Summer 2016

Toward Open and Programmable Wireless Network Edge

Mostafa Uddin
Old Dominion University

TOWARD OPEN AND PROGRAMMABLE WIRELESS NETWORK EDGE

by

Mostafa Uddin
B.S. November 2006, Bangladesh University of Engineering & Technology, Bangladesh

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2016

Approved by:

Tamer Nadeem (Director)

Michele Weigle (Member)

Kurt Maly (Member)

ChunSheng Xin (Member)

Mahadev Satyanarayanan (Member)

ABSTRACT

TOWARD OPEN AND PROGRAMMABLE WIRELESS NETWORK EDGE

Mostafa Uddin
Old Dominion University, 2016
Director: Dr. Tamer Nadeem

Increasingly, the last hop connecting users to their enterprise and home networks is wireless. Wireless is becoming ubiquitous not only in homes and enterprises but in public venues such as coffee shops, hospitals and airports. However, most of the publicly and privately available wireless networks are proprietary and closed in operation. Also, there is little effort from industries to move forward on a path to greater openness for the requirement of innovation. Therefore, we believe it is the domain of university researchers to enable innovation through openness. In this thesis work, we introduces and defines the importance of open framework in addressing the complexity of the wireless network. The Software Defined Network (SDN) framework has emerged as popular solution for the data center network. However, the promise of the SDN framework is to make the network open, flexible and programmable. In order to deliver on the promise, SDN must work for all users and across all networks, both wired and wireless. Therefore, we proposed to create new modules and APIs to extend the standard SDN framework all the way to the end-devices (i.e., mobile devices, APs). Thus, we want to provide an extensible and programmable abstraction of the wireless network as part of the current SDN-based solution. In this thesis work, we design and develop a framework, *weSDN*[1] (wireless extension of SDN), that extends the SDN control capability all the way to the end devices to support client↔network interaction capabilities and new services. weSDN enables the control-plane of wireless networks to be extended to mobile devices and allows for top-level decisions to be made from a SDN controller with knowledge of the network as a whole, rather than device centric configurations. In addition, weSDN easily obtains user application information, as well as the ability to monitor and control application flows dynamically. Based on the weSDN framework, we demonstrate new services such as application-aware traffic management, WLAN virtualization, and security management.

[1] Pronounced as ‘*wee*-SDN’

ACKNOWLEDGMENTS

I would like to express my special appreciation and thanks to my advisor Professor Dr. TAMER NADEEM, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. I would also like to thank my committee members, professor KURT MALY, professor MICHELE WEIGLE, professor CHUNSHENG XIN, and professor MAHADEV SATYANARAYANAN for serving as my committee members even at your busy schedule. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you. I would especially like to thank my lab mates. All of you have been there to support me during my Ph.D. journey.

A special thanks to my family. Words cannot express how grateful I am to my mother, father, mother-in law and father-in-law, for all of the sacrifices that youve made for me. Your prayers for me was what sustained me this far. At the end I would like express appreciation to my beloved wife UMAMA AHMED who spent sleepless nights with and was always my support in the moments when there was no one to answer my queries. Finally, I want to mention my daughter, AAIMA MUSTAFA, whose precious smile dissolves all my tiredness after a long day.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

We are approaching a fundamental shift in the computational era as the penetration of smart devices (e.g., smartphones and tablets) reaches about 30% of the global population by the end of 2013. It is expected in 2019 that out of 9.4 billion mobile subscriptions around the world, 5.6 billion of them (60%) will be linked to a smartphone [48]. Data collected from telecommunication companies and network operators shows that smartphone penetration has broken the 50 percent barrier in the United States by mid-2012 [181], making the feature phone (i.e., classical cell phone) a shrinking minority among US mobile users.

Smart devices have matured as a computing platform and become equipped with several new communication interfaces (Cellular, Wi-Fi, and Bluetooth, acoustic interface using the integrated speaker and microphone, and visual interface using integrated camera and light). With the advancements in microprocessors and more smart computing devices becoming connected, we are seeing the next phase of the Internet populated with traffic primarily from devices (things) communicating with each other forming what is called the Internet of Things (IoT). According to Cisco, the number of IoT connected objects is expected to reach 50 billion by 2020 [7, 30]. The possibilities of what can be accomplished by taking advantage of the Internet of Things can change the way we live and do business.

As the number of smart devices and their applications continues to grow (in 2014, an average of 40,000 applications were added to Apple App Store per month [157]), transmission of mobile traffic data over wireless links (i.e., Wi-Fi and cellular links) is also expected to explode. To cope with the explosion of mobile devices coupled with a growing proliferation of cloud-based applications, best-effort Quality of Service (QoS) is no longer a satisfactory solution and a new breed of intelligent wireless networks is required. More specifically, it is now necessary to have greater visibility and flexible control over the traffic generated from/to the client devices in order to deliver optimal performance and a high Quality of Experience (QoE) to a variety of users and applications

Recent years, a number of researchers have proposed a new research paradigm referred to as *edge computing*, where services and applications run at *wireless network edges* [15, 142],

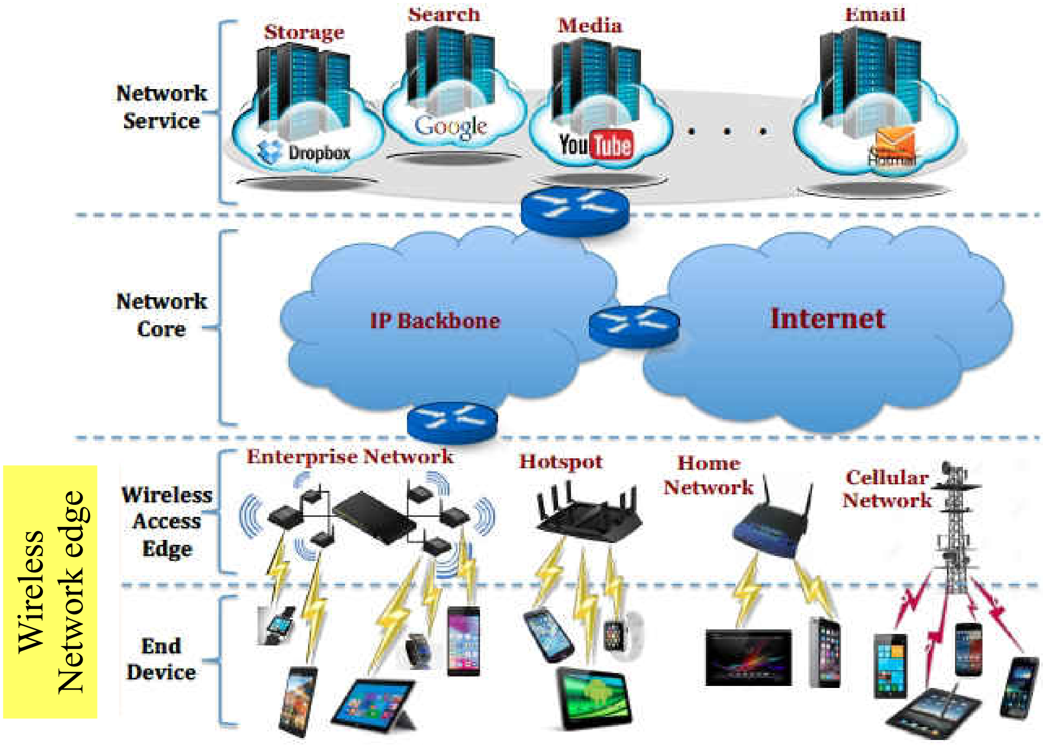


FIG. 1: The wireless network edge consists of wireless access devices and end-devices.

closer to client devices. As shown in Figure 1, wireless network edges is used to refer to both *end devices* (e.g., smartphone, tablets, smart watch etc.) and *wireless access devices*[1] (e.g., WiFi AP, Base Station, Edge router etc.). This trend of pushing computation and storage resources closer to client devices, is motivated by the new breed of applications and services that require low latency, high bandwidth, and privacy [191, 192, 193]. For example, applications, such as real-time gaming, augmented reality, and real-time streaming, are too latency-sensitive to deploy on the cloud. In this context of *edge computing*, it is essential to have flexible and efficient network management at the wireless network edges to support complex network management and configuration tasks such as end-to-end QoS for various network traffic, different traffic engineering schemes with various policies, and efficient load balancing [44, 89, 95, 165, 190].

1.2 PUSHING SDN TO WIRELESS NETWORK EDGES

Recently, the networking community has embraced Software Defined Network (SDN) [33, 62, 78, 186] approaches for designing and managing wireless networks in order to provide performance guarantees to end users by dynamically orchestrating services and policies on network routers, switches, and wireless Access Points (APs). In addition, recently, the Wireless & Mobile Working Group in the Open Networking Foundation (ONF) is investigating various use cases for SDN-enabled APs and is planning to standardize SDN control of wireless APs by extending OpenFlow [104]. Although SDN-enabled APs can control wireless resources for AP-to-client downlink traffic, they cannot control uplink traffic that also interferes with downlink transmissions because WiFi MAC is half-duplex. Thus, AP-side control cannot guarantee the wireless resource for both uplink and downlink. Without resource guarantees, the performance experienced by end users can be highly unpredictable even with high transmission rates of recent 802.11n/ac.

Unfortunately, control of existing SDN frameworks stops at the network edges, either at enterprise edge switches [89], datacenter hypervisor virtual switches [171], or home routers [126]. In wired environments, SDN policy can be effectively enforced on the traffic to/from the end devices without modifying end devices, since the last hop is a point-to-point full-duplex link and transmissions from end devices do not interfere with each other. However, unlike the wired access network, interaction with mobile devices over wireless networks needs to have fine-grained control and be able to selectively and dynamically apply traffic

[1] In rest of the paper, we refer to them as access device.

and wireless resource management techniques to enhance user experience.

Compared to the wired network, a wireless network has its own requirements of different services that include mobility management, traffic flow management, channel configuration, and air-time resource management. In order to address these services, there is a growing interest to bring SDN-based design concepts targeted to different types of wireless networks, which includes home WLAN [126], enterprise WLAN [109, 146, 147, 160], and cellular network [83]. Based on the core idea of SDN, the objective is to decouple and centralize the control functionality of the wireless network from the data traffic to provide flexibility and enhance many parts of the wireless network management. However, unlike the wired network, not necessarily every control decision could be handled centrally for wireless network. For example, in WLAN, each AP needs to make decisions about its modulation format, power, and channel based on Signal to Interference plus Noise Ratio (SINR) estimate. In this case, making such decision centrally in the controller and sending it to the AP needs to reach before the channel state information, from which the decision was derived, has become obsolete. Note that, the wireless channel condition is highly variable, therefore, the time when the decision reaches the AP might not be relevant. Considering these limitations of time-critical functionalities, researchers have only targeted the wireless network management issues that are primarily focused on mobility [109, 146, 147, 160], QoS configuration [146], virtualization [147], and security [73, 85].

In the literature, most of the works of SDN in the wireless network domain are targeted to improve the network management flexibility from the infrastructure perspective. For example, in the cellular network, major efforts towards using the SDN concept is to improve the design of the Radio Access Network (RAN) [63] and the core cellular network [83, 96, 98], which are the key components of the cellular network infrastructure. Similarly, in the Wi-Fi network, researchers have proposed to push SDN-like capabilities to Wi-Fi APs to simplify the implementation of high-level WLAN services, such as virtualization, authentication, association, load-balancing, and hand-off. Thus, in previous SDN works in wireless, there is a lack of interest to bring the end user's experience and perspective into the consideration of managing the wireless network.

However, such infrastructur- based SDN solutions have several limitations. For instance, existing network infrastructure is often considered as a *black box* and proprietary. Therefore, deploying SDN capability in the existing network infrastructure (i.e., cellular core network) for network management and new services is a complex and expensive task. Recently, we have seen project like ECOMP from AT&T that focuses on leveraging cloud technologies and network virtualization to reduce capital and operational expenditures, and to provide operations efficiencies by rapidly on-boarding new services (created by AT&T or 3rd parties) [10].

Considering user's mobility, end-devices move from one network to another. In this case, there can be scenarios where end devices are connected to un-managed networks, such as in coffee shops, metro stations, shopping malls, or in any other public place, where WLAN has no efficient network management mechanism or SDN capability. In this circumstance, the network infrastructure is unable to provide truly end-to-end management and control, in which end users could reap the full benefit of SDN and experience high QoE.

Managing only from the network infrastructure, it is implausible to ensure truly end-to-end QoE. Assume a scenario where an end user is running an on-demand video streaming application in his/her smartphone. In this case, in order to ensure end-to-end QoE for that end user, network infrastructure could allocate a certain amount of bandwidth resource for the video streaming flow. However, due to the uncertainty of last-hop wireless connectivity, the video streaming application might experience much less bandwidth compared to the allocated bandwidth in the network infrastructure. Therefore, without considering the end-device, it is not possible to efficiently allocate network resources and ultimately guarantee truly end-to-end QoE for the end-user.

One of the major shortcomings of infrastructure-based wireless network management is lacking the awareness of the characteristics and the context of the end devices. This sightless management, in fact, makes the current systems unable to provide true end-to-end QoE for end users, which can result in wasting network and end-devices resources in addition to the inability to optimize the QoE for end users. For example, consider a scenario where two users, one with a smartphone playing a video in the background and the other watching a sports game on a tablet, stream videos over the same AP. Now based on the context and characteristics of these two devices, it will be more efficient, in terms of QoE for both users and resource utilization, to allocate more bandwidth to the tablet to have better resolution than the smartphone. This is because the tablet has a bigger screen and thus needs more resolution in addition to the smartphone user playing the video in the background (e.g., listens to music while running other apps). Therefore, extending SDN capability to the end-devices could provide such context-awareness properties for smarter wireless network management.

Network infrastructure is unable to manage uplink 802.11QoS specifications. For example, one client greedily setting its every uplink transmission as high priority can unfairly dominate the air-time resource. Client-side QoS control via SDN APIs can prevent such unfair resource usage across network slices and also within the same slice. Because it is a shared medium, the wireless link is often the most critical from an end-to-end QoS perspective and most likely to define the performance perceived by the application and user. Therefore, by integrating SDN APIs in the end device, we can manage uplink QoS over the shared wireless medium, and provide truly end-to-end QoS control.

Similar to above, we believe having an SDN-like paradigm at end-devices (i.e., smartphone, tablets, etc.) can provide new services and tools that can enhance the user's experience and the interaction with the wireless network from the end user's perspective. Therefore, in this thesis work, we propose to push SDN capability all the way to end-devices to bring the last-hop under the control of the SDN framework to provide an extensible and programmable abstraction of the wireless network edges as part of the current SDN-based solution. Thus, end-devices can efficiently interact with the SDN based wireless infrastructure to optimize and enhance the overall performance of wireless network management. On the other hand, in SDN-incapable wireless network infrastructures, SDN capability at the end-device will at least allow the end-users to have programmable control and monitoring capabilities over the network stacks of the end-devices. Thus, the end-users can have their own intelligent network services that include setting network policies, diagnosing network connectivity, and selecting appropriate interfaces for their network flows.

1.3 THESIS OBJECTIVE AND CONTRIBUTION

In this thesis work, we have designed, developed and implemented a framework called weSDN that extends the existing SDN paradigm to the end devices (i.e., mobile devices) and wireless edge devices (i.e., Wi-Fi APs) by creating new components and APIs. Thus, we provide an extensible and programmable abstraction of the wireless network edge as part of the current SDN-based solution. Based on the framework, we demonstrate three services: application-awareness networking, WLAN virtualization, and security solution. Following are the contributions of our thesis work:

1. We design the weSDN framework, where we extend the SDN framework to network wireless edges (e.g., WiFi APs) and end devices. We extend both the *data plane* and the *control plane* of the SDN framework to support the wireless network.

2. We implement the weSDN framework on a real testbed. We use a UNIX-based platform in implementing the proposed framework.

3. We provide open APIs and programmable capabilities through the weSDN framework to develop new services to address current or future complex wireless network

challenges.

4. We utilize the weSDN framework to develop new services that are non-trivial and challenging to implement in traditional existing wireless network infrastructure. We develop three services: application-aware traffic management, WLAN virtualization, and security solution of mobile devices.

5. Finally, we evaluate the three services using different metrics such as throughput, energy efficiency, loss of packet, network overhead, accuracy of identifying security threat, accuracy of traffic classification, efficiency in time of identifying traffic flows, etc.

1.4 ORGANIZATION

In Chapter 2, we discuss the background of SDN architecture, SDN applications, and OpenFlow. We also discuss about the related work of SDN in the context of wireless networks: cellular and Wi-Fi. In Chapter 3, we describe the overview of our weSDN framework. Then we describe the architecture in detail. Finally, we provide possible services or applications that can be enhanced or created based on the weSDN framework. In Chapter 4, we implement and evaluate a WLAN virtualization system, pTDMA, based on our framework. In Chapter 5, we implement another service based on weSDN framework that provides fine-grained, on-the-fly application and flow-type awareness. We also develop two case studies based on this service that allow us to automatically apply simple traffic management policies on the wireless network edges. In Chapter 6, we design, develop, and evaluate an in-device application-aware network security solution service based on the weSDN framework. This service provides fine-grained network security policies to protect the wireless network communication of sensitive applications running at end-devices. Finally, in Chapter 7, we discuss the possible extension of weSDN framework and new services. In addition, we introduce new research directions of this work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter is organized as follows: in Section 2.1, it begins by describing the SDN concept with examples. Section 2.2 provides an overview of the SDN architecture. It also describes the OpenFlow [104] protocol. Section 2.3 describes briefly about SDN deployment in different wired network environments. Finally, in Section 2.4, we discuss about various related works that have used SDN or SDN-like framework in wireless network domain and smart devices.

2.1 SDN OVERVIEW

This section, discuss about the SDN concept with examples.

