connection shows almost zero delays while the one-hop and two-hop connections show notable delays. To further demonstrate this, we move ONOS controllers from nodes 2 and 4 to nodes 1 and 3. If we still assign each data plane node to its nearest controller, we can find that the average delay is 25% lower, as it is depicted in Figure 3.2(c).

Main Takeaways. *Modern edge network devices (smartphones) can act as controllers to manage other devices. The management delay highly depends on the number of wireless hops and can significantly change for different controller placement strategies (up to* 25% *difference in our testbed).*

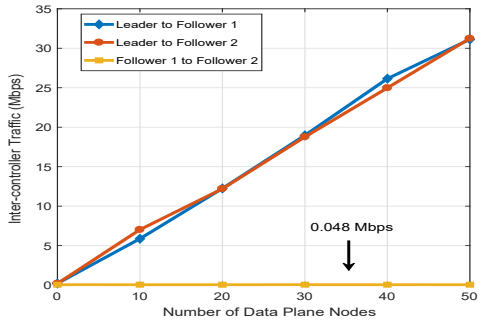
3.2.2 Control Overhead Measurement

In the previous subsection, we focused on the delay required to manage the edge nodes. In this subsection, we will analyze another important factor; the overheads of SDN control. By measuring the overheads of ONOS and OpenDaylight, the most typical commercial multi-controller solutions, we identify two types of control overheads, controller-node overhead and inter-controller overhead. We also summarize two types of synchronization strategies, leaderless synchronization and leader-based synchronization.



(a) Emulation setting.

(b) Control overheads of ONOS.



(c) Control overheads of OpenDaylight.

Figure 3.3: (a) Ring topology in emulations. (b) Controller-node overheads (solid lines) and inter-controller overheads (dot lines) of ONOS controllers. (c) Overheads of OpenDaylight controllers.

Emulation Set-up. Since control overheads depend heavily on the scale of the network, we need to deploy many more nodes than what we have in our testbed if we wish to analyze them. A more accessible way to take large-scale measurements is by running emulations on a virtual network generated by Mininet [86]. This method allows us to test networks with hundreds of nodes and several controllers using a common CPU machine. Specifically, we create a virtual edge network with ring topology and evenly assign nodes to the placed controllers, as shown in Figure 3.3(a). All controllers run a simple built-in application named reactive forwarding.

Measurement Methodology - ONOS. First, we run ONOS as the controller. In order to show the impact of the scale of the network, we take measurements on both types of control traffic with different number of nodes in the virtual network. Figure 3.3(b) verifies that both controller-node traffic and inter-controller traffic grow linearly when the network scales up. What is more, we also consider other factors that have impact. For controller-node traffic, we create a large amount of one-hop *iperf* [169] flows randomly, with a fixed

rate (0.1 flows per second for each node). For inter-controller traffic, we deploy different numbers of controllers (cluster sizes). According to Figure 3.3(b), in all of these situations, the two types of overheads are at the same order of magnitude (up to a few Mbps each). This fact means that both of them are important when deciding the controller placement.

We also analyze the synchronization strategy adopted by ONOS. ONOS has a relatively complicated consensus mechanism, deploying several different algorithms at the same time, which we discuss in detail in the Appendix A.1. However, *the majority of traffic follows a leaderless manner*, i.e., each pair of controllers generates a similar amount of overheads when synchronizing.

Measurement Methodology - OpenDaylight. Next, we replace the controllers with OpenDaylight. Same with ONOS, OpenFlow protocol is adopted for the controller-node communications, leading to similar behaviors. However, the inter-controller traffic pattern is quite different. As shown in Figure 3.3(c), although the overhead also grows linearly with the number of data plane nodes, the overhead is much larger compared with the same setting in ONOS case. Importantly, the overhead is no longer evenly distributed among each pair of controllers. In the 3-controller case, we can identify one controller as a leader, and the remaining two as followers. We find that non-negligible overhead only exists between a leader and a follower, rather than two followers. Therefore, *controller synchronization of OpenDaylight follows a leader-based manner.*

Main Takeaways. *(i) The two types of overheads (inter-controller and controller-node traffic) are at the same order of magnitude in representative scenarios (up to few Mbps each), increasing linearly to the load of controllers. (ii) There are leader-based and leaderless strategies for controller synchronization, leading to different distributions of inter-controller overheads.*

3.3 Problem Modeling

In this section, we build upon the experimentation and emulation results of the previous section to model the controller placement problem in edge networks. We consider a network of a diverse set $\mathcal{N}$ of N edge nodes such as access switches, cellular base stations, set-top boxes, Wi-Fi access points and even mobile devices (e.g., smartphones), as depicted in Figure 3.4. We use the term edge nodes to describe them as they are in close proximity to the end user, unlike core switches and routers of ISP backbone networks or data centers. Our analysis applies to any kind of edge nodes as long as: (i) they are SDN-compatible; (ii) their links have high enough capacity to support the SDN coordination

mechanisms without congestion. Furthermore, the edge nodes can be connected with each other through single or multi-hop paths. Without loss of generality, the nodes generate new flows with uniform rate, normalized to one.

The network is SDN-enabled in the sense that all nodes run virtual switches that support SDN protocol. A controller is placed at the cloud and connects to the edge nodes through in-band or out-of-band control channels. The network operator may decide to place additional controllers in the network. Placing a controller on an edge node requires to locally install and run a controller implementation software such as ONOS [16]. We introduce the binary optimization variable $x_n \in \{0, 1\}$ to indicate whether a controller is placed at node $n \in \mathcal{N}$ ($x_n = 1$) or not ($x_n = 0$). These variables constitute the *controller placement policy*:

$$\boldsymbol{x} = (x_n \in \{0, 1\} \ : n \in \mathcal{N}) \ . \tag{3.1}$$

Due to limited resources, not all the edge nodes may be capable of hosting a controller. To model such cases, we denote by $\mathcal{N}_h \subseteq \mathcal{N}$ the subset of nodes that can play the role of host for a controller, where $N_h = |\mathcal{N}_h|$. Then, we require that:

$$x_n = 0, \ \forall n \notin \mathcal{N}_h \ . \tag{3.2}$$

The network operator will also need to decide the assignment of nodes to controllers, i.e., which controller is responsible for the management of each node. We introduce the binary optimization variable $y_{nm} \in \{0, 1\}$ to indicate whether node $n \in \mathcal{N}$ is assigned to the controller at node $m \in \mathcal{N}$ ($y_{nm} = 1$) or not ($y_{nm} = 0$). Similarly, we denote the cloud node by c, and $y_{nc} \in \{0, 1\}$ indicates the assignment of node n to the controller located at the cloud. These variables constitute the *assignment policy* of the operator:

$$\boldsymbol{y} = (y_{nm} \in \{0, 1\} \ : n \in \mathcal{N}, m \in \mathcal{N} \cup \{c\}) \ . \tag{3.3}$$

Since every node needs to be assigned to a controller, we require that:

$$\sum_{m \in \mathcal{N} \cup \{c\}} y_{nm} = 1, \ \forall n \in \mathcal{N} \ . \tag{3.4}$$

In addition, we require that a controller must be placed at node m in order for node n to be able to assign to it:

$$y_{nm} \leq x_m, \ \forall n, m \in \mathcal{N} \ . \tag{3.5}$$

We also consider that in the leader-based synchronization strategy, one controller acts

as the leader. We use the optimization variable $z_n \in \{0, 1\}$ to indicate whether a node n is the leader or not, and require that:

$$\sum_{n \in \mathcal{N}_h \cup \{c\}} z_n = 1 \ . \tag{3.6}$$

As we showed in Figure 3.2(b), the delay of node management increases rapidly with the topological distance (number of hops) between the controller and the node. Therefore, assigning a node to a controller placed at a nearby edge node instead of the cloud controller can greatly expedite its management. In the model, we denote by d_{nm} (milliseconds) the delay when node n is assigned to the controller at node m. Similarly, we denote by d_{nc} the delay when node n is assigned to the cloud controller. The total (across all nodes) delay is given by:

$$D(\boldsymbol{y}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{N} \cup \{c\}} y_{nm} d_{nm} \ . \tag{3.7}$$

The controllers will continuously exchange messages with the data plane nodes they manage for statistic collection and forwarding table update. As we showed in Figure 3.3(b), the bandwidth overhead of this message exchange can be significant in practice. Moreover, the cost of this overhead would increase with the topological distance between the controller and the node as resources of more links would be consumed. To model this, we denote by w^a_{nm} the overhead cost of assigning node n to the controller at node m. Similarly, the overhead cost of assigning node n to the cloud controller is denoted by w^a_{nc}. The total (across all nodes) assignment overhead cost is given by:

$$W_a(\boldsymbol{y}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{N} \cup \{c\}} y_{nm} w^a_{nm}. \tag{3.8}$$

The controllers will also exchange messages to each other for synchronization purposes. As we show in Section 3.2, different types of controllers may adopt either leader-based or leaderless strategy. We should notice that they may even coexist, in case that a controller adopts multiple consensus algorithms, just as ONOS. Therefore, we model the overheads of each strategy separately.

We begin with the leaderless case. As what Figure 3.3(b) indicates, each pair of controllers exchange messages with *constant rate* as well as messages with *rate that depends on the controller's load*. The latter means that the more nodes are assigned to a controller

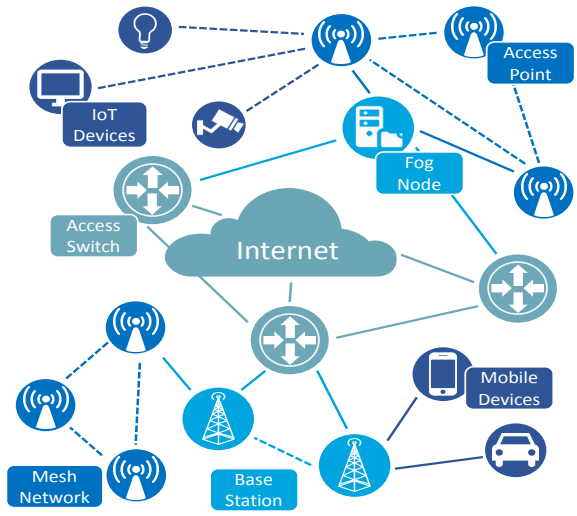


Figure 3.4: An example of an SDN-enabled edge network.

the more messages it exchanges with the rest controllers. Therefore, we can model the synchronization overheads as follows. For the messages exchanged at a constant rate, we denote by $w_{ml}^{con} \geq 0$ the respective overhead cost between controllers at nodes m and l. For the messages exchanged at a rate that depends on the controller load, we denote by $w_{ml}^{dep} \geq 0$ the respective overhead cost between controllers at nodes m and l for each node assigned to controller m. The total (across all controller pairs) leaderless synchronization overhead cost is given by:

$$W_{s1}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} x_m x_l \big(w_{ml}^{con} + w_{ml}^{dep} \sum_{n \in \mathcal{N}} y_{nm} \big). \tag{3.9}$$

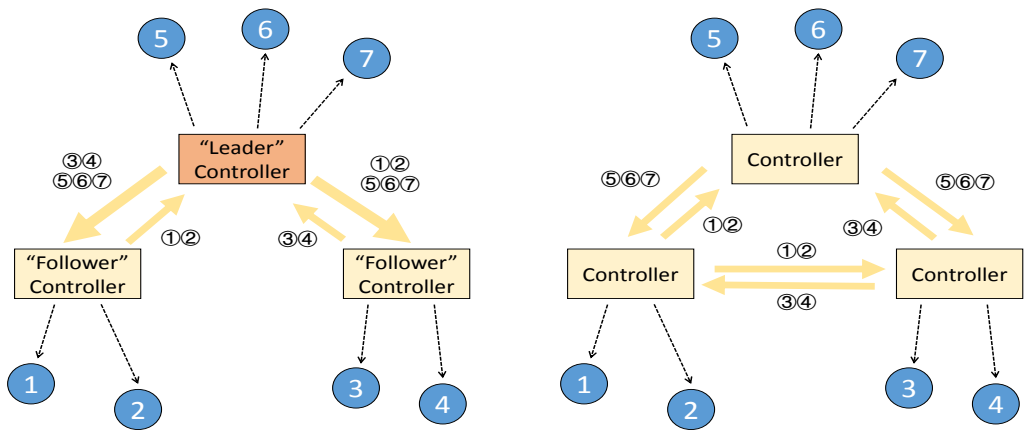
where, with *a slight abuse of notation*, we used the terms x_m and x_l for m and l equal to c in the above summation. These terms are set to one to indicate that a controller is placed at the cloud[1].

For the leader-based case, we have a similar cost expression:

$$W_{s2}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N}_h \cup \{c\}} x_m z_l \big(w_{ml}^{lb\_con} + w_{ml}^{lb\_dep} * N \big). \tag{3.10}$$

where the difference is that the controllers synchronize only with the leader controller, rather than in a peer-to-peer manner, as depicted in Figure 3.5. The constants $w_{ml}^{lb\_con}$ and

[1]We make the same abuse of notation throughout this chapter.



(a) Leader-based (b) Leaderless

Figure 3.5: A comparison between leader-based and leaderless synchronization strategies.

$w_{ml}^{lb\_dep}$ represent the linear relationship between the overhead and the controller load, just like in the leaderless term. However the values can be different.

On the one hand, a *scattered placement* of many controllers across the network would reduce delay and assignment overhead costs since nodes can be managed by controllers at closer proximity. On the other hand, a more *compact placement* of fewer controllers would reduce the synchronization overhead cost. These metrics are contradicting in general, and therefore cannot be all minimized at the same time. Depending on its preferences, the operator would have to find a way to balance delay and overheads. In general, this will require to optimize a function of the following form, where the weight value $\gamma \geq 0$ is used to balance the different metrics:

$$J(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \gamma D(\boldsymbol{y}) + W_a(\boldsymbol{y}) + W_{s1}(\boldsymbol{x}, \boldsymbol{y}) + W_{s2}(\boldsymbol{x}, \boldsymbol{z}). \tag{3.11}$$

By setting $\gamma = 0$, the balanced function reduces to the total overhead costs, neglecting delay. However, by increasing the γ value, more priority is given to the delay. The edge controller placement (ECP, for short) can be expressed as follows:

$$\begin{aligned} & min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}\ J(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \\ & \qquad s.t.\ \text{constraints: } (3.1),\ (3.2),\ (3.3),\ (3.4),\ (3.5),\ (3.6). \end{aligned}$$

The above problem is challenging since it contains discrete variables and a non-linear objective function with cubic and quadratic terms (inside W_{s1} and W_{s2}). In fact, it is not hard

to show that ECP is a generalization of the well-studied facility location problem [158], which is NP-Hard, by allowing facility opening (equivalently controller placement) costs to be non-constant (i.e., the synchronization cost depends on the distance between the controllers).

3.4 Optimization Algorithms

In this section, we address the ECP problem. We start by presenting an optimal algorithm that can be applied to small-scale problem instances. Following that, we present approximation algorithms that scale well with the size of the problem instance.

3.4.1 Small-Scale Optimal Solution

In this subsection, we find an optimal solution by converting ECP to a Mixed Integer Programming (MIP) problem. That is a problem with linear objective function and constraints. This conversion is important since there exist various commercial solvers, such as CPLEX [38], that can be directly used to solve this type of problems.

To obtain the MIP formulation, we apply *standard linearization techniques* [54]. Specifically, we introduce the following two vectors of additional optimization variables:

$$\boldsymbol{\theta} = (\theta_{ml} \in \{0,1\} \ : m, l \in \mathcal{N} \cup \{c\}) \ . \tag{3.12}$$

$$\boldsymbol{\delta} = (\delta_{ml} \in \{0,1\} \ : m \in \mathcal{N} \cup \{c\}), \ l \in \mathcal{N}_h \cup \{c\}) \ . \tag{3.13}$$

$$\boldsymbol{\phi} = (\phi_{mln} \in \{0,1\} \ : m, l \in \mathcal{N} \cup \{c\}, \ n \in \mathcal{N}) \ . \tag{3.14}$$

Then, we add the following linear constraints for θ:

$$\theta_{ml} \leq x_m, \ m, l \in \mathcal{N} \cup \{c\} \ , \tag{3.15}$$

$$\theta_{ml} \leq x_l, \ m, l \in \mathcal{N} \cup \{c\} \ , \tag{3.16}$$

$$\theta_{ml} \geq x_m + x_l - 1, \ m, l \in \mathcal{N} \cup \{c\} \ , \tag{3.17}$$

the following linear constraints for δ:

$$\delta_{ml} \leq x_m, \ m \in \mathcal{N} \cup \{c\}, \ l \in \mathcal{N}_h \cup \{c\}) \ , \tag{3.18}$$

$$\delta_{ml} \leq z_l, \ m \in \mathcal{N} \cup \{c\}, \ l \in \mathcal{N}_h \cup \{c\}) \ , \tag{3.19}$$

$$\delta_{ml} \geq x_m + z_l - 1, \ m \in \mathcal{N} \cup \{c\}), \ l \in \mathcal{N}_h \cup \{c\} \ , \tag{3.20}$$

and the following linear constraints for ϕ:

$$\phi_{mln} \leq \theta_{ml},\ m, l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N}\ , \tag{3.21}$$

$$\phi_{mln} \leq y_{nm},\ m, l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N}\ , \tag{3.22}$$

$$\phi_{mln} \geq \theta_{ml} + y_{nm} - 1,\ m, l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N}\ . \tag{3.23}$$

The D and W_a functions are already linear, so we only need to linearize the W_{s1} and W_{s2} function. This is possible with the new variables, as they can be written as:

$$\widehat{W_{s1}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} \big(\theta_{ml} w_{ml}^{con} + \sum_{n \in \mathcal{N}} \phi_{mln} w_{ml}^{dep}\ \big). \tag{3.24}$$

$$\widehat{W_{s2}}(\boldsymbol{\delta}) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N}_h \cup \{c\}} \delta_{ml} \big(w_{ml}^{lb\_con} + w_{ml}^{lb\_dep} * N\ \big). \tag{3.25}$$

Then, the MIP problem can be expressed as follows:

$$\begin{aligned} min_{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}, \boldsymbol{\phi}}\ & \gamma D(\boldsymbol{y}) + W_a(\boldsymbol{y}) + \widehat{W_{s1}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \widehat{W_{s2}}(\boldsymbol{\delta}) \\ & s.t.\ \text{constraints: } (3.1) - (3.6) \text{ and } (3.12) - (3.23)\ . \end{aligned}$$

The inequalities in (3.15)-(3.17) ensure that θ_{ml} will be zero if at least one (or equivalently the product) of x_m and x_l is zero; otherwise it will be one. It is similar for inequalities in (3.18)-(3.20). Combined with the above, the inequalities in (3.21)-(3.23) ensure that ϕ_{mln} will be zero if at least one (or equivalently the product) of x_m, x_l and y_{nm} is zero; otherwise it will be one.

Various commercial solvers, such as CPLEX, can be directly used to solve a MIP problem. These solvers apply branch-and-bound techniques and can be quite fast for small-scale problem instances. However, in some cases, edge systems can be of extremely large scale, e.g., in IoT architectures, and hence the above MIP problem becomes extremely large, hindering the performance of such branch-and-bound or other computational methods. In the next subsection, we propose a solution method that overcomes this dimensionality problem, and hence extends the range of the systems to which our work can apply.

Algorithm 1: General Approach to Solve ECP

```
z ← 0
for ∀n ∈ N_h ∪ {c} do
    z_n ← 1
    z^n ← z,
    x^n ← Place(z^n), y^n ← Assign(x^n, z^n)
    J^n ← J(x^n, y^n, z^n)
    z_n ← 0
end
n^o ← arg min_n J^n
Output: x^{n^o}, y^{n^o}, z^{n^o}
```

3.4.2 Large-Scale Approximate Solution

We need an algorithm that solves a large-scale ECP instance in a short time, so that the placement strategy can be quickly recalculated periodically or when necessary, e.g., when the network topology changes because nodes join or leave. In this subsection, we present such algorithms for different variants of the problem.

Leader-based case. In some controllers like OpenDaylight, the inter-controller synchronization fully follows leader-based strategy, i.e., $w_{ml}^{con} = 0$ and $w_{ml}^{dep} = 0$. In this case, $W_{s1}(\boldsymbol{x}, \boldsymbol{y})$ can be ignored. Here we describe the steps to solve this problem in Algorithm 1. We note that there exist $N_h + 1$ possible locations to place the leader; the cloud or the edge nodes with sufficient resources to play that role. Let us assume for a moment that we could find a way to solve the subproblem of controller placement and assignment $(\boldsymbol{x}, \boldsymbol{y})$ for a given leader selection $(\boldsymbol{z})$. In that case, we could simply iterate the value of leader for all the $N_h + 1$ possible choices, solve the subproblem for each choice, and pick the solution with the lowest balanced cost. In the rest of this section, we will show how to solve the above subproblem.

Consider the subproblem in which the leader is fixed at node $e \in \mathcal{N}_h \cup \{c\}$. Then, the synchronization overhead cost depends only on the controller placement decisions $(\boldsymbol{x})$:

$$W_{s2}(\boldsymbol{x}, \boldsymbol{z}) = W_{s2}(\boldsymbol{x}) = \sum_{m \in \mathcal{N} \cup \{c\}} x_m \big(w_{me}^{lb\_con} + w_{me}^{lb\_dep} * N \big) \tag{3.26}$$

while we recall that the delay and assignment overhead costs depend only on the assignment decisions $(\boldsymbol{y})$. By defining $f_m = w_{me}^{lb\_con} + w_{me}^{lb\_dep} * N$ and $c_{nm} = \gamma d_{nm} + w_{nm}^{a}$ the

Bound	Main technique	Reference
3	primal-dual	[67]
3	local search	[10]
1.488	LP rounding	[94]

Table 3.1: Approximation ratios for incapacitated facility location problem.

balanced objective can be written as:

$$\sum_{m\in\mathcal{N}_h\cup\{c\}} x_m f_m + \sum_{n\in\mathcal{N}} \sum_{m\in\mathcal{N}_h\cup\{c\}} y_{nm} c_{nm} \ . \tag{3.27}$$

We notice that this is the standard form of the well-studied *uncapacitated facility location problem*. The cost of locating a facility at node m is given by f_m, while the cost of assigning a client at node n to a facility at node m is given by c_{nm}. Various algorithms have been developed to solve this problem in polynomial time with provable approximation ratios, as we summarized in Table 3.1. These include LP rounding [158], primal-dual [67] and local-search [10] methods. So far, the best performance is the 1.488-approximation proposed in [94].

Leaderless case and hybrid case. Next, we turn to a more general case, where any term in the original problem is not necessary to be zero. Similarly, we follow the steps of Algorithm 1, and consider a different approach to solve the subproblem of placement and assignment. We begin by showing that for a given controller placement $\boldsymbol{x}$, the optimal assignment policy $\boldsymbol{y}$ can be easily found. Specifically, we show the following lemma (proved in the Appendix A.2).

Lemma 1. *For a given controller placement $\boldsymbol{x}$, the optimal assignment policy can be described by:*

$$y_{nm} = \begin{cases} 1, \ \textit{if}\ m = \underset{m':x_{m'}=1}{\arg\min} \left[\ \gamma d_{nm'} + w^a_{nm'} + \sum_{l:x_l=1} w^{dep}_{m'l}\ \right] \\ 0, \ \textit{otherwise} \end{cases} \tag{3.28}$$

for each $n \in \mathcal{N},\ m \in \mathcal{N}\cup\{c\}$.

The above lemma indicates that the complexity of the subproblem primarily lies on the optimization of the controller placement policy. Following this intuition, we introduce the *element* X_n to denote the placement of a controller at node n which is equivalent of deciding $x_n = 1$. The set of all possible elements, also called the *ground set*, can be

Bound	Main technique	Reference
$\frac{1}{4}$	uniformly random	[45]
$\frac{1}{3}$	local search (deterministic version)	[45]
$\frac{2}{5}$	local search (randomized version)	[45]
0.41	simulated annealing	[53]
0.42	structural continuous greedy	[46]
$\frac{1}{3}$	greedy (deterministic version)	[29]
$\frac{1}{2}$	greedy (randomized version)	[29]

Table 3.2: Approximation ratios for non-negative submodular function maximization.

defined as follows:

$$G = (X_n : n \in \mathcal{N}_h) \ . \tag{3.29}$$

A subset of elements $X \subseteq G$ corresponds to a controller placement policy $\boldsymbol{x}$, such that $x_n = 1$ if and only if $X_n \in X$. Besides, let $\boldsymbol{x}_X$ be the binary representation of the set of elements X. Then, the objective function B can be expressed as a set function $f : 2^G \to \mathbb{R}$:

$$f(X) = J(\boldsymbol{x}_X, y(\boldsymbol{x}_X)) \tag{3.30}$$

where $y(\boldsymbol{x}_X)$ denotes the optimal assignment policy given the controller placement policy $\boldsymbol{x}_X$ based on equation (3.28).

Next, we consider a well-studied class of set functions called *supermodular* [64].

Definition 1. *Let G be a finite set of elements (ground set). A set function $f : 2^G \to \mathbb{R}$ is called supermodular if for all subsets $A, B \subseteq G$ with $A \subseteq B$ and every element $i \in G \setminus B$ it holds that:*

$$f(A \cup \{i\}) - f(A) \leq f(B \cup \{i\}) - f(B) \tag{3.31}$$

The above definition indicates that the marginal value for adding an element i in a set increases as the respective set expands. We will show that, under certain conditions on the cost values, our objective $f(X)$ can be expressed as a supermodular function. Specifically, we have the following lemma (proved in the Appendix A.3).

Lemma 2. *The set function $f(X)$ defined in (3.30) is supermodular for the case of uniform costs $w_{ml}^{dep} = w_{m'l'}^{dep} = w^{dep}$, $\forall m, l, m', l' \in \mathcal{N}_h \cup \{c\}$.*

Based on Lemma 2, the subproblem can be casted as the *minimization of a supermodular function f*. This type of problems are usually addressed by considering their equivalent

submodular function maximization version. That is, minimizing the supermodular function f (i.e., delay and overhead costs) is equivalent to maximizing the submodular function $\widehat{f}(X) = f^{ub} - f(X)$ (respective delay and overhead cost savings). Here, the constant f^{ub} indicates an upper bound to the highest possible value of $f(X)$.

Given that $\widehat{f}(X)$ is non-negative, there exist various approximation algorithms to maximize it (Table 3.2). Here, an approximation bound β means that the ratio of the value of the approximate solution over the optimal solution value is always at least β, namely $\widehat{f}^{apx}/\widehat{f}^{opt} \geq \beta$. Therefore, we obtain the following theorem.

Theorem 1. *There exists a solution to the subproblem such that* $\widehat{f}^{apx}/\widehat{f}^{opt} \geq \beta$*, where* $\beta \in \{\frac{1}{4}, \frac{1}{3}, \frac{2}{5}, 0.41, 0.42, \frac{1}{3}, \frac{1}{2}\}$.

The Algorithm in the last row of Table 3.2 has the best approximation bound. As it is summarized in Algorithm 2, this algorithm proceeds in N_h iterations which correspond to some arbitrary order $r_1, \ldots r_{N_h}$ of the ground set G. At each iteration, two solutions A and B are maintained, initially set to $\emptyset$ and G respectively. At the nth iteration, the algorithm either adds r_n to A or removes r_n from B. This decision is done randomly and greedily based on the marginal gain of each of the two options. After N_h iterations both solutions coincide, i.e., $A = B$. This is the output of the algorithm.

With the solution to this subproblem with a fixed leader, we follow the same steps in Algorithm 1 as in the leader-based case, to have exhaustive search on all possible leader choices. Finally we get a solution to ECP problem with approximation bounds listed in Theorem 1.

We emphasize that although the approximation bounds of Theorem 1 are shown for the case that w^{dep} values are identical, we make no assumptions on the values of d and w^{con} vectors. Moreover, in the next section we will show that in practice Algorithm 2 achieves a near-optimal solution even for heterogeneous w^{dep} values.

3.5 Evaluation Results

In this section, we evaluate the performance of the proposed algorithms using two real wireless network topologies. Overall, we find that our algorithms achieve excellent performance and multi-fold gains over state-of-the-art methods, especially when the priority is given on the delay rather than the overhead optimization (large γ value). Such tendency exists regardless of the synchronization strategy of controllers. We repeat the same evaluations on both OpenDaylight (Figure 3.6) and ONOS (Figure 3.7), in order to show the

Algorithm 2: Randomized Greedy Algorithm

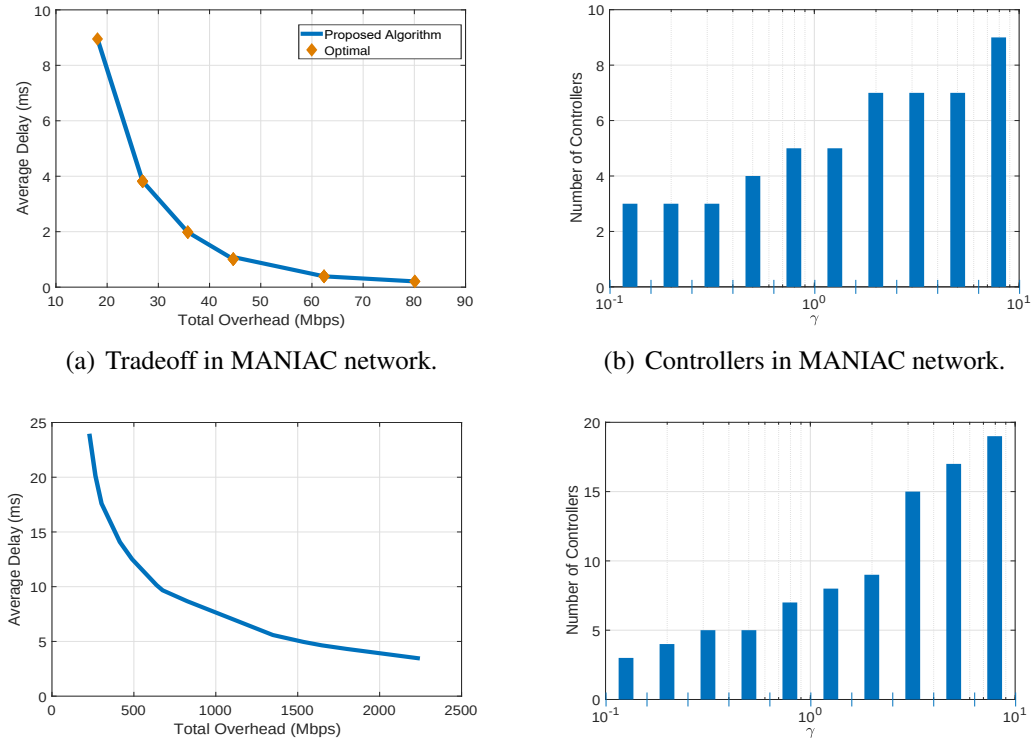
```
A ← ∅, B ← G
for n = 1 to N_h do
    ΔA ← f(A) − f(A ∪ {r_n})
    ΔB ← f(B) − f(B \ {r_n})
    ΔA ← max(ΔA, 0), ΔB ← max(ΔB, 0)
      with probability ΔA/(ΔA + ΔB)* do:
    A ← A ∪ {r_n}
    else (with probability ΔB/(ΔA + ΔB)) do:
    B ← B \ {r_n}
end
Output: A (or equivalently B)
```

* If $\Delta A = \Delta B = 0$, then $\Delta A/(\Delta A + \Delta B) = 1$.

results in a comparative way. We also find that the optimization of delay and overheads indirectly favors two reliability objectives, namely controller-node and inter-controller path loss. Therefore, by balancing delay and overheads, we can also balance the above two reliability metrics.

Evaluation Setup. Throughout the evaluation, we use the MANIAC mobile ad hoc network in [61] and the Barcelona wireless mesh network in [118]. MANIAC contains only 14 nodes, which allows us to execute MIP algorithm and find the optimal solution in reasonable time. On the other hand, Barcelona contains 60 nodes, the evaluation of which verifies the scalability of the algorithms and enriches the results. In each topology, we augment an extra node representing the cloud controller, connected to all other nodes with a large distance. We set this distance value as the half of the graph's diameter. We define delay and overhead costs based on our measurements in previous sections. The overhead cost is the product of the measured traffic volumes in Figure 3.3 and the network distances between the nodes. The specific values depend on the type of controllers. The delay cost d_{nm} is the aggregate delay of the links of the respective shortest path. We set the delay of each link randomly with average value $12.23\ ms$, in accordance with our experiment result in Figure 3.2(b).

Evaluations on OpenDaylight. First, we choose OpenDaylight as the controller, which has totally leader-based synchronization strategy, and the facility location problem based algorithm is available. In accordance with measurement of Section 3.2, we set $w^{a}_{ml} = 0.019 \cdot hops_{ml}$, $w^{lb\_con}_{ml} = 0.207 \cdot hops_{ml}$ and $w^{lb\_dep}_{ml} = 0.62 \cdot hops_{ml}$, where $hops$ indicates the number of hops of the shortest path between the respective nodes. The unit



(a) Tradeoff in MANIAC network.

(b) Controllers in MANIAC network.

(c) Tradeoff in Barcelona network.

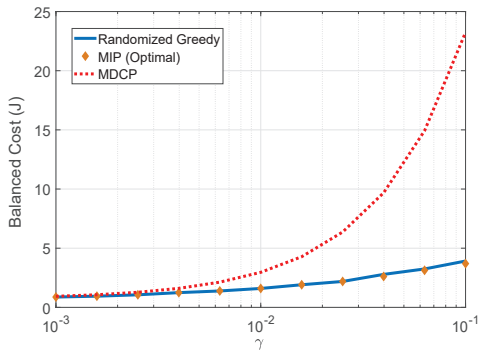
(d) Controllers in Barcelona network.

Figure 3.6: Evaluation results for the extended model, with parameters extracted from the measurements on OpenDaylight.

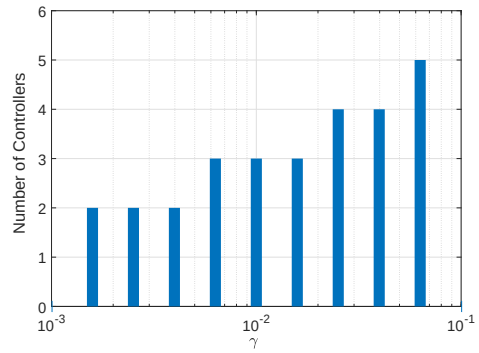
here is $Mbps$.

Figures 3.6(a)-3.6(b) depict the delay-overhead cost trade-off and the number of controllers placed in MANIAC network. In spite of different weights, our algorithm always has a close to optimal performance. The algorithm is capable to balance the delay and overhead cost as well as to place a proper number of controllers. Figures 3.6(c)-3.6(d) depict the results for the Barcelona network, demonstrating the scalability of the algorithm.

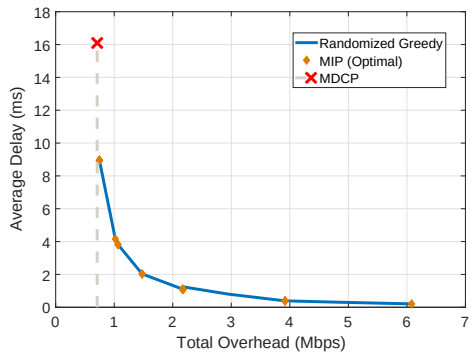
Evaluations on ONOS. Next, we replace the controllers with ONOS. According to our analysis in Section 3.2 and the Appendix, ONOS's synchronization strategy can be regarded as a leaderless one. Therefore we verify both our small-scale MIP method and the large-scale randomized greedy algorithm. We set $w^a_{ml} = 0.019 \cdot hops_{ml}$, $w^{con}_{ml} = 0.04579 \cdot hops_{ml}$ and $w^{dep}_{ml} = 0.00793 \cdot hops_{ml}$ Most of the state-of-the-art methods require that the number of placed controllers is fixed and known in advance (e.g., see the review of related work in Section 3.6). Therefore, it would not be fair to compare the above with our methods which optimize both the number and location of controllers. Interestingly, there is a method in the literature that explores the same solution space with our work. Namely,



(a) Balanced cost for different weight γ.



(b) Number of controllers for different weight γ.

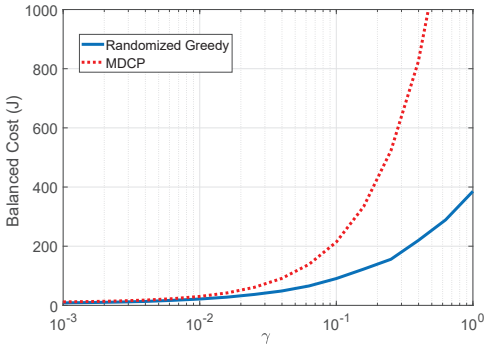


(c) Delay-overhead tradeoff.

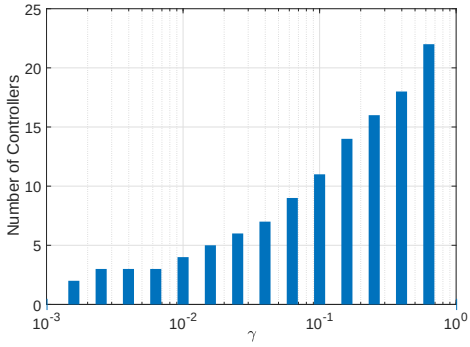
Figure 3.7: Evaluations on MANIAC network [61]. (a) Balanced cost, (b) number of placed controllers and (c) tradeoff between delay and overhead for two different algorithms.

the MDCP method (Algorithm 3 in [165]) is a greedy procedure that places controllers based on the degrees of the nodes and the inter-controller traffic. The latter is estimated by the diameter of the topology graph, rather than based on actual measurements. In next subsections, we will show a detailed comparison among randomized greedy algorithm, MIP and MDCP.

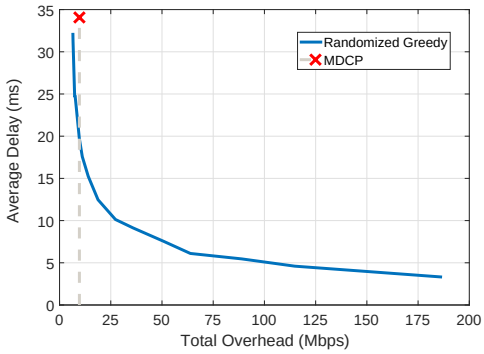
Impact of weight γ. Figure 3.7(a) depicts the evaluation results for the MANIAC network. Running on a common CPU machine, MIP typically takes 30 seconds to return the optimal solution using the built-in optimizer of MATLAB. On the other hand, Randomized Greedy algorithm takes much shorter time. Within 0.1 second, we are able to repeat the algorithm for 200 times and pick the best result. Overall, *Randomized Greedy has a performance very close to the optimal for all values of* γ, while the performance of MDCP drops when γ is large (i.e. more weights are put on the delay rather than overhead cost). In these cases, *our algorithm has up to 3 quarters lower balanced cost than MDCP*.



(a) Balanced cost for different weight γ.



(b) Number of controllers for different weight γ.

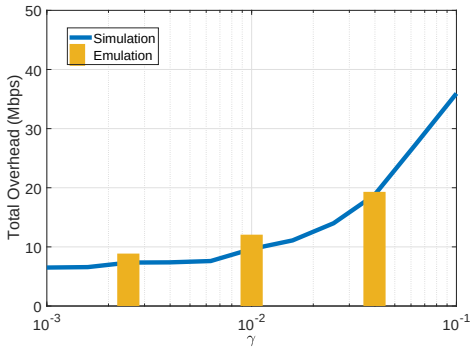


(c) Delay-overhead tradeoff.

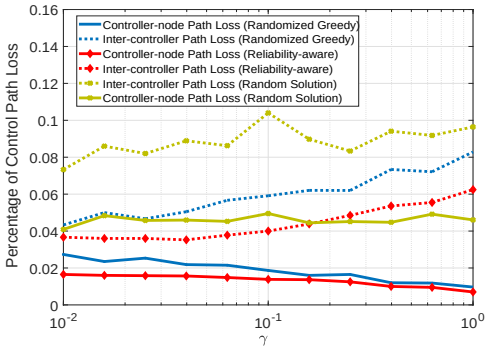
Figure 3.8: Evaluations on Barcelona network [118]. (a) Balanced cost, (b) number of placed controllers and (c) tradeoff between delay and overhead for two different algorithms.

We need to emphasize that many of the previous controller placement works used clustering methods such as k-median [60]. An issue is that these methods do not optimize the number of placed controllers, but take it as an input. In contrast, Randomized Greedy algorithm can adjust the number of placed controllers based on the weight (Figure 3.7(b)). When overhead cost is preferred, the algorithm tends to deploy fewer controllers so that the inter-controller communications reduce. When delay cost is preferred, however, it places more controllers so that the data plane nodes can reach a controller with smaller delay.

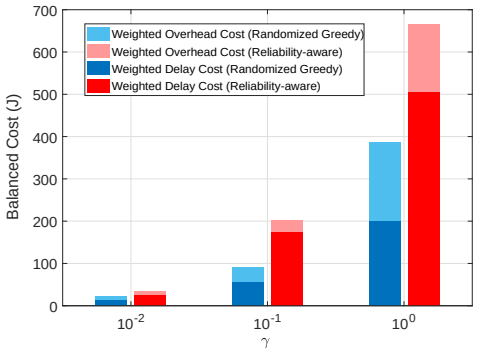
By iterating a large range of weight values, in Figure 3.7(c), we depict the *trade-off curve between the average delay and the total traffic overhead*, in order to show more details of the performance. The usage of measured data of overhead and delay empowers Randomized Greedy algorithm with two improvements over MDCP, which makes decisions based on some topology metrics only, such as network diameter and nodes degree.



(a) Overhead measured by Mininet emulation.



(b) Control path loss for different weight γ.



(c) Balanced cost for different weight γ.

Figure 3.9: (a) Verification on the overhead values by Mininet emulations. (b), (c) Performance comparison with a state-of-the-art reliability-aware algorithm [63] on the Barcelona network.

First, with similar total overhead, the average delay achieved by Randomized Greedy is lower than MDCP. Besides, Randomized Greedy algorithm is able to further decrease the delay by tolerating some additional overhead, and vice versa.

Figure 3.8(a)-3.8(c) repeat the evaluations for the Barcelona network. While MIP cannot run in reasonable time for this larger network, Randomized Greedy algorithm is scalable. It takes 0.8 second in average to finish the algorithm for 200 times. The qualitative results are similar with the MANIAC network. When γ is relatively large, the gains of our algorithm over MDCP are more pronounced. For the same total overhead, the Randomized Greedy algorithm achieves smaller average delay than MDCP.

Emulations vs simulations. To verify the accuracy of our model and evaluation results, we use Mininet to run emulations on the same topology we used for the MATLAB simulations. Mininet allows us to take measurements on the real traffic overhead in the same way as we did in Section II. In Figure 3.9(a), a consistency is shown in both the

value and tendency of overheads between the simulations and emulations. This implies that our model successfully captures the pattern of the control traffic and it is capable for providing guidance in realistic cases.

Interplay between different objectives. Though concentrating on delay and overhead costs, our algorithm will have an impact on other metrics as well. One important metric is the reliability of control paths. Compared with wired networks like data centers, one significant difference in wireless edge networks is that links are generally less reliable. The operator must consider failure scenarios where a portion of links cease working. The work in [63] defines *control path loss*, which is the expected number of disconnected control paths (including both controller-node and inter-controller connections) given the probability of network link failure. This work proposes a simulated annealing algorithm achieving very good reliability. In order to investigate the reliability of our solution, we use this algorithm for comparison. To simplify, we assume every link in the Barcelona network may fail independently with the same probability, e.g., 0.01. The connections to the cloud are usually in another channel and with longer distances, therefore it is reasonable to assume the links have a lower quality. Here we assume they fail with doubled probability. Since the algorithm in [63] is not able to determine the amount of placed controllers, we set the same value as our algorithm.

Figure 3.9(b) depicts the percentage of control path loss. Compared with the reference (Reliability-aware) algorithm, our algorithm achieves a close performance in the controller-node path loss. Although we have larger inter-controller path loss when γ is large, by calculating the performance of random placement solutions, it shows that our algorithm still gets a significantly lower inter-controller path loss than average. Moreover, in practice the Reliability-aware algorithm may not succeed in finding the optimal amount of placed controllers, which may incur a deteriorated performance. Importantly, our algorithm has the ability to balance the two reliability metrics. When γ is large, it tends to decrease the distance between controllers and data plane nodes (so as to reduce delay). As a result, controller-node connections acquire higher reliability. In a similar way, when γ is small, inter-controller connections acquire higher reliability. Meanwhile, Figure 3.9(c) shows that our algorithm achieves significantly lower delay and overhead cost than the reliability-aware reference algorithm.

3.6 Related Works

In this section we present the related work on the SDN controller placement problem. **Delay-centric approaches:** The delay between controllers and data plane nodes is especially critical for quickly exchanging messages required by the SDN protocol. Most of the related works targeted wide area networks that extend over a large geographical distance with large capacity links. In this context, Heller et al. [60] recognized that the problem of placing a given number k of controllers to minimize the average and worst-case delay can be modeled as a k-median and a k-center problem, respectively. Yao et al. [184] explored the impact of controller capacity limitations on the worst-case delay, and proposed an algorithm inspired by the capacitated k-center problem. Zhang et al. [188] experimentally showed that the interaction between the controllers can affect the reaction time perceived by the data plane nodes. Motivated by this, an evolutionary algorithm that finds a Pareto optimal solution with respect to the average controller-node and inter-controller delay metrics was proposed. In the context of a wireless network, Abdel-Rahman et al. [4] modeled the controller-node delay probability assuming the TDMA strategy, and formulated an integer program for minimizing the number of placed controllers. Sudheera et al. [166] augmented to the above objective function components which represent controller-node delay and controller load imbalance costs in a vehicular ad hoc network, and formulated the controller placement problem as a quadratic integer program.

Reliability-centric approaches: Since disruptions in the network can isolate data plane nodes from controllers, it is of great importance to improve the reliability of SDN. Previous works defined various reliability metrics. Hu et al. [63] focused on the minimization of the expected percentage of control path loss which was solved using greedy and simulated annealing based heuristics. The tradeoff between reliability and delay was also explored through simulations. Jimenez et al. [70] proposed an algorithm of different flavor, named k-critical, which creates a robust topology with the minimum number of controllers to deal with network failures and balance the load among the controllers. Guo et al. [56] performed a cascading failure analysis and applied a clustering algorithm to minimize the number of nodes survived at the steady stage. Müller et al. [111] proposed heuristic algorithms to maximize the number of node-disjoint paths between controllers and data plane nodes (path diversity). Lange at al. [85] investigated the delay, load imbalance and number of controller-less nodes (i.e., nodes isolated from controllers) caused by network failures. A simulated annealing algorithm that returns a pareto optimal solution was presented.

Cost-centric approaches: Another aspect of the problem lies on the costs associated to the deployment and operation of controllers. These include the expenses to buy the controllers, if specialized equipment is required, manually install them and connect them in the network. Sallahi et al. [152] proposed an optimal model aiming to calculate the optimal number, location, and type of controllers that minimizes the overall cost. A limitation of this approach is that it requires the solution to an integer program which does not scale well for large networks. Hu et al. [62] studied the problem of minimizing the energy cost associated to the communication between the controllers and data plane nodes and proposed a genetic heuristic algorithm. However, the energy cost associated to the inter-controller communication was not considered. Another operational cost component, which is central to our work, is the bandwidth overhead of inter-controller and controller-node communication. Su et al. [165] and Ksentini et al. [83] considered the minimization of the above two types of overheads and proposed a facility-location inspired heuristic and a bargaining game based pareto optimal solution, respectively. While these are perhaps the closest to our work, they were based on a simpler model for overheads and did not balance overheads with delay. In contrast, our model is driven by empirical measurements on delay and overheads from a multi-controller edge system we developed.

Adaptive approaches: For completeness, we should stress that dynamic algorithms which periodically adapt the controller placement solution were proposed in [14], [173] and [144]. In this way, the required controllers can be dynamically added or deleted and data plane nodes can be reassigned to different controllers based on traffic dynamics. The above algorithms are optimized to minimize the number of reassignments, the number of added controllers and other CAPEX and OPEX costs associated to controllers.

3.7 Summary

In this chapter, we studied the SDN controller placement problem in edge network architectures. Our work combines strong experimentation results along with valid theoretical modeling and analysis. Namely, we built a testbed of a multi-controller edge system and described the sensitivity of delay on the controller placement as well as the magnitude and shape of traffic overheads. Guided by these findings, we presented a methodology that yields a set of optimal controller locations and assignment of nodes to controllers. Evaluation results demonstrated significant performance gains over state-of-the-art methods, and provided insights about the interplay between various performance and reliability objectives.

Chapter 4

Distributed Control Plane: Controller Synchronization Problem

As discussed in the last chapter, the physical distribution of the control plane is a proper way to address scalability and reliability challenges of the centralized design of SDN. However, having multiple controllers managing the network while maintaining a "logically-centralized" network view brings additional challenges. In spite of the controller placement problem, another challenge is how to coordinate the management decisions made by the controllers which is usually achieved by disseminating synchronization messages in a peer-to-peer manner. While there exist many architectures and protocols to ensure synchronized network views and drive coordination among controllers, there is no systematic methodology for deciding the optimal frequency (or rate) of message dissemination. In this chapter, we fill this gap by introducing the SDN synchronization problem: how often to synchronize the network views for each controller pair. Our objective is to maximize the performance of applications of interest which may be affected by the synchronization rate. Using techniques from learning theory, we derive algorithms with provable performance guarantees. Evaluation results demonstrate significant benefits over baseline schemes that synchronize all controller pairs at equal rate.

4.1 Introduction

As shown in the last chapter, in order to keep the flexibility of the SDN control while enhancing the scalability of the system, multiple controller instances can be deployed at different locations in the network. The controllers may be physically distributed across the network, but they should be "logically-centralized". This means that the controllers

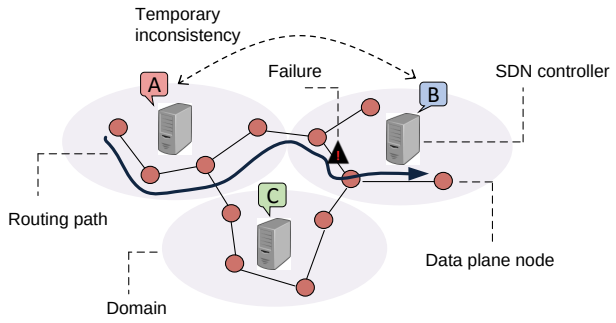


Figure 4.1: Impact of inconsistency among controllers on routing application performance.

should coordinate their decisions to ensure their collective behavior matches the behavior of a single controller.

The coordination among controllers is an active area of research with several protocols proposed thus far [123]. For example, OpenDaylight [105] and ONOS [16], two state-of-the-art controller implementations, rely on RAFT and Anti-entropy protocols for disseminating coordination messages among controllers. Typically, each controller is responsible for a part of the network only, commonly referred to as the controller's *domain*. The messages disseminated by a controller to the other controllers convey its view on the state of its domain (e.g., available links and installed flows). The composition of these messages allow the controllers to synchronize and agree on the state of the entire network.

While different coordination protocols may generate messages of different types and at different timescales, there exist two broad protocol categories [126], [26]. The first category contains the *strongly consistent* protocols which strive to maintain all the controllers synchronized in all times. This is ensured by disseminating messages each time a network change (e.g., a node or link failure) happens followed by a consensus procedure. The second category contains the *eventually consistent* protocols which omit the consensus procedure, yet converge to a common state in a timely manner usually through periodic message dissemination.

Despite its benefits, strong consistency is difficult to ensure in practice as it is challenged by the unreliable nature of network communications. In addition, this approach generates significant *overheads* for message dissemination among controllers which may be prohibitively large especially when applied to wireless networks with in-band control channels of limited capacity [113], [150]. On the other hand, eventual consistency, where controllers are permitted to temporarily have inconsistent views of each other's state, better suits the needs of the above networks, and, thus, can be used to extend the applicability

of distributed controller solutions. Yet, the inconsistent views of controller states can harm the *performance of network applications*.

To illustrate the impact of inconsistency, we consider the toy example with three controllers (A, B and C) and their respective domains in Figure 4.1. Each pair of controllers synchronize periodically, e.g., every few seconds. At some time, controller A receives a request for routing a flow to a destination node inside the domain of B. Controller A will respond by computing and setting up a routing path based on its current view on the state (topology, traffic loads) of its domain and the other domains. However, controller A is not aware if the links on the routing path outside of its domain are still available or have failed (e.g., a failed link in domain B in Figure 4.1) since the last synchronization period. If failures happened, the packets of the flow will have to wait until the next synchronization period, although there is an alternative directly available routing path through the domain of C. Similar problems, if not more serious, can be identified for more advanced traffic engineering applications where inconsistency hinders the effective load balancing and distribution across multiple paths.

The eventually consistent model raises new technical challenges. In particular, it is important to decide *how often* (at what period or rate) to synchronize each pair of controllers in a given network. One might expect that the straightforward policy where all controller pairs synchronize at the same rate would work well. However, some may argue that the synchronization rate should be higher for domains that are more dynamic (with many changes in topology and flow configurations) in order to preserve consistency of the rest domains.

The issue is further complicated by the requirements of the network applications. Previous works [92], [127] showed that certain network applications, like load-balancers, can work around eventual consistency and still deliver acceptable (although degraded) performance. In such cases, some additional effort needs to be made to ensure that conflicts such as forwarding loops, black holes and reachability violation are avoided [57]. Therefore, synchronization policies that completely neglect the specific applications of interest in the network as well as the impact of synchronization rate on their performance may end-up being highly inefficient.

The above questions remain open since, until now, the inter-controller traffic has been often neglected in SDN literature with most of the existing works focusing on the routing and balancing of the data traffic (e.g., see the survey in [82]).

Our goal in this work is to investigate policies for the synchronization among SDN controllers, and focus particularly on the impact of the rate of synchronization on the per-

formance of network applications. This is a complex problem since, in practice, we do not know the function that maps the synchronization rate to application performance. To obtain some quantitative insights on this function, we emulate the performance of two applications of interest, namely shortest path routing and load balancing, using a commercial platform (Mininet) [86] and SDN controller (RYU) [2]. While the results are quite unsteady, the average performance increases with the synchronization rate and saturates eventually showing that a diminishing return rule applies. Next, to overcome the unknown objective challenge, we use elements from the *learning theory*, and propose an algorithm that gradually trains the system and constructs a solution that is with high confidence close to the optimal. The contributions of this work can be summarized as follows:

- We introduce the problem of finding the optimal synchronization rates among SDN controllers in a network with the objective of maximizing the performance of applications. To the best of our knowledge, this is the first work that studies this problem.
- We emulate the performance of two popular applications and obtain insights about the impact of synchronization rates. We use these results to derive an algorithm that gradually trains the system in order to learn the optimal policy.
- We perform evaluations to show the efficiency of our proposed algorithm. We find that benefits are realized compared with the baseline policy that synchronizes all controller pairs at an equal rate.

The rest of the chapter is organized as follows. In Section 4.2, we present our emulation results, based on which we model the problem in Section 4.3 and propose our learning algorithm for maximizing the network application performance in Section 4.4. Section 4.5 presents the evaluation of our proposed algorithm, while Section 4.6 reviews our contribution compared to related works. We conclude our work in Section 4.7.

4.2 Emulation Studies on the Impact of Synchronization Policies

In this section, we provide an emulation study that will illustrate the impact of synchronization rate on the performance of some popular network applications, namely shortest path routing and load balancing.

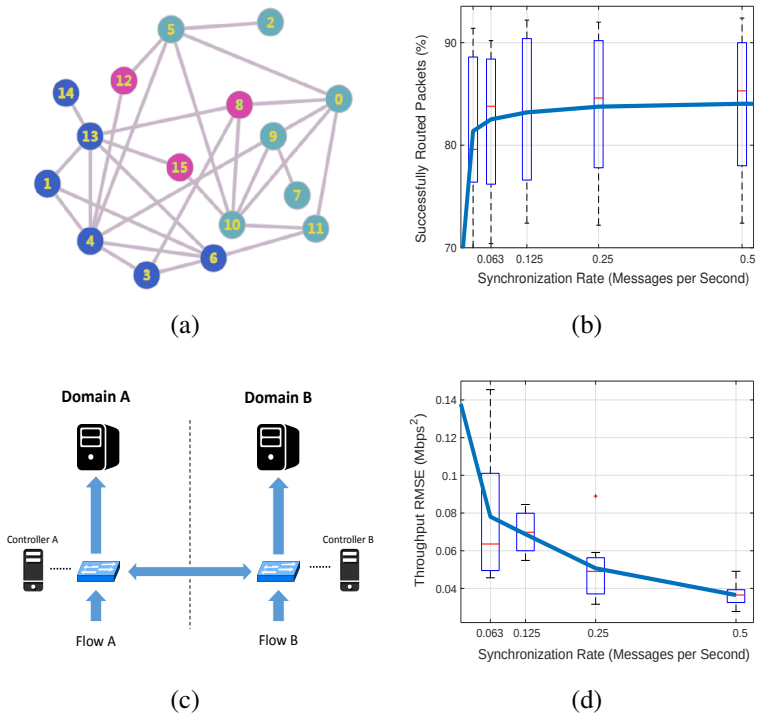


Figure 4.2: Emulation results. Topology and impact of synchronization rate on the performance (box plots and average values) of (a)(b) shortest path routing and (c)(d) load balancing applications.

Emulation setup. We use *Mininet* [86] to emulate virtual networks with several nodes and SDN controllers running on the same CPU machine. Among the set of commercial controllers that are available online we pick *RYU* [2] which is open-source and allows us to develop our own protocols for the synchronization among controllers. Specifically, we implement a simple eventually-consistent protocol which periodically disseminates synchronization messages between each controller pair. Our code is parameterized to allow for any synchronization period. The disseminated messages convey the local views of controllers about the topology and installed flow tables. This information is made available to the controllers by the OpenFlow protocol.

Shortest Path Routing. We first test the performance of a shortest path routing application. With this application, packets are routed to their destination following the path of minimum hop count, calculated by Dijkstra's algorithm. We generate the random network of 16 nodes and 3 controllers, depicted in Figure 4.2(a), where links fail or recover randomly and independently every one second with probability 0.05, and nodes with the same color are managed by the same controller. We further generate data packets with random source-destination nodes. Unless the controllers synchronize at the time of packet

generation, the packet is at risk of following a failed routing path.

The performance of routing application is determined by the number of packets that are successfully routed (without traversing any failed link) to their destinations. We emulate the performance for five different scenarios where all the controller pairs synchronize at the same rate equal to (i) 0.5, (ii) 0.25, (iii) 0.125, (iv) 0.063 and (v) 0.031 (messages per second). This translates to a single message disseminated every 2, 4, 8, 16 or 32 seconds. For each scenario, emulations are run for multiple times and the results are depicted in Figure 4.2(b). Despite a large extent of randomness, we observe that the average performance (calculated over 20 minutes) increases with the synchronization rate and saturates eventually showing that a *diminishing return rule* applies.

Load Balancing. We perform additional emulations to test the performance of a load balancing application. We consider a similar setup with the work in [92], depicted in Figure 4.2(c). That is, we generate a network with two controllers. Each controller manages two nodes, a switch and a server. The switches generate flows uniformly at random. The flows can be routed and queued to any of the two servers. Each controller is aware of the load of the server it manages. It also receives periodic synchronization messages about the load of the other server by the other controller. Each time a new flow is generated, the responsible controller routes it to the server with the currently observed lowest load. However, this may not be the least loaded server in reality, since the controllers are not synchronized at all times.

The emulation results are depicted in Figure 4.2(d). The metric we consider is the root-mean-square deviation (RMSE) of two servers' throughputs. The better the two server loads balance, the lower the value of this metric becomes. Therefore, this metric captures the performance of a load balancing application. For convenience, we claim it the cost function, and denote the performance metric the opposite value of cost function. Then, coinciding with the routing application, we observe that the performance improves with the synchronization rate but gradually saturates showing that a diminishing return rule applies.

4.3 Problem Modeling

Subsequently, we study the objective of maximizing the performance of a network application such as the applications emulated in the previous section. While the objective function is expected to have a curve shape similar to those reported in Figure 4.2, we cannot express in closed-form how exactly the synchronization rates will affect application

performance. Therefore, the objective function is *unknown*.

To overcome the unknown objective challenge, we propose to leverage methods from the *learning theory*. Such methods typically *train* the system by trying-out a sequence of solutions (synchronization rates) over some training period $\mathcal{T} = \{1, 2, \ldots, T\}$ of T time slots, until they can infer a "sufficiently good" solution. To describe such training process, we introduce the vector of synchronization rate variables:

$$\boldsymbol{x} = (x_{ij}^t \in \{0, 1, \ldots, R\} \ : \ \forall i, j \in \mathcal{C}, j \neq i, t \in \mathcal{T}) \tag{4.1}$$

where x_{ij}^t indicates the synchronization rate between controllers i and j tried-out in time slot t. R denotes the maximum possible synchronization rate. We further denote by the vector $\boldsymbol{x}^t$ all the variables in time slot t. We emphasize that the variable values will be typically different from slot to slot as different synchronization rates need to be explored in order to train the system.

Given the synchronization rate vector $\boldsymbol{x}^t$ tried-out in a slot t, the application performance will be $\Psi_t(\boldsymbol{x}^t)$. Here, $\Psi_t(.)$ is an unknown function that governs the application performance in slot t. While the overall function is unknown, the single value $\Psi_t(\boldsymbol{x}^t)$ can be *observed* by the system operator *after* the synchronization rate decision $\boldsymbol{x}^t$ is made, in the end of the slot. For a shortest path routing application, for example, this is possible by measuring the number of data packets that reached their destination in time. Such information is available to the controllers through the TCP acknowledgement packets. The information can be then passed to the system operator (e.g., one of the controllers) which can simply aggregate and sum the respective values.

We emphasize that the function $\Psi_t(.)$ is time slot-dependent, meaning that the performance value might change with time even for the same synchronization rate decision. That is, we may try-out the same synchronization rate vector $\boldsymbol{x}^t = \boldsymbol{x}^{t'}$ in two slots t and t' but observe different performance values $\Psi_t(\boldsymbol{x}^t) \neq \Psi_{t'}(\boldsymbol{x}^{t'})$. Such *uncertainty of observations* is due to the stochastic nature of the network. Intuitively, the performance value will be large if the network happens to be stable in a slot but will be much worse in other slots during which many changes happen.

Despite the uncertainty of observations, the learning method should be able to infer by the end of the training period $\mathcal{T}$ a "sufficiently good" synchronization rate decision $\widehat{\boldsymbol{x}} = (\widehat{x}_{ij} \ : \ i, j \in \mathcal{C}, j \neq i)$. This should, ideally, maximize the *average* performance denoted by an (also unknown) function $\widehat{\Psi}(.) = \mathbb{E}[\Psi_t(.)]$. While the system operator does not know the average performance values, we assume that they do not change over a period

of time (e.g., a few hours). Therefore, the problem can be formulated as:

$$\max_{\widehat{\boldsymbol{x}}} \quad \widehat{\Psi}(\widehat{\boldsymbol{x}}) \tag{4.2}$$

$$s.t. \quad \sum_{i\in\mathcal{C}} \sum_{j\in\mathcal{C}, j\neq i} \widehat{x}_{ij} b_{ij} \leq B \tag{4.3}$$

where B is a constant and inequality (4.3) ensures that the inferred synchronization rate decision will satisfy a resource constraint.

We need to emphasize that the average performance $\widehat{\Psi}(\widehat{\boldsymbol{x}})$ is not the only criterion that determines the efficiency of a learning method. Another important criterion in this context is the *running (or training) time* T, i.e., how many time slots are required for training in order to infer the synchronization rate decision $\widehat{\boldsymbol{x}}$. In the next section, we will propose a learning method that has adjustable average performance and running time.

4.4 Optimization Algorithms

To handle the uncertainty of an observed performance value $\Psi_t(\boldsymbol{x}^t)$, a learning method would typically try-out the *same* synchronization decision $\boldsymbol{x}^t$ *multiple times*, in different time slots. Then, the empirical mean of the observations will be used to estimate the average performance value $\widehat{\Psi}(\boldsymbol{x}^t)$. By repeating the above training process for every possible synchronization decision, an estimate of the entire objective function $\widehat{\Psi}(.)$ can be obtained. However, there exists an *exponential number of possible decisions*; $(R+1)^{C(C-1)}$ decisions in total. Therefore, this approach would require an exponential number of time slots for training, which is clearly not practical.