2.1.1 WHY SDN?

Computer networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes (i.e., devices that manipulate traffic for purposes other than packet forwarding, such as a firewall) with many complex protocols implemented on them. Network operators are responsible for configuring policies to respond to a wide range of network events and applications. They have to manually transform these high level-policies into low-level configuration commands while adapting to changing network conditions. And often they need to accomplish these very complex tasks with access to very limited tools. As a result, network management and performance tuning are quite challenging and thus error-prone [119].

In networks, we deal with two different planes: one is the *data plane* and the other is the *control plane* [112]. The *data plane* takes the decision of forwarding a packet based on some forwarding state. For example, if a packet comes to the router, it looks at its routing table to decide where to forward the packet. The *control plane* is basically responsible for creating the forwarding state. For example, a distributed routing algorithm, Dijkstra's is responsible for creating the forwarding state in the routing table. One of the key attributes of the traditional network is the tight coupling between the data and the control plane, which means that decisions about data flowing through the network are made on-board each network element. In this type of environment, the deployment of new network applications

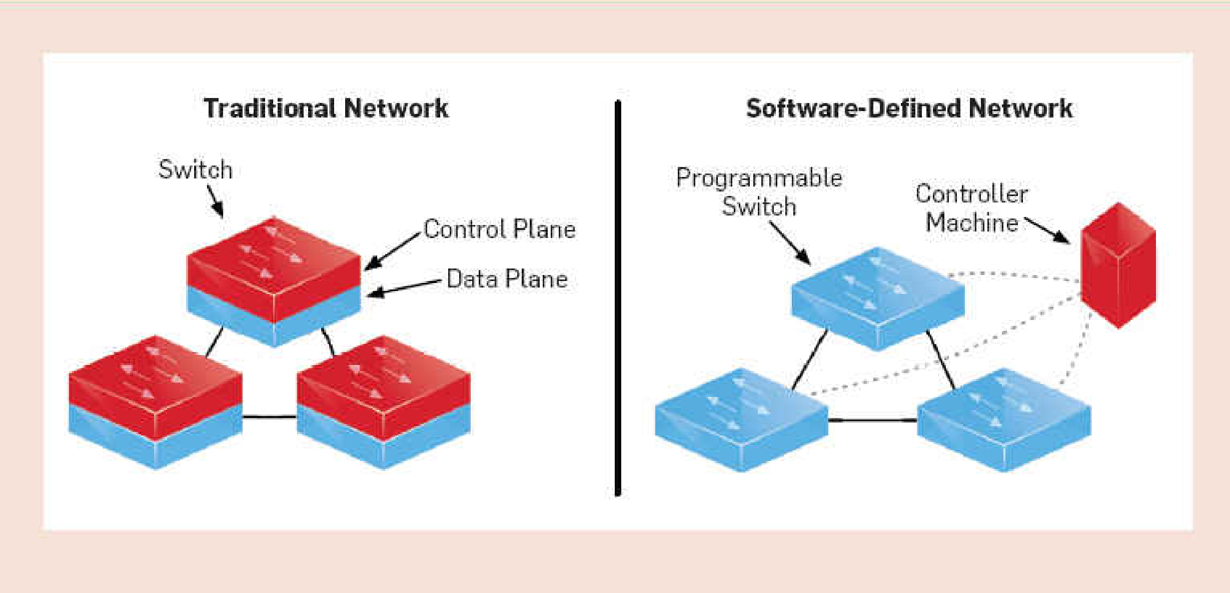


FIG. 2: Architectures of traditional networks and SDN [27]

or functionality is non-trivial, as they would need to be implemented directly into the infrastructure. Therefore, the Internet has become extremely difficult to evolve both in terms of its physical infrastructure as well as its protocols and performance. However, as current and emerging Internet applications and services become increasingly more complex and demanding, it is imperative that the Internet is able to evolve to address these new challenges.

The idea of SDN has been proposed as a way to facilitate network evolution [119]. SDN was developed to facilitate innovation and enable simple programmatic control of the network data-path. Note that the basic objective of SDN is not about improving overall network performances. Instead, it is about providing flexibility of network management that ultimately simplify and enhance different network services.

2.1.2 WHAT IS SDN?

In particular, SDN is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. In SDN, the network intelligence is logically centralized in software-based controllers (the *control plane*), and network devices become simple packet forwarding devices (the *data plane*) that can be programmed via an open interface (e.g., ForCES [45], OpenFlow [104], etc.).

SDN starts from two simple ideas: generalize network hardware so it provides a standard

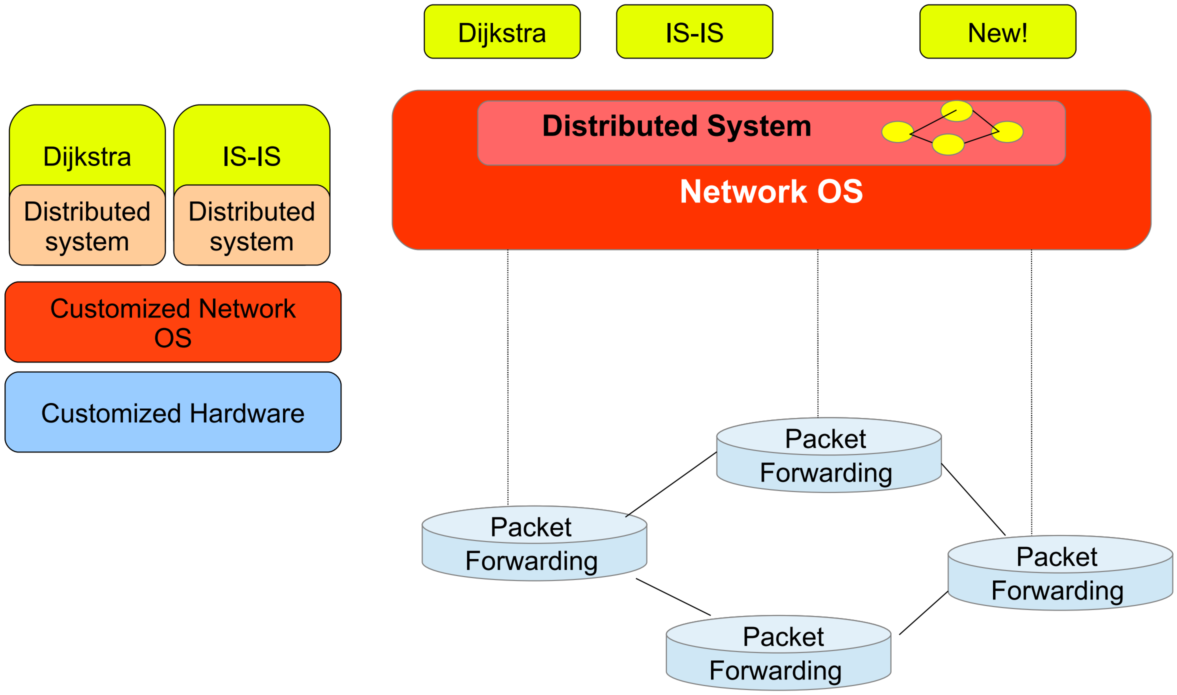


FIG. 3: Example of deploying a new routing algorithm in both traditional network and SDN.

collection of packet-processing functions instead of a fixed set of narrow features, and decouple the software that controls the network from the devices that implement it. This design makes it possible to evolve the network without having to change the underlying hardware and enables expressing network algorithms in terms of appropriate abstractions for particular applications.

Figure 2 contrasts the architectures of traditional networks and SDN. In SDN, one or more controller machines execute a general-purpose program that responds to events such as changes in network topology, connections initiated by end hosts, shifts in traffic load, or messages from other controllers, by computing a collection of packet-forwarding rules. The controllers then push these rules to the switches, which implement the required functionality efficiently using specialized hardware. Because SDN does not specify how controllers are implemented, it can be used to implement a variety of network algorithms, including simple ones such as shortest-path routing, and more sophisticated ones such as traffic engineering. In Figure 2, the separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications, straightforward network virtualization and management, and consolidation of various middleboxes into software control.

2.1.3 SDN EXAMPLE

Consider a network infrastructure consisting of traditional network routers that support two routing algorithms Dijkstra and IS-IS [105]. Figure 3 (left) shows what the traditional router looks like, where it has customized hardware, customized network OS, and routing algorithm modules Dijkstra, IS-IS. Note that, both routing algorithms have components of a distributed system. This distributed system component is responsible for creating the overview of the global network topology. Such a distributed system of creating the network topology is a complex task and is common among all routing algorithms. Now assume network operator needs to deploy a new routing algorithm in the network infrastructure, in such case he/she needs to replace the old router with new router device that supports the new routing algorithm. This process of changing the network infrastructure is complex, costly and time-consuming. However, in this SDN network infrastructure, deploying a new routing algorithm is similar to installing a new application on an Operating System (OS). Figure 3 (right) shows the network OS (i.e., *control plane*) that runs the distributed system for building the global network topology. The routing algorithm that runs on top of the network OS uses the network topology to create rules for forwarding packets for network switches. Thus, SDN decouples the *data plane* devices and allows them to run innovative network applications without replacing the physical network devices.

2.2 SDN ARCHITECTURE

SDN creates a layer network architecture where "control functions" (e.g., routing, security, policy etc.) are decoupled from the "forwarding function" of the network devices (e.g., router, switches etc.) [149, 150]. In addition, it allows control functions to be available as services via APIs to different "business applications". Figure 4, shows the three layers architecture of SDN; *data plane*, *control plane*, and *application plane*. The SDN Controller plane is a logically centralized entity in charge of (i) translating the requirements from the

SDN Application plane down to the SDN *data plane* and (ii) providing the SDN Applications plane with an abstract view of the network topology including statistics and events. An SDN Controller plane consists of one or more “Northbound-API” (NB API) Agents, the SDN Control Logic, and the “Southbound-API” (SB API) driver. The SDN SB API is the interface defined between an SDN Controller and an SDN *data plane*, which provides at least (i) programmatic control of all forwarding operations, (ii) capabilities advertisement, (iii) statistics reporting, and (iv) event notification. An SDN *data plane* comprises an SB API agent and a set of one or more traffic forwarding engines (“datapath”) and zero or more traffic processing functions. These engines and functions may include simple forwarding between the external interfaces or internal traffic processing or termination functions. SDN NB APIs are interfaces between SDN Applications plane and SDN Controllers plane that typically provide abstract network views. In addition, it enables direct expression of network behavior and requirements. On the other hand,“SDN control applications” running in the application layer are programs that explicitly, directly, and programmatically communicate their network requirements and desired network behavior to the SDN Controller via NB APIs. In addition, they may consume an abstracted view of the network for their internal decision-making purposes.

2.2.1 OPENFLOW

OpenFlow [104] standardizes information exchange technique between the *control plane* and *data plane*. In the OpenFlow architecture, we have OpenFlow switch (i.e., Open vSwitch [129]) in the *data plane* network devices, that contains one or more flow tables. Flow tables consist of flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching field?s portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the action was taken by the switch will depend on the instructions defined by the table-miss flow entry. Every flow table must contain a table-miss entry in order to handle table misses. This particular entry specifies a set of actions to be performed when no match is found for an incoming packet, such as dropping the packet, continue the matching process on the next flow table, or forward the packet to the controller over the OpenFlow channel.

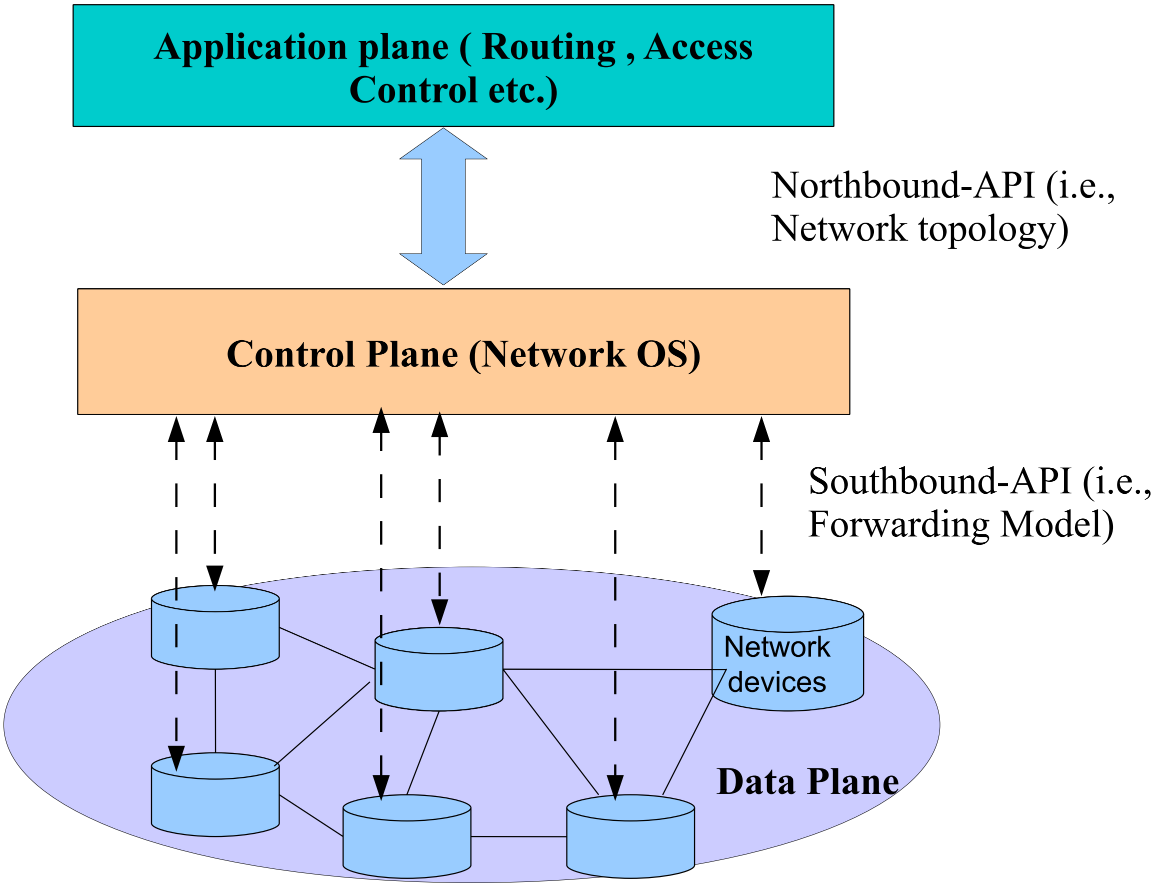

Application plane (Routing , Access Control etc.)
Northbound-API (i.e., Network topology)
Control Plane (Network OS)
Southbound-API (i.e., Forwarding Model)
Network devices
Data Plane

FIG. 4: The Layer Architecture of SDN.

The communication between the controller and OpenFlow switch happens via OpenFlow protocol, which defines a set of messages that can be exchanged between these entities over a secure channel. OpenFlow basically represents the SB API in the SDN layered architecture. Using the OpenFlow protocol a remote controller can, for example, add, update, or delete flow entries from the switch?s flow tables. That can happen reactively (in response to a packet arrival) or proactively.

2.3 SDN APPLICATIONS

There exists a wide variety of SDN applications in different network environments (e.g., data center, enterprise). By decoupling the control and *data plane*s, programmable networks enable customized control, an opportunity to eliminate middleboxes, as well as simplified development and deployment of new network services and protocols. Many novel applications have been implemented with SDN including policy-based access control, adaptive traffic monitoring, wide-area traffic engineering, network virtualization, and others [26, 45, 70, 72, 79, 84, 173]. In principle, it would be possible to implement any of these applications in a traditional network, but it would not be easy: the programmer would have to design new distributed protocols and also address practical issues because traditional switches cannot be easily controlled by third-party programs.

2.3.1 ENTERPRISE NETWORKS

In an enterprise network, SDN can be used to simplify the network by ridding it from middleboxes and integrating their functionality within the network controller. Some notable examples of middlebox functionality that has been implemented using SDN include NAT, firewalls, load balancers [67, 173], and network access control [115]. SDN can also be used to provide unified control and management [58]. The work Ethane [26] is a network architecture designed specifically to address the issues faced by enterprise networks.

2.3.2 DATA CENTERS

An increasingly important consideration in the data center is energy consumption, which has a non-trivial cost in large-scale data centers. The work of Heller et al. [70] ElasticTree, a network-wide power manager that utilizes SDN to find the minimum power network subset which satisfies current traffic conditions and turns off switches that are not needed. A practical example of a real application of the SDN concept and architecture in the context of data centers was presented by Google in early 2012. The company presented at the Open Network Summit [72] a large scale implementation of an SDN-based network connecting its data centers. The work in [79] presents in more detail the design, implementation, and evaluation of B4, a WAN connecting Google?s data centers worldwide.

2.4 SDN FOR WIRELESS NETWORK AND SMART DEVICES

Several efforts have focused on SDN in the context of infrastructure-based wireless access networks, such as cellular and Wi-Fi.

2.4.1 SDN IN CELLULAR

The existing cellular network infrastructure is less flexible in incorporating new services. However, with the fast increase of mobile device usage and the innovation of mobile applications, are pushing the cellular network provider to provide new services. Due to the inflexibility, costly equipment, and complex control-plane, deploying new services in the cellular network is non-trivial. Sometimes deploying a new service in the cellular network infrastructure takes months. Moreover, the current *control plane* of the cellular network are distributed for managing the spectrum, allocating airtime/radio resource, implementing handover mechanism, managing interference, and efficient load-balancing between cells. This distribute nature of the *control plane* among the *data plane* making cellular network less scalable. Therefore, In [96] argue to use SDN for simplifying the design and management of cellular data networks, while enabling new services such as, i) Directing Traffic Through Middleboxes, ii) Monitoring for Network Control and Billing, iii) Seamless Subscriber Mobility, iv) QoS and Access Control Policies, v) Virtual Cellular Operators and vi) Inter-Cell Interference Management. However, this is a position paper with no real implementation or evaluation of the system. In a cellular domain, many such works [83, 128] are basically discussing the conceptual idea of using SDN in cellular, rather than actual implementation and evaluation. The lack of openness in discussing the challenge of deploying SDN in the cellular network is hindering the process of practically applying SDN in cellular. However, we believe in near future cellular company will open-up their mindset in adopting SDN for the cellular network.

One of the first work in cellular, OpenRoads project [188], [187] envisions a world in which users could freely and seamlessly move across different wireless infrastructures which may be managed by various providers. They proposed the deployment of an SDN-based wireless architecture that is backwards-compatible, yet open and sharable between different service providers. However, unlike many previous works in cellular domain, they actually employ a testbed using OpenFlow-enabled wireless devices such as WiMAX base stations controlled by NOX and Flowvisor controllers and show improved performance on handover events. Their vision provided inspiration for subsequent work [96] that attempts to address specific requirements and challenges in deploying a software-defined cellular network.

At the other end of the spectrum, OpenRadio [13] focuses on deploying a programmable wireless *data plane* that provides flexibility at the PHY and MAC layers (as opposed to layer-3 SDN) while meeting strict performance and time deadlines. The system is designed to provide a modular interface that is able to process traffic subsets using different protocols such as WiFi, WiMAX, 3GPP LTE-Advanced, etc. Based on the idea of separation of the decision and forwarding planes, an operator may express decision plane rules and corresponding actions, which are assembled from processing plane modules (e.g., FFT, Viterbi decoding, etc.); an end result is a state machine that expresses a fully-functional protocol.

2.4.2 SDN IN WI-FI

Many of the WLAN enterprise solution brought by Cisco, Aruba, Meru uses a centralized approach to managing wireless access point to allocate a non-overlapping channel, set the power level to reduce interference, authenticate the user etc. For example, Meraki's [106] proprietary solution pioneered the concept of cloud-based management technique for configuring their homogeneous APs remotely through the cloud. However, all such solutions developed by the industries has been proprietary and only works with their APs. Unlike those proprietary systems, our weSDN framework is built upon open source solution of existing SDN frameworks to enable centrally control and configuration among heterogeneous wireless devices. The recent effort of standardizing centrally control AP management protocol (e.g., CAPWAP [184]) shows the urge of such solutions. Even industries like Meru have also declared the effort of bringing the SDN to Wi-Fi network [155].

The promise of SDN is to make the network open, flexible and programmable. To deliver on the promise, SDN must work for all users and across all networks, both wired and wireless, with true interoperability among network components via OpenFlow. OpenFlow [38] is an open and popular SDN framework designed to manage routing policies and other networking-related configurations. There have been efforts to bring OpenFlow to wireless APs (e.g., using OpenFlow together with SNMP [28]). There are prior works [146, 161] that leverages the existing OpenFlow for 802.11network focused on building an experimental platform and managing mobility of devices. However, we believe existing OpenFlow is not able to capture and configure the Wi-Fi network truly. The OpenFlow and OVS work between Layer 3 and Layer 2, which has no control over the wireless frames. Moreover, OpenFlow and OVS cannot accommodate measurements of the wireless medium, report per-frame receiver side statistics, or be used for setting per-frame or per-client transmission settings for the WiFi datapath.

Prior work like Odin [159], has proposed custom protocol rather than extending the OpenFlow protocol for the wireless network. In another example, Dyson [111] proposes a centralized framework to manage APs and clients in enterprise WLANs using a set of APIs at both APs and clients. There are series of work has been done based on the Odin framework, which we believe is not the right approach of making the wireless network programmable and open. Because wireless network contains both wired and wireless network, therefore running two different protocol does not make any sense. We believe, extending the existing wire SDN framework for the wireless network could bring our objective to have open programmable access to the wireless infrastructure so that network-aware applications can communicate directly with the wireless APs and the network can change dynamically in response. And customers can reap the full benefits of SDN.

There are many individual works that try to centrally manage and configure different settings of the Wi-Fi network for performance improvement. For example, CENTAUR [153] utilize the centralization concept to mitigate hidden terminals and to exploit exposed terminals. In another example, DenseAP [110], channel assignment and association related decisions are made centrally by taking advantage of a global view of the network. All such centrally controlled services and algorithms, proposed by the researcher, can run or tested on top of our weSDN framework, where weSDN can provide a common platform to provide enough resource with programmable capability.

Recent work that closely matches with the weSDN approach is vendor-neutral cloud-based centralized framework called COAP [126]. COAP configure, coordinate and manage individual home APs using an open API implemented by these commodity APs. The framework, implemented using OpenFlow extensions, allows the APs to share various types of information with a centralized controller - interference and traffic phenomenon and various flow contexts, and in turn receive instructions configuration parameters (e.g. Channel) and transmission parameters (through coarse-grained schedules and throttling parameters).

Very dense heterogeneous wireless networks have also been a target for SDN. These DenseNets have limitations due to constraints such as radio access network bottlenecks, control overhead, and high operational costs [4]. A dynamic two-tier SDN controller hierarchy can be adapted to address some of these constraints [4]. Local controllers can be used to take fast and fine-grained decisions, while regional (or Global) controllers can have a broader, coarser-grained scope, i.e., that take slower but more global decisions. In such a way, designing a single integrated architecture that encompasses LTE (macro/pico/Femto) and WiFi cells, while challenging, seems feasible.

However, none of the above work brought the client software into the SDN framework or tried direct coordination between the client agent and the network infrastructure (e.g., WLAN APs) to enable an integrated e2e solution. *We believe that running special-purpose client software is akin to extending opaque network middleboxes into end-devices and further fragmenting and complicating the control and management planes.* Instead, a better option would be to extend existing open and powerful SDN APIs and the associated control framework to the client side to support enhanced security, QoS, and WLAN virtualization.

In weSDN, we bring the client device's apps and their network usage into the SDN framework by running an agent (i.e., Local controller) in the client device, which is controlled from the network infrastructure. This concept of centrally managed agent software monitoring and controlling client devices is starting to become mainstream due to the Bring-Your-Own-Device (BYOD) phenomenon in enterprise networks. Enterprises have been deploying management software on their employee devices including personal mobile devices, to ensure device health, security and to protect corporate data stored in the devices [59]. Virtual Private Network (VPN) client software often perform bandwidth and access control on end devices. Mobile WAN optimization solutions use client-side software that interacts with a network proxy to optimize end-to-end performance [162].

With the SDN APIs in clients, weSDN can enforce SDN policies directly on the client uplink traffic, for example, Call Admission Control (CAC) for resource management and Network Access Control (NAC) for security. CAC and NAC can be enforced at network edges but cannot prevent the controller traffic from consuming wireless resources. weSDN can avoid such wasted transmissions and can benefit all WLAN users. weSDN can also better utilize multiple wireless interfaces on a multi-homed device as demonstrated by [190].

Mobile devices today are capable of fulfilling the role of extended SDN control as smartphones, tablets and laptops become more powerful in contrast to CPU-limited wireless APs and Ethernet switches.

2.4.3 SDN IN END-DEVICE

In literature, very few works have proposed to bring or deploy SDN-like paradigm at end-devices. Among them, works like [109, 151] propose to extend SDN-based control capability all the way to end-device. However, instead of using OVS and OpenFlow protocol, all of these works have used a separate software and customized protocol to monitor and control the end-devices. Such different and incoherent design of SDN for end-devices hinder the innovation of bringing the end-devices under the control of SDN framework. Furthermore, it fails to ensure the promise of SDN to make the network open, flexible and programmable. We believe, SDN must work across all networks, both wired and wireless, with true interoperability among network components via OpenFlow protocol and Open vSwitch.

Unlike the above of bringing the end-devices under the control of SDN framework for wireless network management, there have been few works that have deployed Open vSwitch in the mobile devices to create services such as leveraging multiple interfaces and programmable security policies. For example, in [190], authors have demonstrated, how to use OVS in mobile devices to utilize multiple wireless interfaces. Besides that, in [73] authors have presented a new security solution SDN-like framework at end-device that enables fine-grained, application-aware network security policy enforcement on mobile apps and devices. Similarly, in [85] author has developed an SDN-based End-to-end application containment architecture, called SeaCat that provides isolated secure network resource access for medical applications.

2.4.4 SDN IN IOT

The Internet of Things (IoT) is the result of many different enabling technologies such as embedded systems, wireless sensor networks, cloud computing, big-data, etc., which are used to gather, process, infer, and transmit data. Combining all these technologies imposes complex requirements on both the underlying networks and communication mechanisms between large scales of heterogeneous smart devices that communicate over the Internet. However, current network architecture and protocols are not designed to the high level of scalability, high amount of traffic and mobility for IoT. From our survey [65, 120, 133, 141, 167, 168], we find out following three network architectural challenges that could be addressed using SDN-like framework for IoT.

- IoT are often derived from the integration of independently deployed IoT subnetwork, characterized by heterogeneous devices and connectivity capabilities. The co-existence of different types of network technologies such as cellular, WiFi, ZigBee, and Bluetooth, they all must effectively integrate to create a seamless communication platform for IoT. In this case, SDN framework could be used to managing these open, geographically distributed, and heterogeneous networking infrastructure, especially in dynamic environments.

- The IoT multi-networks create opportunities for a wide range of applications with varying service requirements to execute concurrently. In the case, multiple applications might share the network and sensor resources for efficiency. In the heterogeneous IoT setting, different user-defined tasks may run simultaneously given the shared space they operate in, they often share the same sensing/networking resources, with differentiated quality requirements in terms of reliability (packet loss), latency, jitter, and bandwidth. Given the randomized nature of which IoT tasks are required, these applications are often developed, deployed, and triggered in an uncoordinated manner. In this case, the SDN-based framework could be used to optimizing sharing of sensing and communication resources and coordinating messaging.

- The current landscape of IoT uses diverse wireless and/or cellular communication technologies, which makes eavesdropping and vulnerability of channels extremely simple. Also, IoT devices have limited power and computation resources, making complex security schemes infeasible. In particular, various properties, such as confidentiality, integrity, and privacy, must be ensured. Security policies and mechanisms should be specified to guarantee privacy. They should be configurable to suit individual ways to control which of their personal data is being collected, who is collecting those data, and which operations will be performed on such a data. However, in IoT, there is no such infrastructure or servers to manage authentication that achieves the appropriate security policies. Thus, in order to address above issues, SDN-based security solution can be proposed for IoT.

2.4.5 SDN IN EDGE COMPUTING

With the recent trend of IoT devices with diverse and new applications require us to push the data, computation, storage, and application services close to the end users. This concept of bringing cloud-like infrastructure resource and services close to the end-devices is well known as fog computing or edge computing. Edge computing is characterized by its proximity to end users, its dense geographical distribution, and its support for mobility, which provides low latency, location awareness, improved QoS, and heterogeneity support. However, configuring and maintaining these edge computing services for billions of heterogeneous devices is exacerbating the current network management problem. In order to address this problem of edge computing, we need to have fully automated and flexible software solution for network management.

Network Function Virtualization (NFV) is arguably the most appropriate solution in this regard of edge computing [169]. NFV is a technology that provides the ability of dynamically deploying on-demand network services (e.g., a firewall, NAT, a router, a billing service etc.) or user service (e.g., database) whenever and wherever needed. The key idea of NFV is to virtualize the network functions (e.g., NAT, firewall, load balancer etc.) and implement them in SDN-enable virtualization infrastructure. The focus of NFV technology is mostly focused on infrastructure networks. Recently we have seen several works [83] that have used this technology to redesign the cellular Core Network. Thus, the combination of NFV and SDN can bring new architecture design to mobile or wireless networks.

2.5 SUMMARY

Existing SDN work in wireless network domain, mostly targeted to solve a specific network management problem from infrastructure point of view. There is no intension of the previous work to provide flexible and programmable SDN-like capabilities at the end-devices. Furthermore, there is no plan to bring the last-hop of wireless network under the control of existing SDN framework. Very few of the work proposed to have SDN control capabilities in wireless access points. However, instead of using standard software and protocol, these work propose to have specific protocol and software to control the access points. Such incoherent design between wireless and wire part of the network diminish the acceptability of the proposed solutions. Therefore, in this thesis, we work towards the novel idea of pushing SDN-like capabilities all the way to end-devices. Thus, we bring the last-hop under the control of existing SDN framework. In order to do that, we design and develop APIs that provide programmable abstraction of the wireless network edge. This extension of the SDN framework is called wireless extension SDN (weSDN). In next Chapter, we describe details about the weSDN framework and it's architecture.

CHAPTER 3

WESDN: WIRELESS EXTENSION OF SDN

3.1 WESDN OVERVIEW

There is a growing interest of bringing SDN based design concept targeted to different types of wireless networks that includes home WLAN, enterprise WLAN, and cellular network. Among these networks, some are in *controlled* environment, where we can have SDN-based open and programmable control and monitoring capabilities on wireless network infrastructures (e.g., enterprise WLAN, Home WLAN etc.). On the other hand, some networks are in *uncontrolled* environment, where we do not have open access on network infrastructures (e.g., Wi-Fi in coffee shop, WLAN services in any public places). In both environments, we wanted to improve and ensure the QoE of end users. In order to have that, we need better monitoring and controlling capabilities on end-devices. Therefore, we propose to push SDN control capabilities all the way to end-devices.

Existing SDN framework uses Open vSwitch (OVS) [171] in network devices (e.g., switches/routers, APs etc.) to have monitoring and control capabilities. Similarly, in order to bring end-devices under the control of SDN framework, we propose to have OVS in end-devices. We believe this design choice of SDN framework will provide coherent and simple control of both end-devices and network devices of wireless network. However, OVS only support the Ethernet interfaces not the wireless interface. Therefore, we propose to extend the OVS to support the wireless interface. Thus we wanted to bring end-devices, as well as the last-hop of wireless network, under the control of SDN framework. In this chapter, we discuss details about the proposed framework, called *weSDN*[1] (wireless extension of SDN), that extends SDN control capability all the way to end devices to support client↔network interaction capabilities and new services.

Figure 5 shows the overall architecture of weSDN framework, wherein data plane, we have end-devices in addition with wireless Access Points (APs) and switches/routers. In weSDN, we extend the OVS, called “OVS wireless extension”, to support the wireless interface of both the APs and the end devices. Thus we bring the wireless interface of both the APs and end devices under the control of SDN. Typically, the SDN controller

[1] Pronounced as ‘*wee*-SDN’

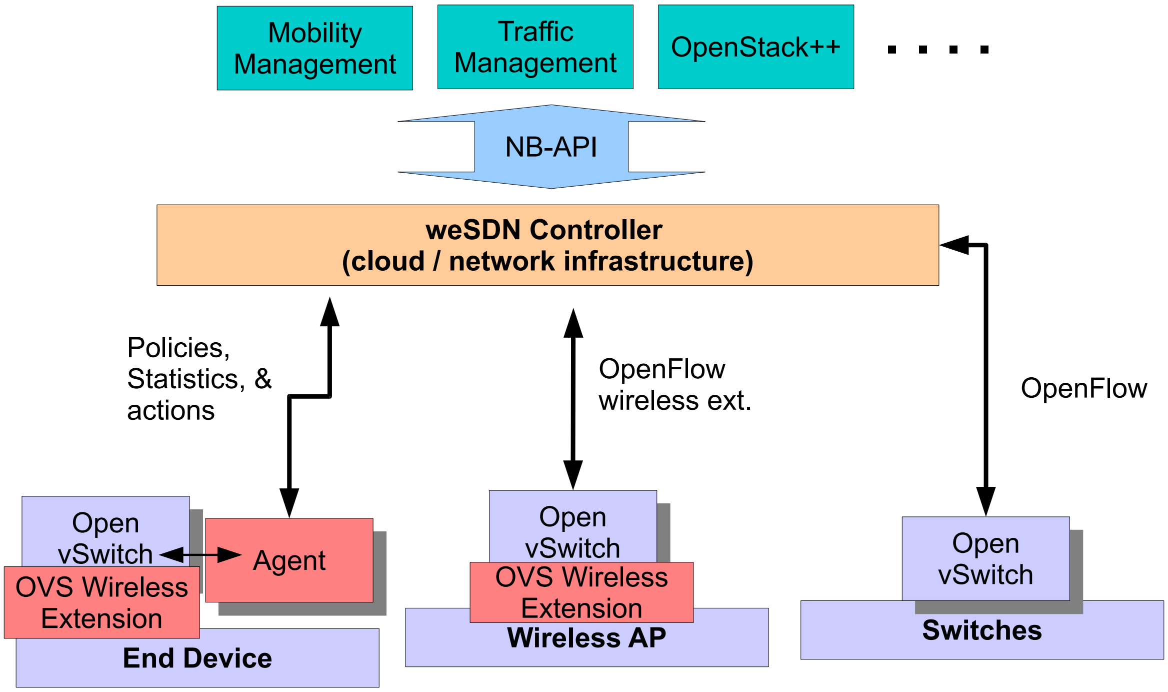


FIG. 5: Overall architecture of Wireless extension of SDN (weSDN) Framework.

uses OpenFlow protocol to collect per-flow statistics and apply actions on OVS. In weSDN, we extend OpenFlow protocol to collect a new set of per-flow statistics and apply actions on end devices or wireless APs. This extension, called “OpenFlow wireless extension”, provides new APIs to weSDN controller or to the “Agent” to monitor and control per-device/per-application flows for better wireless network management.

This extension of OpenFlow protocol or Southbound API, between the controller and the wireless APs, create new functionalities in the SDN control plane for controlling the wireless network. Furthermore, weSDN controller can create Northbound API for making the wireless network functionalities available to the network application developer. For example, the network module (i.e., Quantum) of OpenStack/OpenStack++ can directly talk with weSDN controller through Northbound APIs to get real-time status about the last hop wireless connection between the end device and the AP. This information could help the cloud application to adapt their services based on the wireless network condition of the end device. Note that, weSDN controller is a part of SDN control plane. In *controlled* environment, weSDN controller resides in network infrastructure, and in *uncontrolled* environment, weSDN controller resides in cloud.