To overcome the high dimensionality of the synchronization decision space, we could leverage learning methods proposed recently that do not require the estimation of the objective function at every possible decision. For instance, the *ExpGreedy algorithm* proposed in [160] can infer a close-to-optimal decision in *polynomial-time* provided that the objective function follows a diminishing return rule, as the one observed in the emulation results in Figure 4.2. Still, however, the running time of this algorithm may be too large for our problem, as we show numerically in the next section, hindering its application in practical scenarios.

Based on the above, we propose an alternative more-practical learning algorithm for which we can flexibly adjust the running time by setting appropriate values to its input parameters. We refer to this algorithm as *Stochastic Greedy* and summarize it in Algorithm

3. To ease presentation, we have assumed that the resource costs are equal and normalized to one for all the controller pairs, i.e., $b_{ij} = 1 \; \forall i, j$. However, the algorithm and analysis can be easily extended for heterogeneous resource costs.

In a nutshell, the Stochastic Greedy algorithm starts with the all-zero synchronization decision and then gradually constructs the decision to be returned by iteratively increasing by 1 the synchronization rate of a single controller pair. This procedure will end when the B resource constraint is reached, i.e., after B iterations. Each iteration requires multiple time slots for training so as to be confident that the controller pair selected to increase its rate by 1 will improve the average performance more than other controller pairs. The length of the training period can be adjusted by two input parameters σ and τ. The value of σ is between 1 and $C(C-1)$, while τ can take any positive integer value.

Formally, the algorithm maintains a synchronization rate decision $\widehat{\boldsymbol{x}}$, initially set to the zero vector $\mathbf{0}$ (line 1). It spends the first τ time slots trying out the zero synchronization decision and uses the τ observations to estimate $\widehat{\Psi}(\mathbf{0})$ (lines 2-3). In the next B iterations (lines 4-12), the algorithm will iteratively select a controller pair and increase the respective synchronization rate by 1, updating $\widehat{\boldsymbol{x}}$. At each iteration $k = 1, 2, \ldots, B$, the algorithm will initially pick σ random pairs of controllers as candidates (line 5). For each such pair $p = 1, 2, \ldots, \sigma$, the synchronization decision $\widehat{\boldsymbol{x}}'$ will be set accordingly (line 7) and τ time slots will be spent to estimate $\widehat{\Psi}(\widehat{\boldsymbol{x}}')$ (lines 8-9). The marginal performance gain of switching from decision $\widehat{\boldsymbol{x}}$ to $\widehat{\boldsymbol{x}}'$, denoted by $D(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{x}}')$, will be estimated (line 10). Among the σ candidate controller pairs, the algorithm will include in the current decision $\widehat{\boldsymbol{x}}$ the pair with the maximum estimated marginal performance gain (line 12).

The algorithm spends τ time slots to estimate $\widehat{\Psi}(\widehat{\boldsymbol{x}})$ for $\widehat{\boldsymbol{x}} = \mathbf{0}$, and $\sigma\tau$ more slots for each iteration. Therefore, the total running (or training) time is $T = \tau + \sigma\tau B$ time slots. The following theorem describes the average performance of the algorithm. Since the algorithm makes random decisions, the average performance bound holds in expectation.

Theorem 2. *Algorithm 3 achieves average performance $\widehat{\Psi}(\widehat{\boldsymbol{x}})$ that is in expectation a factor $1 - e^{-(1-\epsilon)\mu}$ from the optimal where $\epsilon = e^{-\sigma \frac{B}{C(C-1)R}}$ and μ is the expected fraction of the observed marginal gain in a slot over the actual marginal gain.*

We defer the proof of the theorem to the Appendix B. We emphasize that the average performance bound depends on the value of μ. This value captures the uncertainty of the observed performance values since the changes in network state may be unevenly distributed across the time slots. If $\mu = 1$, it means that the performance value does not depend on the time slot of observation and hence the estimated maximum performance

Algorithm 3: Stochastic Greedy with (σ, τ) input

Initialize $\widehat{\boldsymbol{x}} = \mathbf{0}$;
Try out $\boldsymbol{x}^t = \widehat{\boldsymbol{x}}$ and observe $\Psi_t(\boldsymbol{x}^t)$ $\forall t \in \{1, \ldots, \tau\}$;
Estimate $\widehat{\Psi}(\widehat{\boldsymbol{x}}) = \frac{1}{\tau}\sum_{t=1}^{\tau} \Psi_t(\boldsymbol{x}^t)$;
for *each iteration* k *from* 1 *to* B **do**
  Pick σ random controller pairs p for which $\widehat{x}_p < R$;
  for *each picked pair* p *from* 1 *to* σ **do**
    Set $\widehat{\boldsymbol{x}}' = \widehat{\boldsymbol{x}}$ where $\widehat{x}'_p = \widehat{x}_p + 1$;
    Try out $\boldsymbol{x}^t = \widehat{\boldsymbol{x}}'$ and observe $\Psi_t(\boldsymbol{x}^t)$
     $\forall t \in \{(k-1)\sigma\tau + p\tau + 1, \ldots, (k-1)\sigma\tau + p\tau + \tau\}$;
    Estimate $\widehat{\Psi}(\widehat{\boldsymbol{x}}') = \frac{1}{\tau}\sum_{t=(k-1)\sigma\tau+p\tau+1}^{(k-1)\sigma\tau+p\tau+\tau} \Psi_t(\boldsymbol{x}^t)$;
    Set $D(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{x}}') = \widehat{\Psi}(\widehat{\boldsymbol{x}}') - \widehat{\Psi}(\widehat{\boldsymbol{x}})$;
  end
  Update $\widehat{\boldsymbol{x}} = \text{argmax}_{\widehat{\boldsymbol{x}}'} D(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{x}}')$;
end
Output: $\widehat{\boldsymbol{x}}$;

will be the actual one. However, as the μ value goes to 0 the observations become more uncertain.

Another issue is that the performance bound in Theorem 2 holds in expectation, which means that it may be violated in practice. Therefore, it is important to bound the extent to which this happen, as we show in the following theorem.

Theorem 3. *Algorithm 3 achieves average performance* $\widehat{\Psi}(\widehat{\boldsymbol{x}})$ *that is a factor* $1-e^{-(1-\epsilon)(1-\gamma)\mu}$ *from the optimal with probability* $1 - e^{-\frac{\gamma B \tau}{2}}$ *for any* $\gamma \in (0, 1)$.

The average performance bounds of our algorithm can be better understood through an example. In particular, consider the system with $C = 5$ controllers, $B = 10$ available resources and $s = 30$ seconds per time slot. By picking $\sigma = 5$ out of the 20 possible controller pairs and $\tau = 3$ time slots per try-out, the total running (training) time of the algorithm will be about one hour. Moreover, if the observed marginal performance gains are 50% or more of the actual ones ($\mu = 0.5$) and $R = 1$, the average performance achieved by the algorithm will be in expectation at least 37% of the optimal. Picking a larger σ value will increase the average performance (cf. Theorem 2). Picking a larger τ value will increase the probability that the performance bound is not violated (cf. Theorem 3).

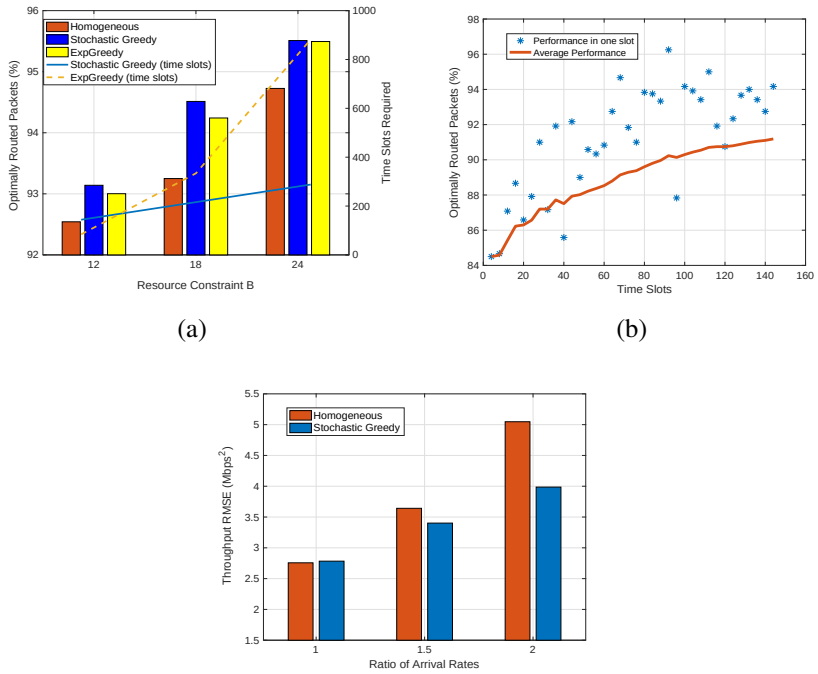


Figure 4.3: (a) Performance and training time for different resource budgets and (b) learning process under the shortest path routing application. (c) RMSE cost for different ratios of flow arrival rates under the load-balancing application.

4.5 Evaluation Results

In this section, we carry out evaluations to show the benefits of the proposed algorithm. Overall, we find that benefits are realized compared with the baseline algorithm that synchronizes all the controller pairs at equal rate (referred to as Homogeneous). Moreover, our Stochastic Greedy algorithm achieves better performance-training time tradeoff than a state-of-the-art learning algorithm (ExpGreedy in [160]).

Evaluation Setup. We choose the same network topologies and applications as in our emulations in Section 4.2 (16-node shortest path routing and 2-server load balancing). We compare our Stochastic-Greedy algorithm with both the Homogeneous and ExpGreedy algorithms. To eliminate randomness, we run each algorithm 10 times and take the average value.

Shortest Path Routing. We first consider the shortest path routing application in the 16-node network. A performance metric of interest for this application is the percentage of packets that are optimally routed to their destinations, i.e., following paths of the same

number of hops as the optimal path. Figure 4.3(a) depicts the performance for different resource budgets B. We notice that *the proposed Stochastic-Greedy algorithm routes optimally more packets than Homogeneous and ExpGreedy algorithms*. The training time required by our algorithm increases linearly with B. On the other hand, the time of ExpGreedy increases more dramatically, which shows that our algorithm is more scalable. Specifically, *our algorithm requires around 200 time slots (about an hour and a half) for training while ExpGreedy may consume more than 800 time slots (6-7 hours)*, which may be prohibitively large in practice.

Figure 4.3(b) illustrates the learning process when Stochastic Greedy is run for $B = 18$, $\sigma = 2$ and $\tau = 4$. Although in each time slot the algorithm observes a performance value with large randomness, it is able to allocate resources to proper pairs and increase the average performance over time.

Load Balancing. Finally, we examine the load balancing application. Similar to the emulations in Section 4.2, we randomly generate flows at two switches. We define one time slot as 60 seconds. Under the same B value, we compare the Stochastic Greedy and Homogeneous algorithms for various flow arrival rates. When the arrival rates at the two switches are equal, the Homogeneous algorithm should be optimal because of the symmetry. In this case, as Figure 4.3(c) shows, our algorithm gets almost the same RMSE cost. Next, we set different arrival rates at the two switches. As a result, when the ratio of arrival rates gets larger, our algorithm leads to lower cost than the Homogeneous algorithm. For example, our algorithm can decrease the RMSE by around 20% when the ratio of flow arrival rates at the two switches is equal to 2.

4.6 Related Work

Distributed SDN controller deployments require a coordination protocol among controllers, which could easily generate significant amount of control traffic, e.g., see the measurement studies in [113]. However, the control traffic is often neglected in literature with most of the existing works focusing on the routing and balancing of the data traffic, e.g., see [171] and the survey in [82].

Few recent works suggested the dynamic adaptation of synchronization period (or rate) among controllers in an eventual-consisted system so as to improve the performance of network applications while maintaining a scalable system [11], [151]. While interesting and relevant, the above works did not provide any mathematical formulation or optimization framework. To the best of our knowledge our work is the first to systematically study

the synchronization problem and propose optimization and learning methods.

4.7 Summary

In this work, we studied the problem of finding the optimal synchronization rates among controllers in a distributed eventually-consistent SDN system. Our objective was to maximize the performance of applications which may be affected by the synchronization decisions, as highlighted by emulations on a commercial SDN controller. For this objective, we characterized the complexity of the problem and proposed algorithms to achieve the optimal synchronization rates. Evaluation results demonstrated significant performance benefits over the baseline policy that synchronizes all controller pairs at equal rate.

Chapter 5

Distributed Control Plane: Orchestrating Heterogeneous Networks

Having investigated multiple problems during the distributed deployment of the control plane, there are still challenges existing in applying such architecture to Radio Access Networks (RANs), through which the IoT devices are connected to each other or the Internet. Nowadays, multiple alternative radio access technologies exist (e. g. ,LTE, WLAN, and WiMAX), and there is no unifying abstraction to compare and compose from diverse technologies. In addition, access networks may belong to different providers where competitions may exist.

In this chapter, we propose to adopt the Software-Defined Radio Access Network (SD-RAN) architecture and virtualization technologies for heterogeneous RAN slicing across multiple providers. A central component in our architecture is a service orchestrator that interacts with multiple network providers and service providers to negotiate resource allocations that are jointly optimal. We propose a double auction mechanism that captures the interaction among selfish parties and guarantees convergence to optimal social welfare in finite time. We then demonstrate the feasibility of our proposed system by using open source SD-RAN systems such as EmPOWER (WLAN) and FlexRAN (LTE).

5.1 Introduction

5G technologies will revolutionize mobile networks and push them to the limit. Besides the significant improvement in efficiency and capacity, the network has better support to a wide range of services with distinct requirements by virtualization. On the same physical infrastructure, multiple virtual networks are established as slices, and network resources

are isolated into each slice to meet the requirement of different services.

An end-to-end virtualization involves slicing not only in the core but also in the radio access networks (RANs). One challenging problem of RAN slicing is the coexistence of heterogeneous radio access technologies (RATs) such as LTE, WLAN and WiMAX, where types of resources are not identical and cannot be allocated under a uniform mechanism. Moreover, it is common for a mobile device (such as a smartphone) to have multiple network interfaces and utilize them simultaneously. Therefore, slicing and radio resource allocation across multiple RANs is required.

Aiming towards more advanced coordination of heterogeneous RANs, 3GPP has developed standards like LTE-WLAN Aggregation and LTE-WLAN Radio Level Integration. Several slicing architectures across RANs have also been proposed [32][27]. These solutions usually enforce changes or deploy new components in the network infrastructure. They also assume RANs are owned by the same network provider or they are fully cooperative. These requirements cannot be satisfied in a more general case when multiple network providers have private infrastructures and compete in selling resources to services.

Software-Defined Radio Access Network (SD-RAN) brings new possibilities to this problem. By extending the SDN concept of a centralized and programmable management layer to RANs, radio resource allocations at physical devices (e.g., eNodeBs of LTE and Access Points of WLAN) can be achieved in a flexible manner through a central controller.

Building on the flexibility of SD-RAN, we proposed a novel architecture towards heterogeneous RAN slicing. Our architecture includes a slicing orchestrator which coordinates multiple network providers and service providers to reach a joint slicing allocation by negotiations. In our architecture, the SD-RAN controller of each network provider has an associated agent which runs as an SDN application and takes part in the resource allocation mechanism. Compared with existing approaches, our architecture has more flexible and modular support of heterogeneous RANs. As a network application, it can be dynamically deployed on SD-RAN platforms owned by either the same or different operators without any infrastructure changes and is not limited within specific RATs. At the same time, it takes advantage of the functions that already exist in the RAN such as access control, handover and resource abstractions.

Specifically, we make the following contributions:

- We design an architecture of a novel type of orchestrator which realizes network slicing across heterogeneous radio access technologies and SD-RAN platforms, taking diversity of network services, competing network owners and users' multi-connectivity

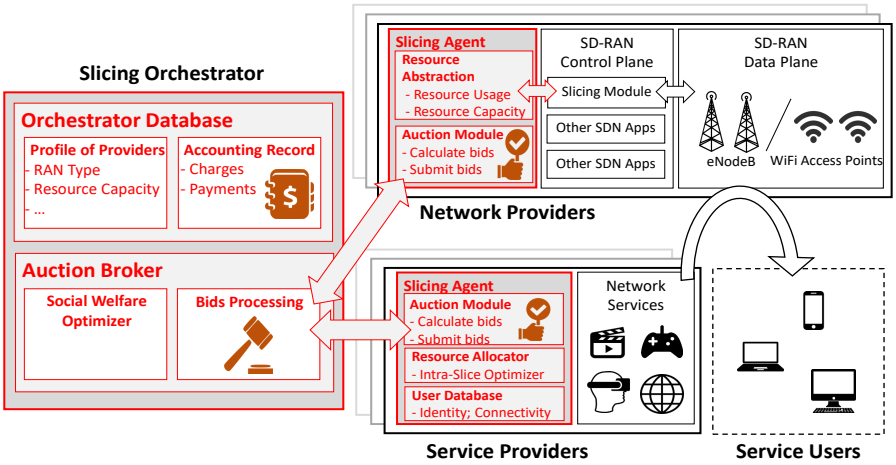


Figure 5.1: Architecture of proposed system. The new components we introduce (Slicing Orchestrator and Slicing Agents) are marked in red.

into account.

- We propose theoretical models capturing interactions and competitions among different roles (e.g., network providers and service providers) during slicing configuration. We guarantee optimal social welfare by an iterative double auction algorithm.
- We develop a prototype of our proposed architecture based on state-of-the-art SD-RAN open-source projects and real commercial mobile devices. We evaluate our system by taking measurements from multiple realistic use-cases and evaluating multiple performance metrics.

The rest of the chapter is organized as follows. In Section 5.2, we propose our system design and establish mathematical models for each component. In Section 5.3 we provide algorithms to optimize the resource allocation among multiple providers. We describe our testbed setup and evaluate the proposed architecture in Section 5.4. We list related works in Section 5.5 and summarzie the whole work in Section 5.6.

5.2 System Design and Problem Modeling

5.2.1 Overview

In this section, we propose an architecture and system model for achieving network slicing across heterogeneous SD-RANs. In a typical slicing scenario, there will be multiple

network providers of same or different radio access technologies (RAT) represented by a set $\mathcal{K} = \{1, 2, ..., K\}$, and multiple service providers represented by $\mathcal{M} = \{1, 2, ..., M\}$. Each service provider owns a slice with a certain amount of isolated resources (e. g., power, bandwidth, speed). Such resource slicing (isolation) tasks are challenging, as they should permit a slice to purchase resources from more than one RAN, and make decisions on the amount of resources to allocate that satisfy the demands of all network and service providers.

To solve this problem, a key component in our design is the *Slicing Orchestrator*. It is a centralized entity establishing connections to all network and service providers. However, it is owned and operated by a third party different from the network and service providers. Although the orchestrator does not have the full access to either control or private information of each RAN, it is capable of managing the competitions among network and service providers by holding *auctions*. The purpose of the orchestrator is to maximize the slicing efficiency, which is represented by *social welfare* maximization, while making profits for itself. Social welfare is typically defined as the sum of utilities of every agent involved in the auction. In order for heterogeneous providers to communicate with the Slicing Orchestrator through a uniform protocol, a *Slicing Agent* is deployed at each network and service provider, which is another significant component in our design. We do not require any modifications or new components in the devices of network and service users.

Figure 5.1 shows the overall system architecture. To demonstrate the incentive of deploying the orchestrator and the auction mechanism, we first model the problem in aspects of both service and network providers.

5.2.2 Service Provider Slicing Agent

Each slice is owned by a service provider. We assume that a Slicing Agent is deployed for requesting resources from network providers. The Slicing Agent aggregates the demands of all service users to estimate the amount of resource required.

User Connectivity Profile. A service provider $m \in \mathcal{M}$ usually has multiple users to serve through its slice. The connectivity of a user $i \in \mathcal{I}_m$ can be denoted by a vector $\beta_{\mathbf{i}} = (\beta_{ki})_{k \in \mathcal{K}}$. β_{ki} is a non-negative number representing factors such as the link quality (e.g., $(0, 1]$ depending on the path loss). The values can be determined according to related indicators (e.g., Channel Quality Indicator (CQI) in LTE, Received Signal Strength Indicator (RSSI) in 802.11) reported by users.

Intra-slice Resource Allocation. The service provider should consider all its users

when requesting resources for its slice, which is an optimization problem of intra-slice resource allocation. Suppose a service provider m requests a certain amount of resources from every RAN k denoted by a vector $\mathbf{x_m} = (x_{mk})_{k\in\mathcal{K}}$, then it allocates a portion of them, $\mathbf{z_i} = (z_{ki})_{k\in\mathcal{K}}$ to each user i. Depending on the resource allocated, user i has its utility $u_{mi}(\mathbf{z_i})$ (the form varies based on the type of service). In order to maximize the sum of all user utilities, an optimization problem should be solved by service provider m:

$$U_m(\mathbf{x_m}) = \max_{\mathbf{z_i}} \sum_{i=1}^{I_m} u_{mi}(\mathbf{z_i}) \tag{5.1}$$

$$\text{s.t.} \sum_{i=1}^{I_m} \frac{z_{ki}}{\beta_{ki}} \leq x_{mk}, \forall k \in \mathcal{K} \tag{5.2}$$

$$z_{ki} \geq 0, \forall i \in \mathcal{I}_m, k \in \mathcal{K} \tag{5.3}$$

We can make an assumption that user utility $u_{mi}(.)$ is an increasing and concave function, which holds in most scenarios, e.g., elastic traffic [142], or services that guarantee fairness [75].

Objective during Slicing. The Slicing Agent determines $\mathbf{x_m}$, the amount of resources to request which maximizes the service provider's own interest, denoted by the utility function $U_m(\mathbf{x_m})$ of this service provider m.

5.2.3 Network Provider Slicing Agent

Similarly, a Slicing Agent exists as an application of each SD-RAN platform, determining the type and amount of resources allocated to different services.

Resource Abstraction. Although an SD-RAN may have its own abstraction of radio resources (e.g., Resource Blocks in LTE, airtime control in WLAN), it is able to quantify them as the performance level of the same network metric (e.g., downlink throughput), which will be a crucial function of the Slicing Agent. If we consider one specific network metric in this way, the resource offered by a network provider k can be denoted by a vector $\mathbf{y_k} = (y_{km})_{m\in\mathcal{M}}$. Without loss of generality[1], capacity C_k limits the amount of resources that can be offered, i.e., $\sum_{m=1}^{M} y_{km} \leq C_k$.

Objective during Slicing. Similar to the service provider, the Slicing Agent of a net-

[1]Although the capacity can be dependent of k in the cases such as Wi-Fi channel conflicts, the orchestrator introduced later is capable to notify each provider to prevent such conflicts. In addition, it is easy to extend the algorithm described in the next section to other forms of linear constraints.

work provider aims at maximizing its own profit, i.e., minimizing a cost function $V_k(\mathbf{y_k})$. This cost occurs because of the operation and management overheads of the RAN, as well as the opportunity cost since the network provider cannot use these resources for other purposes. In this work, we mainly discuss the representative cases where $V_k(\mathbf{y_k})$ function is increasing and convex. [37]

5.2.4 Slicing Orchestrator

A service provider requests resources for its slice which maximize its utility, while network providers aim at minimizing their costs. Since these goals are at conflict it is hardly possible to achieve it without negotiations through a third party. This is the job of the Slicing Orchestrator which leads to an agreement of resource requests and offers through double auctions. In other words, a legal slicing scheme requires $y_{km} = x_{mk}$ for any service provider m and network provider k. Beyond that, we consider maximizing the total utility of all providers, which can be regarded as a social welfare optimization problem:

$$\max_{\mathbf{x_m},\mathbf{y_k}} \sum_{m=1}^{M} U_m(\mathbf{x_m}) - \sum_{k=1}^{K} V_k(\mathbf{y_k}) \tag{5.4}$$

$$\text{s.t. } y_{km} = x_{mk}, \forall k \in \mathcal{K}, m \in \mathcal{M} \tag{5.5}$$

$$\sum_{m=1}^{M} y_{km} \leq C_k, \forall k \in \mathcal{K} \tag{5.6}$$

$$y_{km} \geq 0, x_{mk} \geq 0, \forall k \in \mathcal{K}, m \in \mathcal{M} \tag{5.7}$$

Although the Slicing Orchestrator acts as a centralized component connecting to Slicing Agents of all providers, it cannot solve this problem directly. First, it is a reasonable assumption that each provider is selfish and cares about their own utility or cost, rather than the social welfare. Second, the orchestrator does not have full access to information private to network and service providers. More specifically, providers do not always have the incentive to reveal their utility and cost functions $U_m(\mathbf{x_m})$ and $V_k(\mathbf{y_k})$. Moreover, the orchestrator should be profiting during the resource allocation in order to maintain itself. In the next section, we will introduce a double auction mechanism to solve this problem, where the Slicing Orchestrator is the broker and each Slicing Agent is a bidder.

5.3 Optimization Algorithms

In this section, we introduce the methods to solve the social welfare optimization problem described in the last section. We also analyze the benefit of proposed mechanism theoretically in comparison with other possible slicing architectures.

5.3.1 User Utility Optimization

First, we focus on the properties of the service provider utility function $U_m(\mathbf{x_m})$. It is the aggregation of each single service user's utility, where a subproblem of the intra-slice resource allocation exists, as stated in (5.1).

If $\mathbf{x_m}$ has been determined by the orchestrator, the service provider can directly solve this subproblem by itself. For example, the unique optimal can be efficiently found by applying Karush-Kuhn-Tucker (KKT) conditions [28]. What is more, $U_m(\mathbf{x_m})$ has following important property:

Lemma 3. *$U_m(\mathbf{x_m})$ is an increasing and concave function.*

Proof. For the monotonicity, with an increased x_{mk}, allocating the marginal value to any arbitrary user i improves $u_{mi}(\mathbf{z_i})$ and $\sum_{i=1}^{I_m} u_{mi}(\mathbf{z_i})$. Therefore, the optimal allocation $U_m(\mathbf{x_m})$ is increasing as well.

For the concavity, define $\mathbf{z} = (z_{ki})_{k\in\mathcal{K}, i\in\mathcal{I}_m}$, and

$$f(\mathbf{z}, \mathbf{x_m}) = \begin{cases} \sum_{i=1}^{I_m} u_{mi}(\sum_{k=1}^{K} z_{ki}) & \text{if constraint (5.2) holds} \\ -\infty & \text{otherwise} \end{cases}$$

$f(\mathbf{z}, \mathbf{x_m})$ is a concave function of both $\mathbf{z}$ and $\mathbf{x_m}$. According to [28], its partial maximization (i.e., $U_m(\mathbf{x_m})$) preserves concavity. □

5.3.2 Iterative Double Auction

The result of the subproblem above implies the concavity of the social welfare function, making it possible for us to adopt a double auction mechanism similar to [66] optimizing the resource allocation during slicing.

By applying KKT conditions and introducing Lagrange multipliers, the problem (5.4) has a unique optimal solution because of the concavity. However, it cannot be acquired

without information of $U_m(\mathbf{x_m})$, $V_k(\mathbf{y_k})$. Instead, we consider the following alternative optimization problem:

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = & \sum_{m=1}^{M} \sum_{k=1}^{K} (p_{mk} \log x_{mk} - \frac{a_{km}}{2} y_{km}^2) \\ & - \sum_{k=1}^{K} \lambda_k (\sum_{m=1}^{M} y_{km} - C_k) \\ & - \sum_{k=1}^{K} \sum_{m=1}^{M} \mu_{mk} (x_{mk} - y_{km}) \end{aligned} \tag{5.8}$$

where $\boldsymbol{\lambda} = (\lambda_k \geq 0)_{k \in \mathcal{K}}$ and $\boldsymbol{\mu} = (\mu_{mk} \geq 0)_{k \in \mathcal{K}, m \in \mathcal{M}}$ are Lagrange multipliers. There are undetermined parameters $\mathbf{a_k} = (a_{km} \geq 0)_{k \in \mathcal{K}, m \in \mathcal{M}}$ in this alternative problem, which are the bids that the broker expects each network provider k to submit. Similarly, $\mathbf{p_m} = (p_{km} \geq 0)_{k \in \mathcal{K}}$ are bids from each service provider m. Two sets of rules are required for the auctions. First, we need *allocation rules* to solve this alternative problem. Second, *payment rules* will guide the providers to submit bids determining the parameters of this alternative problem, which should lead the optimal solution coincide with the original problem.

Allocation Rules. Optimal results $\mathbf{x}^*$ and $\mathbf{y}^*$ of this alternative problem can be calculated by KKT conditions of $L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}, \boldsymbol{\mu})$:

$$x_{mk}^* = \frac{p_{mk}}{\mu_{mk}^*}, \qquad y_{km}^* = \frac{\mu_{mk}^* - \lambda_k^*}{a_{km}} \tag{5.9}$$

$$\mu_{mk}^* (x_{mk}^* - y_{km}^*) = 0, \qquad \lambda_k^* (\sum_{m=1}^{M} y_{km}^* - C_k) = 0 \tag{5.10}$$

Equation (5.9) shows the allocation rules of the double auction, revealing how the orchestrator determine the amount of resources allocated to each slice based on bids received.

Payment Rules. By comparing the alternative problem (5.8) with the original one (5.4), we notice that they have the same optimal solution $\mathbf{x}^*$ and $\mathbf{y}^*$ only when:

$$p_{mk} = x_{mk}^* \frac{\partial U_m(\mathbf{x_m^*})}{\partial x_{mk}}, \; a_{km} = \frac{1}{y_{km}^*} \frac{\partial V_k(\mathbf{y_k^*})}{\partial y_{km}} \tag{5.11}$$

The orchestrator applies payment rules to induce bidders submitting the above values. More specifically, the broker charges $g_m(\mathbf{p_m})$ to each service provider m for the resource

Algorithm 4: Iterative Double Auction

$t \leftarrow 0$
Initialize $\mathbf{x}^{(0)}, \mathbf{y}^{(0)}, \boldsymbol{\lambda}^{(0)}, \boldsymbol{\mu}^{(0)}$
IsConverged$\leftarrow$ False
while *IsConverged is False* **do**
  The broker announces $\boldsymbol{\lambda}^t, \boldsymbol{\mu}^t$
  Each service provider m submits bids $\mathbf{p_m}^{(t+1)}$ by (5.12); each network provider n submits bids $\mathbf{a_n}^{(t+1)}$ by (5.13)
  The broker calculates $\mathbf{x}^{(t+1)}$ and $\mathbf{y}^{(t+1)}$ by (5.9)
  The broker calculates $\boldsymbol{\lambda}^{(t+1)}$ and $\boldsymbol{\mu}^{(t+1)}$ by:
    $\mu_{mk}^{(t+1)} = (\mu^{(t)} + s^{(t)} \cdot (x_{mk}^{(t)} - y_{km}^{(t)}))^+$
    $\lambda_k^{(t+1)} = (\lambda_k^{(t)} + s^{(t)} \cdot (\sum_{m=1}^{M} y_{km}^{(t)} - C_k))^+$
    $\forall k \in \mathcal{K}, m \in \mathcal{M}$, and $s^{(t)} > 0$ is the step size of gradient descent.
  if $|\frac{p_{mk}^{(t+1)} - p_{mk}^{(t)}}{p_{mk}^{(t)}}| < \epsilon_1$ *and* $|\frac{a_{km}^{(t+1)} - a_{km}^{(t)}}{a_{km}^{(t)}}| < \epsilon_2$, $\forall k \in \mathcal{K}, m \in \mathcal{M}$ **then**
    IsConverged $\leftarrow$ True
  end
  $t \leftarrow t + 1$
end
Output: $\mathbf{x}^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\lambda}^{(t)}, \boldsymbol{\mu}^{(t)}$

it bids to request, and pays $h_k(\mathbf{a_k})$ to each network provider k for the resource it bids to offer.