In weSDN framework, weSDN controller interacts with the "agent" that runs in end devices to collect statistics and apply network policies. However, unlike wireless AP, "agent" in end devices do not use standard OpenFlow protocol to communicate with the weSDN controller. This is because, end device and weSDN controller use in-band control channel for their interaction, which basically goes over a shared wireless medium. Therefore, in order to reduce the overhead of control communication over wireless medium, openflow is not used for end-device ↔weSDN controller connection. Instead, "agent" is used as a proxy for the OVS that is running in the end-device. "Agent" collect statistics from the OVS and summarizes them before sending it to the SDN controller. Thus, the SDN control plane associate with the end-device, can easily obtains the client application's network demand and uses it for applying proper network policies, such as traffic scheduling, End-to-End QoS control, security settings etc. Finally, based on the given network policies, "agent" applies corresponding actions or commands on the OVS or the *scheduler*.

3.2 WESDN ARCHITECTURE

weSDN framework allows the wireless network to be managed in the unison with their wired counterparts. In addition, it provides the wireless network edges to have the programmable capability to address the complexity, dynamic nature, and the uncertainty of the wireless network. The core component of the weSDN framework is in wireless APs and end devices. In this chapter, we specially focus on describing the detail architecture of weSDN framework for wireless APs and end devices.

As shown in Fig. 6, weSDN framework has three components both in end devices and wireless APs: (1) Scheduler (Linux qdisc) (2) Flow manager (e.g., OVS kernel module, XFRM framework), and (3) local controller (OVS client, application-flow mapping). However, in terms of functionalities and implementations, these components might very between wireless APs and end devices. In weSDN framework, we have another component, called weSDN controller that interact with the end devices and the wireless APs. This component can reside either in cloud (uncontrolled environment) or in the network infrastructure (controlled environment). In following subsections, we describe details about the components.

3.2.1 FLOW MANAGER

The main objective of the Flow manager is to apply per-flow policies and to collect per-flow statistics. This component consists of a software OpenFlow switch, (i.e. OVS kernel module[2]), XFRM (i.e., IPsec) framework, and weSDN extension. In existing SDN, OVS measures only per-flow statistics such as packet count, and byte count. In weSDN, we extend the OVS to collect additional metrics, such as packet sizes, inter-arrival packet time, burst duration, data rate and inter-burst time for the flow manager. The flow manager sends these additional statistics and metrics to the SDN controller through the local controller for airtime traffic management, load balancing, for example, in order to minimize the scheduling latency experienced by real-time applications.

Flow manager leverages OVS to apply policies per-flow, such as correct QoS markings (IP DSCP/TOS)for end-to-end QoS provisioning and correct mapping to 802.11 QoS queues, (e.g. ensure that no P2P traffic gets queued in the Voice queue), access control rules to allow or block or modified the packets of a certain flow. Furthermore, in flow manager we extend the OVS to provide application or flow-type aware policies. In addition, flow manager leverages the XFRM framework to apply per-flow security policies (i.e., IPsec) to secure end-to-end communication.

In flow manager, we propose to extend the OVS further to interact with WiFi driver to configure, for example, its power-save settings, transmission (TX) rate, TX power and also to collect 'per-client' or 'per-flow' wireless statistics such as RSSI, data rate, retransmission count, TX mode and drop count. This wireless extension called weSDN extension, also open up new APIs that allow the SDN control plane or local controller to better manage airtime resource. For example, VoIP flow in one end device suffering hidden-interference will show high drop counts and we can schedule the client in a different time window to avoid interference. The 'per-flow' stat is useful because we can prioritize a VoIP flow over a p2p flow when both suffer high wireless drops.

3.2.2 SCHEDULER

The scheduler is Linux Qdisc [43] that applies prioritization and rate-limiting to incoming/outgoing flows. Note that, the Qdisc is the major building block of Linux network stack on which all of Linux network traffic control is built. OVS implementation today already leverages Qdisc to implement OpenFlow QoS APIs – prioritization and rate-limiting. To

[2] OVS kernel module is called *Datapath*

implement airtime scheduling, which is specific to wireless, weSDN can extend either Qdisc or WiFi driver. (We like to compare the two options later.) This “weSDN extension” module, either in Qdisc or WiFi driver, starts/stops dequeuing of the outgoing flow based on the airtime schedule given by the control plane. Furthermore, “weSDN extension” module in scheduler can also interact with the wireless driver to control the dequeuing event based on the number of packets exists in the driver buffer. Thus, scheduler makes sure not much get packet get piled up in the driver buffer. Scheduler can also be used to apply traffic obfuscating technique on the network flows, such as by changing the inter-arrival time of the packet. In order to do that, scheduler control the dequeue event of packet from the Qdisc.

Ideally, the ability to schedule airtime usage of ‘each flow’ or ’each client’ is desirable for finer-grained airtime/QoS control and we can further extend Traffic Control (tc) APIs to signal per-flow schedules down to the “weSDN extension” module. But this would complicate the controller and scheduler implementations. Though we are open to the possibility of per-flow airtime scheduling, at this point, we assume airtime scheduling is done per-device. The design and implementation of the scheduler will differ between end device and AP. Where in AP, we are in the favor of maintaining per-end device vport in the OVS, and maintaining one Linux Qdisc per-vports. Thus we can control the per-device downlink traffic from the AP.

3.2.3 LOCAL CONTROLLER

Local Controller is a user space application that consists of OVS user-space client (i.e., `ovs-vswitchd`,`ovs-ofctl`, `ovs-vsctl`) and traffic control (i.e., `tc`) to control the Flow Manager and Scheduler. The Local Controller, in end devices, provides application-awareness and generates *flow-to-application mappings* by monitoring active network sockets (`netstat` or `ss` logs) and their `pid/uid` bindings that are available in Android and other major Linux-based operating systems. The local controller in end devices applies application-aware policies by inserting flow rules into flow manager corresponding to each application. Note that in end devices, the local controller knows which socket/flow belongs to which application, it can easily apply appropriate application-specific policy, assigned by the SDN controller or set directly by the end user [195]. Local controller can also interact with other user-space network management daemon (i.e. IKE daemon, wpa supplicant) to configure the network settings, and apply policies. For example, local controller can interact with IKE daemon, in addition with *flow-to-application mappings*, to apply

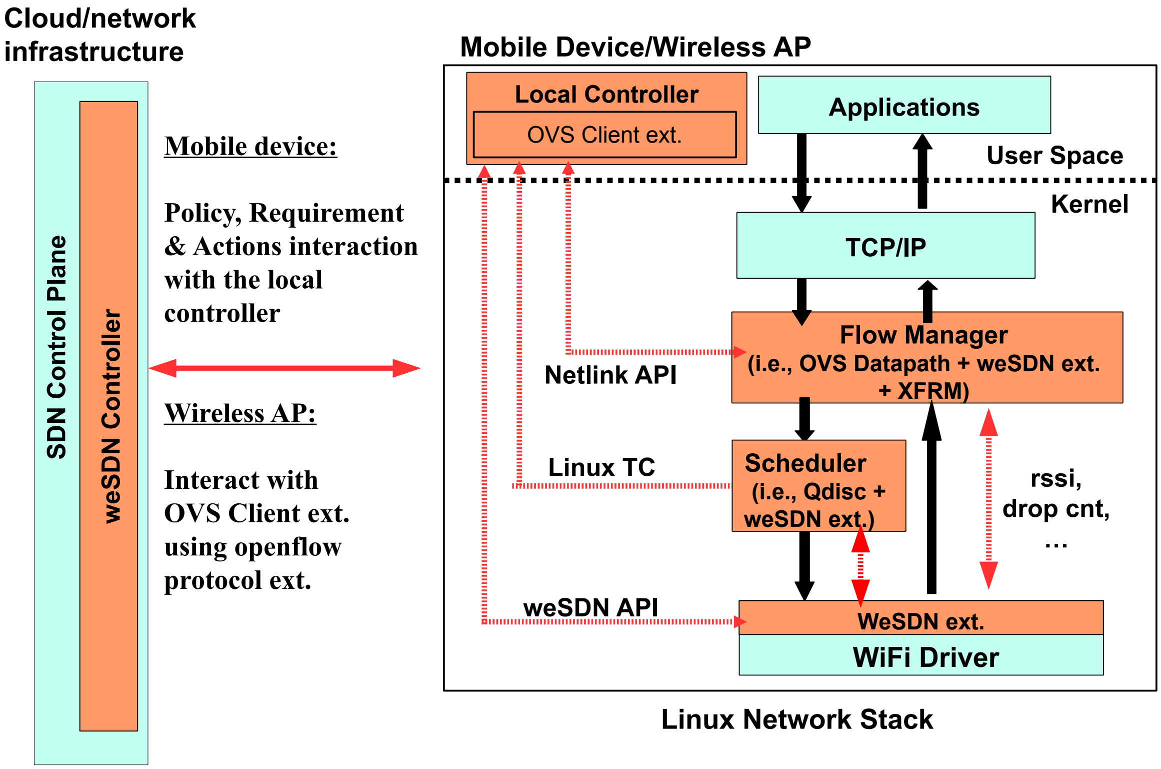

Cloud/network infrastructure
SDN Control Plane
weSDN Controller
Mobile device:
Policy, Requirement & Actions interaction with the local controller
Wireless AP:
Interact with OVS Client ext. using openflow protocol ext.
Mobile Device/Wireless AP
Local Controller
OVS Client ext.
Applications
User Space
Kernel
TCP/IP
Flow Manager
(i.e., OVS Datapath + weSDN ext. + XFRM)
Netlink API
Linux TC
Scheduler
(i.e., Qdisc + weSDN ext.)
rssi, drop cnt, …
weSDN API
WeSDN ext.
WiFi Driver
Linux Network Stack

FIG. 6: Detail architecture of weSDN framework for Wireless AP and end-device.

application-aware IPsec policies per-flow. Local controller use extended OpenFlow APIs to read per-flow stats from the flow manager. In addition, local controller use Linux Traffic Control (TC) or weSDN API to control the "weSDN extension" depending on its location either in Qdisc or WiFi driver.

3.2.4 WESDN CONTROLLER

This component controls the Scheduler and the Flow Manager through Local Controller. In weSDN, the interaction between the wireless AP and the weSDN controller happens through OpenFlow protocol. In order to support the addition interaction for the wireless network, OpenFlow protocol has "wireless extension" module. However, the Local controller in end devices does not use OpenFlow to coordinates with the weSDN controller.

Instead, weSDN controller communicates with the local controller of the end device, as server-client, in following manners.

local controller → weSDN controller: Local controller “aggregate” currently running applications’ flow resource (e.g., airtime demand) and QoS requirements (e.g., latency tolerance), and send the aggregated requirements to the weSDN controller. This client-side aggregation reduces control overhead and improves scalability. This also protects user privacy because the local controller can provide network requirements to the global controller without revealing which applications are running on the device. This is in contrast to current network-based application detection solutions that employ Deep Packet Inspection (DPI), looking into the user payload.

weSDN controller → local controller: weSDN controller provides per-application policies (e.g. access control policy, security policy etc.,) and QoS profiles to the local controllers. For example, an enterprise IT may allow a certain VoIP application for guest users but not for employees. The IT admins may have pre-profiled a QoS spec for certain video applications. In addition, weSDN controller applies proper actions back to the local controllers. For example, based on the received per-client aggregate airtime demands, the SDN controller provides the schedules to the local controller, which then program scheduling mechanism in the scheduler.

weSDN controller may also directly manage the device components (OVS and qdisc) using OpenFlow, and since OVS supports multiple controllers, it can be managed by both local controller and weSDN controllers. Unlike wired SDNs that typically have an out-of-band control channel between switches and the controller, we have to use the same wireless interface for data and control; and reliable and scalable in-band control is hard in wireless due to power-saving and interference. Thus, we argue that *the weSDN controller is better to communicate only with the local controller, which runs as a proxy for the device components*, in semi-real-time synchronously e.g., once every ‘N’ beacon cycles, and also asynchronously as needed, e.g., to adapt to sudden changes in application demands. Besides that, there can be use-case scenarios, where the local controller can act independently to interact with the flow manager and the scheduler without the input or the feedback from the weSDN controller.

3.3 WESDN SERVICES

Based on the weSDN framework, we envision to have new services to address the complexity of the modern day wireless network. In following subsections, we describe some example of such new services.

3.3.1 WLAN VIRTUALIZATION

Wireless LANs (WLANs) are becoming ubiquitous not only in homes and enterprises but in public venues such as coffee shops, hospitals, and airports. Along with this trend, the need to virtualize WLAN infrastructure is surging. This is in order to enable more effective sharing of wireless resources by a diverse set of users with diverse requirements. For example, mobile carriers want to obtain a guaranteed share of RF resources on public WiFi infrastructure (e.g., in an airport) to offload their subscribers' data traffic to WiFi. Enterprises/Homes want to virtualize their WLAN infrastructure to create differentiated service networks, e.g., an employee/parents network vs. a guest/kids network. This kind of virtualization is already happening in cellular networks, e.g., Mobile Virtual Network Operators [12], and there is a clear need to extend this capability to WLANs.

The killer application thus far for SDN has been network virtualization, applied primarily to the wired Ethernet infrastructure. So, why not apply the same SDN mechanisms to virtualize the WLAN infrastructure using SDN-enabled wireless access points (APs)? There is a fundamental problem with applying the existing SDN framework to virtualizing WLANs and providing each virtual network with some specified network resources. The fundamental problem stems from the fact that the last WLAN hop is a shared wireless medium and most WiFi clients today support only contention-based MAC, leaving no means for AP to control client-to-AP uplink transmissions. Consider an example of an enterprise wanting to guarantee 50% share of airtime for employee devices. If there are nine guest devices and only one employee device, all with always backlogged traffic to send, the employee will get only about 10% share of air-time.

Without an uplink control capability, SDN-enabled APs can only control wireless resource (channel and airtime) usages for AP-to-client downlink transmissions [71, 185] and also apply SDN policies on the up-stream traffic already received from the clients, but this is not enough to realize WLAN virtualization that requires guaranteed wireless resource share, for both uplink and downlink, for each network slice. Without resource guarantees, the performance experienced by end users will be highly unpredictable even with high transmission rates of recent 802.11n/ac. Therefore, extending the SDN capability to the end devices allow the weSDN to have such virtualization capability through controlling both uplink and downlink wireless resource.

3.3.2 APPLICATION-AWARENESS NETWORKING

The increasing in the number of mobile devices and corresponding various mobile applications and services, network traffic and is growing significantly in addition to introducing new traffic types. Typically, mobile applications generate various types of flows for different objectives. For example, Skype applications can do the voice, video, screen sharing, file sharing, and Instant Messaging (IM) flows as well as the background traffic for signaling, analytics, advertisement, etc. These applications diversity and various flow types have different QoS requirements and security/resource implications. In addition, different users may have different policies and requirements even for the same category of applications and flow types. For example, giving the possibility of information leakage from the commercial VOIP applications, enterprises tend to block these applications or some of their flow types while enabling more robust and secure business VOIP applications and their flow types. The network administrator might require controlling the same traffic flow with different priority/policy based on the locality. For example, in a campus WLAN scenario, streaming movies applications such as Netflix and Amazon are throttled down in specific positions on campus such as the library or the study places while, on the other hand, get fair bandwidth in other places such as dormitory. Thus, the ability to accurately recognize individual applications and the various traffic flow types in real-time within each application is very critical to have greater visibility and control over the traffic generated from the end devices. In weSDN, the design of extending the OVS and OpenFlow to extract new traffic flow feature can allow us to have fine-grained and real-time application-awareness at the network edge.

3.3.3 SECURING WIRELESS NETWORK EDGES

In addition to many regular applications, the patient runs many sensitive mHealth apps on their personal mobile devices. According to the report of March 2013 from Research2Guidance [136], there were about 97,000 mHealth apps across 62 app stores that targeted healthcare professionals, medical or nursing students, and patients. The mHealth apps running in the mobile devices tend to use Wi-Fi, as a prominent network interface, to send a large amount of sensitive to the internet. Unlike the regular apps, those mHealth apps might require extra security for their traffic between the AP and the mobile device. However, due to the broadcast nature of Wi-Fi links, wireless traffic are exposed to any eavesdropping adversary within the WLAN. Therefore, studies show, many physicians, as well as patients, feel vulnerable and insecure to use sensitive mHealth apps in their personal smart device. Such concern could limit the willingness and potential benefit of using mHealth apps. Therefore, it is important to address the security threat of using Wi-Fi.

The weSDN framework can provide a transparent and configurable technique of securing the wireless traffic between the mobile devices and the AP. The weSDN can address the security concerns of the WLAN such as, securing the unencrypted sensitive data traffic between the mobile device and the AP, apply traffic shaping technique to hide the side-channel information about inferring user's input to the sensitive apps (i.e., mHealth apps), and analyzing per-client MAC layer traffic statistics to infer traffic jamming attack from unusual network activities. Furthermore, the application-awareness capability of the weSDN allows us to configure security option only on the sensitive app's traffic without impacting the regular app's traffic. Such flexible way of just securing the targeted apps' traffic make the user comfortable and confident.

3.3.4 SUPPORT MULTIPLE NETWORK INTERFACES

Nowadays smartphones are equipped with multiple wireless interfaces (Wi-Fi, LTE, HSPA+, and Bluetooth). Unfortunately, the current design of Linux network stack implementation in off-the-shelf mobile devices are not capable of utilizing multiple interfaces. Furthermore, not necessarily all network interfaces perform (i.e., bandwidth, delay, packet loss) similarly all the time. Therefore, we require a dynamic and programmable system to leverage all available network interfaces according to their merit. There is a large body of work that seeks to exploit the diversity of connectivity options in mobile wireless networks. Unlike many of the previous work, we wanted to provide the end device to have the flexibility of choosing multiple wireless interfaces. Based on the weSDN framework, one can create one internal virtual port (vport) and bind all the available wireless network interface with the OVS wireless extension. The applications in end device only send the packets to the vport, later based on the network demand and performance availability the packet is forwarded to the right network interface. Before sending to the actual physical network interface OVS make appropriate changes in the packet header.