In this case, each service provider m determines its bid that maximizes their payoff:

$$\mathbf{p_m^*} = \arg\max_{\mathbf{p_m}} (U_m(\mathbf{x_m}) - g_m(\mathbf{p_m})) \tag{5.12}$$

Similarly, each network provider k makes decisions according to:

$$\mathbf{a_k^*} = \arg\max_{\mathbf{a_k}} (-V_k(\mathbf{y_k}) + h_k(\mathbf{a_k})) \tag{5.13}$$

Payment rules should make these results coincide with Equation (5.11). It can be calculated by combining the allocation rules (5.9) with bids expressions (5.11)(5.12) and (5.13). As a result, the payments and charges are proportional to the resources demanded/offered:

$$g_m(\mathbf{p_m}) = \sum_{k=1}^{K} p_{mk}, \qquad h_n(\mathbf{a_k}) = \sum_{m=1}^{M} \frac{(\mu_{mk} - \lambda_k)^2}{a_{km}} \tag{5.14}$$

Iterative Algorithm. The allocation and payment rules above are parameterized by

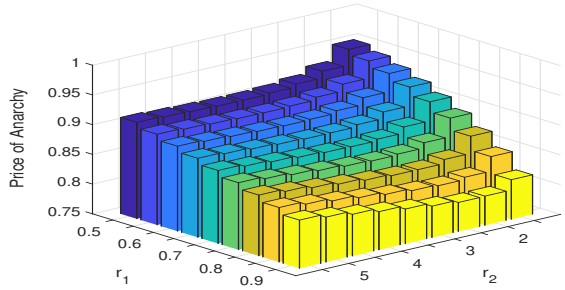


Figure 5.2: PoA with different utilities and costs when disabling the orchestrator.

the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, which can be calculated in a gradient descent manner by running auctions of multiple rounds. The procedure of the iterative double auctions is listed in Algorithm 4.

Defining a Lyapunov function summing the quadratic drifts of $\boldsymbol{\lambda}^{(t)}$ and $\boldsymbol{\mu}^{(t)}$, the convergence can be proved [66] under the assumption that bidders are price-takers, where they passively accept the price raised by the broker, rather than strategically exert impact on it. It is true in perfect competition market, which is reasonable in our architecture because we are considering multiple network and service providers with limited information of each other.

According to (5.12)(5.13), the algorithm is efficient and individually rational, i.e., optimal social welfare is reached when every bidder maximizes their own payoffs. This conclusion can easily be extended to the case where a service provider owns multiple slices or a network provider owns multiple sets of RAN infrastructures. The provider can simply make decisions for each of its slice/infrastructure independently.

The algorithm is scalable in aspects of both computing (concave minimization with linear constraints) and synchronization overheads ($\mathcal{O}(M \cdot K)$ messages in each round). In addition, it is straightforward to demonstrate that the profit $\sum_{m=1}^{M} g_m(\mathbf{p_m^*}) - \sum_{k=1}^{K} h_k(\mathbf{a_k^*})$ is always non-negative. Therefore, the orchestrator faces no problem of maintaining itself (it never runs a loss) and has the incentive to hold auctions.

5.3.3 Social Welfare Improvement

The optimal social welfare achieved by introducing the Slicing Orchestrator is nontrivial. In particular, we demonstrate that while our method can guarantee optimality with Price of Anarchy (PoA) $= 1$, alternative distributed architectures without centralized control

(where providers directly negotiate with each other as described below) can lead to sub-optimal performance with PoA < 1. Thus the central orchestrator of our architecture is an essential component for RAN slicing.

The Stackelberg game is a typical model to depict such distributed architectures and widely discussed in related literature [50][76], in which a leader and followers take actions sequentially. The interaction of each service provider m and network providers in this alternative architecture can therefore be captured by a two-stage Stackelberg game:

- Stage 1: The service provider announces P_m, the price it is willing to pay for every unit resource offered.
- Stage 2: Each network provider k submits $x_{mk} = y_{km}$.

To analyze the price of anarchy (PoA) for the Stackelberg game model, we need to make a few additional assumptions. First, we assume network providers are able to interact with every service provider independently, by assuming $V_k(\mathbf{y}_k) = \sum_{m=1}^{M} V_{km}(y_{km})$. Secondly, the capacity of resources is no longer a constraint. We also assume all information are public. These assumptions actually weaken the practicality of such models. A strength of our proposed design is that the difficulties due to these assumptions are avoided. Moreover, we demonstrate that the Stackelberg game has inferior social welfare even if all these extra assumptions are satisfied.

The game has an equilibrium. At Stage 2, given the price P_m, a network provider responds maximizing its payoff:

$$y^*_{mk}(P_m) = \arg\max_{y_{km}}(P_m \cdot y_{km} - V_{km}(y_{km})) \tag{5.15}$$

Anticipating the response, the service provider will determine the price in Stage 1 as:

$$P^*_m = \arg\max_{P_m}(U_m(\mathbf{y}^*_m(P_m)) - P_m \cdot \sum_{k=1}^{K} y^*_{mk}(P_m)) \tag{5.16}$$

Correspondingly, the social welfare under the equilibrium is:

$$SW_{equil} = \sum_{m=1}^{M} U_m(\mathbf{y}^*_m(P^*_m)) - \sum_{k=1}^{K} V_k(\mathbf{y}^*_k(\mathbf{P}^*_m)) \tag{5.17}$$

We introduce Price of Anarchy (PoA), the ratio of social welfare between the worst equilibrium and the centralized optimal solution, as a metric to demonstrate the benefits

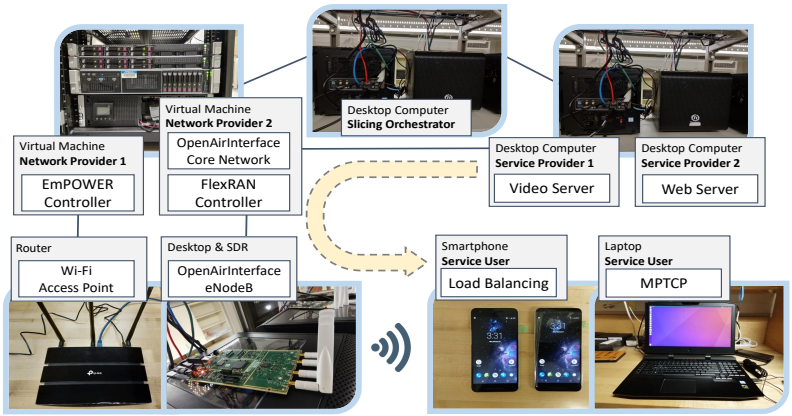


Figure 5.3: Testbed setup and experimentation scenario of two services (video streaming and web browsing) and two RANs (WLAN and LTE).

gained by setting up an orchestrator. For instance, we consider following power functions as utility and cost, i.e., $U_m(\mathbf{x}_m) = A \cdot (\sum_{k=1}^{K} x_{mk})^{r_1}$ and $V_k(y_{km}) = B_k \cdot \sum_{m=1}^{M} y_{km}^{r_2}, \forall k \in \mathcal{K}, r_1 \in (0,1), r_2 \in (1,\infty)$. Then the equilibrium social welfare SW_{equil} can be calculated following (5.15) and (5.16). It is also trivial to acquire the unique optimal solution SW_{opt} by making $x_{mk} = y_{km}$ and taking derivatives of $\sum_{m=1}^{M} U_m(\mathbf{x}_m) - \sum_{k=1}^{K} V_k(\mathbf{y}_k)$. Finally the PoA has following expression:

$$PoA = \frac{SW_{equil}}{SW_{opt}} = \frac{r_2^{\frac{2r_1}{r_1-r_2}} - r_2^{\frac{2r_2}{r_1-r_2}} \cdot r_1}{r_2^{\frac{r_1}{r_1-r_2}} - r_2^{\frac{r_2}{r_1-r_2}} \cdot r_1} \tag{5.18}$$

The PoA is impacted by the extent of concavity/convexity of utility/cost functions. In Figure 5.2 we calculate PoA in different combinations of r_1 and r_2. The results are lower than 0.8 in worst cases, indicating that our proposed architecture is able to improve the social welfare by more than 25% in specific scenarios. Later in the evaluation section, we will also demonstrate this improvement in realistic settings.

5.4 Evaluation Results

In this section, we evaluate our proposed architecture and algorithm with both our implementation in real devices and numerical simulations in large-scale network topologies.

5.4.1 Testbed Setup

To quantitatively evaluate the performance of realistic scenarios, we build a testbed containing heterogeneous SD-RANs, multiple network services and different types of user devices, as shown in Figure 5.3.

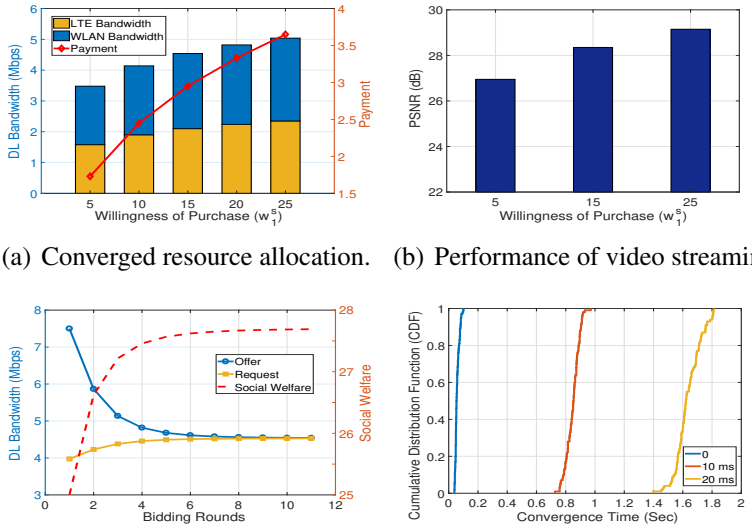
We set up two network providers, one of LTE and another of WLAN. In the LTE network, a desktop computer (3.6 GHz, 16 GB of RAM) with USRP B210 deploying OpenAirInterface [120] eNodeB works as the data plane on LTE band 7. We also deploy virtual machines in a server (HP ProLiant DL360) running components of LTE core network (HSS, MME and SPGW) and the FlexRAN control plane. The WLAN network has similar setting, while deploying EmPOWER control plane and using a router (TP-Link AC1750) as data plane on 802.11g, channel 11 instead. Control planes of both RANs run our Network Provider Agent. The details of our prototype implementation are described in Appendix C.

We also set up two service providers (one web server and one video streaming server) in another desktop computer, both deploying our Service Provider Agent. The Slicing Orchestrator exists in third desktop computer. 1Gb/s Ethernet links are set between RAN data planes and RAN control planes, as well as the agents and orchestrator.

Two types of user devices, one HP Omen laptop and two Nexus 6P Android smartphones are deployed for experiments. The laptop connects to LTE network with a Huawei E3372 LTE USB modem. Besides, we consider multiple approaches enabling multi-connectivity and deploy different solutions in user devices.

5.4.2 Small-Scale Experimentation Results

Video Streaming Service. First, we consider a scenario where a single service provider requests downlink bandwidths from both LTE and WLAN providers, in order to support HTTP video streaming to a laptop through MPTCP v0.93 [179]. We assume that network providers have cost functions $V_k(\mathbf{y_k}) = w_k^n \cdot \sum_{m=1}^{M} y_{km}^2$. Here we have $k = 1$ for LTE and $k = 2$ for WLAN. According to the actual performance of infrastructure, we set $C_1 = 15Mbps$ and $C_2 = 9Mbps$. On the user side, we choose the quality of received video as its utility. Although it would not be easy to deduct its exact relationship with downlink bandwidth, the network provider can estimate it using an elastic utility function $u_{mi}(\mathbf{z_i}) = w_m^s \cdot \sum_{k=1}^{K}(1 - e^{-\alpha \cdot z_{ki}})$, $m = 1, i = 1$, which is general to cover various services [142]. Value of α can be estimated depending on the video bitrate. We use a 1080P video (around 7000 kbps) for experimentation, and assume $\alpha = 1.6$ correspondingly.



(a) Converged resource allocation. (b) Performance of video streaming.

(c) Convergence of an auction. (d) Time to finish auctions.

Figure 5.4: The (a) resource allocation and payment schemes determined by double auction with different weights of service provider utilities. (b) Actual performance of video streaming by measuring PSNR. (c) Number of auction rounds and (d) actual time required to finish the algorithm.

We fix $w_1^n = 0.2$, $w_2^n = 0.1$ and run the system with different w_1^s values, which represent the willingness of the service provider to purchase resources for its slice. Figure 5.4(a) shows the results of proposed double auction algorithm, in which the service provider requests a share of bandwidth from both RANs. With a larger w_1^s, the service provider acquires more resources, indicating the capability of proposed system to balance the offer and request with different utilities/costs of providers. The result is concave in w_1^s, which suits the video streaming service because redundant bandwidth beyond the video bitrate adds little value. The orchestrator receives payments from the service provider and make compensations to network providers. In this scenario, these two amounts are balanced. And it is intuitive that the service provider pays more for larger requests.

We also measure the actual performance of video streaming. After the whole video is streamed, we quantify the received video quality by measuring its peak signal-to-noise ratio (PSNR). A larger PSNR value indicates a smaller quality loss during streaming. Figure 5.4(b) shows how the PSNR values increase with larger bandwidth allocated.

Another crucial performance metric of proposed system is the time spent on finishing the slice configuration. Figure 5.4(c) shows the procedure of the iterative double auction. With gradient descent step size 0.1, the offers and requests quickly converge in 11 bidding rounds. We also measure the actual time spent on finishing an auction as Figure 5.4(d), in

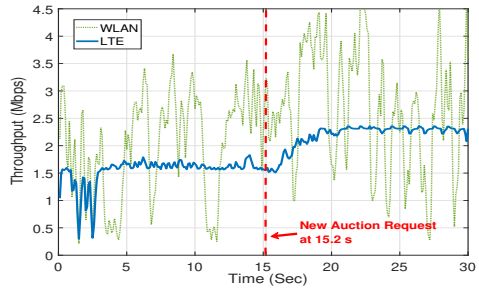
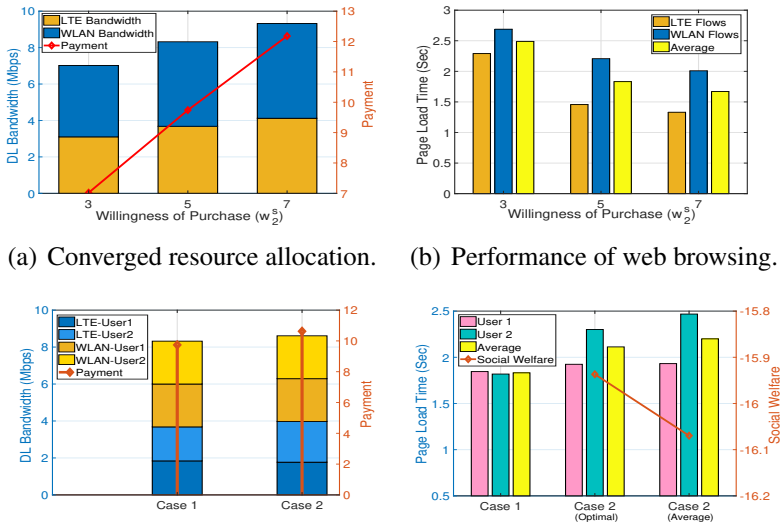


Figure 5.5: Real-time MPTCP throughput monitoring of video streaming. The service provider increases w_1^s from 5 to 25 and starts a new auction during the transmission.

which we hold auctions for 100 times and plot the cumulative distribution function (CDF). With all participants placed in the same room with wired connections, it always takes less than 0.1 second. Then we add a simulated delay on all outgoing traffic from the Slicing Orchestrator. The auctions can still be finished quickly within 1 and 2 seconds, when the delay is set to $10ms$ and $20ms$.

Furthermore, to verify above conclusions from another aspect, we monitor the traffic of video streaming through both RANs in real time. As depicted in Figure 5.5, initially the throughput from each RAN is consistent with the slicing scheme in Figure 5.4(a). Then, we assume that the service provider changes its willingness w_1^s from 5 to 25 and therefore starts a new auction at the 15th second. As a result, the throughput starts to increase within 1 second, and becomes stable again within 5 seconds, indicating the whole procedure of slicing has been finished. In the figure, the WLAN throughput shows fluctuations, because we set the temporal interval of the curve as 0.1 second to better indicate the system's dynamic response. The queueing-based slicing mechanism of EmPOWER cannot achieve the same level of fine-grained control as the Resource Block allocation of LTE. However, it is able to follow the auction result correctly on a larger time scale, e.g., when measuring the average bandwidth of every 1 second. Therefore, we assert that the auction and slicing mechanisms work smoothly as designed and is flexible enough to adapt dynamic changes of user demands.

Web Browsing Service. We then consider another scenario of two Android smartphones surfing the Internet to investigate different performance metrics. Instead of MPTCP, we consider another multi-connectivity case, the load balancing assignment of flows to different network interfaces. With each smartphone, we send the same amount of HTTP requests to download a large HTML page (around 1.2 MB) through both LTE and WLAN connections. And we measure the average page load time as the performance metric. Cor-



(a) Converged resource allocation. (b) Performance of web browsing.

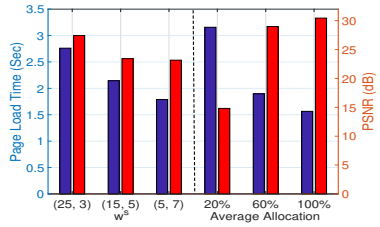
(c) Adaptive adjustment of requests. (d) Performance comparison.

Figure 5.6: The (a) resource allocation, payment schemes and (b) performance of web browsing service. (c)(d) shows how the service provider adjusts its bids depending on the signal strength of its users.

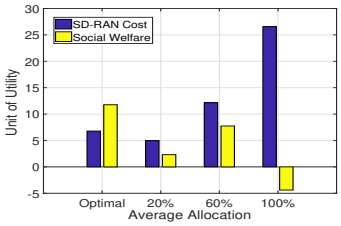
responding to this metric, the service provider may choose a different utility function in the form of minimum potential delay fairness, $u_{mi}(\mathbf{z_i}) = w_m^s \cdot \sum_{k=1}^{K}(-1/z_{ki})$, $m = 2, i = 1, 2$. We assume the setting of network providers are the same as in the last scenario.

Figure 5.6(a) shows that the auction algorithm also succeeds in balancing the requests and offers with this new utility function. We repeat the download for 100 times using both network interfaces and take the average value of page load time for each slicing scheme. As Figure 5.6(b) indicates, the delay decreases if the service provider requests more bandwidth for its users.

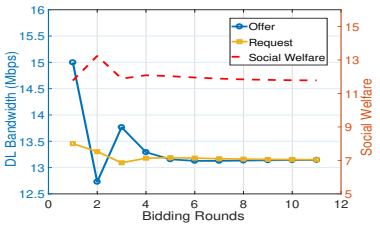
We also investigate how the proposed system is able to tackle with device mobility. Keeping $w_2^s = 5$, we change the location of the second smartphone so that its LTE signal strength falls to around -115 dB from -95 dB. (The WLAN signal strength is less impacted.) In this case, the user can report its signal strength to the service provider, reflecting in an updated β_{12} parameter in the formulation. Then the service provider can call a new auction to adjust its bids. Figure 5.6(c) compares the auction results before (Case 1) and after (Case 2) the movement, considering which the service provider requests more LTE bandwidth, and allocates a larger portion of it to its second user. In Figure 5.6(d), we measure the performance of this new allocation, in comparison of an average allocation, where two smartphones still acquire the same portion of resources. It indicates that there is an improvement on the load time of the second user. The negative impact of worse signal



(a) Competition between services.



(b) RAN costs and social welfare.



(c) Convergence of an auction.

	CPU (%) WLAN	LTE	Memory (MB) WLAN	LTE
w/o Agent	27.7	21.9	106.5	194.2
Idle	27.7	21.9	159.7	234.7
Busy	38.2	25.5	159.7	234.7

(d) CPU and memory costs.

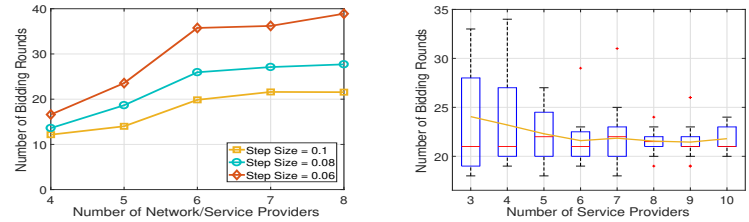
Figure 5.7: (a) Performance of two different services under their competition. (b) Cost and social welfare comparisons between the optimal and average allocations. (c) Number of bidding rounds until convergence. (d) CPU and memory consumption of SD-RANs.

is not totally eliminated, because the service provider needs to pay for the extra resource allocated. However, it is able to find a balance and achieve the optimal social welfare.

Multiple Slices. The proposed design also handles competitions among service providers. In this scenario, we run the two services above simultaneously. Figure 5.7(a) shows the performance of two services under different combinations of w_1^s and w_2^s. The service provider with higher purchase willingness is able to achieve better performance over the other one. We compare these results with the case in which slicing is not applied, e.g., the WLAN applies no queueing policies, and the LTE eNodeB allocates the same number of resource blocks to every user. We examine cases in which the eNodeB offers 20%, 60% and 100% of resource blocks. These plans lead to different costs as well, as depicted in Figure 5.7(b). In each case, either great performance degradation or significant additional cost incurs, and the resources allocated to two services are also severely imbalanced. All these factors lead to a worse (and even negative) social welfare than our optimal result.

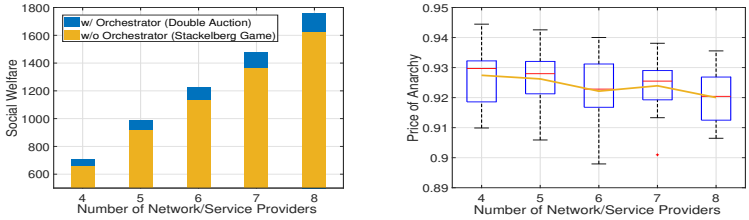
The time required for convergence does not dramatically increase with more bidders. Figure 5.7(c) shows the procedure of an auction with $w_1^s = 15$ and $w_2^s = 5$. Here the social welfare appears to decrease with time, because the constraints are not yet satisfied. The final result is still optimal. The performance of larger scale auctions will be further analyzed in the next subsection.

In Figure 5.7(d) we measure the CPU usage (of two cores) and memory consumption



(a) Convergence speed of different scales.

(b) Range and average of convergence speed.



(c) Social welfare of different architectures.

(d) PoA without an orchestrator.

Figure 5.8: (a) The number of bidding rounds required for convergence when the network scales up. (b) Box plots and average values of auctions among 8 network providers and different number of service providers. (c) Social welfare of proposed architecture where an orchestrator holds Double Auctions and another architecture where providers compete as a Stackelberg game. (d) Price of Anarchy in slicing games without an orchestrator.

of SD-RAN components. The first column shows a baseline, the consumption of the original SD-RAN controller (EmPOWER and SDN controller for WLAN, FlexRAN and OpenAirInterface EPC for LTE) without the deployment of our agents. In the second column, the Slicing Agents are deployed. When not processing auctions, no additional CPU resource is required. Only a small extra portion of memory is occupied. In the third column we keep initiating auctions with an interval of 1 second, therefore the Slicing Agents are busy bidding and implementing the slicing schemes, leading to larger while still affordable CPU usage. From the results shown in the table, the Slicing Agent is lightweight and does not exert heavy extra burden on the SD-RAN controller.

5.4.3 Large-Scale Simulation Results

Scalability. Having verified our design and implementation in the small-scale testbed, we now run simulations of larger network topology to guarantee that the performance of proposed design will not degrade when the network scaling up. More specifically, we consider multiple RANs and services in a $100m \times 100m$ area. Among K RANs, the

first two are LTE while remaining are WLAN providers each with an Access Point. Each of M different service providers has I users with dual-connectivity of LTE and WLAN. Entities above are distributed uniformly in this area. A user's LTE provider is randomly assigned with uniform probability, and it connects to its nearest WLAN access point. Each LTE network covers the whole area with $\beta_{ki} = 1$, while WLAN's β_{ki} is proportional to the spectral efficiency of Shannon formula, following Rayleigh fading depending on the distance between the user and the Access Point. All other parameters (e.g., w_k^n, w_m^s, α) and utility/cost functions are identical to the testbed, except that we multiply each of them with a random factor uniformly distributed in $[0.9, 1.1]$, and enlarge the capacity by I times consistent with the increasing amount of users.

We investigate the impact of network scale by observing the speed of convergence with different numbers of network and service providers. Figure 5.8(a) depicts the number of bidding rounds until convergence with up to 8 network providers and 8 services (each has 10 users). It does not grow dramatically with more providers participating the auction. Besides, the convergence speed can be adjusted by setting proper step size of gradient descent at the orchestrator.

Noticing that the marginal increase of bidding rounds becomes even slighter with more providers, we investigate it further and have Figure 5.8(b) changing the number of service providers (and users) while keeping 8 network providers. Both variance and average values are larger when there are fewer services, because users sparse in the area are more likely to result in unbalanced resource requests to each RAN, which need more iterations to converge. Due to features shown above, our approach has good scalability.

PoA. In Figure 5.8(c) and 5.8(d), we investigate the social welfare improvement compared with the Stackelberg game model without an orchestrator, as stated in the previous section. we plot the box plot and the average values of PoA in different topology, indicating an improvement of social welfare between 7% and 10% in most cases.

5.5 Related Work

Slicing in Heterogeneous RANs. Among SD-RAN approaches above, [146] aims at having centralized control on not only WiFi access points but also LTE eNodeB. Similarly, architectures of applying control and slicing over heterogeneous RANs are discussed in [32] and [27]. More specifically, [73] and [6] address the control architecture and resource allocation problem across LTE and WLAN. [91] discusses the slicing problem in heterogeneous cellular networks. Most of these works adopt centralized management, ignoring

interactions and competitions occur among RANs owned by different parties. In our work we utilize game theoretic modeling and mechanism design to deal with self-interested parties.

Game Theory in Slicing and Resource Allocation. Game theory [122] and mechanism design [121] has been widely applied to slicing and resource allocation of wireless networks [31, 58]. Congestion games and Price of Anarchy (PoA) [149] have been analyzed for network slicing [41]. The authors in [189, 190] design combinatorial auctions for efficient spectrum resource allocation. [182] proposes truthful auctions to enforce cooperation among wireless relay nodes in a network. The authors in [52] propose a share-constrained proportional allocation for network slicing games. Compared with existing works, we take more realistic factors into consideration at the same time, including heterogeneous RATs, services and multi-connectivity of users.

Mobile Data Offloading. Another possible way to make better use of multiple RATs for a cellular network operator is offloading its traffic to third-party owned WiFi access points [90]. Both centralized algorithms [95, 136] and game theory models [89, 128, 66] are developed towards this approach. Mobile data offloading schemes merely consider interaction between two specific RATs, with the cellular operator as the game leader. Our proposed architecture is not RAT-dependent, and demonstrates the advantages of introducing an orchestrator taking the leader role instead of one of the network providers. This ensures that any single network provider cannot monopolize the market.

5.6 Summary

In this chapter, we have proposed a new architecture for resource slicing across multiple selfish network providers using diverse technologies. Our proposed double auction mechanism guarantees convergence to optimal social welfare in finite time. Our central Slicing Orchestrator enables a unified resource abstraction to compare and compose resources exposed by diverse RAN technologies. We have demonstrated the feasibility of our architecture by deploying our orchestrator along with open source RAN slicing systems such as EmPOWER and FlexRAN.

Chapter 6

Local Intelligence: Learning-enabled Protocol-Independent Packet Classification

Security threats arising in massively connected IoT devices have attracted wide attention. It is necessary to equip IoT gateways with firewalls to prevent hacked devices from infecting a larger amount of network nodes. The match-action mechanism of SDN provides the means to differentiate malicious traffic flows from normal ones. However, vulnerabilities of IoT devices and heterogeneous protocols coexisting in the same network challenge the capability of SDN protocols such as OpenFlow. Despite the efforts of distributed control plane described in the previous chapters, programmability and intelligence are also required at the data plane to overcome these challenges.

In this chapter, we leverage the high level of data plane programmability brought by the P4 language and design a novel two-stage deep learning method for attack detection tailored to that particular language. Our method is able to generate flow rules that match a small number of header fields from arbitrary protocols while maintaining high performance of attack detection. Evaluations using network traces of different IoT protocols show significant benefits in accuracy, efficiency and universality over state-of-the-art methods.

6.1 Introduction

Internet of Things (IoT) interconnects a multitude of devices interfacing with the physical world as sensors and actuators, facilitating their communication towards accomplishing

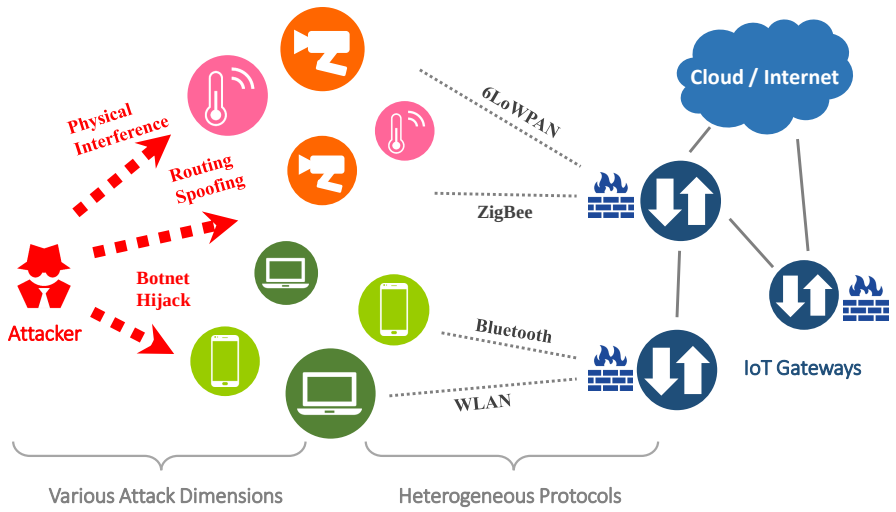


Figure 6.1: Firewalls deployed at IoT gateways targeting various types of attacks in heterogeneous protocols.

assigned tasks. In such networks with massively interconnected devices, security is a major concern. A large amount of insecure IoT devices have become targets of botnet attacks [80], leading to some of the most potent DDoS attacks in history. IoT devices are vulnerable to more types of attacks compared with other devices [8], such as network attacks in different protocols (e.g., RFID, Zigbee, 6LoWPAN) and even physical attacks. Therefore, it has been a big challenge to guarantee the security of an IoT network.

Traditional methods to secure an IoT device require the deployment of *physical* and *application layer* protection in it, e.g., by strengthening the authentication and encryption during data transmission. However, such approaches usually involve firmware and even hardware modifications, taking a relatively long time period. Devices in which security policies are not updated in time will increase the risk of being hacked and becoming sources of infection to other devices. To prevent malware from spreading, *network layer* security approaches are also necessary. For example, firewalls can be deployed at IoT gateways, monitoring and separating malicious from normal traffic, as depicted in Figure 6.1.

SDN provides a flexible framework for network management and is widely adopted in IoT networks. This flexibility can be exploited for the development and dynamic reconfiguration of network layer security mechanisms. By separating control and data planes, SDN protocols such as OpenFlow [103] make it possible to develop such mechanisms in a logically centralized and programmable manner. OpenFlow-enabled switches process incoming packets through match-action flow rules received from the controller checking

specific header fields (e.g., MAC and IP addresses, TCP port, etc.) and performing actions such as forwarding or dropping accordingly.

A firewall can be developed by generating flow rules through *machine learning* algorithms, which have been demonstrated as a promising method for identifying attacks from even unknown or encrypted traffic flows [99]. However, this method presents several limitations. Specifically:

1. *Limitations in Learning Models*. The training features used by the machine learning algorithm are often the specific header fields of the packet. However, heterogeneous IoT protocols may have distinct packet header structures, leading to a problem that the feature extraction process and even the whole learning algorithm should be specifically *redesigned* for every different protocol. Besides, the *manual* feature extraction adds difficulty to achieve optimal performance.