When an application sends a packet, the fate of a packet, in terms of the interface it is sent on and in what order, is determined the moment the IP addresses are added and the packet is bound to a particular interface. Therefore, when the mobile devices switches/handoff to new wireless interface or new Wireless AP, an on-going TCP connection

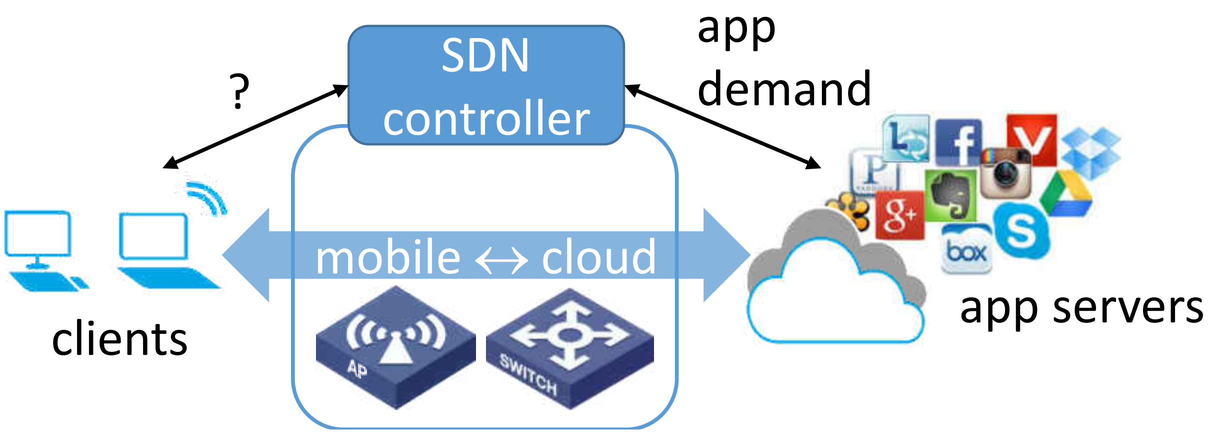


FIG. 7: Mobile-Network-Cloud interactions.

cannot be handed over to a new interface, without re-establishing state. Often wireless driver maintains large packet buffer (i.e., 100 packets) both in APs and end devices. So when the end device moves from one wireless AP to another, all the downlink packets in the buffer gets dropped, which might create TCP timeout event. Based on the weSDN framework, in wireless AP, one could reduce the number of packet in the wireless driver buffer by controlling the dequeue event of the traffic scheduler qdisc. Later, in handoff scenario, the qdisc packet can be forwarded to new wireless AP using OVS actions.

3.3.5 MOBILE(CLIENT)↔SDN(NETWORK)↔CLOUD/CLOUDLET INTERACTIONS

Recent mobile cloud/cloudlet applications require guaranteed network performance more than conventional client-server applications as such mobile applications incur tight and real-time interactions with cloud and sometimes offload network-intensive workloads to cloud [41] or cloudlets [145]. In a wired infrastructure, SDN APIs is used to guarantee performance by dynamically coordinating network edges, mostly Ethernet switches. Recent SDN controllers also interact directly with cloud-based application servers to communicate application specific requirements. This SDN (network)↔cloud (server) interaction, together with the existing mobile (client)↔cloud (server) interaction [34], aims to deliver application-optimized network performance and eventually bottleneck-free computing experience to users.

Mobile (client)↔SDN (network) interaction is very essential to complete the loop between mobile, SDN and cloud (Figure 7). Knowing the application requirement from the cloud (server), it is important for the SDN (network) to provide performance guarantee to the end device. Similarly, Mobile(client)↔SDN(network) interaction allow SDN to provide network condition to the cloud application Unfortunately, the control of the existing SDN frameworks stops at the network edge, either at enterprise edge switches [89], datacenter hypervisor virtual switches [171] or home routers [126]. In wired environments, SDN policy can be effectively enforced on the traffic to/from end devices without modifying the end devices since the last hop is a point-to-point full-duplex link and transmissions from end devices do not interfere with each other. However, unlike the wired access network, the wireless client in its interaction with the network needs to have fine-grained control and be able to selectively and dynamically apply traffic and wireless resource management techniques to enhance user applications experience. The weSDN framework of extending the SDN paradigm to mobile clients could provide optimal network performance between the cloud/cloudlet and the wirelessly-connected clients.

3.3.6 MONITORING AND CONTROLLING THE BANDWIDTH OF MULTIPLE VIDEO PLAYERS

Increase of video traffic is creating big challenges for video providers in guaranteeing a satisfactory level of viewing experience to the end users. In addition to the huge increase in the resources required to serve more streaming demands, the unstable nature of wireless links and the frequent changes in network loads create another obstacle toward providing a high quality video service. In order to overcome most of the above challenges, HTTP adaptive video streaming technology was introduced, along with other great features for streaming videos. Therefore, most of video providers nowadays such as YouTube, Netflix, and Vimeo have already adapted this technology in streaming videos.

However, despite the aforementioned features, practical implementation, represented in today's commercial players, reveals that viewing quality can still suffer with this protocol under certain situations and conditions. For instance, our measurements show that YouTube app can suffer from several serious issues that would directly impact the QoE of the end user including instability in the perceived quality, a long starting-up time, bandwidth underutilization, and unfairness between the users. Mostly these issues are likely experienced when multiple concurrent players compete over the same bottleneck due to the intermittent traffic generated by streaming protocol as indicated by several studies [3, 74, 82]. As confirmed by our experiments, most of these issues are mainly caused by the aggressive competition between the players for the available network resources. Therefore, any future effort that aims to enhance the performance of video players and the viewing quality for all clients, should concentrate on mitigating this competition in such a way that ensures the fairness among the player and, maximizes the utilization of the available bandwidth.

Having multiple users concurrently streaming videos through the same WiFi access point (at shopping center, coffee shop, etc.), or the same base station of cellular network is a very common scenario. Therefore, looking beyond an individual case, and make the optimization global (over multiple concurrent streams) is essential for an efficient streaming solution. To this end, weSDN can apply possible technique, for instance, is to monitor and control the bandwidth utilized by each stream at the end-device. Each time a new stream joins the network, the system should instantly react and reallocate a fair portion of the bandwidth to the new stream. This technique might require reallocating the bandwidth, and distributing it again among all players. However, to avoid major degradation in the quality of the existing (old) players, the system should be aware of the conditions and status of all players and the characteristics of their flows before performing bandwidth reallocation.

3.4 ADVANTAGES OF HAVING SDN CAPABILITY AT END-DEVICES

In this section, we describe some of the benefits that the SDN on end devices can bring to users and network illustrated with several practical examples.

Removing the dependency on network infrastructure: Existing network management techniques and policies are currently enforced from network infrastructure. Consequently, the policies that are applied on a certain network might not be applicable in other networks. For example, the WLAN in an enterprise environment usually has different policies than the WLAN in public spaces. However, there are several scenarios where users need to apply certain policies on the applications running on their mobile device regardless of what network they connected to. For example, the parents can set a policy at home gateway that allows the devices of their children to only access the internet (or social networks such as Facebook or Twitter) during the evening. However, when the children leave home and join other public networks (e.g., schools network), they can have an access to the internet (or social networks) at any time. Therefore, having the SDN capabilities on the end device can persistently enforce the parent policies regardless of the network their children might connect to. Moreover, having SDN capability with right set of APIs on an end-device can provide the flexibility for users to customize the policies for each network. For example, consider a scenario where a user uses an application that shares GPS location with the server. Now, the user wants to block the access to this application when he/she is at home. However, whenever the user is in office location, he/she allows the access of the application.

Device context-aware network management: One of the major shortcomings of infrastructure based wireless network management is lacking the awareness of the characteristics and the context of the end devices. This sightless management, in fact, makes the current systems unable to provide true end-to-end QoE for end users which can result in wasting network and end-devices resources in addition to the inability to optimize the QoE for end users. For example, consider a scenario where two users, one with smartphone playing a video in the background and the other watches a sports game on a tablet, stream videos over the same AP. Now based on the context and characteristics of these two devices, it will be more efficient, in terms of QoE for both users and resource utilization, to allocate more bandwidth to the tablet to have better resolution than the smartphone. This is obviously because the tablet has a bigger screen and thus needs more resolution in addition to the smartphone user plays the video in the background (e.g., listens to music while running other app). Therefore, extending SDN capability to the end-devices could provide such context-awareness properties for smarter wireless network management.

Fine grained control and reliable monitoring capability: In end devices, SDN-like paradigm allows to have application-aware flow control and monitoring capabilities. As a network flow can be easily bound to its application using socket connection listener tools (e.g., netstat, ss) on the end device, it is possible, with OVS and OpenFlow protocol, to both control and monitor the network flows of different applications. Furthermore, services like SmartEdge (chapter 5) could even provide flow-type awareness at the end-device as many of mobile applications generate various types of flows (e.g., video chat, voice chat, video stream, etc.). For example, Skype users can do voice calls, video calls, screen sharing, file sharing, and Instant Messaging (IM) as well as background traffic for signaling, analytics, and advertisement. These different flows intuitively have different network loads, QoS requirements, and security/resource policies, thus it is very critical to the QoE to accurately and efficiently recognize the running applications and their various traffic flows in real-time. Such capabilities are potentially valuable for properly applying network policies and ensuring QoS, security, and resource efficiency in addition to allowing for better next-generation Internet infrastructure design and efficient content distribution systems.

Enabling user/app-network interaction: As managing the network today is getting more complex with the explosion of smart devices and applications, ensuring a good level of QoE for end users becomes a challenge. In fact, all the existing techniques for inferring the end users' level of satisfaction and preferences fail to provide an accurate measurement under various conditions and scenarios. For example, consider a scenario when a student needs to upload a large submission for an impending deadline. Since the uploading is typically treated by the network as a background traffic, the student might experience a large uploading delay and not be able to submit in time. However, the network could treat the uploading as the most important task of any other network activity if we enable the student to interact with the network and express his/her needs and preferences resulting in significant improvement in the network performance and thus QoE. Moreover, extending the SDN to the end devices can also open the door for application-network interaction which can enhance the performance, in many cases, without user intervention. For instance, it is not possible for the network to detect whether the user who plays a video is currently experiencing a stall in the playback. The video application, On the other hand, is not only aware of this situation, but also can prevent the stall before it happens through the feedback sent to the network. Hence, enabling user/app-network interaction can derive the network resource management to optimize the user QoE. Typically, the objective of having SDN capability at end-devices is to collect and provide information on the current network conditions to both users and application developers, gathering both users' and apps feedbacks, and then developing novel wireless protocols based on these feedbacks.

3.5 SUMMARY

In the next three chapters, we describe the three services, WLAN virtualization, and application-awareness networking and securing wireless network edges that we have implemented based on weSDN framework. In this chapter, we describe different components of weSDN architecture in general. In following three chapters, we elaborate the design and the implementation of these components for the three different services.

CHAPTER 4

PTDMA: WLAN VIRTUALIZATION

4.1 WLAN VIRTUALIZATION

WLAN virtualization is becoming a key to enable a more effective sharing of wireless resources by a diverse set of users with diverse requirements. For example, mobile carriers want to obtain a guaranteed share of RF resources on public WiFi infrastructure (e.g., in an airport) to offload their subscribers' data traffic to WiFi. Enterprises/Homes want to virtualize their WLAN infrastructure to create differentiated service networks, e.g., an employee/parents network vs. a guest/kids network. This kind of virtualization is already happening in cellular networks, e.g., Mobile Virtual Network Operators [12], and there is a clear need to extend this capability to WLANs.

The existing SDN mechanism for virtualizing the wired Ethernet infrastructure, for example using OpenFlow-enabled wireless access points (APs), is not directly applicable to virtualize the WLANs. Because, the last WLAN hop is a shared wireless medium and most WiFi clients today support only contention-based MAC, leaving no means for APs to control client-to-AP uplink transmissions. Consider an example of an enterprise wanting to guarantee 50% share of airtime for employee devices. If there are nine guest devices and only one employee device, all with heavy backlog traffic to send, the employee will get only about 10% share of air-time. Previous approaches for APs to control clients' uplink 802.11 or TCP transmissions were either intrusive (dropping uplink 802.11 frames or downlink TCP ACKs), unfair by causing starvation (greedy use of Self-CTS) or hard to be adopted (requiring changes of 802.11 protocol) [17, 156, 194]. Furthermore, most of these techniques do not provide a good resource guarantee, since they only indirectly control uplink transmission probabilities and can not prevent aggressive UDP streaming from monopolizing a link.

As a proof-of-concept, we use weSDN framework to design, implement and evaluate a WLAN virtualization service that slices end devices using a Time Division Multiple Access (TDMA) like airtime scheduling, named pseudo TDMA (pTDMA), that runs on top of 802.11 MAC. By using Linux Qdisc on end devices, pTDMA virtualizes (separates) airtime resource between network slices while minimizing contention between clients within a slice.

pTDMA also allows client WiFi interfaces to more efficiently utilize their active time and to sleep longer outside of the given transmission windows. We will present weSDN's network virtualization capability and its improved power-efficiency from our prototyping on Android phones.

4.2 PTDMA

pTDMA, using TDMA-like 'coarse-grained' airtime scheduling on top of 802.11 CSMA/CA MAC, virtualizes airtime resource between network slices. Note that we do not (and cannot) guarantee absolute interference-free airtime, which is impossible in unlicensed bands. pTDMA guarantees a share of airtime duration for each slice to 'attempt' medium access while avoiding contention between different slices and minimizing contention between clients within a slice; but it cannot completely avoid interference.

We assume the wireless "channel" resource is under control by the centralized Radio Resource Management (RRM) solutions in enterprise WLANs [9, 36, 37]. We define the role of pTDMA to manage airtime share between virtual network instances (their clients) that collocate in space and channel. RRM can compute airtime available for each channel and AP based on the measured contention and interference and feed the airtime availability to our global pTDMA scheduler.

The key concept of pTDMA is to *separate airtime slices used by different network instances* so that traffic for different networks do not overlap and wireless contention is confined to traffic internal to one network instance. Fig. 8 illustrates a simplified example of the airtime resource shared by two network instances: employee network and guest network with 50:50 time share. As a simple baseline, the entire time slice (max 50%) can be open to all clients of each network instance. The clients and APs of each network instance will contend for the medium access based on the 802.11 MAC. If the operator also wants to manage contention and airtime usage within each instance, the instance's airtime slice can be further divided into multiple time windows and scheduled based on each individual clients' QoS requirements and traffic pattern provided by the local controllers.

4.2.1 SCHEDULING PRINCIPLES

There are myriads of scheduling schemes in the literature that addressed other important

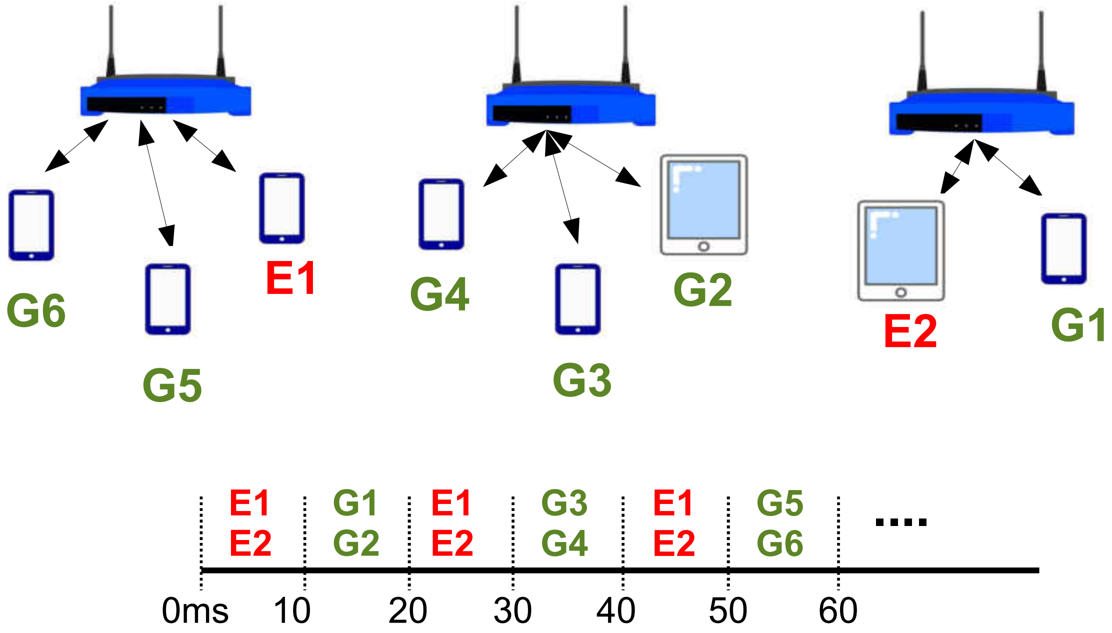


FIG. 8: pTDMA scheduling example.

factors like work-conservation, fairness, and interference [17, 19, 87, 137, 152]. For example, efficiently reassigning unused airtime resource to those in need (work-conservation) is critical. When two clients are known to have hidden interference, we can avoid scheduling the two in the same window. We rely on the previous work and future work for the detailed design of the scheduling algorithm and this paper discusses high-level principles and constraints specific to running pTDMA in WLANs. Ultimately, the scheduling mechanism is up to each network operator who can program the algorithm in the SDN controller.

Because the pTDMA qdisc operates above the WiFi driver, we can not tightly control per-packet transmission timing to a microsecond level as conventional TDMA does. Also, the tight TDMA scheduling can perform worse in the presence of unexpected interference or burst traffic, which 802.11 CSMA MAC is supposed to deal with. Thus, *pTDMA scheduling unit is not per-packet basis but is a larger time window during which a client can transmit and receive multiple packets.* (As 802.11 MAC aggregation typically use 4ms time limit, this could be a good minimum limit. We use 10ms in our prototype.) Because perfectly estimating future traffic pattern and demand is difficult, we also schedule multiple clients in a common window to help maximize channel utilization while controlling the number of contending clients. Finding a good balance between multiplexing gain and contention overhead has been studied in ref. [138] and the pTDMA scheduler can leverage the previous study.

pTDMA should carefully determine the interval between two consecutive time windows of a client to meet currently active application's delay and jitter requirement. Consider an example when video streaming and VoIP applications, running on the same client, have similar inter-burst intervals but their bursts are interleaved. pTDMA can delay one flow to match its burst pattern to the others' and schedule the client's ontime windows to fit the matched pattern; delaying the streaming flow is more desirable because a streaming buffer can absorb additional delay while VoIP has a more stringent delay requirement. Keeping the interval time constant is desirable to control jitter and to provide consistent TCP performance. In addition to the interval, the window size (scheduling unit) must be set carefully to meet various application requirements, while balancing channel utilization and contention overhead.

Optimally deciding the interval & window size and scheduling clients over time windows are challenging problems, especially when there are many network slices and client devices. In this case, scheduling all clients of each network instance in a window (the baseline approach) can simplify the problem and increase the chance to meet the basic requirement – guarantee airtime share between network instances.

4.2.2 DOWNLINK CONTROL AND POWER-SAVING

For AP-to-client downlink control, we may implement a similar pTDMA scheduler on APs but its implementation may be complicated because the AP has to control timings for every client. As an alternative, we found WMM-Power Save (WMM-PS) mechanism that triggers AP's downlink transmission of buffered data to a client by the client's uplink transmission [6]. (This is different from legacy power saving, in which a client typically waits till the next beacon to transmit or receive.) WMM-PS is a part of Wi-Fi Certification program and implemented in most client devices. *We leverage WMM-PS to indirectly confine downlink transmissions to the time window controlled by the client's pTDMA scheduler.*

Typically WMM-PS is used for VoIP or video traffic that has a regular burst pattern. Best effort applications (email, web and file transfer) are handled by legacy power saving. When backlogged packets are present for TX or RX, e.g., from bulk file transfer, most WiFi drivers stay in the Constant Awake Mode (CAM), as opposed to power saving mode, even when they do not obtain constant channel access due to contentions. Because pTDMA controls contention in each window, it allows the WiFi interface to more efficiently utilize its active time and to sleep longer outside of the ontime window. Thus, pTDMA makes WMM-PS also attractive for best effort traffic. In addition, weSDN OVS can detect two flows with their burst patterns interleaved and combine their patterns appropriately in the pTDMA qdisc such that the inter-burst time of the combined flow is maximized, increasing the sleeping time. In addition, we will show *pTDMA and WMM-PS can improve power-efficiency without losing throughput performance.*

4.3 PROTOTYPE IMPLEMENTATION

In this section, we describe a scaled-down version of weSDN framework to prototype the pTDMA service on top.

4.3.1 ARCHITECTURE

We implemented a pTDMA scheduler using Linux multiq [1] qdisc as a basis. pTDMA qdisc represents the wireless extension of the weSDN architecture (Fig. 6). Implementing it in the driver could give tighter control over airtime usage, but due to practical issues of less accessibility we prototyped in the qdisc.

In Flow Manager, we use OVS to measure the per flow statistics and also take QoS actions on per-flow per-app. We also prototyped the OpenFlow-Wireless extension by conveying WiFi stats (such as RSSI) in Linux packet buffer (`skbuff`) forwarded from the driver to the OpenFlow datapath but did not use the stats for pTDMA scheduling in this implementation. Finally, the SDN Controller communicates with the local controller to set the schedule for pTDMA qdisc.

4.3.2 CHALLENGES

Millisecond level synchronization, instead of microsecond required by conventional per-packet TDMAs, is needed to enforce pTDMA schedules across clients. Mobile phone GPS or NTP can provide such accuracy as we observed from our Android phones.

Driver buffering delay is known to be large enough to cause application performance to drop under certain conditions (Bufferbloat [60]) because WiFi (and also Ethernet) drivers typically have a fairly large (100-300 or more) ring packet buffer. The time to drain the buffer can take hundreds of milliseconds or more when the buffer is filled with max MTU

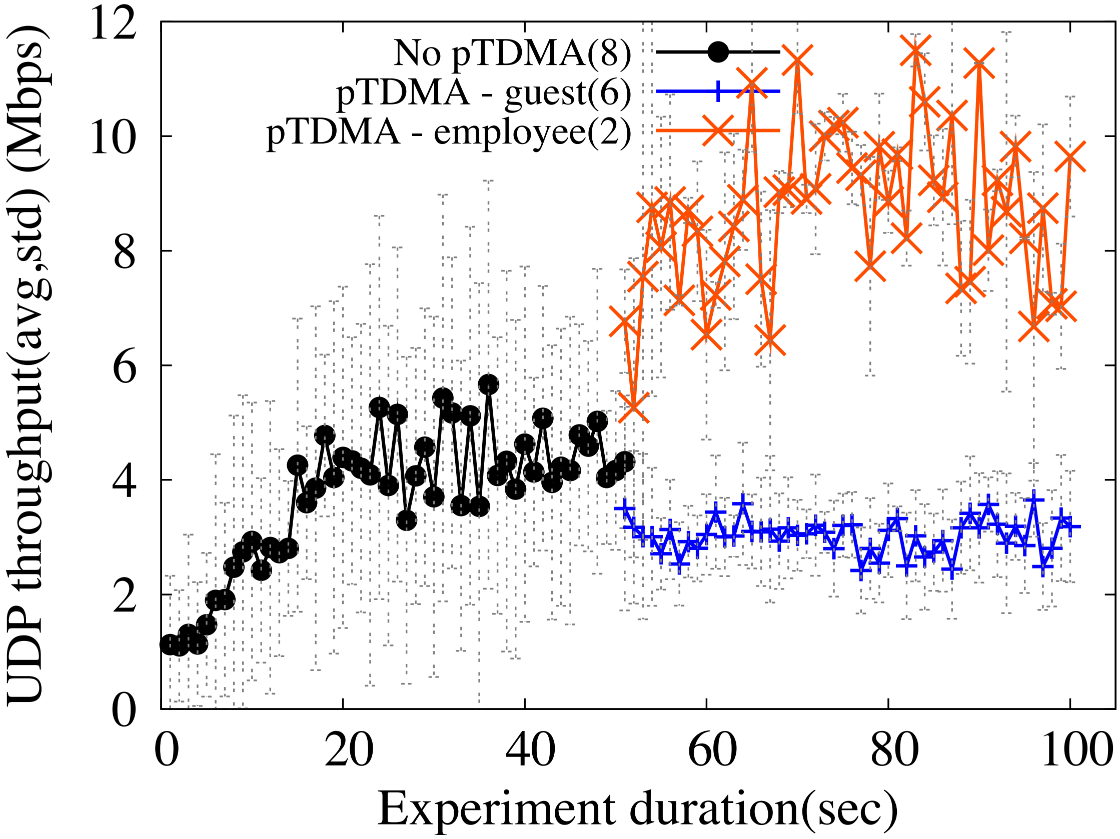


FIG. 9: Testing with 2 employees and 6 guest devices with 50:50 airtime share for evaluating UDP throughput.

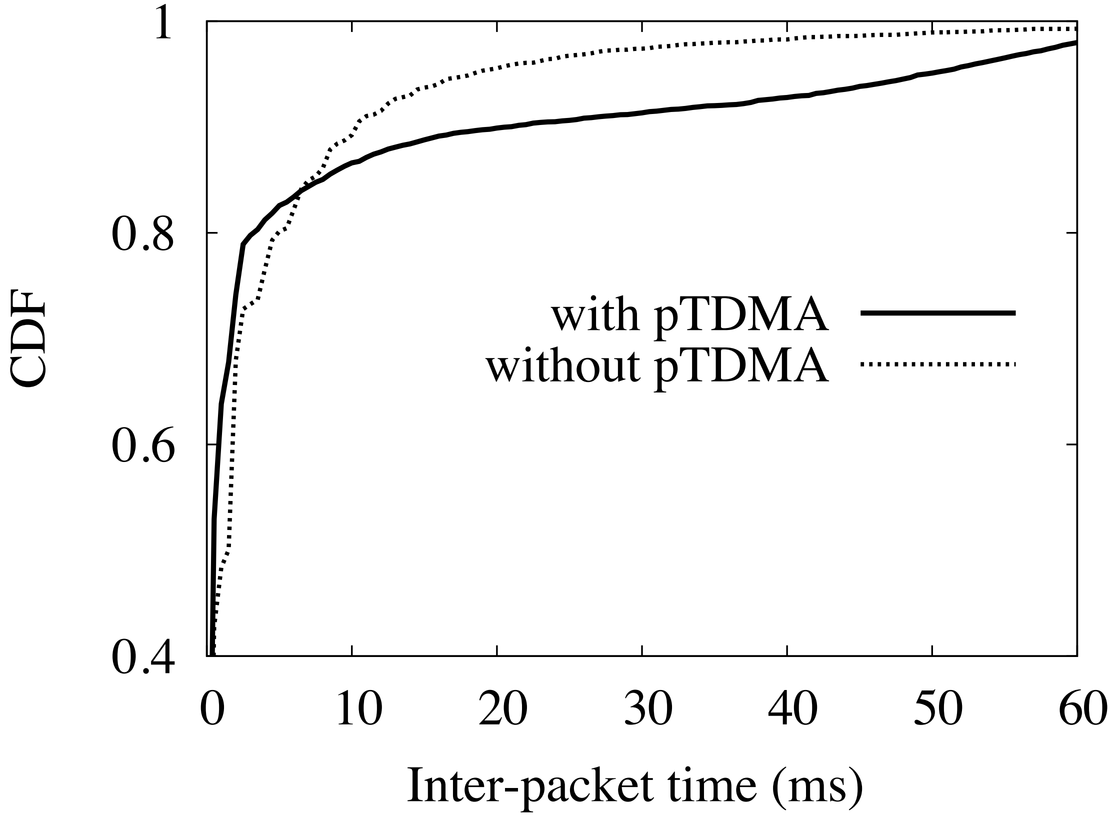


FIG. 10: Guest1 UDP packet interval.

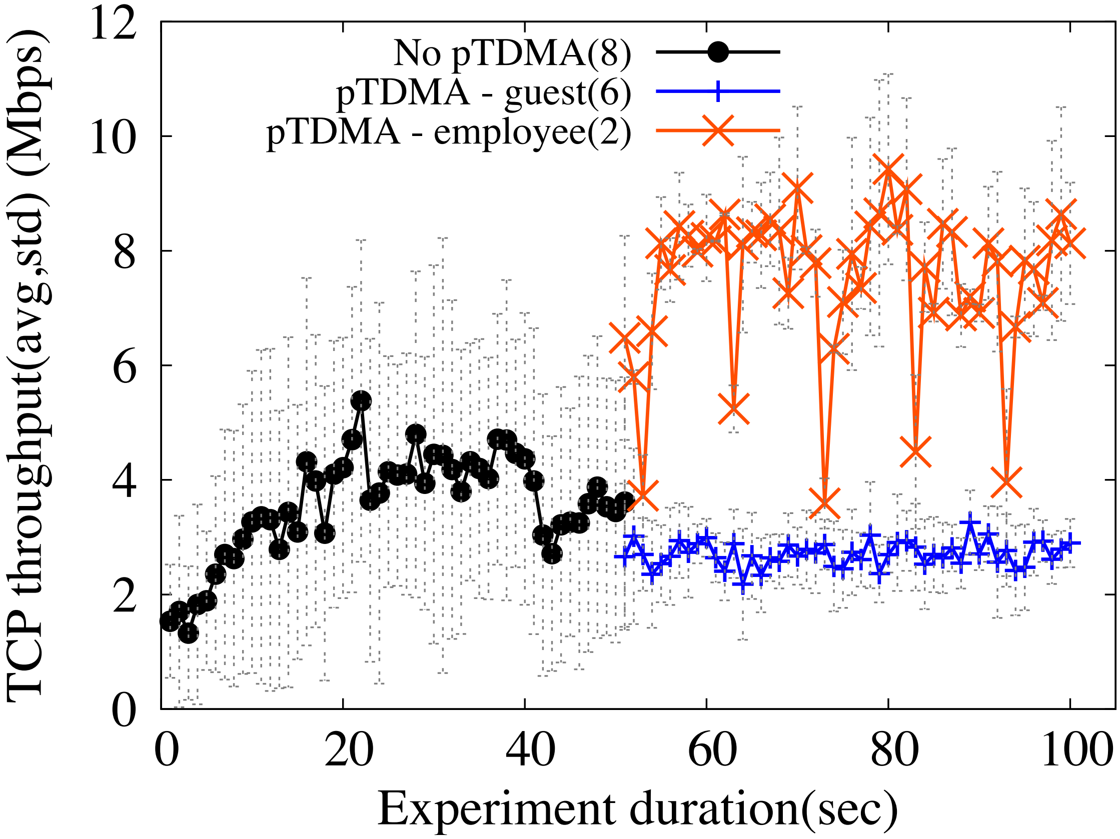


FIG. 11: Testing with 2 employees and 6 guest devices with 50:50 airtime share for evaluating TCP throughput.

sized packets and the wireless throughput is slow. This can hurt pTDMA because the driver may consume more airtime than scheduled to drain all the buffered packets even after pTDMA qdisc stops. As a solution to bufferbloat, Ethernet drivers started to implement Byte Queue Limits (BQL), where the limit is dynamically set based on number of bytes the NIC dequeued recently [39]. In our prototype, we took a similar BQL approach into the WiFi driver while providing reasonable buffer space for 802.11 MAC aggregation.

802.11 beacons: we believe pTDMA scheduling cycle does not have to be tied to the beacon interval, which can differ across APs. We do not control or coordinate beacon timing, but rather inform the beacon TX schedule to the pTDMA scheduler, which then leaves some time after each beacon, for multicast traffic and unicast traffic for legacy power saving devices.

4.3.3 EVALUATION

We prototyped weSDN client-side components on eight Google Nexus 4 Android phones. We formed two network slices – an "employee" network with 2 devices and a "guest" network with 6 devices – and applied the same pTDMA schedule of Fig. 8 providing 50:50 airtime share to the two virtual slices, but with all 8 devices connected to one physical AP.

In Fig. 9, all devices are sending uplink iperf UDP at 12 Mbps, and the pTDMA scheduling is initiated at 50 sec. The graph plots the average throughput over the devices in each network slice with the high and low water-mark bars presenting the standard deviations. The first 20 seconds are impacted by low wireless speeds from fresh starts and were excluded in the following analysis. We clearly see the employee average is almost 3X larger than the guest average, indicating pTDMA provides the intended airtime share. pTDMA also improves fairness with lower temporal variations compared to non-pTDMA, as pTDMA reduces the number of simultaneously contending nodes in a window (from 8 to 2 in the employee network slice and from 8 to 6 in the guest slice). As we have only two employee devices, the "employee" slice shows higher statistical variation. Guests (6 devices) have much smaller variation than non-pTDMA (8 devices) thanks to the reduced contention of pTDMA. The reduced contention also increased net throughput, by 3 to 10% for different schedules.

To assess power-saving improvement, we plot inter-packet transmission intervals of one guest device in Fig. 10. pTDMA gives longer interval time during which the WiFi interface can sleep. While at this time, we did not directly measure device power consumption, we compute the total time the interface would sleep assuming a 5 ms timer for WMM-PS to detect inactivity and go to the sleep state: it can sleep 80% of entire run time with pTDMA while only 28% without pTDMA. Fig. 11 shows that the increased transmission intervals in the pTDMA schedule do not adversely impact TCP performance.

We tested different schedules, e.g., only one employee in a window or three guests in a window, and observed very similar results. Some schedules increased aggregate throughput (by 10%) while deviating from the intended employee:guest throughput ratio, suggesting a need for future study.

Finally, we evaluate pTDMA for end-to-end application performance. We have selected two popular applications, skype and youtube, to evaluate end-to-end performance.

Typically, HTTP video players like youtube have two main states: *buffering state*, in which the players tries to fill its buffer with video frames at the beginning of the streaming session, and *steady state*, in which the player starts periodically requesting video chunks after the buffer is filled up. The adaptive player typically maintains two thresholds, an upper and lower thresholds. The player pauses downloading video chunks as soon as the buffer filled and reached the upper threshold, and it resumes downloading once the buffer drops to the lower threshold. Figure 12 illustrates these two states in which the player starts buffering video chunks at the beginning of the streaming session, and when the buffer reaches its max threshold at time 70, the player goes off and enters the steady state. At time 78, the player goes on again and send a chunk request to the server when the buffer goes below the minimum threshold. This behavior recurs repeatedly until the video chunks are fully downloaded. These intermittent traffic pattern can be addressed as ON/OFF periods.

We found that, during steady state, pTDMA have no impact on the QoE of video streaming. As long as the, available bandwidth is enough to support the video encoding bit rate, we have not experience any degradation of video quality and video stalling. However, during buffering state, pTDMA might elongated the starting delay. But delay will be negligible if the available bandwidth is enough to support the requested video streaming.

Unlike the video streaming application, skype is real-time app that is time sensitive. In order to evaluate the impact of pTDMA on the performance of skype app, we create a controlled experiment setup. In the setup, we use one smartphone to generate iperf [164] UDP traffic to create the background traffic. Using the iperf, we create 4Mbit/s UDP data traffic as background. We run the skype in Android Nexus 4 smartphone, and we also enable the skype video technical report option to measure skype's throughput, frame rate, packet loss rate, and jitter. We run one skype app over cellular connection and another the one on our targeted WLAN.