2. *Limitations in OpenFlow*. The match fields of OpenFlow are *predefined* and *fixed*. Many IoT headers cannot be parsed by it, e.g., compressed IPv6 headers in 6LoWPAN packets, or application layer protocols such as MQTT and RESTful API. As a result, no proper flow rules can be created in these cases. Although OpenFlow can be extended with user-defined headers by OpenFlow Extensible Match (OXM), it has limited functionality and hardware support in the above scenarios.

P4 language [24] provides possible solutions to the above challenges. Unlike OpenFlow which focuses on the control plane (i.e., the controller), P4 makes the data plane (i.e., the switches) programmable as well. Specifically, the packet headers are customizable by operators with the position and width provided, and table lookup can be conducted on these newly defined headers by the switches. This feature is especially meaningful in IoT scenarios, where support of different IoT protocols can be added by defining their headers [172].

Motivated by the above, we propose a new framework for IoT security and a corresponding learning algorithm which take advantage of the P4 language. Figure 6.2 illustrates its differences compared with the existing OpenFlow-based methods. The proposed method operates in two stages. In Stage 1, a learning algorithm trains *a dilated Convolutional Neural Network (Dilated CNN)* with raw packet bytes, skipping the step of manual feature extraction. In Stage 2, *a proper set of header field definitions* is inferred from the trained neural network, based on which flow rules for blocking traffic (dropping packets) are generated and installed in the IoT gateway (data plane switch). This method is applicable to heterogeneous IoT protocols. Besides, it is designed to take the constraints of

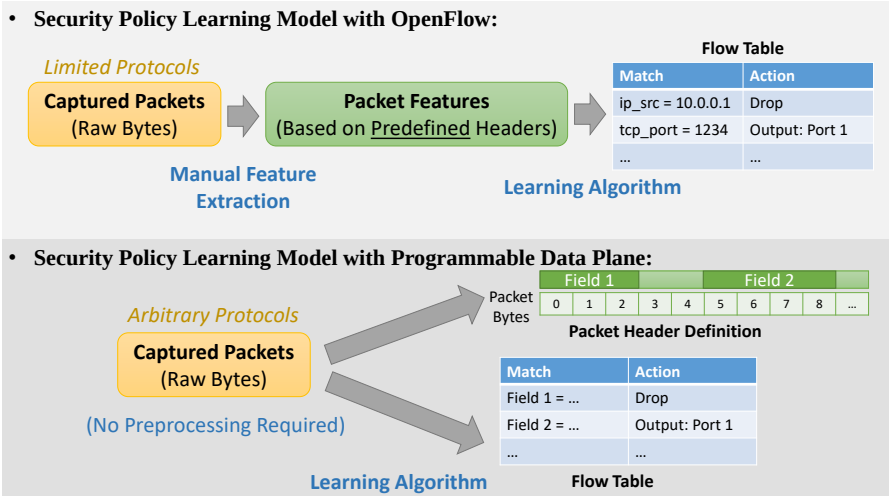


Figure 6.2: The learning process based on OpenFlow method and P4 language.

switch memory cost and packet processing speed into consideration, realizing a trade-off between accuracy and efficiency.

The contributions of this work can be summarized as follows:

- *IoT Security Framework*. We propose a new framework for securing IoT networks and devices. Taking advantages of the programmable data plane of P4 language, we aim at developing a universal, highly accurate and efficient solution to identify malicious traffic flows of multiple IoT protocols.

- *Learning Algorithm (Stage 1)*. We propose a learning algorithm that trains a dilated Convolutional Neural Network (CNN) with raw packet bytes to set up a traffic classifier. This approach skips the step of manual feature extraction of OpenFlow based methods and thus requires minimum data preprocessing.

- *Header Field Definition (Stage 2)*. We develop a method for converting the abstract features learned in the trained CNN into a particular set of header fields, so that a proper set of flow rules can be installed at the IoT gateway. This way, the classification can be realized as a switch function at the IoT gateway for lower memory cost and faster processing speed.

- *Experimental Datasets*. We conduct experiments to create our own new datasets of IoT traffic and multiple types of attacks. With them as well as publicly available datasets, we evaluate the performance of the proposed framework and algorithm in

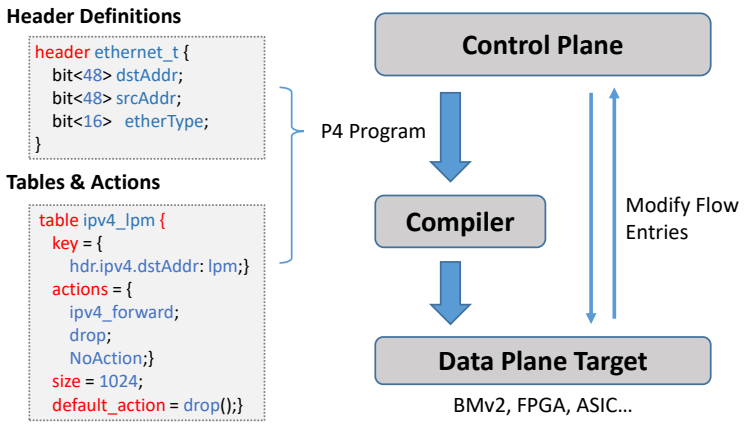


Figure 6.3: The protocol independence and reconfigurability of P4 language.

all aspects. The results show that our method makes proper choices of header fields achieving a better attack (intrusion) detection accuracy level than state-of-the-art OpenFlow based methods (performance) while being also able to handle heterogeneous IoT protocols (universality). At the same time, the line speed of packet processing is maintained (efficiency).

The rest of this chapter is organized as follows. Section 6.2 presents our IoT security framework. Sections 6.3 and 6.4 define and solve the header field definition problem based on the constructed CNN. The experimentation results are presented in Section 6.5, while we review our contribution compared to the related works in Section 6.6 and conclude our work in Section 6.7.

6.2 System Design

The proposed system has two components. The first part is the control plane, an SDN controller which is a software entity hosted in a node with sufficient computation capacity, e.g., a conventional cloud server or an edge cloud node. The second part is the data plane, which can be an IoT gateway. We consider the case that the IoT gateway is programmable by supporting the P4 language [24].

P4, or Programming Protocol-independent Packet Processors language is designed with reconfigurability and protocol independence. More specifically, the control plane (controller) is able to define how a data plane device (switch or gateway) parses a packet in a programmable and automated way (reconfigurability). First, one or more headers are

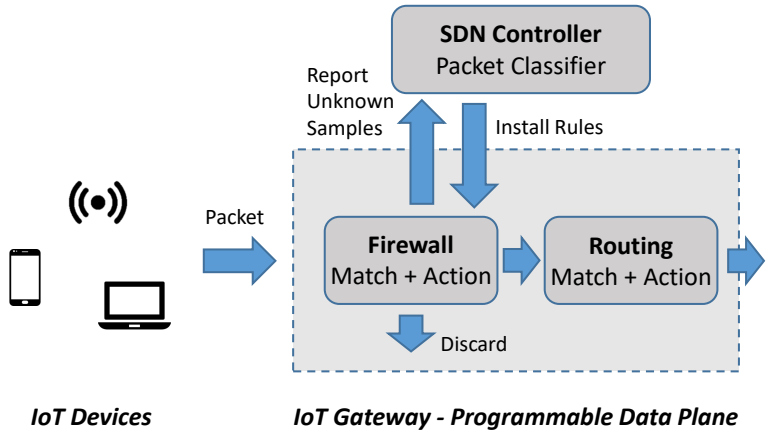


Figure 6.4: The control and data planes of the proposed framework, both programmable.

defined as a list of fields given their positions and widths in bits. Then, a parser works as a state machine to extract headers, following a series of match+action tables, which is similar to OpenFlow, except that header fields are not predefined (protocol independence). The whole workflow is depicted in Figure 6.3.

As shown in [172], a P4-enabled gateway is capable of serving IoT devices of heterogeneous network protocols. Our aim is to use the IoT gateway to identify malicious incoming traffic flows (e.g., from a hijacked IoT device) before they are routed to other domains and devices. We program the IoT gateway to execute a firewall function before the routing function. The firewall keeps a match+action table recording the features of known packets, which are the values of certain packet header fields. These fields will be checked inside the incoming packets and marked as normal or malicious based on the flow rules installed in the table. Normal packets will be passed to the routing function without modifications. On the other hand, actions can be defined to handle the malicious packets, e.g., blocking them or forwarding them to a honeypot. The flow rules are generated by the SDN controller, where a classifier is deployed and responsible for judging whether a flow is malicious or not. The controller is able to convert classification results into header field definitions and flow rules to install them in the firewall at the IoT gateway either reactively or proactively. The whole architecture is depicted in Figure 6.4.

Two key problems are required to be solved in the proposed system. First, we need to find algorithms for classifying packets with high accuracy. Second, P4 match-action tables should be generated, making classification a data plane function which achieves line-speed

packet processing. Besides, the solution we expect should be universal for heterogeneous IoT protocols, i.e., neither algorithm redesign nor protocol-dependent data preprocessing is required. In the next two sections, we will formally propose a formulation and a two-stage solution corresponding to the two key problems.

6.3 Problem Modeling

Assumptions. To model the two problems, we consider a scenario of one IoT network domain equipped with one gateway along with its SDN controller. This scenario can be easily extended into a multi-domain or multi-gateway topology by deploying the same solution in each domain. The gateway is responsible for identifying attacks among all traffic flows going through it, so that it can block current and future packets of the attack flow to prevent it from spreading, e.g., a hijacked device outside the domain infecting devices inside the domain, and vice versa. We assume that the security of the gateway itself and its SDN controller is not compromised.

Packet Classification. The features that can be used for classifying network traffic can be divided into two types, the *packet-level features* (e.g., IP address, TCP port, payload length), and the *flow statistics* (e.g., packet count, duration). The programmable data plane of P4 brings opportunities for defining new packet-level features, not restricted to OpenFlow's pre-defined collection, which is particularly important for the IoT network where heterogeneous protocols coexist. Besides, previous studies like [177] claim several other merits of learning directly from packet bytes, including the higher accuracy and the ability to classify encrypted traffic. Therefore, our work is focused on the packet-level features type of classification.

We use the first N bytes of the packet as features for classification. The packet can be thus represented by a vector $\mathbf{x} = (x_1, x_2, ..., x_N)$ where each element $x_i \in [0, 1]\ \forall i \leq N$ is a number converted from a byte. If the length of a packet is less than N, zero padding is applied. A classifier in the control plane should provide a function $F(\mathbf{x})$ judging the packet. We consider a binary output indicating whether the packet belongs to a normal traffic flow (i.e., $F(\mathbf{x}) = 0$) or a malicious one (i.e., $F(\mathbf{x}) = 1$). We can directly extend the method for multiple output values where the gateway takes different actions depending on the type of attack.

Header Fields Definition. While the control plane can check the bytes inside the packet one-by-one (and therefore compute the $F(\mathbf{x})$ value), such fine-grained classification may not be possible in the dataplane (IoT gateway) as this would require to install a huge

number of flow rules for all possible combinations of the N bytes. This is not feasible since it would lead to unrealistic memory cost and latency of lookup and processing packets. Taking advantage of P4, any *substring* of packet bytes can be regarded as a *header field* by the gateway, based on which flow rules will be generated. Therefore, we can effectively limit the number and length of flow rules, as well as the associated packet processing latency, by carefully defining a small number of packet byte substrings as header fields at the gateway.

Formally, we define the *Header Fields Definition* $H = \{h_k, k = 1, 2, ..., K\}$ which is a set of K substrings of bytes. [39] investigates various P4-enabled devices to show that the number of header fields has an impact on the performance. Therefore, we require that $K \leq K_{max}$ where $K_{max} << N$ so that to ensure a maximum memory cost and packet processing latency requirement is met. Each element $h_k = (a_k, a_k + L_k)$ is a substring starting from the a_k-th byte of the packet and ending at the $(a_k + L_k - 1)$-th byte, with its length L_k. These substrings should not overlap with each other, i.e., $a_{k+1} \geq a_k + L_k$ for any k, to avoid wasting memory. Unlike the traditional definition of header fields, each of which contains a specific type of information (e.g., network address or port number), we do not restrict that every substring defined by our method corresponds to a clear entity. Instead, *we aim for an algorithm capable in learning the meaning and importance of different substrings*, so that it can minimize the requirement of data preprocessing and be applicable to heterogeneous IoT protocols.

Based on the Header Fields Definition H, the information actually extracted from a packet $\mathbf{x}$ is $\mathbf{x}^H = (x_{a_1}, ..., x_{a_1+L_1-1}, ..., x_{a_K}, ..., x_{a_K+L_K-1})$. Therefore, the packet classification executed at the gateway follows a different function from $F(\mathbf{x})$, which depends on the definition of header fields H. We denote this function by $F^H(\mathbf{x}^H)$. Our goal is to find proper H and $F^H(\mathbf{x}^H)$ functions which satisfy the constraints mentioned above and are able to predict the packet classification at a high accuracy.

6.4 Algorithms and Learning Models

6.4.1 Overview

We solve the two problems specified in the previous section in *two stages* as depicted in Figure 6.5. In Stage 1, we build and train a *neural network (NN)* as the packet classifier. The training is based on raw packet bytes without considering the definition of header fields. This classifier will be deployed at the control plane. In Stage 2, we calculate

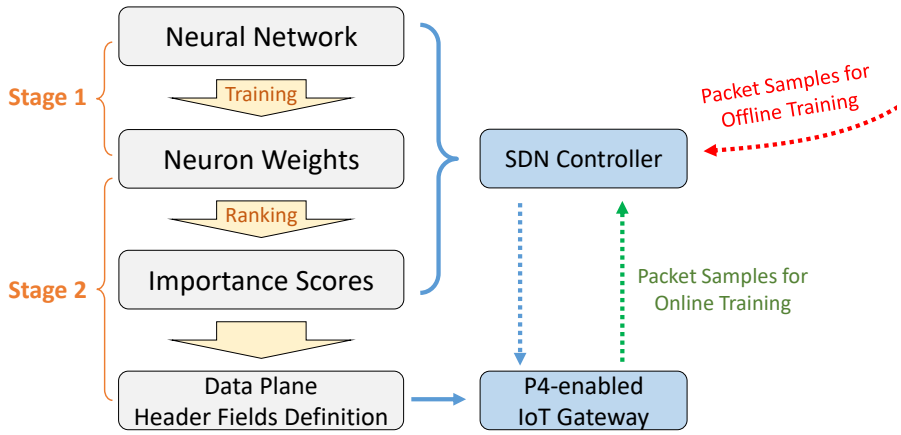


Figure 6.5: Illustration of the proposed two-stage learning approach. Packet classification is realized by the SDN control plane in Stage 1, followed by header field definition and implementation at the IoT gateway in Stage 2.

importance scores for each possible substring of packet bytes using the information from the trained NN (Neuron Weights), and then select non-overlapping substrings with largest scores to be included in the header field definition, which will be installed at the gateway (data plane) along with a match+action flow table.

Initially, the NN is trained offline with captured network traces. The trained NN is then deployed at the controller as the packet classifier. For the data plane, both proactive and reactive operating modes are available according to different scenarios. In the first mode, the controller installs both header field definitions and corresponding flow rules from training data proactively at the gateway. The gateway can therefore process new incoming packets at the line speed without forwarding them to the controller. In the second mode, the controller can proactively install header field definitions only, and install flow rules in a reactive way by replying to the gateway's queries. This mode incurs less memory cost in the gateway but increases latency due to the controller-gateway communication each time when the gateway receives unknown packets.

After the initial offline training, with the gateway sampling new packets and sending them to the controller, the two-stage process can be repeated in an online manner optionally, as long as the labels of packets can be acquired by the controller as well. The controller can also dynamically update the header field definition by compiling a new P4 program. All these operations are supported by the P4 specification.

6.4.2 Stage 1: Neural Network Structure

We apply methods of supervised learning for the packet classification. In particular, trained with a labeled dataset (i.e., large amount of packets marked as either malicious or normal), the classifier should be able to infer the expected output of a new input (the function $F(\mathbf{x})$). A Neural Network (NN) [59] is a computing system for supervised learning. It consists of several hidden layers and an output layer. Each layer is constructed by building blocks called neurons. For example, if we arrange the neurons of each layer in an array with index n (corresponding to the byte index of the packet), assign another index $i = 1, 2, ..., I_t$ for each layer t and take the packet byte vector $\boldsymbol{x}$ as the input, the output of a neuron in the first hidden layer is:

$$c_{ni}^{1} = f(\boldsymbol{w^{1;ni}} \cdot \boldsymbol{x} + \boldsymbol{b^{1;ni}}) \tag{6.1}$$

The output of each layer is the input of the next layer. For the neuron in the t-th hidden layer ($t > 1$), the output is:

$$c_{ni}^{t} = f(\boldsymbol{w^{t;ni}} \cdot \boldsymbol{c^{t-1}} + \boldsymbol{b^{t;ni}}) \tag{6.2}$$

where $\boldsymbol{w^{t;ni}}$ is a 2D vector of trainable weights, $\boldsymbol{b^{t;ni}}$ is a bias term, and f is a non-linear activation function.

Among various NN structures, we adopt 1D Dilated Convolutional Neural Network (Dilated CNN) [174] as depicted in Figure 6.6. In each hidden layer t, connections are local and dilated with step size 2^{t-1}. In other words, each neuron with index i only takes two rows of neurons with indices i and $i + 2^{t-1}$ in its last layer as the inputs. Neurons in the same layer share the same weight values. The output of the hidden layer neurons can be represented in the following way:

$$c_{ni}^{1} = f(w_{\alpha}^{1} \cdot x_n + w_{\beta}^{1} \cdot x_{n+1} + b^{1}) \tag{6.3}$$

$$c_{ni}^{t} = f(\boldsymbol{w_{\alpha}^{t}} \cdot \boldsymbol{c_{n}^{t-1}} + \boldsymbol{w_{\beta}^{t}} \cdot \boldsymbol{c_{n+2^{t-1}}^{t-1}} + \boldsymbol{b^{t}}), \forall t > 1 \tag{6.4}$$

where $\boldsymbol{w_{\alpha}^{t}}$ and $\boldsymbol{w_{\beta}^{t}}$ are two 1D vectors of trainable weights.

This structure brings two major benefits. First, for any hidden layer neuron c_{ni}^{t}, its inputs are limited in the range between packet bytes x_n and x_{n+2^t-1}, which means that we can establish a correspondence between a neuron c_{ni}^{t} and a substring $(n, n + 2^t)$ following the denotation in the last section. Second, the neuron receptive field is 2^t, increasing exponentially with the network depth. With T hidden layers, we can find neurons corre-

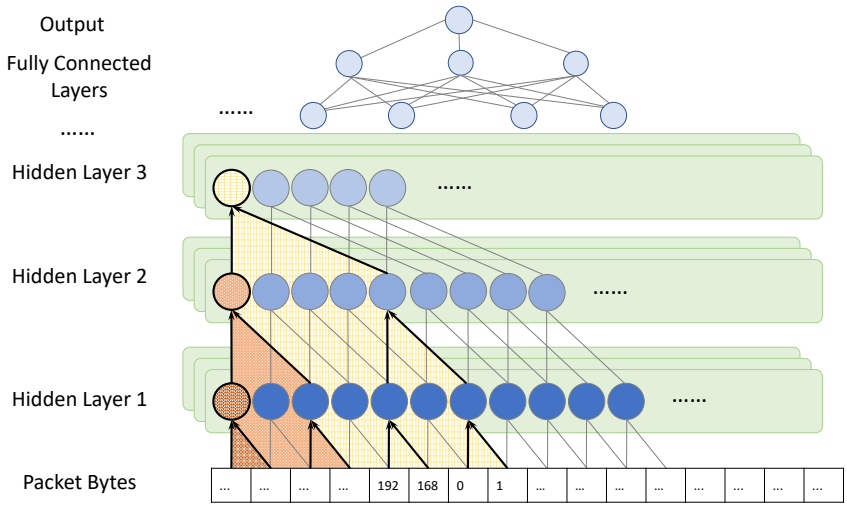


Figure 6.6: Structure of the dilated convolutional neural network (Dilated CNN) for packet classification.

sponding to any potential header field of length $2, 4, 8, ...$, up to 2^T bytes. In other words, with a limited amount of layers, we are able to cover a wider range of packet substrings. This is beneficial in both representing the packet structure better and training the neural network more efficiently. After convolutional layers, we have fully-connected layers, the last of which has a single neuron taking the weighted sum of the last hidden layer outputs as the final result. This structure can be easily extended to multi-class classification, as long as we set up more neurons in the output layer.

6.4.3 Stage 2: Header Field Definition

In the next stage, we adopt a neural network pruning [186] technique to the trained network. Pruning compresses the neural network by reducing the number of neurons. With smaller memory and calculating costs, pruning facilitates the processing of NN in IoT scenarios [175], where the capacity of devices may be limited. However, besides this benefit, our main purpose is to deduct an optimal set of header field definition based on the results of pruning, therefore enabling the line-speed packet processing in a P4-enabled gateway.

Pruning leads to an *importance score* of each neuron. Neurons with higher importance scores play a more crucial role in the classification. According to [186], we apply the Inf-FS [148] algorithm to calculate the importance scores of neurons in the last hidden layer. Then, the importance scores are calculated for the remaining layers in a backpropagation manner.

Leveraging the one-to-one correspondence between neurons and header fields in the proposed CNN structure, we extend the notion of importance score from neurons to header fields. Unlike [186] that suggests to greedily select neurons with highest importance scores, our problem has additional constraints, e.g., that the header fields should not overlap with each other. Therefore, we propose a new problem formulation.

The input of the problem includes the importance scores of all neurons in each hidden layer t. We denote the importance score of neuron c_{ni}^t by s_{ni}^t. By summing these values, we denote the importance score of a potential header field $(n, n+2^t)$ by $S_n = \sum_i s_{ni}^t$. Then, we obtain the following optimization problem:

$$\max_{\boldsymbol{y}} \sum_{n=1}^{N^t} y_n * S_n \tag{6.5}$$

$$s.t. \sum_{n=1}^{N^t} y_n \leq K_{max} \tag{6.6}$$

$$y_n * y_{n+j} = 0, \quad \forall n < N^t, j < L \tag{6.7}$$

$$L = 2^t, \quad N^t = N - L + 1 \tag{6.8}$$

where $\boldsymbol{y} = (y_1, y_2, ..., y_{N^t})$ is the vector of variables to optimize, representing all possible substrings of length 2^t in the first N bytes of the packet. The binary element y_n indicates whether to select substring $(n, n{+}2^t)$ in the header field definition ($y_n = 1$) or not ($y_n = 0$).

To solve this problem, we propose to use *Dynamic Programming* [19]. A *Bellman equation* can be easily defined based on two states; K as the amount of selected header fields and n_0 as the starting byte of the latest selected header field. We then have the following equations:

$$V(1, n_0) = S_{n_0}, \qquad \forall n_0 \leq N^t$$

$$V(K, n_0) = \max_{n+L \leq n_0} V(K-1, n) + S_{n_0}, \qquad \forall n_0 \leq N^t, K > 1$$

Based on the above equations, any $V(K, n_0)$ value can be calculated by *recursion*. The maximum of our objective function is therefore $\max_{n_0 \leq N^t} V(K_{max}, n_0)$. As described in Algorithm 5, an optimal set of header fields H can be selected with reasonable $O(K_{max} * N)$ time complexity.

The parameters K_{max} (i.e., maximum number of header fields) and $L = 2^t$ (i.e., length of one header field) can be determined according to the capacity of different types of P4-enabled devices [39]. In general, a tradeoff between accuracy and cost can be achieved by

Algorithm 5: Optimal Header Fields Selection

Input: $S_1, S_2, ..., S_{N^t}, K_{max}, L$

1 **for** $n_0 \leq N^t$ **do**

2 $\quad V(1, n_0) = S_{n_0}$;

3 $\quad H(1, n_0) = \{(n_0, n_0 + L)\}$;

4 **end**

5 **for** $K = 2, 3, ..., K_{max}$ **do**

6 $\quad$ **for** $n_0 \leq N^t$ **do**

7 $\quad\quad n^* = \arg\max_{n+L \leq n_0} V(K-1, n)$;

8 $\quad\quad V(K, n_0) = V(K-1, n^*) + S_{n_0}$;

9 $\quad\quad H(K, n_0) = H(K-1, n^*) \cup \{(n_0, n_0 + L)\}$;

10 $\quad$ **end**

11 **end**

12 $n^* = \arg\max_{n \leq N^t} V(K_{max}, n)$;

Output: $H = H(K_{max}, n^*)$

adjusting these parameters. With fewer or shorter header fields, some different traffic flows may be regarded as the same one by the gateway, negatively affecting the classification accuracy. With more or longer header fields, however, it takes larger memory cost to store flow rules, and may slow down the packet processing in some implementations. In the next section, we will evaluate the exact impact of these parameters on different performance metrics.

6.5 Evaluation Results

To demonstrate the benefits of our P4-based IoT security approach, we perform evaluations using various real traffic datasets. We begin with presenting the datasets and algorithms that will be later used to generate the evaluation results.

6.5.1 Setup

First, we use the following two publicly-available datasets of IoT network traffic:

- *ISCX Botnet 2014 Dataset [15].* This is a collection of botnet traffic traces from multiple well-known datasets. The types of traffic are mainly HTTP, P2P and IRC. This dataset is already divided into the training set and test set. The test set has more diversity than the training set, in order to evaluate the detection of unknown attacks. It is originally gathered for statistics-based classification and contains a

huge amount of packets, therefore we sample 10% of the packets from each flow for packet-level training. We also randomly modify the IP fields because all malicious flows are remapped to fixed IP addresses in the original data.

- *CICAAGM Android Dataset [87].* This publicly available dataset captures the traffic of Android applications in real smartphones, including 250 adware, 150 malware and 1500 benign applications. Besides HTTP, there are also massive HTTPS traces, a large portion of which is SSL/TLS-encrypted. The raw packet bytes are available through PCAP files. We sample 1000 successive packets from each class of the trace for packet-level training and testing.

We also make our *own* efforts to create two *new* datasets using network simulators and real IoT devices we deploy, containing unique threats to IoT devices. These datasets contain protocols that OpenFlow cannot handle. On the contrary, we will demonstrate that P4 and our algorithm work well on them.

- *Cooja Network Simulator Dataset.* [88, 101] analyze different types of attacks in 6LoWPAN networks through the RPL routing protocol with the help of Contiki operating system and its Cooja simulator. Adopting similar methods, we run simulations of 10-node IoT networks with random topologies, and set up a malicious node conducting Version Number Attack and Increased Rank Attack. We collect packet bytes of both malicious and normal traffic flows to generate our dataset.

- *Waspmote IoT Sensor Dataset.* We also create a new dataset with measurements on real IoT devices (not simulator) we deploy. Specifically, we install temperature, humidity and luminosity sensors on a Waspmote [97] Smart Cities Pro sensor board. It periodically sends 802.15.4 frames to the gateway containing sensor data. If the electrical connection from a sensor to the board is impeded, the device will still send packets in the same format but with the wrong values. This is indeed categorized as a physical attack on sensors rather than network attack. However, we will demonstrate that our method is also effective in detecting such unconventional attacks.

In each dataset except the first one which has already been split, we randomly pick 80% of the samples for training, and the remaining 20% for testing. We implement several state-of-the-art algorithms and make comparisons with our method. In particular:

- *Proposed P4-based Method.* In Stage 1, we build the deep neural network of the proposed structure with 4 convolutional layers each with 64 filters and the ReLU

activation function [114], followed by two fully-connected layers with 100 and 50 neurons. At each hidden layer, a 0.05 dropout rate is set to avoid over-fitting. We keep the hyperameters unchanged when training with different datasets. In Stage 2, we produce the header field definition and install the corresponding flow rules to the IoT gateway.

- *OpenFlow-based Methods*. As a comparison, we consider classification methods based on OpenFlow protocol, representing SDN without programmable data plane. We limit the features of classification within the predefined header fields of MAC, IP, TCP and UDP protocols according to the OpenFlow specification. As stated in [115], multiple machine learning techniques can be applied to these features, among which we choose two representative methods, Decision Tree (DT) and Support Vector Machine (SVM).

- *1D Convolutional Neural Networks (1D-CNN)*. We also consider other deep learning approaches for packet classification which (similar to our method) take packet bytes rather than some specific header fields as the input. We implement two 1D-CNN imitating the structures and hyperameters in [177, 99], denoted by CNN-1 and CNN-2. These CNNs provide the same type of output as our Stage 1 output. However, they are not capable in producing a header field definition as Stage 2 of our method does. In other words, the classification cannot be executed as a switch function for line-speed packet processing.

We implement DT and SVM models using scikit-learn [129] library, and implement NNs in TensorFlow [3]. To verify the header field definition calculated by our algorithm, we also conduct emulations with Mininet [86] and P4 behavioral model software switch (BMv2) [35]. The experiments are conducted on a desktop computer with Intel Core i7-7700 Processor, 16 GB RAM and GeForce GTX 1060 graphics card.

We evaluate the performance of the classification algorithms using as metric not only accuracy, but also precision and recall. We denote the number of correctly identified malicious packets by TP and incorrectly identified ones by FP. We denote the number of correctly identified normal packets by TN and incorrectly identified ones by FN. The metrics are calculated as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.9}$$

$$precision = \frac{TP}{TP + FP}, \quad recall = \frac{TP}{TP + FN} \tag{6.10}$$

Considering that the datasets have uneven class distributions (where malicious samples account for around 30% in each dataset, except the Cooja dataset with around 10% malicious samples), we also calculate the F1 score defined as the harmonic mean of precision and recall:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{6.11}$$

6.5.2 Classification (Stage 1) Performance

In this subsection, we evaluate Stage 1 of the proposed method. We compare the classification performance of the proposed dilated convolutional neural network with the two other CNN structures as well as with the DT and SVM OpenFlow-based methods.

Method	Accuracy	Precision	Recall	F_1
DT	0.790	0.694	0.659	0.676
SVM	0.773	0.706	0.544	0.615
CNN 1	0.907	0.897	0.816	0.854
CNN 2	0.909	0.903	0.816	0.857
Proposed	**0.911**	0.904	0.822	**0.861**

Table 6.1: Performance metrics of the Dilated CNN on ISCX dataset.

ISCX Botnet. We train and test all the algorithms on the ISCX dataset. Table 6.1 shows the accuracy of each algorithm. Compared with methods based on OpenFlow headers, the CNNs (including our approach) that take raw bytes as the input have significantly better performance. We also find that CNN-based methods outperform other algorithms in both precision and recall rates, leading to higher F1 score.

Method	Accuracy	Precision	Recall	F_1
DT	0.890	0.833	0.771	0.801
SVM	0.895	0.933	0.646	0.780
CNN 1	0.882	0.833	0.738	0.782
CNN 2	0.898	0.870	0.760	0.811
Proposed	**0.908**	0.927	0.736	**0.820**

Table 6.2: Performance metrics of the Dilated CNN on CICAAFM dataset.

CICAAGM dataset. We perform similar training and testing on the CICAAGM Android dataset, which contains a larger diversity of traffic flows including SSL/TLS encrypted ones. The results are depicted in Table 6.2. Although the performance difference

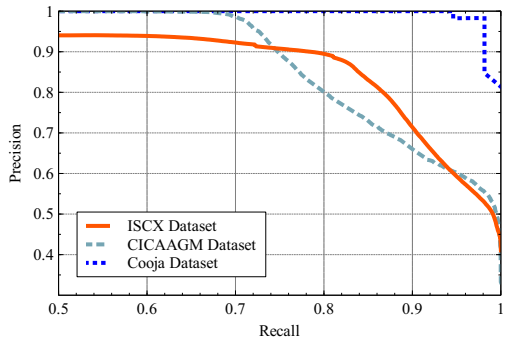


Figure 6.7: The precision-recall curve on different datasets.

is not as large as in the ISCX dataset, our algorithm still achieves highest accuracy than the other algorithms. We note that while the SVM OpenFlow-based method reaches higher precision, it severely degrades the recall value, leading to a lower F1 score.