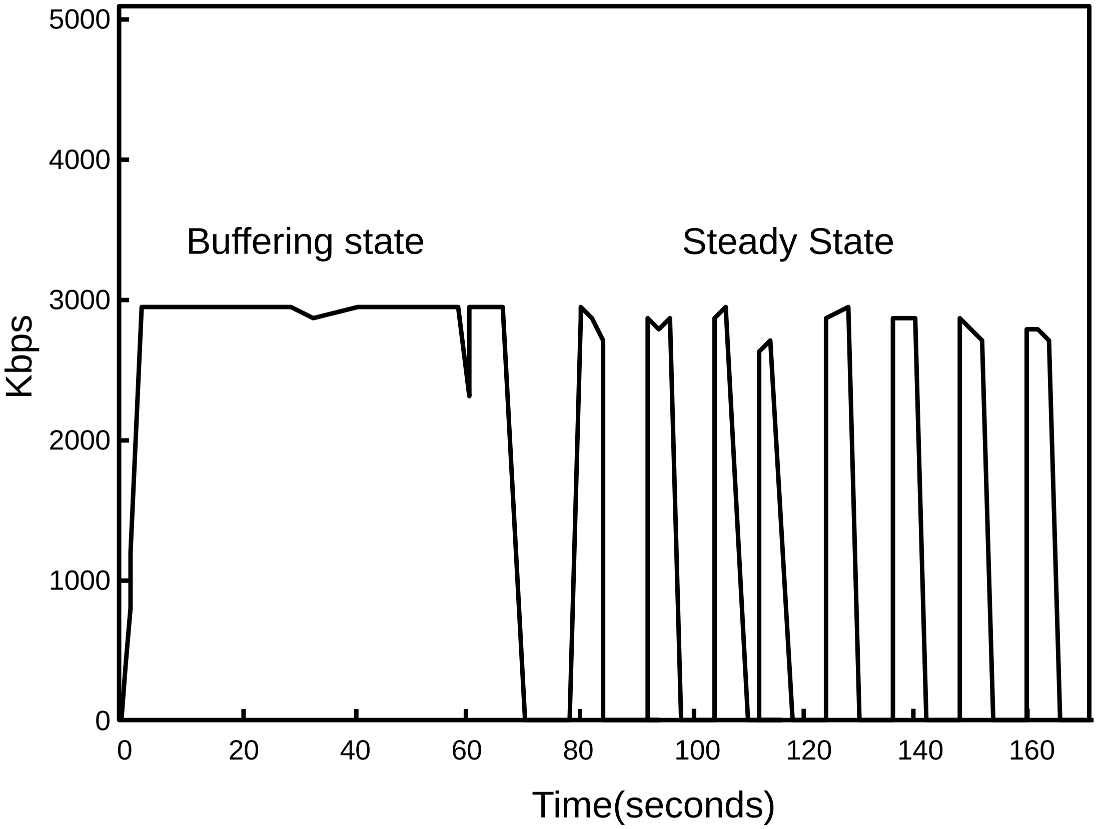


FIG. 12: Player state: buffering state and steady state.

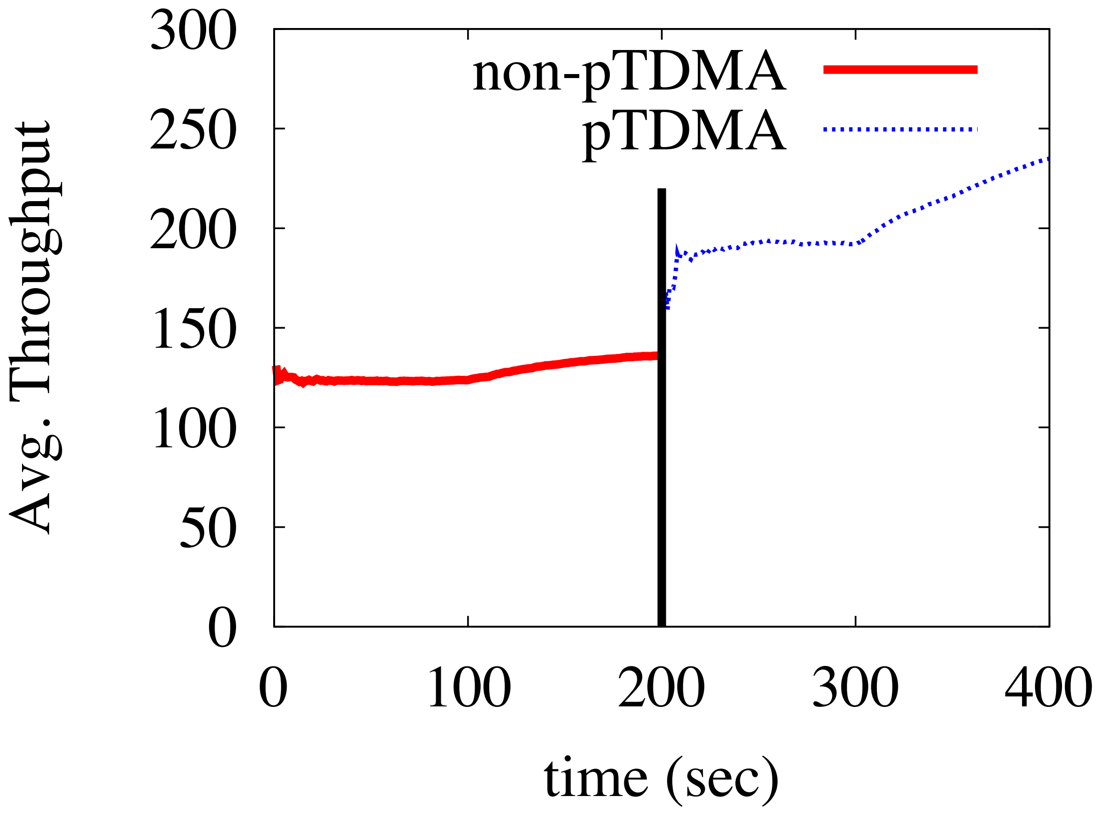


FIG. 13: Throughput changes for skype app from non-pTDMA to pTDMA.

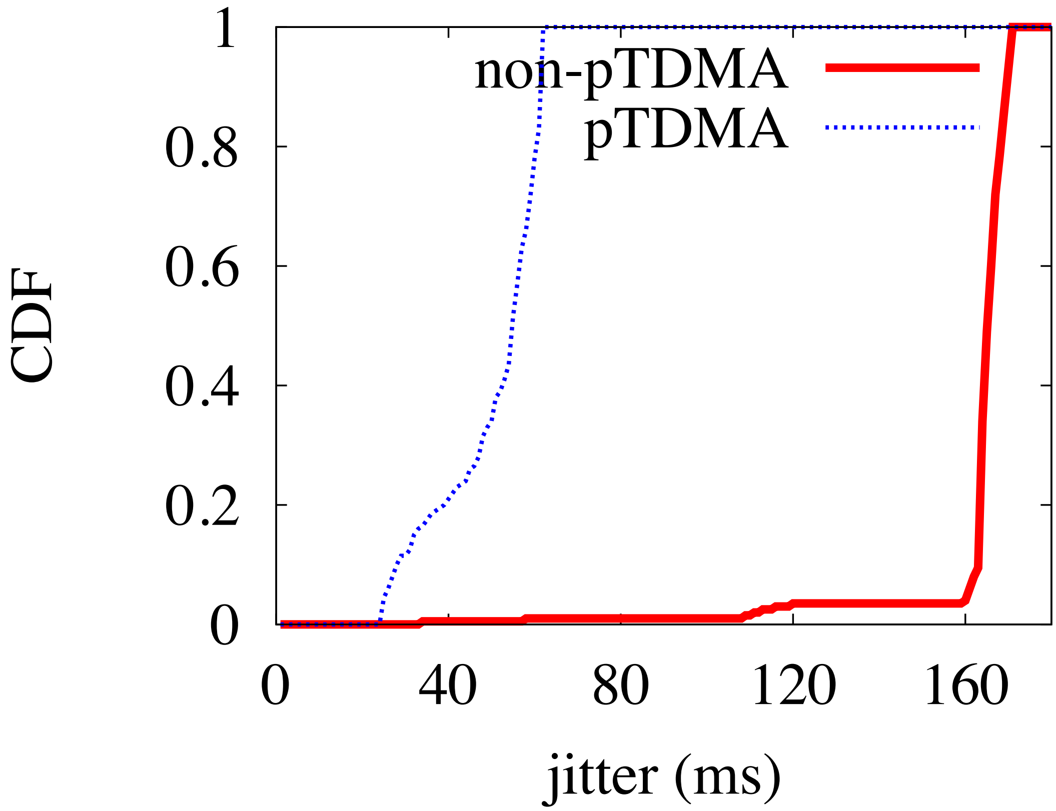


FIG. 14: Comparison of jitter between non-pTDMA and pTDMA for skype app.

Figure 13 shows the throughput changes when our targeted WLAN move from non-pTDMA scheme to pTDMA scheme. We initiate the pTDMA scheme after 200 sec. The figure 13 shows skype have higher throughput during pTDMA scheme compare to non-PTDMA due to collision of background traffic. We also have seen that, in both cases, the frame rate was changing between 10 to 15. However, during pTDMA, the video resolution was 320x240, and in non-pTDMA case video resolution was switching between 320x240 and 160x240. In addition, figure 14 shows that, pTDMA shows significantly less jitter then the non-pTDMA.

4.4 RELATED WORK & DISCUSSION

WLAN infra virtualization: Current APs can host multiple SSID networks or 'virtual APs' (e.g., employee vs. guest) on an AP radio [189] as a basis to control authentication, security and Ethernet-side bandwidth but they do not guarantee wireless resource share to each SSID network. Throttling Ethernet bandwidth may indirectly control wireless airtime usage of uplink TCP traffic, but with limited granularity; uplink UDP can not be controlled at all [23]. Recent work of OpenRadio [71], OpenRAN [185], CloudMAC [170] and Odin [160] extends SDN control to wireless APs. The Wireless & Mobile Working Group in Open Networking Foundation (ONF) is investigating various use cases for SDN-enabled APs and planning to standardize SDN control of wireless APs by extending OpenFlow [55]. SDN-enabled APs can control wireless resource for downlink traffic. However none of these work are considering SDN on end devices or addressing uplink control issue, and therefore they cannot control uplink traffic. Uncontrolled uplink transmissions can interfere with other uplink traffic and downlink traffic, so SDN-enabled APs can not guarantee uplink airtime or even downlink airtime. By carefully tuning 802.11e QoS parameters on the AP, we can probabilistically control downlink airtime shares and be more immune to uplink traffic, but this breaks 802.11 QoS mechanism or limits the number of virtual networks to four (802.11 QoS classes) [113].

Client-side solutions: Implementing per-packet TDMA MAC in WiFi driver has been demonstrated to virtualize airtime [156] but its throughput and scalability is challenged by the lack of sub-millisecond level 1) hardware control from the driver software and 2) clock synchronization across devices. SplitAP [17] loosely controls uplink airtime by shaping client's outbound traffic using Click router but it causes under-utilization of airtime. In contrast, pTDMA achieves tighter airtime control and also improves aggregate throughput and power-efficiency. The work in [190] deployed OVS on Android to efficiently utilize multiple network interfaces on a multi-homed device. They also introduced a local controller to manage OVS but the coordination with other devises or the rest of the SDN framework was not discussed.

Application-awareness: MultiNets [118] and Delphi [44] have proposed mechanisms to use multiple network interface of the smartphones based on the policies (i.e. energy saving, throughput performance, data usage cost, delay sensitivity), manually given by the user [118] or specified by each application [44]. Rather than relaying on users or apps to specify their objectives, weSDN monitors and analyzes network flows to learn the app's network demands real-time. pTDMA changes the transmission patterns of applications and thus may affect Quality of Experience (QoE) of application users. Recent work [2] shows the possibility of monitoring network flows to estimate the QoE, which weSDN can leverage to improve pTDMA scheduling.

Interference is another hurdle in achieving RF resource guarantees in unlicensed bands. pTDMA can control interference within and also between weSDN network slices.

External interferences can be handled by existing work on interference monitoring and mitigation [134, 140, 154]. Enterprise Radio Resource Management solutions today aim to realize such centralized schemes by tightly monitoring and reacting to interference at down to 4 min adaptation cycle [9]; recent small business and home APs are also controlled by 'cloud' for better radio resource management [37]. Even when external interference is unmanaged, pTDMA can still control the airtime 'share' among network slices, while each share will experience the external interference and deal with it based on 802.11 MAC.

Client WiFi driver modification: We believe that the WiFi protocol and the client WiFi stack are better to be kept intact to develop and deliver the new weSDN framework reliably across a multitude of end devices. We believe a pTDMA scheduler can be implemented in the WiFi driver without breaking the protocol standard and stack.

However, though there have been many innovative solutions that require "only" driver changes [35, 152, 156], none of them have been widely adopted. We found that end-device manufacturers tend to avoid making any custom changes on the drivers shipped by the chipset vendors because it is hard to maintain the custom changes throughout the future chipset/driver releases and restricts flexibility of end-device manufacturers into which radio chipsets they can use in future designs. Thus, our current prototype leveraged Linux standard components – Qdisc and OVS – as building blocks. Both components exist in recent Android kernel source. Due to the different nature of wireless and 802.11 QoS, additional control knobs and statistics from the driver will surely benefit weSDN, and we look forward to seeing such new knobs.

Incremental deployment: We had to root the Android devices to install OVS and pTDMA qdisc kernel modules, though they're available in the Android source tree. (On the other hand, changing the WiFi driver required re-imaging the entire kernel.) To ensure that heterogeneous end devices can participate in and benefit from weSDN, the kernel modules need to be natively integrated into the stack OS by the device manufacturers and similar support from other operating systems are needed.

To support hybrid deployments and phased migrations, weSDN and non-weSDN end devices will need to co-exist in the same WLAN environment especially in non-enterprise settings where synchronized deployment over all client devices is difficult. To continue to provide air time guarantees to the weSDN clients, non-weSDN clients can be controlled via mechanisms proposed earlier, such as dropping 802.11 data frames to lead them to backoff.

4.5 SUMMARY

In this chapter, we used the weSDN framework to implement a WLAN virtualization service that effectively guarantees airtime shares to network slices using a Time Division Multiple Access like airtime scheduling, named pseudo TDMA (pTDMA).

CHAPTER 5

SMARTEDGE: TOWARD MAKING WIRELESS NETWORK EDGES TRAFFIC-AWARE

5.1 INTRODUCTION

The growth of network-connected smart devices, such as smartphones, tablets, smart TVs, Health monitoring devices, home security system etc., has been accelerating and they are becoming an integral part of our daily life. The increase in the number of smart devices and corresponding various applications and services results in a significant growing of network traffic as well as the introduction of new traffic types. In addition, different mobile applications generate various types of flows for different objectives. For example, Skype application can have the voice, video, screen sharing, file sharing, and Instant Messaging (IM) flows as well as the background traffic for signaling, analytics, advertisement, etc. These applications diversity and various flow types might have different QoS requirements and security/resource policies.

In addition, with the new trend of pushing computation and storage resources closer to client devices for low latency, high bandwidth, and privacy [191, 192, 193], it is essential to have flexible and efficient network management at wireless network edges (i.e., end-devices and access devices) to support complex network management and configuration tasks such as end-to-end QoS for various network traffic, different traffic engineering schemes with fine-grained policies, and efficient load balancing [44, 89, 95, 165, 190]. In order to support such services, wireless network edges urge to have light-weight, on-fly fine-grained application and flow type awareness.

Recently, in both cellular [95] and WLAN enterprise [83], we have seen the effort of pushing the SDN capabilities at the wireless-edges (e.g.,WiFi Access Points (APs), Base Station) to enables fine-grained policy enforcement and performance optimization [44, 89, 95, 165, 190]. In applying fine-grained policies at wireless network edges, it is essential for the network operator to have greater visibility and control over the traffic generated from the client devices to provide optimal performance and high quality of experience. Therefore, with the recent advent of pushing SDN at the edge, we need to build tools that have the ability to accurately and efficiently recognize individual mobile applications (e.g., facebook, skype, tango, fringe etc.) and it is various traffic flows (e.g., video chat, voice chat, video stream, etc.) in real-time.

Existing SDN solutions, for application and flow-type awareness, depends on the centralize deep packet inspection (DPI) engine [5, 40, 54] for traffic classification. Unfortunately, unlike the edge data center machines, many devices of the wireless network edges are not computationally powerful enough for supporting DPI based solution [5, 125]. In addition, DPI-based methods have the limitation of providing fine-grained traffic classification due to payload encryption, privacy issues, and tunneling transfer [125, 197]. Therefore, SDN-based centralized solutions of traffic awareness are inefficient and impose significant overhead (e.g., delay) in order to control the traffic of the wireless network edges [54]. Furthermore, there could be scenarios, such as in coffee shops, metro stations, shopping mall, etc., where WLAN infrastructure is unable to provide application and flow-type aware network management services for the end-devices. In these cases, it is implausible to support truly end-to-end management and control, in which end users could reap the full benefit of SDN and experience high Quality of Experience (QoE).

In this chapter, we present our design, development, and evaluation of a lightweight, flexible and real-time application and flow types awareness service, called *SmartEdge*, based on the weSDN framework for wireless network edges, and specifically for end-devices. *SmartEdge* is the first of its kind that provides on-fly fine-grained visibility and control over the network traffic generated by different applications and corresponding various flow types running on wireless network edges. In *SmartEdge*, we push the SDN-like paradigm all the way to end-devices, by extending and deploying the SDN data layer (i.e., Open vSwitch (OVS)), and the OpenFlow protocol [104]) on both *end devices* and *access device*, to extract new flow statistics such as packet sizes, directions, sequences, and timestamps, where a 'flow' is identified by 5-tuple fields. Note that existing OVS and OpenFlow protocol is incapable of collecting such statistics from the network flows. Therefore, in *SmartEdge* we extend both OVS and OpenFlow protocol to collect such statistics per network flow.

In addition, we develop and integrate a hierarchical Machine Learning (ML) based solution in OVS for traffic classification. State-of-the-art ML-based approaches are only classifying applications to coarse-grained classes such as P2P, email, news, web browsing, VOIP and game application [24, 50, 81, 117, 139, 175]. Unlike identifying coarse-grained mobile application classes or protocols, *SmartEdge* focus on fine-grained detection of actual mobile applications name (e.g., Facebook, Skype, Tango, Fringe etc.) and it is different flow types (e.g., video chat, voice chat, video stream, etc.). In *SmartEdge*, we introduce new features (i.e., DWT) to extracts high-order frequency and temporal information from packet sizes and arrival timestamps, which improve the ML-based traffic classification detection accuracies from 75-89% (using the state-of-the-art [24, 117, 175]) to 85-98%.