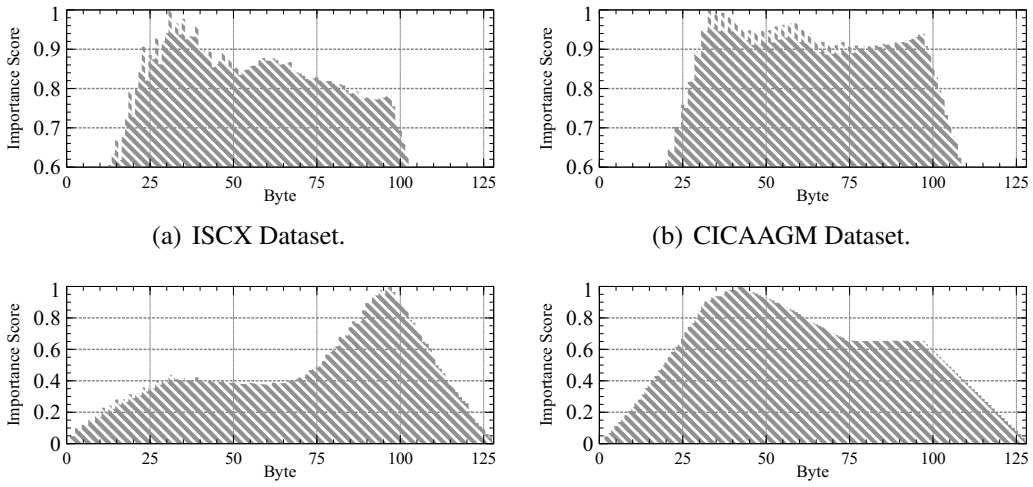
Dataset	Cooja		Waspmote	
Method	Accuracy	F_1	Accuracy	F_1
CNN 1	0.998	0.991	0.995	0.993
CNN 2	0.994	0.971	0.998	0.996
Proposed	**0.995**	**0.973**	**1.00**	**1.00**

Table 6.3: Performance metrics of the Dilated CNN on other datasets.

Cooja dataset and Waspmote dataset. The Cooja and Waspmote datasets are relatively simple, each with smaller amount of samples and only two types of attacks. However, the former contains compressed 6LoWPAN headers, and the latter has abnormalities which can only be identified from the packet payload rather than the headers. Therefore, *the packets are not readable and can no longer be classified by the OpenFlow-based methods (i.e., DT and SVM).*

As shown in Table 6.3, all three CNNs are capable of identifying the RPL routing attacks and sensor physical attacks with accuracy higher than 99%. The performance metrics of different methods are generally at the same level. Except being slightly worse than the CNN-1 in the Cooja dataset, our proposed network has superior performance in accuracy and F1 score. Especially, it achieves perfect predictions in the Waspmote dataset.

Performance Tradeoff. We are also interested in the tradeoff between the different performance metrics. In some scenarios, the false alarms must be controlled, otherwise system failures can happen. To achieve this, we can apply a threshold to the CNN output.



(a) ISCX Dataset. (b) CICAAGM Dataset.

(c) Cooja Dataset. (d) Waspmote Dataset.

Figure 6.8: Distributions of single-byte importance scores in different datasets.

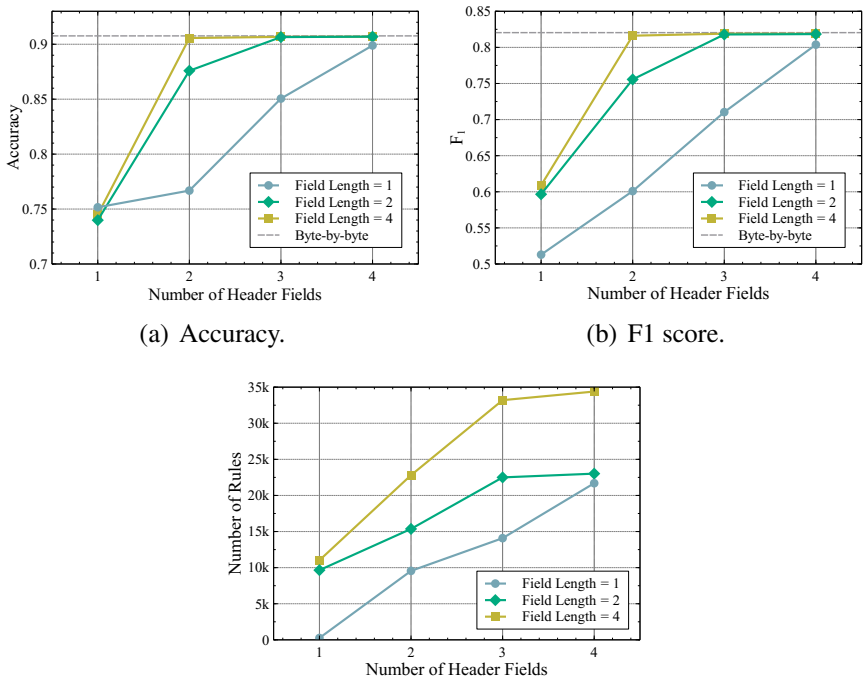
We depict the respective precision-recall curves for different thresholds in Figure 6.7 for all datasets except the Waspmote dataset where perfect predictions have been reached. We notice that in all datasets there is a space to increase the precision further at a cost of the recall.

Main Takeaways. (1) P4-based learning methods with packet bytes as the input can achieve better classification performance compared with OpenFlow-based learning methods that take as input predefined header fields. They can also handle heterogeneous protocols and application layer contents of packets, where OpenFlow-based methods are not applicable. (2) Our proposed Dilated CNN structure achieves similar or better performance than other state-of-the-art CNN approaches that take the same input (packet bytes).

6.5.3 Header Field Definition (Stage 2) Performance

The classification performance benefits in the previous subsection are important but not surprising. It was expected that taking packet bytes rather than predefined headers as input to the learning algorithm achieves superior classification performance as the classifier design space is larger. Still, the above results quantified the exact performance improvement we can achieve and verified the suitability of our proposed Dilated CNN structure compared to other CNN structures.

The main contribution of our work, however, lies on the implementation of the intrusion detection function directly inside the data plane (P4-enabled IoT gateway). This is important because it enables line-speed packet processing that is not available in the other



(a) Accuracy.

(b) F1 score.

(c) Memory cost.

Figure 6.9: Accuracy, precision and memory cost with different header fields selected in CICAAGM dataset.

learning methods like CNN-1 and CNN-2. To achieve this, Stage 2 of our learning method uses the trained Dilated CNN to define a particular set of packet byte substrings as header fields that will be used by the gateway to install flow rules. Therefore, matched packets will be directly handled by the gateway without requiring to be forwarded to the SDN controller or another remote firewall function. In the sequel, we elaborate on the header field definition and corresponding classification performance achieved by Stage 2 of our algorithm.

Profiles of Importance Scores. Following the procedure described in Section 6.4.3, we calculate the importance scores for all substrings of length $1, 2, 4, 8$ and 16 in the first $N = 128$ byte positions. For example, Figure 6.8 depicts the results of importance scores (after normalization) for every single byte.

The profiles of the datasets show different and complicated tendencies. However, there are also some intuitive results:

- *ISCX dataset* (with IP addresses masked). The algorithm highly scores both TCP/UDP fields and some positions in the application layer.

- *CICAAGM Android dataset*. The curve has three peaks in the IP address field, the

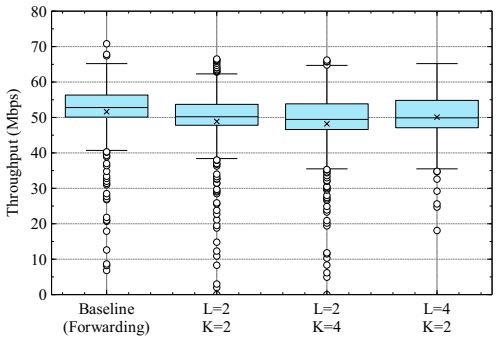


Figure 6.10: Throughputs with different header field definitions.

TCP port field and application layer. This distribution implies that the classifier makes predictions based on information from headers of multiple network layers, which is an advantage of adopting SDN and P4. For example, in the case where the packet payload is SSL/TLS encrypted, even if the classifier is not able to parse application-layer information, it is able to make predictions based on TCP/IP headers with a high accuracy. On the other hand, the application-layer headers reveal much information in those packets without encryption.

- *Cooja dataset*. High importance score is given to 97-th byte. It is reasonable because all attacks occur through DODAG Information Object (DIO) messages [88] of 96 bytes. The algorithm takes the packet length into account when making classification.

- *Waspmote dataset*. The algorithm successfully assigns highest importance scores to Byte 31, 32 and Byte 36, 37 in every 802.15.4 frame which store the the sensing data in question.

These distributions demonstrate that the importance scores calculated by our method successfully identify header fields that are crucial in classifying packets.

Impact of Header Fields on Accuracy. The proposed Dynamic Programming algorithm (Algorithm 5) will select as header fields the substrings of the packet bytes that have the highest importance scores. Taking the CICAAGM Android dataset as an example, Figure 6.9(a) and 6.9(b) show the accuracy and F1 score as we increase the number of header fields we match in the gateway node (K_{max} equal to 1, 2, 3 or 4) and for different header field lengths (L equal to 1, 2 or 4). The byte-to-byte approach corresponds to the packet classifier in Stage 1 of our method described in the previous subsection. Intuitively,

the performance improves with the number of header fields. According to the results, it is not necessary to have a large number of header fields. *With three 2-byte-long fields or two 4-byte-long fields, the classification is almost as accurate as the byte-to-byte approach*. The difference is around 0.1% in accuracy values.

Impact of Header Fields on Costs. Next, we examine the costs associated with the header field definition, measured by the number of flow rules stored in the gateway node. Since more rules lead to a larger memory occupancy and more queries to the control plane, we need to keep their number as low as possible. Figure 6.9(c) shows that the number of rules required for classification increases with both the length and the number of header fields selected. Therefore, *a tradeoff exists between the accuracy and cost*. The balance point can be achieved by adjusting the values of K_{max} and L parameters in our algorithm. Notice that although the number of all possible values of a header field increases exponentially with its length, the growth is not drastic in practice. From the evaluation results, the tendency is closer to a linear growth.

We need to emphasize that the proposed intrusion detection mechanism does not incur much additional costs in other aspects such as the network latency and throughput, because it only adds an one-time table lookup in the packet processing procedure. To verify this intuition, we create a virtual network with one BMv2 switch and two hosts using the Mininet emulation platform. We implement several sets of header field definitions and flow tables similar to the results in Figure 6.9. We use this virtual network to measure the maximum throughput achieved by our mechanism for different K and L choices and compare it with the baseline L2 forwarding mechanism that does not perform any intrusion detection. The results are depicted in Figure 6.10. We notice that the maximum throughput is reduced by less than 10% compared with the baseline, i.e., the line speed of packet processing is maintained. In the same scenario, we have another approach that forces packets to go through an application-layer single-thread analyzer based on Scapy [22] before being forwarded, which represents the case if adopting solutions similar as CNN-1 and CNN-2 in the last subsection. In this case, no larger throughput than 1 Mbps is achieved. Therefore, it is extremely beneficial to implement the intrusion detection as a switch function inside the IoT gateway with the help of the programmable data plane feature.

Optimal Selection of Header Fields. Last but not least, to demonstrate that the importance scores are proper metrics for the data plane definition, we compare the optimal selection of header fields in our algorithm with random selections. As shown in Table 6.4, with the same number of selected header fields, the performance of our algorithm is sig-

Method	Optimal		Random	
# of Fields	Accuracy	F_1	Accuracy	F_1
1	0.740	0.597	0.689	0.325
2	0.876	0.757	0.765	0.490
3	0.907	0.818	0.775	0.557
4	0.907	0.818	0.802	0.639

Table 6.4: Comparisons between the proposed algorithm and random selected header fields. (The length of each field is 2 byte in both cases.)

nificantly better, with more than 10% accuracy and around 20% more F1 score than the random selection.

Main Takeaways. A similar level of packet classification accuracy as the byte-to-byte approach can be achieved by merely matching a small number (two or three) of header fields appropriately selected based on the importance scores in the associated neural network. When implemented as a P4 switch function at the IoT gateway, this approach requires low memory and latency cost and incurs small throughput loss for table lookup (less than 10%, i.e., line speed is maintained), while alternative application-layer intrusion detection mechanisms would cause a multi-fold throughput reduction to achieve the same level of functionality.

6.6 Related Work

Security problems of IoT devices have attracted wide attention. [8] and [5] provide comprehensive surveys of IoT attacks and classify them into various types. New types of attacks different from traditional networks threat IoT security, including a variety of attack methods in IoT protocols such as Zigbee and 6LoWPAN [30, 134, 101], as well as physical attacks targeting the sensors and actuators [49, 159]. These works suggest adding authentication mechanisms to the devices. However, a network-level security solution is also necessary for preventing malware from spreading among vulnerable IoT devices, such as botnets [9]. Our firewall implementation at the IoT gateway complements the device-level authentication for a more powerful security guarantee.

Network-level security approaches can be grouped into two categories. The first category applies machine learning methods on specific packet headers [93]. For example, [116] applies learning on 6LoWPAN headers. Kalis [106] provides knowledge-driven solution detecting IoT attacks, while DÏoT [119] and IoT Sentinel [107] identify the IoT

device types by learning. Though these methods are effective, they usually require pre-knowledge from protocol definitions or device manufacturers. Due to the large diversity of IoT devices and protocols, we explore another direction leading to a more universal solution for heterogeneous IoT systems in case that such pre-knowledge is not available.

The second category classifies packets based on raw packet bytes rather than header fields. Machine learning methods, especially neural networks are also widely applied for it, such as [177, 99, 178]. These approaches have high accuracy and are not limited to specific protocol or device types. However, they can only be deployed in a remote server/host rather than a switch (IoT gateway). Therefore, packets cannot be processed at the line speed.

Our work focuses on combining the merits of the two approaches above, developing intrusion detection as a switch function at the IoT gateway and at the same time not relying on assumptions of device and protocol types. Benefiting from their programmable, flexible and efficient packet processing capabilities, recent developments in SDN make the implementation of such switch function possible. For example, Sensor OpenFlow [100] and SDN-Wise [51] extend OpenFlow protocol in this direction. Besides, there is an increasing research interest in deploying and managing P4 switches. [98] aims at aggregating sensor data from multiple packets by P4 header operations. [172] achieves multi-protocol switching of IoT services by deploying P4-enabled switches. Our proposed security framework is a similar approach which leverages P4 for IoT scenarios.

6.7 Summary

In this chapter, we studied new opportunities for enhancing security in the IoT network brought by the programmable data plane. Namely, we proposed a two-stage deep learning method based on P4 language that first trains a neural network as the packet classifier and in a later stage selects packet byte substrings as header fields and installs appropriate flow rules to realize intrusion detection functionality inside the IoT gateway. Evaluation results on publicly available and newly developed datasets of IoT scenarios demonstrated the performance benefits and universality of the proposed method compared with state-of-the-art OpenFlow-based methods. Importantly, the results verified that a more favorable tradeoff between detection accuracy, memory cost, latency and throughput can be achieved by the proposed method.

Chapter 7

Local Intelligence: Binarization Techniques towards Scalability

In this chapter, we continue to discuss the usage of the programmable data plane in the IoT security application. Apart from handling heterogeneous protocols, the local intelligence brought by the programmable data plane devices also has great potential as a scalable solution for efficient security-aware packet processing, especially when combined with the distributed control plane approach. Based on this concept, we extend the architecture proposed in the last chapter for scenarios of larger scales, which may contain multiple edge network domains. We implement another learning-based algorithm, the Binarized Neural Network (BNN) directly in the data plane for high-efficient packet classification. We also address the scalability issue by adopting a federated learning approach which is capable of training the learning model across multiple edge domains with small communication overheads. We develop a prototype using the P4 language and perform evaluations. The results demonstrate that a multi-fold improvement in latency and communication overheads can be achieved compared to state-of-the-art learning architectures.

7.1 Introduction

As discussed in the last chapter, machine learning algorithms such as neural networks are widely adopted for classifying incoming packets, which can be deployed at the network layer as a powerful tool to secure the IoT networks. Taking the values of packet header fields and flow statistics as input features, these algorithms are able to learn the pattern of attacks from collected network traces and make predictions for future inputs with high accuracy.

However, in the traditional network architecture, the data plane devices only execute simple functions such as specific packet header fields matching and table lookup. When an unknown packet arrives at a switch, it will be forwarded to a remote server or host where the learning algorithms run. The delay incurred makes it unlikely to process packets at a high speed. In addition, a large number of flow rules will be generated in this procedure and have to be stored in the switches, whose memory is usually limited and becomes another bottleneck [84].

The development of SDN and the programmable data plane concept bring new opportunities towards addressing the above challenges. SmartNIC products and P4 language [24] enhance the capability of the switch itself, which is now capable of offloading services that are traditionally run in remote servers with general (and powerful) CPUs [170].

Binarized Neural Network (BNN) [36] can be used to deploy machine-learning-based packet classification in the form of in-network services inside the switches. BNN compresses all the weights of a neural network into single bits, therefore significantly reducing the computation and memory requirement of performing the inference to a level that a data plane switch may afford. It also converts all computations (e.g., real-valued dot production and activation functions) into bitwise operations, which are supported by typical programmable data plane switches.

While the use of BNNs can expedite the inference process by enabling the offloading of it directly on the data plane switch level, there still exist challenges about the training process of these learning models. It is unclear how to train the BNNs in a scalable manner e.g., in large networks with many interconnected edge domains, many gateways and switches. When a new attack pattern appears only in specific domains, other gateways should also be informed, even if the attacker's packets do not go through them, so as to make more efficient training decisions in future. Meanwhile, the communication overheads either among gateways or between the gateways and the cloud being responsible for the training should also be considered. Even worse, it is possible that edge domains are controlled by multiple parties who do not want to share their network traces with others for training, since the information leak itself is another security threat.

Federated learning [104] is a technique suitable for online training in this scenario, which aggregates local weight updates from each gateway without asking their collected packets, and then calculates new model parameters for gateways. We explore a novel way of combining federated learning and BNN to set up a scalable packet classification architecture with high performance and low costs while preserving the privacy of network traces.

Specifically, we make the following contributions:

- We propose a learning framework for packet classification combining BNNs and federated learning achieving high accuracy with low memory and communication costs. To the best of our knowledge, this is the first work combining these concepts together.

- We design an architecture based on programmable network switches for providing security service to multi-party edge device owners while performing packet classification at the line speed of the switches and updating learning models in a scalable manner.

- We develop a prototype of the proposed architecture in P4 language and evaluate its performance and costs in a network testbed with real devices and traffic traces. We find that a multi-fold improvement in latency and communication overheads can be achieved compared to state-of-the-art learning architectures.

The remainder of this chapter is organized as follows. We describe the main challenges of the packet classification at the network edge and propose a system architecture in Section 7.2. In Section 7.3, we describe the learning model inference and training mechanisms, as well as the federated learning framework. Section 7.4 demonstrates how such architecture and mechanisms are implemented as a prototype and evaluates its performance. We discuss our contribution over related works in Section 7.5 and conclude the chapter in Section 7.6.

7.2 System Design

In this section, we describe the architecture design of the proposed system for network security. The system consists of a central cloud and several edge network domains. For each domain, there is a gateway node responsible for forwarding packets from and to the devices of that domain. It also performs packet classification to identify attacks from normal traffic flows. Each gateway is SDN-enabled with separated control and data plane i.e., an edge controller and a switch. Both planes are programmable. Previous works have shown the feasibility and benefits of this type of gateway design and implementation for edge networking scenarios [172]. In this work, we make a step further and propose specific mechanisms for effective packet classification achieving high accuracy with low memory

and communication costs. We first list a number of challenges we need to address before presenting the proposed mechanisms.

7.2.1 Challenges

A high-performance architecture for packet classification at the network edge has multiple requirements:

- *High Accuracy & Low False Alarm Rate*. The gateway should be capable to identify attacks from normal flows. Besides, the false alarms (normal packets incorrectly classified as attack packets) must be kept to a low rate, otherwise normal packets may be blocked and network functions will be hampered.

- *Line-Speed Packet Processing*. The gateway should perform the packet classification by itself instead of forwarding packets to a remote host or server and waiting for reply. This requires the classification algorithm (inference process of the learning algorithm) to be lightweight enough so that the gateway can run it locally in real time.

- *Model Updates*. An edge domain can be highly dynamic with new devices joining the network and new traffic flows generated over time. The gateway should be able to use the new network traces to improve the classification algorithm, i.e., re-train the model over time. The training task can be offloaded to the control plane or remote cloud server, but the updated model must be finally downloaded to the gateway data plane.

- *Scalability and Privacy*. It is common in an edge networking scenario that the amount of devices and domains is large. A solution can hardly scale up unless the communication overheads between the cloud server and gateways during training can be controlled in a reasonable manner. In addition, devices of different edge domains may belong to different owners who are not willing to share their network traces for training.

7.2.2 Design Choices

In order to meet all requirements above, we choose the binarized neural network (BNN) and federated learning as the main components of our architecture. We describe each component in the following, as depicted in Figure 7.1.

Gateway Data Plane (Programmable Switches). The data plane refers to a packet forwarding device with programmability such as P4-enabled switches, SmartNICs and FPGAs. A BNN is deployed in each gateway's data plane for classifying incoming packets. The data plane extracts certain bits from incoming packet's header as the BNN input and a binary output (i.e., attack or normal traffic) is acquired by a series of bitwise operations. After this inference process, the gateway performs ordinary packet forwarding for normal traffic and is able to send attack samples to the control plane if online training is active. *With both the classification and forwarding functions inside the data plane, line-speed packet processing can be achieved.*

Gateway Control Plane (Edge Controllers). Each gateway is managed by a separate edge controller with a general CPU or GPU. The controller may be deployed locally in the gateway or in another host within the same domain. The controller maintains a neural network with the same structure as in the data plane, except that the weights and activation functions are not binarized. This neural network is used for re-training the classification algorithm over time by performing backward propagation with the new network traces collected by the data plane. The controller also keeps an API writing weight values to the data plane, and an API communicating with the cloud server for federated learning. The detailed methods will be introduced in the next section.

Cloud Server. For scalable training of the classification algorithm, a federated learning technique [104] is deployed in the cloud server. The federated learning can be regarded as a service provided by the cloud, and each gateway can choose whether to subscribe to this service, decided by its owner. Each gateway subscribing to the service, after each epoch of local training, it sends the local updates to the cloud that acts as the aggregator. When the aggregator receives messages from all the gateways, it calculates the new model weights based on the local updates and broadcasts the new model weights to the gateways.

The procedures of BNN inference in the data plane, model training in the control plane and weight aggregation in the cloud as well as the implementation details of these mechanisms will be described in the next sections.

7.3 Problem Modeling and Algorithms

In this section, we propose a new packet classification problem formulation which takes multiple edge domains into consideration and describe how we adopt BNN and federated learning techniques to solve it.

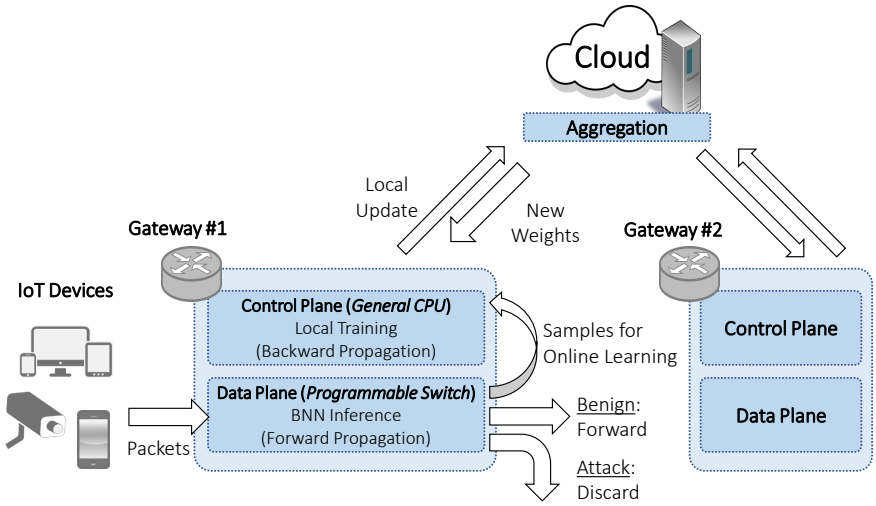


Figure 7.1: An architecture deploying BNN and federated learning for network security at the edge.

7.3.1 Problem Formulation

We consider a system of a cloud server c and N edge network domains. Each domain contains a gateway which is the pair of a data plane switch and its edge controller (collocated with the switch or hosted in a different device within the same domain). The set of all gateways is denoted by $\mathbf{N}$.

A data plane switch is able to parse headers of different protocols contained in a packet and determine where the packet should be forwarded (or blocked) according to specific header fields, which can be regarded as *packet-level features*. The switch may also use *flow-level features* such as the packet/byte count of a flow to make appropriate forwarding decisions. It is straightforward to represent both types of features by a bit string. Therefore, given a group of features supported by the gateway, we can concatenate them with a fixed sequence to get a 1D vector. Each element of the vector is binary, i.e., either -1 or $+1$. We denote this vector as x_0, which is the input for the packet classification.

The purpose of packet classification is to find a function $\hat{y} = f_n(x_0)$ at each gateway $n \in \mathbf{N}$, where $\hat{y}$ is a 1D binary vector indicating the prediction of the packet type. For example, as a simple case, $\hat{y}$ has only one binary element, taking value of $+1$ if the incoming packet belongs to a normal traffic flow, or -1 if it belongs to an attack.

Algorithm 6: Inference Process

Input : x_0: binary input sample
$W^b_{n,l}$: binary weights of layer l in gateway n's data plane
Output: y: binary prediction

1 **for** $l = 1 : L - 1$ **do**
2 $\quad x_l \leftarrow sign(XnorDotProduct(x_{l-1}, W^b_{n,l}))$
3 **end**
4 $y \leftarrow sign(XnorDotProduct(x_{L-1}, W^b_{n,L}))$

7.3.2 Inference: Binarized Neural Networks

To achieve line-speed packet processing, we require that an incoming packet is classified directly in the gateway instead of forwarded to the edge controller or any other remote server. In other words, each gateway n executes $f_n(x_0)$ in its data plane independently without help from either its edge controller or gateways of other domains.

Neural network is one of the most popular methods for packet classification. However, it requires a large amount of dot product operations on real-valued vectors, as well as activation functions which are usually non-linear. Originally designed for packet forwarding, most data plane devices do not support these operations. To overcome this difficulty, we deploy BNN [36] that has weights of only binary ($+1$ or -1) values and sign function as the activation function. More specifically, consider a neural network with L fully-connected layers. We denote the neuron weights of layer l by a 2D vector $W^b_{n,l}$ and denote the input of this layer by x_{l-1}. Then, the output of layer l is:

$$x_l = sign(x_{l-1} \cdot W^b_{n,l}) \tag{7.1}$$

If both x_{l-1} and $W^b_{n,l}$ are binary vectors, this operation is equivalent to the Hamming weight of two bit strings' XNOR. Similarly, the whole inference procedure of L layers is described in Algorithm 6. In the next section, we will demonstrate how we implement it completely in a programmable data plane device.

7.3.3 Training: Federated Learning Technique

To classify packets with high accuracy, a neural network needs to be trained in order to get optimal weights. Although BNN is efficient when performing the inference, it cannot be trained directly because gradients cannot be calculated from binary functions. We adopt a similar method as [36], which keeps the real-valued weights denoted by W_n. When

calculating the loss function by forward propagation, binary weights are used. However, during the backward propagation as the next step, real-valued gradients are calculated and applied for the weight update. In our approach, we store W_n and perform the backward propagation in the *edge controller* of the gateway n, leaving the data plane for binary forward propagation only. Besides this one-time training, it is also possible for the data plane to report the inference results of incoming packets to its controller in real time, so that training can be performed again over time in the controller to improve the classification accuracy.

[143] suggests that replacing the output layer with real-valued weights and activation functions during the forward propagation will positively impact the accuracy in practice. Such improvement is also possible in our architecture. The data plane can send to the controller the output bit string of its BNN's last hidden layer and make the controller finish the calculation of the output layer using the real-valued weights. The details of the interaction between control and data planes will be described in the next section.

So far, we have discussed the BNN training within one edge domain. In a network with N domains, each domain's gateway may receive different packet samples. In order to learn more comprehensive attack patterns, we adopt federated learning [104] across all domains by connecting all gateway controllers to a cloud server. In federated learning, each gateway calculates the weight gradients with a batch of local input samples and sends the local updates to the cloud. Receiving updates from all gateways, the cloud will aggregate them and announce new weight values.

Scalability of federated learning is one of our main concerns. With a large number N of domains, the communication overheads between controllers and the cloud are not negligible if each controller reports all its real-valued weight updates in every learning batch. To save bandwidth, we take another binarization approach, SignSGD[18]. According to this method, each gateway now reports the 1-bit sign of local updates. Then, the cloud will have a "majority vote" and announce the result, which are also single bits. More specifically, we denote a local update of gateway n by g_n, then the new weights after communicating with the cloud are calculated by:

$$W_n^{t+1} = W_n^t + \delta^t * sign[\sum_{n=1}^{N} sign(g_n)] \tag{7.2}$$

where δ^t is the learning rate. Both down-link and up-link messages during federated learning are compressed to single bits, while the convergence persists as proven in [18]. The

Algorithm 7: Training Process

Input : X_n, Y_n: batch of inputs and labels trained at gateway n
$L(\hat{Y}_n, Y_n)$: loss function
W_n^t: real-valued weights in gateway n's control plane
$W_n^{b,t}$: binary weights in gateway n's data plane
δ^t: learning rate

Output: W_n^{t+1},$W_n^{b,t+1}$: updated weights of each gateway

1 **for** $n \in$ **N do**
2 $\quad \hat{Y}_n \leftarrow ForwardPropogation(X, W_n^t, W_n^{b,t})$
3 $\quad g_n \leftarrow BackPropogation(L(\hat{Y}_n, Y), W_n^t)$
4 **end**
5 (At the cloud) $\Delta W \leftarrow \delta^t * sign[\sum_{n=1}^{N} sign(g_n)]$
6 **for** $n \in$ **N do**
7 $\quad W_n^{t+1} \leftarrow W_n^t + \Delta W$
8 $\quad W_n^{b,t+1} \leftarrow sign(W_n^{t+1})$
9 **end**

complete BNN federated learning process is described in Algorithm 7.

Intuitively, SignSGD is expected to cooperate well with BNN because W_n^b will not change unless the update to W_n is large enough, i.e., from a negative value to a positive one or the other way around. Updates without impact on W_n^b will become a waste of resources. On the other hand, (7.2) appears to be a suitable way of updating. We will further show the efficiency of this proposed method in the evaluation section.

7.4 Evaluation Results

In this section, we deploy the proposed architecture and algorithms in a network testbed and evaluate them with a mixture of emulations and real device experiments to demonstrate the performance and costs in multiple aspects.

7.4.1 Testbed Setup

As indicated in the previous sections, the BNN simplifies the calculation process and therefore can be supported by different types of data plane devices with programmability. We choose the representative P4 language and its BMv2 [35] software switches to develop our prototype. The details and some crucial codes are presented in Appendix D.

We set up a network testbed containing multiple desktop computers with Linux oper-

ating system, connected through Ethernet cables. Each domain as well as the cloud server is represented by one computer. Each domain contains multiple hosts and one gateway, which are deployed in a Mininet [86] virtual network. The BNN implemented inside the data plane contains one fully-connected hidden layer with 120 neurons and a single-neuron output.

We consider the following publicly available datasets containing network traces to train and test the packet classification algorithm.

- *CICIDS2017* [156]. This dataset has a labeled record of multiple types of attacks and benign flows. Statistics are summarized for each flow. We take two thirds of records for training and the remaining for testing. We convert the layer-4 destination port, bidirectional total amount of packets and bytes into a 144-bit input vector to the BNN. All these statistics can be easily acquired by a P4-enabled switch.

- *ISCX Botnet 2014* [15]. This dataset collects heterogeneous botnet and malware traffic in realistic scenarios as well as non-malicious traffic. Its test set contains larger diversity than the training set to evaluate whether an algorithm is able to handle unknown traffic patterns. For the evaluations, we replay the TCP and UDP flows in this dataset to the gateway. Different from the last dataset, we choose a very common group of packet-level features, 5-tuple (IP addresses, layer-4 protocol and ports) and IP packet length as a 120-bit input vector.

7.4.2 Performance of Inference

First, we concentrate on Algorithm 6 and evaluate the classification performance within the scope of one domain and one gateway. Ignoring the federated learning method temporarily, we conduct an offline training on the gateway's BNN with the complete dataset and Adam [78] optimizer. For comparison, we also adopt other state-of-the-art learning algorithms, including the decision tree (DT) and linear support-vector machine (SVM) methods implemented by scikit-learn [129], as well as another neural network (denoted by NN) having the same structure as our BNN except that the activation function is non-linear (sigmoid function) and all weights are real-valued with 32-bit precision. Comparison with this NN will indicate if the binarization leads to performance loss.

We measure similar metrics as the last chapter, which are accuracy, precision, recall rates and F-1 scores that characterize the performance of inference.

Flow-Level Classification. Table 7.1 contains our measurement of accuracy, precision and recall rates on CICIDS2017 dataset, where algorithms classify a flow based on several

statistics. We observe that the real-valued NN has the same level of performance with DT. Our proposed BNN method has only slightly lower accuracy (0.6%) after the binarization. It also behaves better than SVM. At the same time, the BNN compresses the memory required for weight value storage to $1/32$ compared with the real-valued NN and makes it possible to run the algorithm as a data plane switch function (at the line speed of the switches). Besides, although DT has a good performance here, it lacks an effectively training algorithm in a distributed manner [12]. In contrast, we will demonstrate how the BNN can be trained across different domains using the federated learning framework in the next subsection.

Method	Accuracy	Precision	Recall	F_1
BNN	**0.983**	0.966	0.963	**0.965**
NN	0.989	0.967	0.987	0.977
DT	0.989	0.962	0.993	0.977
SVM	0.957	0.889	0.937	0.913

Table 7.1: Performance metrics of BNN on CICIDS2017 dataset.

Packet-Level Classification. While we have shown that our method is valid when performing classification based on flow statistics, we now concentrate on the packet-level features, i.e., matching on header fields, which permits the switch to react to incoming packets in real time. This is the major use case of the proposed method as a switch function. We measure performance metrics on the ISCX dataset with such packet-level features as inputs in Table 7.2. As in the previous table, we observe that the binarization incurs minor accuracy loss only (1.05%). Besides, BNN behaves better than both DT and SVM (6% and 7% more accuracy) under this setting.

Method	Accuracy	Precision	Recall	F_1
BNN	**0.945**	0.945	0.766	**0.846**
NN	0.953	0.992	0.767	0.865
DT	0.900	0.735	0.767	0.751
SVM	0.890	0.700	0.763	0.730

Table 7.2: Performance metrics of BNN on ISCX dataset.

A high recall rate is especially important for packet classification, since the incorrect blockage of non-malicious traffic (false negatives) may hamper normal network functionalities. Therefore we also measure the precision and recall rates in Table 7.2 and calculate the F-1 score, which shows a similar tendency as the accuracy performance.

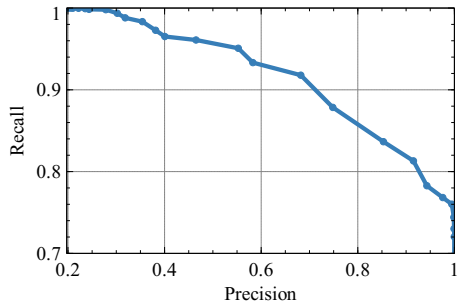
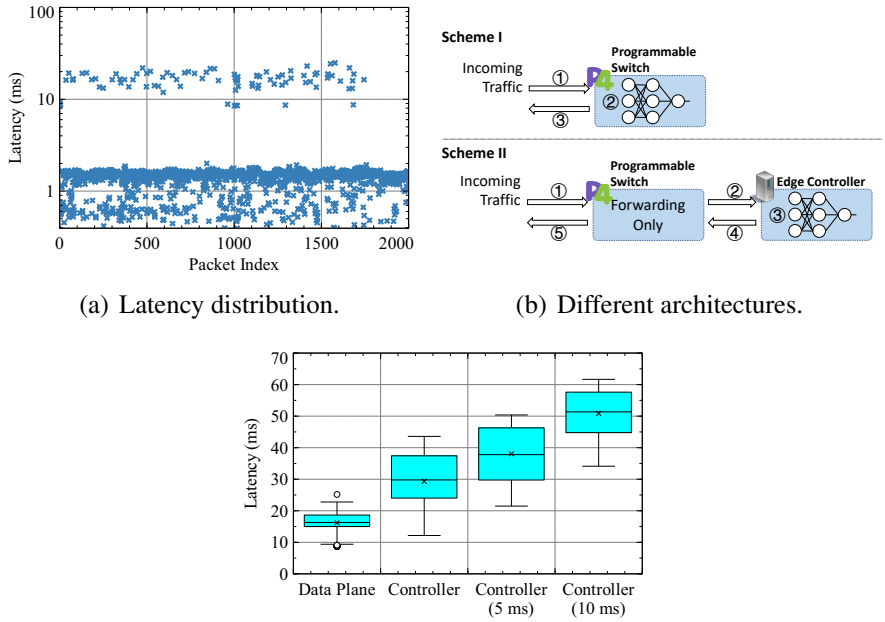


Figure 7.2: The precision-recall curve.

Moreover, by adjusting the threshold of the Hamming weight calculated in the output layer, a tradeoff can be achieved as depicted in Figure 7.2, which means that a better (higher) recall rate can be acquired at a cost of sacrificing some precision.

Packet Processing Latency. We next examine how the line-speed packet classification can be achieved in our proposed architecture. We send a subset of the ISCX dataset containing 2000 successive packets from a host to the gateway. As described in last section, the gateway data plane (the programmable switch) keeps both the BNN and a flow table matching the source IP addresses and TCP/UDP ports of incoming packets. In order to measure the network latency of every packet correctly, the switch marks the packets of malicious flows in the DSCP field instead of dropping them. Figure 7.3(a) plots the distribution of network latency of each packet. A small portion (around 5%) of packets are processed with a larger latency, having an order of magnitude of 10 ms. These are unknown input samples the gateway encounters for the first time without having a table entry, and therefore the switch uses the BNN to process them. The remaining 95% packets are processed with a much smaller latency (less than 2 ms), because they just require a one-time flow table match operation.

We next focus on the latency caused by running BNN in the control plane, which involves more complicated calculations. We deploy an alternative architecture (Scheme II in Figure 7.3(b)) where the neural network is deployed in the edge controller within the same domain. In this case, the data plane switch has to forward an unknown packet to the controller before making forwarding decisions. This is similar to the traditional intrusion detection approaches. To evaluate the performance of the two different architectures, we disable the flow table and make the BNN to process all packets. The box plots of latency are depicted in Figure 7.3(c). We notice that both the average value and the variation of packet processing latency are lower when deploying the BNN directly in the data plane.



(a) Latency distribution.

(b) Different architectures.

(c) Latency of processing unknown samples.

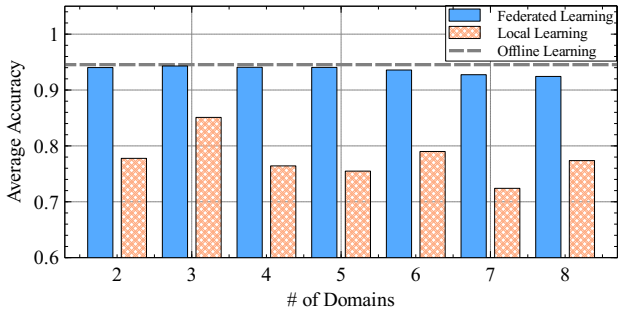
Figure 7.3: Packet processing latency evaluations of BNN inference as a switch function in the data plane.

Moreover, unlike the emulation environment, there is usually also propagation delay between the data and control planes in reality. Therefore, we introduce extra delay at the link of the control path (the third and forth box plots). As a result, the packet processing latency increases accordingly, demonstrating further the efficiency of our programmable data plane approach.

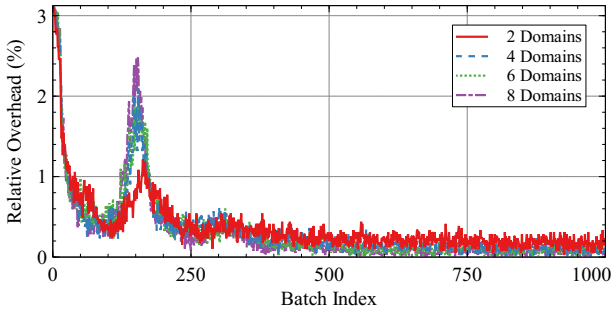
Main Takeaways. (1) The proposed BNN method performs packet classification with high accuracy based on both flow-level (flow statistics) and packet-level (header fields) features. (2) The BNN method outperforms several state-of-the-art learning methods in accuracy and F-1 score, with only slight performance loss during the binarization. (3) Implementing BNN in the data plane as a switch function achieves faster packet processing speed (line speed) than traditional approaches that deploy similar functions in a remote host.

7.4.3 Performance of Federated Learning

Having shown the performance of the proposed architecture within a single domain, we now extend the scenario to a multi-domain network and evaluate the federated learning



(a) Accuracy of different learning schemes.



(b) Convergence and control message overhead.

Figure 7.4: The (a) accuracy and (b) control message overheads during federated learning with the network scaling.

method (Algorithm 7). We assume that there are N domains each containing a gateway with the same P4 program. Correspondingly, the dataset is split into N subsets, and each gateway can only get access to one of them.

Accuracy with Distributed Training. First, we consider a case without federated learning (denoted as local learning), where each gateway does not connect to the cloud and is trained based on its subset only. We evaluate the trained BNN in each domain's gateway with the original test set. The average accuracy is depicted by red bars with cross texture in Figure 7.4(a), which severely degrades (less than 80% in the worst case compared with 94.5% when training with the complete dataset). On the other hand, if the federated learning described in Algotihm 7 is adopted during training, we can get an accuracy (blue bars in Figure 7.4(a)) which is almost as good as the offline trainig with the complete dataset. Such conclusion holds with different N values.

Communication Overhead. Although federated learning makes it possible to have a scalable solution for training gateways in multiple domains, the communication overhead of both uploading (gateways sending local updates) and downloading (the cloud announcing the aggregated update) will be a problem, especially when there is a large amount of

domains, which is the reason why we apply the binarization technique the second time during this communication. We analyze two types of traffic overheads; between the cloud and gateway controllers, as well as between each gateway's control and data planes.

When analyzing the overheads, we compare with traditional federated learning approaches, where local updates are updated with real values usually represented by 32 bits. Then, the cloud will aggregate updates by calculating the average values. It will broadcast the aggregated weight updates also in 32 bits. It is straightforward that the SignSGD method we adopt will significantly reduce the traffic overheads between the cloud and each edge controller, because only a single bit for every weight is required in our approach, leading to $1/32$ up-link traffic overhead. The same analysis can also be applied for down-link overhead.

The control message overhead from a gateway controller to the data plane switch updating the binarized neural weights also decreases. Another benefit of replacing the real-valued weights with single bits is that the controller does not need to send a control message if all binarized weights of the same neuron remain unchanged after training with a new batch. Therefore, less messages and overheads are required when the BNN converges. In Figure 7.4(b), we plot the control message overhead between all pairs of control and data planes during the first one thousand batches of federated learning. With the network converging quickly after training with 500 batches, the overhead reduces to less than 0.5% compared with the case that we use the real-valued NN and traditional federated learning method.

Main Takeaways. The proposed architecture enabled by federated learning leads to (1) much more accurate classification compared with training each gateway independently, and (2) small traffic overheads in communications between the cloud and edge controllers, as well as between the control and data planes.

7.5 Related Work

Learning Methods. Machine learning has been widely used for packet classification and intrusion detection such as approaches in [177] [99] promising high accuracy. However, a remote host or server is typically required to run the learning algorithm, introducing additional latency and preventing packets from being processed at the line speed of the switches. This is true even for the SDN-based learning methods [7] where learning is performed in the control plane (SDN controller) and the data plane (switches) only plays the role of flow table storing and matching. To overcome this limitation, we seek for a data

plane-compatible algorithm for higher processing speed.

Binarized Neural Networks. BNN is a type of neural network with only binary weights and activation functions [36], the inference process of which can be converted into bitwise operations. [143] demonstrates that BNN can achieve much faster speed and cost less memory while maintaining a high level of accuracy. Such features make it suitable for embedded devices with limited capacity [102]. [161] and [162] attempted to implement BNN in smart network devices. We make similar attempts while also performing realistic networking tasks, i.e., packet classification. In addition, we propose an online training scheme, which is scalable by adopting federated learning techniques.

Distributed / Federated Learning. For better scalability, neural networks can be trained in a distributed manner. Furthermore, the concept of federated learning is proposed [104], which keeps the training data locally to preserve privacy. Federated learning has been applied for the security issue in edge scenarios, e.g., IoT [119] and mobile networks [12]. Reducing communication overhead is a major concern in distributed and federated learning. One promising approach is to quantize or binarize the weight updates, such as SignSGD [18]. The distributed learning procedure also shows good compatibility with programmable data plane devices. [154] and [96] propose in-network methods for accelerating the aggregation phase of distributed training. In this work, we explore methods for effective intrusion detection at the network edge by combining the advantages of federated learning, BNN and programmable data plane.

7.6 Summary

In this chapter, we explored scalable methods for enhancing security at the network edge with SDN and programmable data plane. We designed an architecture running BNNs in edge gateways as switch functions to detect attacks from incoming packets. We also proposed a federated learning framework for gateways of multiple edge network domains to learn new attack patterns online and collaboratively. Evaluations on a real prototype we developed demonstrate that our method can achieve line-speed packet processing with high classification accuracy and low false alarm rate. Moreover, our solution is scalable with small communication overheads between the control and data planes of each edge domain, as well as between the cloud and each edge controller.

Chapter 8

Conclusions

This dissertation proposes solutions for better adopting SDN to wireless environments, especially the IoT networks. We discuss two approaches, the distributed deployment of the control plane and the local intelligence enabled by the programmable data plane. Such architectures take advantage of the centralized and programmable network management, while keeping the scalability and the compatibility of heterogeneous protocols which are required by wireless networking.

Under this framework, we make contributions in multiple aspects. We investigate the efficiency and costs of different processes during the network control architecture deployment, including the SDN controller placement, SDN controller synchronization and radio resource allocation. For each process, we establish mathematical models based on actual measurements on real networking devices. Then, we adopt proper techniques such as the combinatorial optimization, convex optimization and game theory to derive optimal or sub-optimal solutions with low time complexity.

When exploring the control architecture, we focus on the realistic performance of networking applications. We pick several most representative applications of IoT to verify our optimization algorithms, including the routing and load balancing, video and web services, as well as the IoT security. Evaluation results are provided to demonstrate the performance improvement when applying the proposed architecture and algorithms.

We develop several prototypes of our system design based on programmable hardware and open-source projects. We build a networking testbed with real devices to ensure our designs and models are practical. We also release open-source codes of our implementations. We hope that the reproducibility of our results will facilitate future research efforts for the benefit of research community.

8.1 Future Work and Open Problems

The current results have indicated several directions for future researches. One possible direction is to extend the proposed architecture to support more advanced wireless standards such as the 5G New Radio (5G NR), which plays an increasingly important role in IoT applications. Although our control architecture design provides general compatibility to heterogeneous RATs, several additional factors need to be considered. On the one hand, the new access technology brings more IoT use cases and therefore new objectives which require optimization, e.g., the energy consumption of lightweight devices. On the other hand, 5G utilizes mmWave frequency bands with features such as massive MIMO and highly directional antennas, which must be taken into consideration when modeling and optimizing the deployment process.

The combination of learning and the programmable data plane also deserves further exploration. So far, we focus on the packet classification application, where the programmable switches take relatively simple actions such as dropping malicious packets. Nevertheless, the programmable data plane language such as P4 supports more sophisticated operations including header modification and even stateful packet processing. It is potential to extend the learning approach to more network applications, such as the traffic engineering and Quality of service (QoS) control. At the same time, we will find the most appropriate learning models correspondingly. For example, to deploy graph-based neural networks so that the relationship among different devices in the network topology can be better captured.

Appendix A

Controller Placement: Emulation Setting and Proofs

A.1 Controller Traffic Analysis

Controller-node Traffic. When mentioning controller-node traffic, we refer to the traffic between the controllers and data plane nodes. Namely, controllers and nodes exchange various messages through a specific protocol, which is OpenFlow in most cases, including periodic heartbeat messages and statistic requests/replies. It is intuitive that such overheads grow when the network scales up. Moreover, there are also overheads related to the routing of packet flows. When a new flow is generated and a node receives packets that cannot be matched in its forwarding table, it will report to the controller through *PacketIn* messages. On the contrary, the controller may install new forwarding rules to nodes using *FlowMod* messages. Therefore, the overhead is also influenced by the number of flows. The more frequently new flows emerge, the larger overhead is needed to install forwarding rules.

Inter-controller Traffic. In ONOS, multiple protocols for controller synchronization coexist [113]. The first one is the anti-entropy gossip protocol [145]. With an interval, a controller sends the information (network topology, flow tables, etc.) within its domain (nodes assigned to it) to a random peer controller. The second algorithm is RAFT. ONOS uses it to synchronize controller-node assignments within the cluster, generating an overhead also related to the controller load. In Figure A.1, we classify the captured inter-controller traffic in the same setting as Section 3.2. Compared with RAFT traffic, the flow tables synchronization following gossip protocol and the heartbeat massages between each pair of controllers generate the majority of overheads. Besides, although RAFT is leader-based, ONOS deployed multiple RAFT clusters. Every controller acts as leader in

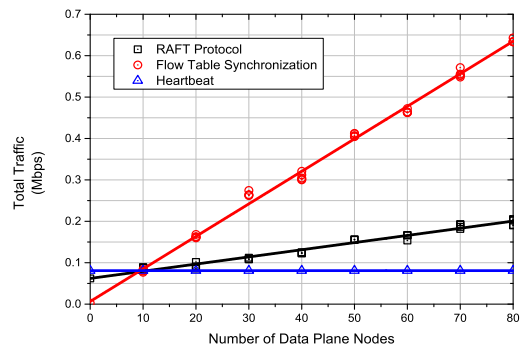


Figure A.1: Different inter-controller messages in ONOS.

specific clusters. Taking all of above into consideration, we can regard the synchronization of ONOS controllers as a leaderless approach. In OpenDaylight, RAFT [124] is the only protocol for controller synchronization, therefore all the traffics are leader-based.

A.2 Proof of Lemma 1

Consider an optimal assignment policy $\mathbf{y}^o$ with a node n^o being assigned to the controller at a node m^o, i.e., $y_{n^o m^o} = 1$. Clearly, it should be $x_{m^o} = 1$. Let us assume that there is another node $m^h \neq m^o$ such that $x_{m^h} = 1$ and:

$$\gamma d_{nm^h} + w^a_{nm^h} + \sum_{l:x_l=1} w^{dep}_{m^h l} < \gamma d_{nm^o} + w^a_{nm^o} + \sum_{l:x_l=1} w^{dep}_{m^o l} \,. \tag{A.1}$$

By reassigning node n^o to m^h instead of m^o the delay and assignment overhead cost are reduced by $d_{nm^o} + w^a_{nm^o} - d_{nm^h} - w^a_{nm^h}$. At the same time, the synchronization cost is reduced by:

$$\sum_{l:x_l=1} w^{dep}_{m^o l} - \sum_{l:x_l=1} w^{dep}_{m^h l} \tag{A.2}$$

since the load-independent part of the synchronization cost is not affected by the above reassignment. Hence, the value of the objective function J is reduced by:

$$\gamma d_{nm^o} + w^a_{nm^o} + \sum_{l:x_l=1} w^{dep}_{m^o l} - \gamma d_{nm^h} - w^a_{nm^h} - \sum_{l:x_l=1} w^{dep}_{m^h l} \overset{(A.1)}{>} 0 \tag{A.3}$$

This contradicts our assumption of optimality of $\mathbf{y}^o$.

A.3 Proof of Lemma 2

Since the positively weighted sum of supermodular functions is also supermodular it suffices to show that each of the following three functions is supermodular.

$$f_{s,con}(X) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} \mathbb{1}_{\{X_m \in X\}} \mathbb{1}_{\{X_l \in X\}} w_{ml}^{con} \tag{A.4}$$

$$f_{s,dep}^n(X) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} \mathbb{1}_{\{X_m \in X\}} \mathbb{1}_{\{X_l \in X\}} w^{dep} y(\boldsymbol{x}_X)_{nm} \tag{A.5}$$

$$f_a^n(X) = \sum_{m \in \mathcal{N} \cup \{c\}} y(\boldsymbol{x}_X)_{nm} (\gamma d_{nm} + w_{nm}^a) \tag{A.6}$$

Here, $f_{s,con}(X)$ and $f_{s,dep}^n(X)$ denote the constant and load-dependent synchronization overhead cost, respectively. For a given node $n \in \mathcal{N}$, the function $f_a^n(X)$ denotes the total (delay and overhead) assignment cost.

Let us consider two placement sets A and B where $A \subset B \subset G$. We add an element $X_k \in G \setminus B$ to both placement sets. In other words, we place a controller at node k.

1) For the function $f_{s,con}$ and the placement set A, the marginal value of element X_k is:

$$\sum_{l \in \{l' : X_{l'} \in A\} \cup \{c\}} w_{kl}^{con} + \sum_{m \in \{m' : X_{m'} \in A\} \cup \{c\}} w_{mk}^{con} \ . \tag{A.7}$$

Since the above value increases if we replace A with $B \supset A$, the function $f_{s,con}$ is supermodular.

2) For the function $f_{s,dep}^n(X)$ and any placement set, the marginal value of element X_k is w^{dep} (which is independent of the assignment policy). Hence, the function $f_{s,dep}^n(X)$ is modular, which is a special class of supermodular functions.

3) For the function $f_a^n(X)$, we distinguish between two cases. **In the first case**, according to the placement set B, node n is assigned to the controller at node j where $\gamma d_{nj} + w_{nj}^a \leq \gamma d_{nk} + w_{nk}^a$. Then, the marginal value of element X_k is zero. For the placement set A, node n is assigned to the controller at node j'. If $\gamma d_{nj'} + w_{nj'}^a \leq \gamma d_{nk} + w_{nk}^a$ then the marginal value is again zero. However, if $\gamma d_{nj'} + w_{nj'}^a > \gamma d_{nk} + w_{nk}^a$ then the marginal value is $\gamma d_{nk} + w_{nk}^a - \gamma d_{nj'} - w_{nj'}^a < 0$. **In the second case**, according to the placement B, node n is assigned to a controller at node j where $\gamma d_{nj} + w_{nj}^a > \gamma d_{nk} + w_{nk}^a$. Then, the marginal value of element X_k is $\gamma d_{nk} + w_{nk}^a - \gamma d_{nj} - w_{nj}^a$. For the placement set A, node n is assigned to a controller at node j' where it must be $\gamma d_{nj'} + w_{nj'}^a \geq \gamma d_{nj} + w_{nj}^a$ since $A \subset B$. Hence, the marginal value is $\gamma d_{nk} + w_{nk}^a - \gamma d_{nj'} - w_{nj'}^a \leq \gamma d_{nk} + w_{nk}^a - \gamma d_{nj} - w_{nj}^a$.

Appendix B

Controller Synchronization: Proofs

B.1 Proof of Theorem 2

To facilitate the presentation of the proof, we describe an alternative representation of the synchronization rate decisions using the following set of elements (ground set):

$$\mathcal{G} = \{g_{ij}^r \ : \ \forall i, j \in C, j \neq i, r \in \{1, \ldots, R\}\} \tag{B.1}$$

Here, each of the elements $\{g_{ij}^1, g_{ij}^2, \ldots, g_{ij}^R\}$ indicates a separate message disseminated between controllers i and j. Each subset of elements $\widehat{\mathcal{X}} \subseteq \mathcal{G}$ indicates a synchronization policy $\widehat{\boldsymbol{x}}$ where the synchronization rate $\widehat{x}_{ij}$ is equal to the number of the aforementioned elements included in $\widehat{\mathcal{X}}$.

We denote by the subsets $\mathcal{A} \subseteq \mathcal{G}$ and $\mathcal{O} \subseteq \mathcal{G}$ the solution returned by the Stochastic Greedy approximation algorithm and the optimal, respectively. We also denote by the subset $\mathcal{A}_k = \{\alpha_1, \ldots, \alpha_k\} \subseteq \mathcal{A}$ the solution returned by the Stochastic Greedy algorithm after the first $0 \leq k \leq B$ iterations. Then, we compute the probability that the set $\mathcal{S}$ of σ elements that is randomly picked by Stochastic Greedy at iteration $k+1$ does not overlap with the optimal set $\mathcal{O}$ besides of the elements already included in $\mathcal{A}_k$:

$$\begin{aligned}
\Pr[\mathcal{S} \cap (\mathcal{O} \setminus \mathcal{A}_k) = \emptyset] &= \Big(1 - \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{|\mathcal{G} \setminus \mathcal{A}_k|}\Big)^{\sigma} \\
&\leq e^{-\sigma \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{|\mathcal{G} \setminus \mathcal{A}_k|}} \\
&\leq e^{-\sigma \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{C(C-1)R}}
\end{aligned} \tag{B.2}$$

where the first inequality is because $(1-x)^a \leq e^{-ax}$ for any $x \in (0, 1)$. The second

inequality is because $|\mathcal{G}| = C(C-1)R$. Then, we have:

$$\begin{aligned}\Pr[\mathcal{S} \cap (\mathcal{O} \setminus \mathcal{A}_k) \neq \emptyset] &\geq 1 - e^{-\sigma \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{C(C-1)R}} \\ &\geq (1 - e^{-\sigma \frac{B}{C(C-1)R}}) \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{B} \\ &= (1-\epsilon) \frac{|\mathcal{O} \setminus \mathcal{A}_k|}{B} \end{aligned} \tag{B.3}$$

where the second inequality is because the function $1 - e^{-\sigma \frac{x}{C(C-1)R}}$ is concave with respect to $x \in [0, B]$. The last equality is because of the definition of ϵ.

At iteration $k+1$, Stochastic Greedy adds the element α_{k+1} to the solution $\mathcal{A}_k$ which is estimated to maximize the marginal gain $\widehat{\Psi}(\mathcal{A}_{k+1})$ - $\widehat{\Psi}(\mathcal{A}_k)$. However, the element with the real maximum marginal gain may be different, namely $\alpha'_{k+1} \neq \alpha_{k+1}$. Given that α_{k+1} is picked after τ try-outs, the following equation holds:

$$\begin{aligned}&\widehat{\Psi}(\mathcal{A}_k \cup \{\alpha_{k+1}\}) - \widehat{\Psi}(\mathcal{A}_k) = \\ &\quad \Big(\sum_{t=1}^{\tau} \frac{\mu_{k+1}^t}{\tau}\Big)\Big(\widehat{\Psi}(\mathcal{A}_k \cup \{\alpha'_{k+1}\}) - \widehat{\Psi}(\mathcal{A}_k)\Big) \end{aligned} \tag{B.4}$$

where μ_{k+1}^t is the ratio of marginal gains according to try-out $t = 1, 2, \ldots, \tau$. Each μ_{k+1}^t value is taken from a distribution with mean value μ.

By definition, $\widehat{\Psi}(\mathcal{A}_k \cup \{\alpha'_{k+1}\}) - \widehat{\Psi}(\mathcal{A}_k)$ is at least as much as the marginal value of an element randomly chosen from the set $\mathcal{S} \cap (\mathcal{O} \setminus \mathcal{A}_k)$ (if non-empty). This is actually an element randomly chosen from the entire set $\mathcal{O} \setminus \mathcal{A}_k$, since the set $\mathcal{S}$ itself is randomly chosen. Thus, we have:

$$\begin{aligned}&\widehat{\Psi}(\mathcal{A}_k \cup \{\alpha'_{k+1}\}) - \widehat{\Psi}(\mathcal{A}_k) \\ &\geq \Pr[\mathcal{S} \cap (\mathcal{O} \setminus \mathcal{A}_k) \neq \emptyset] \frac{\sum_{o \in \mathcal{O} \setminus \mathcal{A}_k} (\widehat{\Psi}(\mathcal{A}_k \cup \{o\}) - \widehat{\Psi}(\mathcal{A}_k))}{|\mathcal{O} \setminus \mathcal{A}_k|} \\ &\geq \frac{1-\epsilon}{B} \sum_{o \in \mathcal{O} \setminus \mathcal{A}_k} (\widehat{\Psi}(\mathcal{A}_k \cup \{o\}) - \widehat{\Psi}(\mathcal{A}_k)) \\ &\geq \frac{1-\epsilon}{B} (\widehat{\Psi}(\mathcal{O}) - \widehat{\Psi}(\mathcal{A}_k)) \end{aligned} \tag{B.5}$$

where the second inequality is because of (B.3). The third inequality is due to the rule of

diminishing returns. By combining (B.4) and (B.5) we obtain:

$$\begin{aligned}\widehat{\Psi}(\mathcal{A}_{k+1}) - \widehat{\Psi}(\mathcal{A}_k) &= \widehat{\Psi}(\mathcal{A}_k \cup \{\alpha_{k+1}\}) - \widehat{\Psi}(\mathcal{A}_k) \\ &\geq \frac{(1-\epsilon)\frac{\sum_{t=1}^{\tau}\mu_{k+1}^t}{\tau}}{B}(\widehat{\Psi}(\mathcal{O}) - \widehat{\Psi}(\mathcal{A}_k)) \end{aligned} \tag{B.6}$$

By induction, we can show that:

$$\begin{aligned}\widehat{\Psi}(\mathcal{A}_B) &\geq \Big(1 - \big(1 - \frac{(1-\epsilon)\frac{\sum_{k=1}^{B}\sum_{t=1}^{\tau}\mu_k^t}{B\tau}}{B}\big)^B\Big)\widehat{\Psi}(\mathcal{O}) \\ &\geq \big(1 - e^{-(1-\epsilon)\frac{\sum_{k=1}^{B}\sum_{t=1}^{\tau}\mu_k^t}{B\tau}}\big)\widehat{\Psi}(\mathcal{O}) \end{aligned} \tag{B.7}$$

Since the μ_k^t values are drawn from a distribution with mean value μ, it will be $\frac{\sum_{k=1}^{B}\sum_{t=1}^{\tau}\mu_k^t}{B\tau} = \mu$ in expectation, which concludes the proof.

B.2 Proof of Theorem 3

Let $\mu_1^1, \ldots, \mu_B^\tau$ be the marginal gain ratios associated with the $B\tau$ try-outs of the Stochastic greedy algorithm. Since $\mu_k^t \in (0,1)$, $\forall t, k$ with mean value μ, we can apply the Chernoff bound and obtain for each $\gamma \in (0,1)$:

$$\Pr[\frac{1}{B\tau}\sum_{k=1}^{B}\sum_{t=1}^{\tau}\mu_k^t < (1-\gamma)\mu] < e^{-\frac{\gamma\mu B\tau}{2}} \tag{B.8}$$

Therefore, with probability $1 - e^{-\frac{\gamma\mu B\tau}{2}}$, the empirical mean value will be at least as much as $(1-\gamma)\mu$. With the same probability, the performance will be at least as much as $1 - e^{-(1-\epsilon)(1-\gamma)\mu}$.

Appendix C

Access Network Orchestration: Prototype Implementation

We develop a prototype of proposed Slicing Agents and Slicing Orchestrator and test them over several SD-RAN controllers. In this section, we introduce some details of our implementation.

We define two protocols in the system. First, RESTful APIs are exposed to operators for the database update and lookup. The orchestrator is open for network and service providers to register in the system, update their information or quit the system. It also keeps an account recording the history charges and compensations caused by auctions to each provider. Similarly, the agent has APIs for a service provider to update the profiles of its users, including the list of RANs a user is connecting to, the signal quality it receives and its identification in each RAN. The double auction is also initiated at the side of service provider, by specifying the type of resource to request, the agent will send an auction message to the Slicing Orchestrator.

Once the auction message is accepted by the orchestrator, it will broadcast to all agents to start an auction through the second protocol. The auction proceeds automatically by the communications between the orchestrator and agents, following the steps described in Algorithm 4. The auction protocol defines several types of messages, representing actions during a double auction such as bidding and parameter updating. When the orchestrator ensures the convergence of the algorithm, it broadcasts a message to end the auction, and sends a summary to each bidder, containing the final slicing scheme and a bill of charges/compensations. The auction module of our prototype is implemented in Python 3, with bidding decisions calculated by Scipy [72] optimizer.

Receiving the slicing scheme, the agent of network provider calls its SD-RAN con-

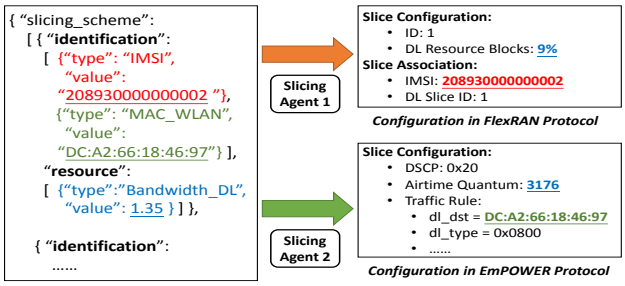


Figure C.1: Abstract of a slicing scheme as the result of an auction, from which agents of network providers extract information and convert it into a readable format for heterogeneous SD-RANs.

troller to actually execute slicing. The Slicing Agent works as a bridge enabling the interaction between Slicing Orchestrator and SD-RAN controller. It establishes southbound communications with the SD-RAN following the controller's protocols and interfaces (which may be heterogeneous). As an instance, we implement the downlink bandwidth auction with FlexRAN LTE controller and EmPOWER WLAN controller. FlexRAN identifies a user by its International Mobile Subscriber Identity (IMSI), and realizes bandwidth slicing by allocating specific number of Resource Blocks (RBs), the smallest resource unit of an LTE frame. On the other hand, EmPOWER marks flows classified by OpenFlow [103] rules with a Differentiated Services Code Point (DSCP) header, and applies the Airtime Deficit Round Robin (ADRR) packet scheduling policy [147] for downlink bandwidth slicing. Correspondingly, the Slicing Agent has two major tasks. First, for each user it picks the proper identification (e.g., IMSI for FlexRAN and OpenFlow fields for EmPOWER) from the a multiple ones provided by the service provider. Then, it will translate the amount of bandwidth requested into the unit which the SD-RAN controller adopts, e.g., number of Resource Blocks for FlexRAN, and airtime portion for EmPOWER. Figure C.1 shows the details of above example about the resource and user identity abstraction. Although protocol-dependent, development of such a module is not a bottleneck when a new SD-RAN joins the coordination of Slicing Orchestrator. In our prototype, lines of this module's codes account for less than 10% in total. And remaining codes are identical for all providers. This enables the modular and fast deployment of proposed architecture in heterogeneous RANs with no infrastructure modification and minor code development required.

Appendix D

Network Security: Prototype Implementation

In this section, we develop a prototype of the architecture ad algorithms proposed in Section 7.2 and 7.3. Among various available programmable data plane methods, we choose the representative P4 language [24] to implement our system. P4 is capable of achieving relatively complicated logic of packet header parsing and stateful processing, and it can be compiled for various targets, i.e., different types of software/hardware switches.

D.1 P4 Data Plane

The data plane device (gateway) in each domain runs a P4 program which is the key component of our proposed architecture. It is responsible for the following functionalities.

Feature Extraction. Protocol-independence is one of the most significant features of P4. By defining different network protocol headers in a P4 program, the data plane device is able to extract any header fields (e.g., fields of IP, TCP and even application layer protocols like HTTP) from an incoming packet and interpret them as bit strings. We concatenate several such strings together as the input of the BNN. Moreover, P4 also provides multiple ways (e.g., meters, counters and registers) to extract flow-level statistics. Such features can be used as the input of the BNN in the same way.

BNN Implementation. We use a register to store each BNN neuron's weight as a bit string. The registers are stateful so that they can be written and read dynamically. When processing each layer, bitwise XNOR operations are performed between the input bit string and every neuron in the layer. The activation function can be realized by calculating the Hamming weight of the XNOR output. Although P4 does not provide built-in functions

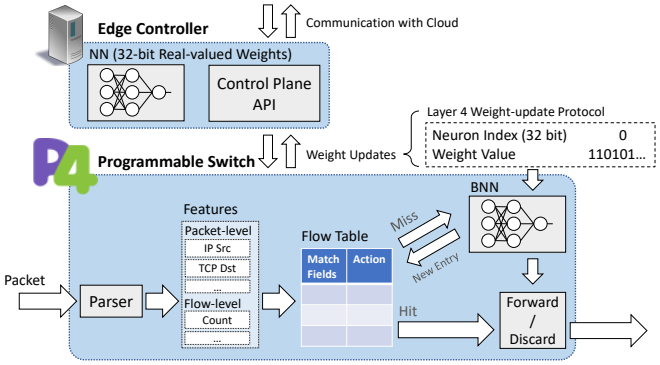


Figure D.1: P4-based prototype of the proposed gateway in one domain.

for it, there are various works [110] providing algorithms that enable fast calculations, and the parallel algorithm among them can be easily implemented in P4. Figure D.2 roughly shows how BNN can be implemented using the P4 grammar and data structure. In addition, we also implement the same logic in C language for supported devices.

Packet Forwarding. The BNN can coexist with layer-2/3 or any custom packet forwarding mechanism in the same P4 program. In this prototype, we consider a simple case where a packet from the flow regarded as an attack will be directly discarded. We combine the BNN with a flow table matching the incoming packet's 5-tuple. If the packet hits an entry in the flow table, it will be processed accordingly without being sent to the BNN. Otherwise, the BNN performs inference and adds a new entry to the table. *In both cases, line-speed processing is achieved, and this method further improves the efficiency as well as reduces computation costs.* The whole workflow of the data plane is depicted in Figure D.1.

Control Protocol. We define a new layer-4 protocol for the control plane to update the weight values of the data plane. It contains two header fields as shown in Figure D.3, the index of target neuron and a bit string representing the weight values of this neuron. When the data plane device receives a packet carrying this header from the controller, it will neither forward this packet nor call the BNN. Instead, it writes the new weight value to its register. This protocol can also be used by the data plane to send the output of the BNN's last hidden layer to the controller during the online training process, as described in the previous section.

```
// an example of 120-bit input and 120 neurons in each layer
control MyIngress(...) {
  register<bit<120>>(1024) weights;
  bit<120> Input = 0;
  bit<120> NextLayerInput = 0;
  bit<1> Activated;

  action Activation(bit<120> NeuronInput){
    bit<8> popcnt = ...  // calculate Hamming weight
    Activated = popcnt>60;
    NextLayerInput = NextLayerInput<<1 + (bit<120>) Activated;
  }

  action LayerProcess(bit<10> IndexOffset){
    bit<120> weight = 0;
    weights.read(weight, (bit<32>)IndexOffset+0);
    Activation(~(weight^Input));
    weights.read(weight, (bit<32>)IndexOffset+1);
    Activation(~(weight^Input));
    ...  // process all neurons in the same way
    }

  apply{
    ...
    // a function extracting header fields and statistics
    BuildInput();
    LayerProcess(0);        // first layer processing
    Input=NextLayerInput;
    NextLayerInput=0;
    LayerProcess(120);      // second layer processing
    Input=NextLayerInput;
    NextLayerInput=0;
    LayerProcess(240);      // third layer processing
    ...
  }
  ...
}
```

Figure D.2: Implementing BNN with P4 codes

```
typedef bit<120> MaxInputSize;
header weightupdate_t {
    bit<32>     index;
    MaxInputSize    value;
}
...
parser MyParser(...){
    ...
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
            17: parse_udp;
            6:  parse_tcp;
            61: parse_weightupdate;
            default: accept;
        }
    }

    state parse_weightupdate {
        packet.extract(hdr.weightupdate);
        transition accept;
    }
}
```

Figure D.3: An example of P4 header definition for weight updates

D.2 Control Plane and Cloud Server

We deploy another host with a general CPU in the same domain as the controller for each gateway. In order to perform online training, each controller should hold a neural network with real-valued (rather than binary) weights. We implement such networks by TensorFlow [3] and use Scapy [22] for the communications with the data plane. We also deploy a server as the cloud for federated learning. It receives local updates from each controller through UDP packets and conducts the aggregation. We evaluate this prototype with different topologies, which will be described in detail in the next section.

Bibliography

[1] Floodlight. https://github.com/floodlight/floodlight.

[2] Ryu sdn framework. https://ryu-sdn.org/.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[4] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, and M. El-Nainay. *On Stochastic Controller Placement in Software-defined Wireless Networks*. IEEE WCNC, 2017.

[5] Fadele Ayotunde Alaba, Mazliza Othman, Ibrahim Abaker Targio Hashem, and Faiz Alotaibi. Internet of things security: A survey. *Journal of Network and Computer Applications*, 88:10–28, 2017.

[6] Ali Saeed Dayem Alfoudi, Mohammed Dighriri, Gyu Myoung Lee, Rubem Pereira, and Fung Po Tso. Traffic management in lte-wifi slicing networks. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 268–273. IEEE, 2017.

[7] Pedro Amaral, Joao Dinis, Paulo Pinto, Luis Bernardo, Joao Tavares, and Henrique S Mamede. Machine learning in software defined networks: Data collection and traffic classification. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–5. IEEE, 2016.

[8] Ioannis Andrea, Chrysostomos Chrysostomou, and George Hadjichristofi. Internet of things: Security vulnerabilities and challenges. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 180–187. IEEE, 2015.

[9] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1093–1110, 2017.

[10] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *in Proc. SIAM Journal on Computing, vol. 33, no. 3,*, pages 544–562, 2004.

[11] M. Aslan and A. Matrawy. Adaptive consistency for distributed sdn controllers. *IEEE Networks*, 2016.

[12] Evita Bakopoulou, Balint Tillman, and Athina Markopoulou. A federated learning approach for mobile packet classification. *arXiv preprint arXiv:1907.13113*, 2019.

[13] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2017.

[14] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. In *in Proc. IEEE CNSM*, 2013.

[15] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*, pages 247–255. IEEE, 2014.

[16] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6.

[17] Massimo Bernaschi, Filippo Cacace, Giulio Iannello, Massimo Vellucci, and Luca Vollero. Opencapwap: An open source capwap implementation for the management and configuration of wifi hot-spots. *Computer Networks*, 53(2):217–230, 2009.

[18] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[19] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[20] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. Openstate: Programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51, 2014.

[21] Roberto Bifulco and Gábor Rétvári. A survey on the programmable data plane: Abstractions, architectures, and open problems. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–7. IEEE, 2018.

[22] Philippe Biondi et al. Scapy, 2011.

[23] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.

[24] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[25] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[26] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. V. Ramos, and A. Bessani. Design and implementation of a consistent data store for a distributed sdn control plane. *IEEE EDCC*, 2016.

[27] Amina Boubendir, Fabrice Guillemin, Christophe Le Toquin, Marie-Line Alberi-Morel, Frédéric Faucheux, Sylvaine Kerboeuf, Jean-Luc Lafragette, and Barbara Orlandi. Federation of cross-domain edge resources: a brokering architecture for network slicing. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 415–423. IEEE, 2018.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[29] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *in Proc. IEEE FOCS*, 2012.

[30] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou, and Jiming Chen. Ghost-in-zigbee: Energy depletion attack on zigbee-based wireless networks. *IEEE Internet of Things Journal*, 3(5):816–829, 2016.

[31] Dimitris E. Charilas and Athanasios D. Panagopoulos. A survey on game theory applications in wireless networks. *Computer Networks*, 54(18):3421 – 3430, 2010.

[32] Tao Chen, Honggang Zhang, Xianfu Chen, and Olav Tirkkonen. Softmobile: Control evolution for future heterogeneous mobile networks. *IEEE Wireless Communications*, 21(6):70–78, 2014.

[33] Junguk Cho, Binh Nguyen, Arijit Banerjee, Robert Ricci, Jacobus Van der Merwe, and Kirk Webb. Smore: Software-defined networking mobile offloading architecture. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*, pages 21–26, 2014.

[34] Thomas Clausen and Philippe Jacquet. Rfc3626: Optimized link state routing protocol (olsr), 2003.

[35] P4 Language Consortium et al. Behavioral model (bmv2), 2018.

[36] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[37] Costas Courcoubetis and Richard Weber. *Pricing communication networks: economics, technology and modelling*. John Wiley & Sons, 2003.

[38] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[39] Huynh Tu Dang, Han Wang, Theo Jepsen, Gordon Brebner, Changhoon Kim, Jennifer Rexford, Robert Soulé, and Hakim Weatherspoon. Whippersnapper: A p4 language benchmark suite. In *Proceedings of the Symposium on SDN Research*, pages 95–101. ACM, 2017.

[40] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 7–12, 2013.

[41] Salvatore D'Oro, Francesco Restuccia, Tommaso Melodia, and Sergio Palazzo. Low-complexity distributed radio access network slicing: Algorithms and experimental results. *IEEE/ACM Transactions on Networking (TON)*, 26(6):2815–2828, 2018.

[42] Sinem Coleri Ergen. Zigbee/ieee 802.15. 4 summary. *UC Berkeley, September*, 10(17):11, 2004.

[43] Melike Erol-Kantarci and Sukhmani Sukhmani. Caching and computing at the edge for mobile augmented reality and virtual reality (ar/vr) in 5g. *Ad Hoc Networks*, pages 169–177, 2018.

[44] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[45] U. Feige, V. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *in Proc. IEEE FOCS*, 2011.

[46] M. Feldman, J. Naor, and R. Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm. In *in Proc. ICALP*, 2011.

[47] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd annual international conference on mobile computing and networking*, pages 127–140, 2017.

[48] Xenofon Foukas, Navid Nikaein, Mohamed M Kassem, Mahesh K Marina, and Kimon Kontovasilis. Flexran: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 427–441, 2016.

[49] Kevin Fu and Wenyuan Xu. Risks of trusting the physics of sensors. *Communications of the ACM*, 61(2):20–23, 2018.

[50] Vojislav Gajić, Jianwei Huang, and Bixio Rimoldi. Competition of wireless providers for atomic users. *IEEE/ACM Transactions on Networking*, 22(2):512–525, 2014.

[51] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 513–521. IEEE.

[52] Pablo Caballero Garces, Albert Banchs, Gustavo de Veciana, and Xavier Costa-Pérez. Network slicing games: Enabling customization in multi-tenant mobile networks. *CoRR*, abs/1612.08446, 2016.

[53] S. O. Gharan and J. Vondrak. Submodular maximization by simulated annealing. In *in Proc. ACM/SIAM SODA*, 2011.

[54] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research, vol 22, no. 1, pp. 180-182*, 1974.

[55] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[56] M. Guo and P. Bhattacharya. *Controller Placement for Improving Resilience of Software-defined Networks*. IEEE ICNDC, 2013.

[57] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao. Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks, vol. 68, pp. 95-109*, 2014.

[58] Ummy Habiba and Ekram Hossain. Auction mechanisms for virtualization in 5g cellular networks: Basics, trends, and open challenges. *IEEE Communications Surveys & Tutorials*, 20(3):2264–2293, 2018.

[59] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[60] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *in Proc. HotSDN*, 2012.

[61] A. Hilal, J. N. Chattha, V. Srivastava, M. S. Thompson, A. B. MacKenzie, L. A. DaSilva, and P. Saraswati. *CRAWDAD dataset vt/maniac*, 10:15783.

[62] Y. Hu, T. Luo, N. C. Beaulieu, and C. Deng. The energy-aware controller placement problem in software defined networks. *IEEE Communications Letters, vol. 21, no.4, pp. 741-744*, 2017.

[63] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan. Reliability-aware controller placement for software-defined networks. In *in Proc. IFIP/IEEE IM*, 2013.

[64] V. P. Il'ev. An approximation guarantee of the greedy descent algorithm for minimizing a supermodular set function. *Discrete Applied Mathematics, vol. 114, no. 1-3, pp. 131-146*, 2001.

[65] Intel. Intel dpdk: Data plane development kit.

[66] George Iosifidis, Lin Gao, Jianwei Huang, and Leandros Tassiulas. A double-auction mechanism for mobile data-offloading markets. *IEEE/ACM Transactions on Networking (TON)*, 23(5):1634–1647, 2015.

[67] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. In *in Proc. Journal of the ACM (JACM), vol. 48, no. 2,*, pages 274–296, 2001.

[68] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[69] Nadeem Javaid, Arshad Sher, Hina Nasir, and Nadra Guizani. Intelligence in iot-based 5g networks: Opportunities and challenges. *IEEE Communications Magazine*, 56(10):94–100, 2018.

[70] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia. On the controller placement for designing a distributed sdn control layer. In *in Proc. IFIP Networking*, 2014.

[71] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 163–174.

[72] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[73] Saehoon Kang and Wonyong Yoon. Sdn-based resource allocation for heterogeneous lte and wlan multi-radio networks. *The Journal of Supercomputing*, 72(4):1342–1362, 2016.

[74] Sukhveer Kaur, Japinder Singh, and Navtej Singh Ghumman. Network programmability using pox controller. In *Proc. Int. Conf. Commun., Comput. Syst.(ICCCS)*, volume 138, pages 134–138, 2014.

[75] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[76] Hongseok Kim and Marina Thottan. A two-stage market model for microgrid power transactions via aggregators. *Bell Labs Technical Journal*, 16(3):101–107, 2011.

[77] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[78] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[79] Hlabishi I Kobo, Adnan M Abu-Mahfouz, and Gerhard P Hancke. A survey on software-defined wireless sensor networks: Challenges and design requirements. 5:1872–1899.

[80] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[81] A Koshibe, A Baid, and Ivan Seskar. Towards distributed hierarchical sdn control plane. In *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, pages 1–5. IEEE.

[82] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. In *Proc. of the IEEE, vol. 103, no. 1,*, pages 14–76, 2015.

[83] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham. On using bargaining game for optimal placement of sdn controllers. In *in Proc. IEEE ICC*, 2016.

[84] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. What you need to know about sdn flow tables. In *International Conference on Passive and Active Network Measurement*, pages 347–359. Springer, 2015.

[85] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management, vol. 12, no. 1, pp. 4-17*, 2015.

[86] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[87] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309. IEEE, 2017.

[88] Anhtuan Le, Jonathan Loo, Yuan Luo, and Aboubaker Lasebae. The impacts of internal threats towards routing protocol for low power and lossy network performance. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000789–000794. IEEE, 2013.

[89] Joohyun Lee, Yung Yi, Song Chong, and Youngmi Jin. Economics of wifi offloading: Trading delay for cellular capacity. *IEEE Transactions on Wireless Communications*, 13(3):1540–1554, 2014.

[90] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile data offloading: How much can wifi deliver? In *Proceedings of the 6th International COnference*, page 26. ACM, 2010.

[91] Ying Loong Lee, Jonathan Loo, Teong Chee Chuah, and Li-Chun Wang. Dynamic network slicing for multitenant heterogeneous cloud radio access networks. *IEEE Transactions on Wireless Communications*, 17(4):2146–2161, 2018.

[92] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized? state distribution trade-offs in software defined networks. *ACM HotSDN*, 2012.

[93] Chuanhuang Li, Yan Wu, Xiaoyong Yuan, Zhengjun Sun, Weiming Wang, Xiaolin Li, and Liang Gong. Detection and defense of ddos attack–based on deep learning in openflow-based sdn. *International Journal of Communication Systems*, 31(5):e3497, 2018.

[94] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *in Proc. Information and Computation, vol. 222,*, pages 45–58, 2013.

[95] Yong Li, Guolong Su, Pan Hui, Depeng Jin, Li Su, and Lieguang Zeng. Multiple mobile data offloading through delay tolerant networks. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 43–48. ACM, 2011.

[96] Youjie Li, Iou-Jen Liu, Yifan Yuan, Deming Chen, Alexander Schwing, and Jian Huang. Accelerating distributed reinforcement learning with in-switch computing. 2019.

[97] Libelium. Waspmote, n.d.

[98] Yi-Bing Lin, Shie-Yuan Wang, Ching-Chun Huang, and Chia-Ming Wu. The sdn approach for the aggregation/disaggregation of sensor data. *Sensors*, 18(7):2025, 2018.

[99] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, pages 1–14, 2017.

[100] Tie Luo, Hwee-Pink Tan, and Tony QS Quek. Sensor openflow: Enabling software-defined wireless sensor networks. *IEEE Communications letters*, 16(11):1896–1899, 2012.

[101] Anthea Mayzaud, Remi Badonnel, and Isabelle Chrisment. A taxonomy of attacks in rpl-based internet of things. *International Journal of Network Security*, 18(3):459–473, 2016.

[102] Bradley McDanel, Surat Teerapittayanon, and HT Kung. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260*, 2017.

[103] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[104] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 3, 2017.

[105] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE.

[106] Daniele Midi, Antonino Rullo, Anand Mudgerikar, and Elisa Bertino. Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 656–666. IEEE, 2017.

[107] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE, 2017.

[108] C Mims. Forget'the cloud';'the fog'is tech's future. *The Wall Street Journal*, 2014.

[109] Masoud Moshref, Apoorv Bhargava, Adhip Gupta, Minlan Yu, and Ramesh Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 61–66, 2014.

[110] Wojciech Muła, Nathan Kurz, and Daniel Lemire. Faster population counts using avx2 instructions. *The Computer Journal*, 61(1):111–120, 2017.

[111] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos. *Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability*. IEEE GLOBECOM, 2014.

[112] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82, 2007.

[113] A. Muqaddas, A. Bianco, P. Giaccone, and G. Maier. Inter-controller traffic in onos clusters for sdn networks. In *in Proc. IEEE ICC*, 2016.

[114] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[115] Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa, and Baijian Yang. Predicting network attack patterns in sdn using machine learning approach. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 167–172. IEEE, 2016.

[116] Mohamad Nazrin Napiah, Mohd Yamani Idna Bin Idris, Roziana Ramli, and Ismail Ahmedy. Compression header analyzer intrusion detection system (cha-ids) for 6lowpan communication protocol. *IEEE Access*, 6:16623–16638, 2018.

[117] Jorge Navarro-Ortiz, Pablo Romero-Diaz, Sandra Sendra, Pablo Ameigeiras, Juan J Ramos-Munoz, and Juan M Lopez-Soler. A survey on 5g usage scenarios and traffic models. *IEEE Communications Surveys & Tutorials*, 22(2):905–929, 2020.

[118] A. Neumann, E. López, and L. Navarro. *An evaluation of BMX6 for Community Wireless Networks*. CNBuB, 2012.

[119] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Minh Hoang Dang, N. Asokan, and Ahmad-Reza Sadeghi. Dïot: A crowdsourced self-learning approach for detecting compromised iot devices. *CoRR*, abs/1804.07474, 2018.

[120] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. Openairinterface: A flexible platform for 5g research. *SIGCOMM Comput. Commun. Rev.*, 44(5):33–38, 2014.

[121] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 129–140, 1999.

[122] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[123] Y. E. Oktian, S. Lee, H. Lee, and J. Lam. Distributed sdn controller system: A survey on design choice. *Computer Networks, vol. 121, no. 5, pp. 100-111*, 2017.

[124] D. Ongaro and J. K. Ousterhout. *In Search of an Understandable Consensus Algorithm*. in USENIX Annual Technical Conference, 2014.

[125] Edward J Oughton, William Lehr, Konstantinos Katsaros, Ioannis Selinis, Dean Bubley, and Julius Kusuma. Revisiting wireless internet connectivity: 5g vs wi-fi 6. *arXiv preprint arXiv:2010.11601*, 2020.

[126] A. Panda, C. Scott, and A. Ghodsi al. Cap for networks. *ACM HotSDN*, 2013.

[127] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker. Scl: Simplifying distributed sdn control planes. *USENIX NSDI*, 2017.

[128] Stefano Paris, Fabio Martisnon, Ilario Filippini, and Lin Clien. A bandwidth trading marketplace for mobile data offloading. In *2013 Proceedings IEEE INFOCOM*, pages 430–434. IEEE, 2013.

[129] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[130] Mugen Peng, Shi Yan, Kecheng Zhang, and Chonggang Wang. Fog-computing-based radio access networks: Issues and challenges. *Ieee Network*, 30(4):46–53, 2016.

[131] Kostas Pentikousis, Yan Wang, and Weihua Hu. Mobileflow: Toward software-defined mobile networks. *IEEE Communications magazine*, 51(7):44–53, 2013.

[132] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Rfc3561: Ad hoc on-demand distance vector (aodv) routing, 2003.

[133] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 117–130, 2015.

[134] Pavan Pongle and Gurunath Chavan. A survey: Attacks on rpl and 6lowpan in iot. In *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–6. IEEE, 2015.

[135] Petar Popovski, Kasper Fløe Trillingsgaard, Osvaldo Simeone, and Giuseppe Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *IEEE Access*, 6:55765–55779, 2018.

[136] Konstantinos Poularakis, George Iosifidis, and Leandros Tassiulas. Deploying carrier-grade wifi: Offload traffic, not money. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 131–140. ACM, 2016.

[137] Konstantinos Poularakis, Qiaofeng Qin, Liang Ma, Sastry Kompella, Kin K Leung, and Leandros Tassiulas. Learning the optimal synchronization rates in distributed sdn control architectures. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1099–1107. IEEE, 2019.

[138] Qiaofeng Qin, Nakjung Choi, Muntasir Raihan Rahman, Marina Thottan, and Leandros Tassiulas. Network slicing in heterogeneous software-defined rans. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2371–2380. IEEE, 2020.

[139] Qiaofeng Qin, Konstantinos Poularakis, George Iosifidis, Sastry Kompella, and Leandros Tassiulas. Sdn controller placement with delay-overhead balancing in wireless edge networks. *IEEE Transactions on Network and Service Management*, 15(4):1446–1459, 2018.

[140] Qiaofeng Qin, Konstantinos Poularakis, Kin K Leung, and Leandros Tassiulas. Line-speed and scalable intrusion detection at the network edge via federated learning. In *2020 IFIP Networking Conference (Networking)*, pages 352–360. IEEE, 2020.

[141] Qiaofeng Qin, Konstantinos Poularakis, and Leandros Tassiulas. A learning approach with programmable data plane towards iot security. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 410–420. IEEE.

[142] Veselin Rakocevic, John Griffiths, and Graham Cope. Performance analysis of bandwidth allocation schemes in multiservice ip networks using utility functions. In *Teletraffic Science and Engineering*, volume 4, pages 233–243. Elsevier, 2001.

[143] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[144] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha. *Optimal Controller Placement in Software Defined Networks (SDN) Using a Non-zero-sum Game*. IEEE WoWMoM, 2014.

[145] R. Van Renesse, D. Dumitriu, V. Gough, and C. Thomas. Efficient reconciliation and flow control for anti-entropy protocols. In *in Proc. ACM Workshop on Large-Scale Distributed Systems and Middleware*, 2008.

[146] Roberto Riggio, Mahesh K Marina, Julius Schulz-Zander, Slawomir Kuklinski, and Tinku Rasheed. Programming abstractions for software-defined wireless networks. *IEEE Transactions on Network and Service Management*, 12(2):146–162, 2015.

[147] Roberto Riggio, Daniele Miorandi, and Imrich Chlamtac. Airtime deficit round robin (adrr) packet scheduling algorithm. In *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 647–652. IEEE, 2008.

[148] Giorgio Roffo, Simone Melzi, and Marco Cristani. Infinite feature selection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4202–4210, 2015.

[149] Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

[150] S.-C.-Lin, P. Wang, I. F. Akyildiz, and M. Luo. Towards optimal network planning for software-defined networks. *IEEE Transactions on Mobile Computing*, 2018.

[151] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer. Towards adaptive state consistency in distributed sdn control plane. *IEEE ICC*, 2017.

[152] A. Sallahi and M. St-Hilaire. Optimal model for the controller placement problem in software defined networks. *IEEE Communications Letters, vol. 19, no.1, pp. 30-33*, 2015.

[153] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 150–156, 2017.

[154] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[155] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 525–538, 2016.

[156] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[157] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[158] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *in Proc. ACM STOC*, 1997.

[159] Yasser Shoukry, Paul Martin, Yair Yona, Suhas Diggavi, and Mani Srivastava. Pycra: Physical challenge-response authentication for active sensors under spoofing attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1004–1015. ACM, 2015.

[160] A. Singla, S. Tschiatschek, and A. Krause. Noisy submodular maximization via adaptive sampling with applications to crowdsourced image collection summarization. *AAAI*, 2016.

[161] Giuseppe Siracusano and Roberto Bifulco. In-network neural networks. *arXiv preprint arXiv:1801.05731*, 2018.

[162] Giuseppe Siracusano, Davide Sanvito, Salvator Galea, and Roberto Bifulco. Deep learning inference on commodity network interface cards.

[163] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, volume 20, 2017.

[164] Statista. Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions).

[165] Z. Su and M. Hamdi. Mdcp: Measurement-aware distributed controller placement for software defined networks. In *in Proc. IEEE ICPADS*, 2015.

[166] K. Sudheera, K. Liyanagea, M. Maa, and P. H. J. Chong. Controller placement optimization in hierarchical distributed software defined vehicular networks. *Computer Networks, vol. 135, pp. 226-239*, 2018.

[167] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, Taehong Kim, and Jiyoung Kwak. On performance of opendaylight clustering. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 407–410. IEEE, 2016.

[168] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 115–120, 2012.

[169] Ajay Tirumala. Iperf: The tcp/udp bandwidth measurement tool. *http://dast. nlanr. net/Projects/Iperf/*, 1999.

[170] Yuta Tokusashi, Huynh Tu Dang, Fernando Pedone, Robert Soulé, and Noa Zilberman. The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, pages 21:1–21:16, New York, NY, USA, 2019. ACM.

[171] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou. Adaptive resource management and control in software defined networks. *IEEE Transactions on Network and Service Management, vol. 12, no. 1, pp. 18-33*, 2015.

[172] Mostafa Uddin, Sarit Mukherjee, Hyunseok Chang, and TV Lakshman. Sdn-based multi-protocol edge switching for iot service automation. *IEEE Journal on Selected Areas in Communications*, 36(12):2775–2786, 2018.

[173] Md Tanvir Ishtaique ul Huque, Weisheng Si, Guillaume Jourjon, and Vincent Gramoli. Large-scale dynamic controller placement. *IEEE Transactions on Network and Service Management*, 14(1):63–76, 2017.

[174] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[175] Marian Verhelst and Bert Moons. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, 2017.

[176] G. Wang, Y. Zhao, J. Huang, and W. Wang. The controller placement problem in software defined networking: A survey. *IEEE Network, vol. 31, no. 5, pp. 21-27*, 2017.

[177] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017.

[178] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 24, 2015.

[179] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI*, volume 11, pages 8–8, 2011.

[180] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning?: Toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, pages 25–33, New York, NY, USA, 2019. ACM.

[181] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu. Minimizing flow statistics collection cost of sdn using wildcard requests. In *in Proc. IEEE Infocom*, 2017.

[182] Dejun Yang, Xi Fang, and Guoliang Xue. Truthful auction for cooperative communications. In *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '11, pages 9:1–9:10, 2011.

[183] Mao Yang, Yong Li, Depeng Jin, Lieguang Zeng, Xin Wu, and Athanasios V Vasilakos. Software-defined and virtualized future mobile and wireless networks: A survey. 20(1):4–18.

[184] G. Yao, J. Bi, Y. Li, and L. Guo. On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters, vol. 18, no. 8, pp. 1339-1342*, 2014.

[185] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller, and Fabian Schneider. Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, 2017.

[186] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[187] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919, 2017.

[188] T. Zhang, P. Giaccone, A. Bianco, and S. D. Domenico. The role of the inter-controller consensus in the placement of distributed sdn controllers. *Computer Communications, vol. 113, no. 15, pp. 1-13*, 2017.

[189] Lei Zhong, Qianyi Huang, Fan Wu, and Guihai Chen. TRADE: A truthful online combinatorial auction for spectrum allocation in cognitive radio networks. *Wireless Communications and Mobile Computing*, 15(9):1320–1330, 2015.

[190] Xia Zhou, Sorabh Gandhi, Subhash Suri, and Haitao Zheng. ebay in the sky: Strategy-proof wireless spectrum auctions. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, MobiCom '08, pages 2–13, 2008.

[191] Noa Zilberman, Yury Audzevich, G Adam Covington, and Andrew W Moore. Netfpga sume: Toward 100 gbps as research commodity. *IEEE micro*, 34(5):32–41, 2014.