We also implemented *SmartEdge* in real test-bed in order to evaluate the efficiency and the overhead of the framework. In implementing *SmartEdge*, we extend and deploy both the kernel module (i.e., "datapath") and the user space (i.e., "vswitchd") of the OVS to provide application and flow awareness at wireless network edges. We describe more details about the *SmartEdge* in section 5.3. We also evaluate *SmartEdge* in real test-bed in regards to energy efficiency, throughput reduction, and computation overhead. Finally, we develop two proof-of-concept prototype of application and flow-type aware policies based on *SmartEdge* at wireless network edges.

5.2 RELATED WORK

5.2.1 APPLICATION-AWARE SDN

Ensuring E2E QoS (Quality of Service) and providing intelligent capabilities in network edges, is one of the key reason for conducting research in the direction of app-aware SDN network. There is two type of approaches that has been taken to make the SDN controller more application aware. In one approach, require the user to input directly or to make custom integration between SDN controller and each application through API call to let the SDN controller know about the start/end of application flows and about their QoS requirement [2, 89, 165]. In another approach, Deep-Packet Inspection were applied to leverage unique per-app signatures to distinguish network flows from different applications/application protocol [132]; but they are unable to differentiate the various type of network flows generated from one application. Moreover, real-time detection of QoS demanding flows would require constant monitoring and analysis of network flows, which is costly to implement using DPI on CPU-limited AP and switches.

SDN *data layer* (i.e. OpenFlow switches) and *control layer* only analyze Layers 2-4 packet header fields, SDN cannot recognize packet's application or flow type information. Note that traditional schemes that use the port number and IP address for application/flow classification are not reliable in many cases since nowadays applications start to use non pre-defined or dynamic port numbers [42, 200]. Therefore, existing applications/flow identification schemes require direct user/application inputs [44, 52, 89] or custom integration with application servers [165]

On the other hand, techniques based on Deep Packet Inspection (DPI) can be more accurate, but incur high computation cost and require manual signature maintenance [20]. Moreover, many applications today are delivered via end-to-end encrypted channels, such as HTTPS and SRTP, thus limiting the reliability of DPI-based approaches and making the signature maintenance more difficult. Despite such difficulties, DPI is a popular solution with the SDN framework. In many commercial solutions, DPI is deployed with the SDN control layer, called DPI engine. However, in order to support time-critical application and minimizing latency for applying policy directly, industry have proposed to implement DPI in the SDN data layer, especially in virtual switches (i.e. Open vSwithch).

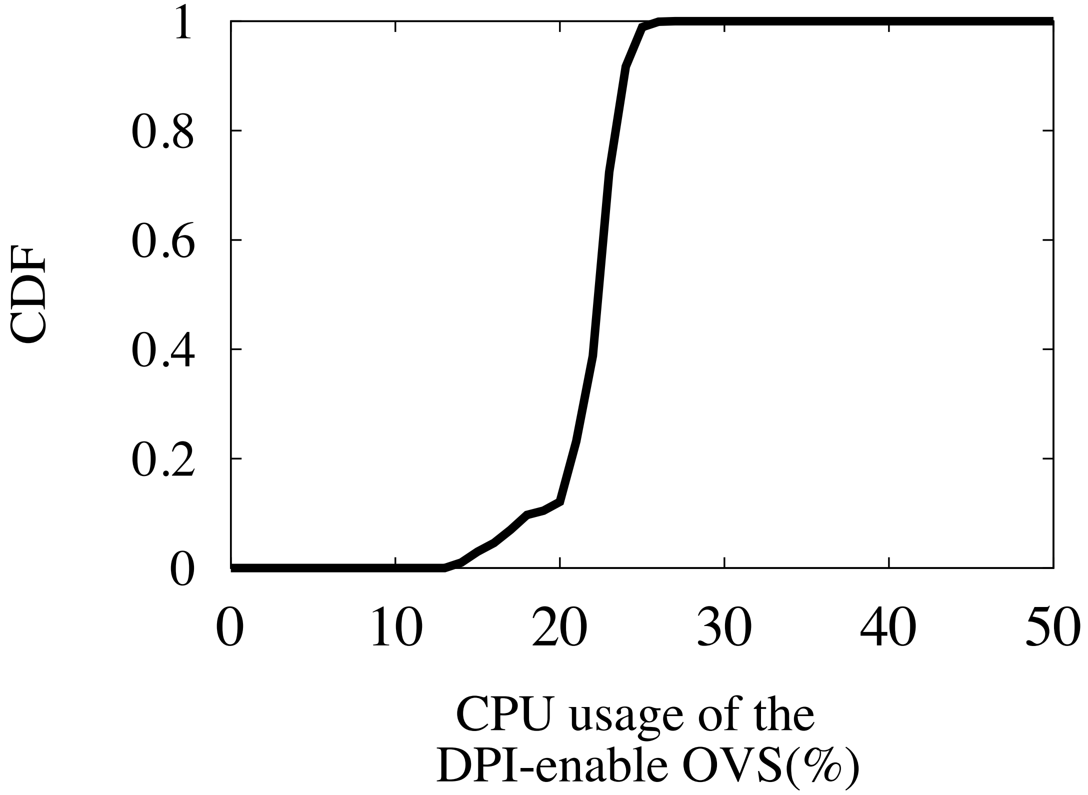


FIG. 15: Distribution of CPU usage overhead of the the DPI-enable OVS for the new “upcall” operation in the mobile device.

We found, there is not much attention in analyzing the performance of DPI with SDN framework. More specially, we are interested to find out the performance of running DPI software in Open vSwitch. DPI uses the initial set of packets of a flow to identify the application or application protocol. Therefore, according to one implementation [54], DPI-enable Open vSwitch (OVS) use a new “upcall” action in datapath that sends the initial set of packets of a flow to the user-space for analyzing in DPI software. In order to mimic the scenario, we create a similar action in OVS for sending first 8 packets of a flow to user-space, which is the maximum amount of initial packets require to identify an app or app's protocol in DPI. Figure 15 shows the CPU usage of just having such "upcall" action in mobile devices, which is quite significant in terms of mobile phone usage. Note that this CPU usage does not include the additional processing of the DPI engine. However, several earlier studies [20, 81, 125, 197] have already claimed that DPI engines are computationally expensive, which make such DPI based solution inapplicable in wireless edges.

5.2.2 ML-BASED TRAFFIC CLASSIFIER

A major advantage of ML-based approaches [22, 49, 88, 117, 131, 179] is that the classifier does not require examining packet payload content, instead, it only requires flow level features such as the packet sizes and timestamps. This results in a much lower computational cost than DPI-based solutions [179] and can correctly identify encrypted traffic. There are a wide range of work on ML based traffic classification in the past, for example [86, 92, 93]. In *SmartEdge* we only focus on lightweight, real-time, fine-grained ML-based classifier based on flow level features. Previously, researchers have used Netflow [77] for collecting flow level features for real-time traffic classification[25, 81, 139]. In *SmartEdge*, for the first time, we propose to extend and use OpenFlow [104] protocol with Open vSwitch (OVS) [171] to collecting flow level features for traffic classification. Note that, OpenFlow provides flexible network control, and fine-grained network monitoring capability.

ML approaches have so far been restricted to traditional internet applications with coarse-grained classifications such as web, P2P, and VoIP [88]. Unlike traditional applications, mobile applications (e.g., facebook, skype, tango, fringe etc.) often generate various types of flows (e.g., video chat, voice chat, video stream, etc.). Several studies show how different users generate different flow types of the same application [16, 76, 99]. Note that, different flow types of different applications have different network loads, QoS requirements, security/resource policies, etc. Thus, it is very critical to be able to accurately and efficiently recognize individual mobile applications and its various traffic flows in real-time. There are several recent works [124] which focus on a more detailed classification goal such as identifying the exact application name. However, due to fast-changing and dynamic nature of the mobile application, traffic classification with good accuracy is a challenging task. Despite that, in *SmartEdge*, we propose multiple levels of ML based classification model with the new set of flow feature to provide more practically applicable application and flow-type classification technique for wireless network edges.

The limitation of supervised classification – falsely classifying a new app flow into one of the known apps – seems to not have been studied in the context of network traffic classification, it has been studied in the machine learning literature [29] in the context of outlier detection.

Existing outlier detection algorithms [29] relied on traditional clustering techniques such as *k-means* that requires knowledge on the number of clusters, which a network operator would have no idea to start off with because an application can have any number of clusters and some clusters are shared by multiple apps. Moreover, there is no way to control the size (tightness) of the formed clusters that directly affects the new app detection accuracy.

Existing work for classifying various flow types from each application was either relying on DPI [**?**], limited to a specific application [18], or unable to bound packet collection window time suitable for real-time classification [116]. Ref. [116] is the closest to our approach in terms of training and classifying on sub-flows, but they require a certain number of packets to form a sub-flow, being unable to guarantee real-time detection. Moreover, they used only low-order statistics as ML features, unlike DWT of our choice, resulting in only 80% accuracy (for 15 tested apps) in spite of using much larger datasets for training.

5.2.3 CHALLENGES OF EXISTING TRAFFIC IDENTIFICATION TECHNIQUE

The traditional approach of traffic identification is challenging to apply directly on mobile traffic identification. One of the reason is that mobile applications predominantly uses HTTP/HTTPS to communicate with their host services. In addition, mobile applications do not use specific protocols or IP ports with the distinctive feature. These strategies at the application-level have essentially made port based traffic classification inaccurate and hence ineffective [108]. Also, a widespread use of cloud and Content Delivery Network (CDN) services invalidates mobile application identification approaches based on hostname and IP addresses. In many scenarios, mobile applications can tunnel traffic inside other applications for ease of implementation. For example, Zynga offers several games that run on facebook platform, uses HTTP for its network traffic. In that case, network operator needs to know not just that there is HTTP traffic on the network but also that there is Zynga poker within facebook traffic being carried over HTTP. This fine-grained information about the application in the network traffic enables better QoS, billing, and security.

An alternatives solution to the hostname and port-based method is the DPI techniques, which consist of looking for characteristic signatures (or patterns) in the packet payloads. Although DPI approach provides high accuracy for supported applications, they cannot cover diverse mobile applications in a scalable and timely manner due to their high development & maintenance costs [80]. Therefore, with the rapid growth of mobile devices and mobile applications, DPI approach faces the challenge of scalability. Moreover, DPI schemes fail to classify encrypted packets. As more mobile applications are using encrypted content and data, more concerns about the future of Deep Packet Inspection [40, 54]. To overcome this issue, current DPI schemes uses heuristic classification techniques for encrypted traffic, which is mostly unreliable and slow.

5.3 *SMARTEDGE* DESIGN

Figure 16 shows the full software stack of *SmartEdge*, where at the bottom, every network flows pass through the *feature engine* component. This component collects flow-level statistics as well as flow information (i.e., port, IP address, QoS priority, protocol etc.) for individual flows, and sends it to the *feature extraction* component. Flow-level statistics are series of packet sizes and packet arrival timestamps collected during a certain time window (i.e., 200ms, 2000ms) at the starting of the flow. The *feature extraction* component extracts set of features from the collected flow statistics, and feeds them to the *classifier* component. Finally, the *classifier* component uses the classification model database to classify each flow into the corresponding application (e.g., Vimeo, Skype, Facebook etc.) and the flow-type (video chat, voice chat, video stream etc.). After flow classification, the *agent* applies corresponding policies on the flow, provided by the *policy controller*. The *agent* basically works as an intermediate interface for applying application and flow -aware policies. In following subsections, we describe the three components; *feature engine*, *feature extraction*, and *classifier* in details. These three are the primary design components of *SmartEdge* .

5.3.1 FEATURE ENGINE

Feature engine component is basically an extended version of a programmable software switch, which in this case is Open vSwitch (OVS) [171]. Note that OVS support number of protocols like NetFlow, sFlow, and OpenFlow to collect flow-level information. Typically all the previous traffic-classification work based on DPI or ML, has the drawback of high cost and complexity of collecting flow-level information and statistics. Therefore, in *feature engine* we leverage and extend OVS and OpenFlow protocol to simplify and automate the

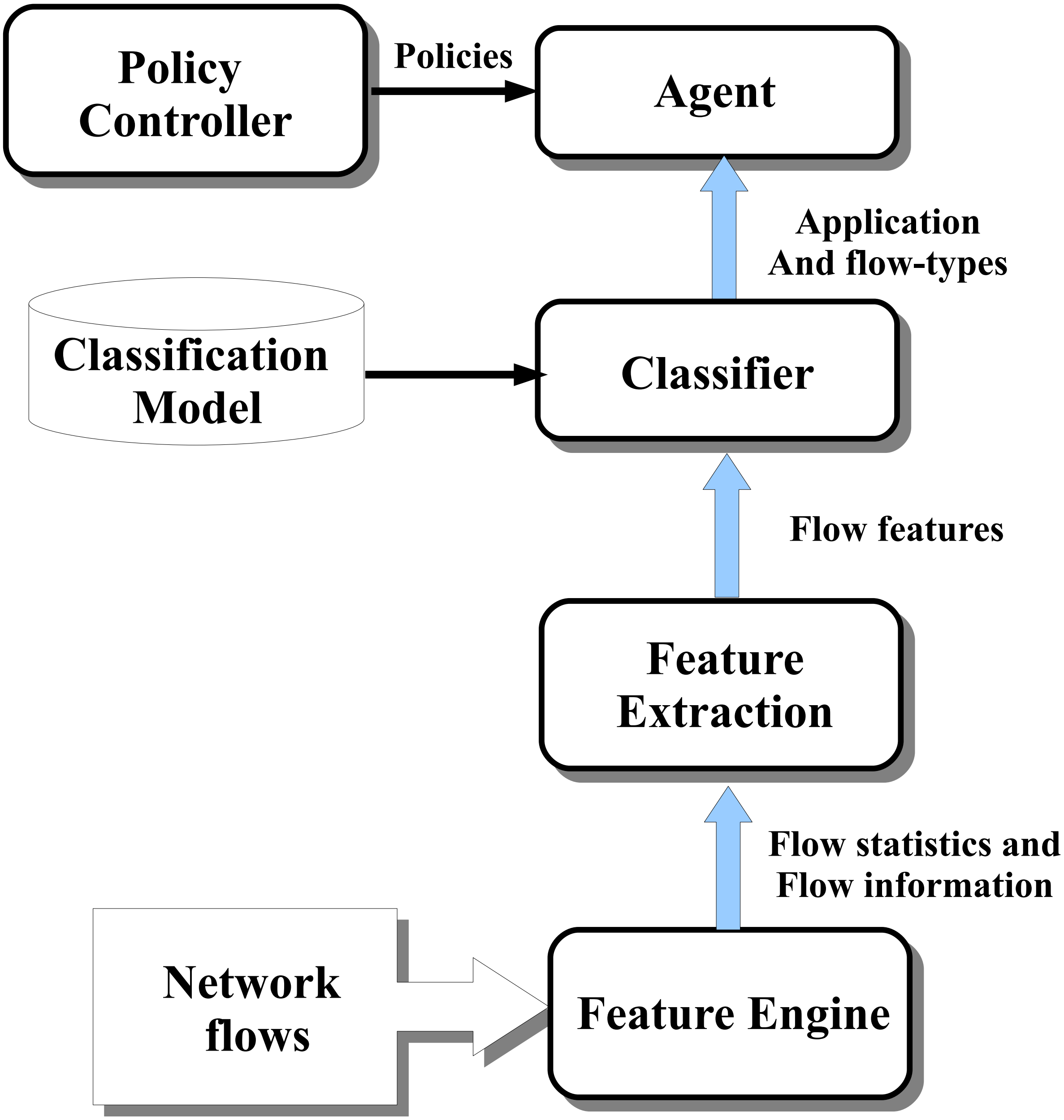


FIG. 16: The design overview of the application and flow-aware SmartEdge.

process of collecting flow-level statistics and flow information.

In *feature engine*, we extend OVS data structure to collect metadata information of individual packets such as packet size, packet arrival timestamp, port address, IP address, protocol, TCP flag, TOS bits, etc. Furthermore, we extend OVS to aggregate the individual packets metadata information into flow-level statistics such as series of packet-size, packet inter-arrival times, etc. Note that, we collect flow-level statistics for a short duration of at the beginning of the flow life-time.

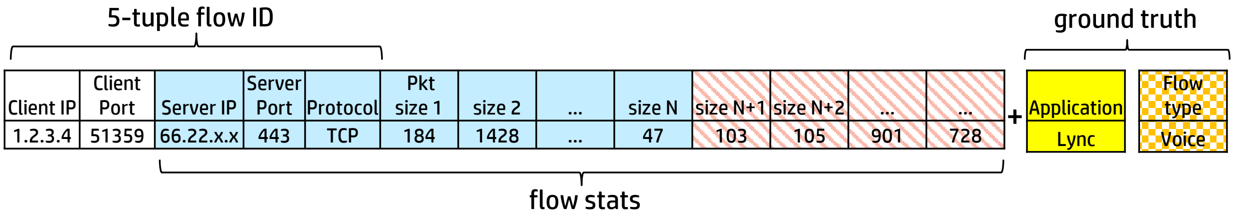
In SmartEdge, we also extend the OpenFlow protocol to collect flow-level statistics and flow-information (IP address, port, protocol, TOS) from OVS for each unclassified flow. In extending both OVS and OpenFlow protocol, we ensure no network performance degradation and minimal computation overhead. Furthermore, in extending OVS and OpenFlow protocol, we reuse and leverage the existing source code of the original OVS to reduce the overall source code and memory usage.

5.3.2 FEATURE EXTRACTION

The *Feature extraction* component extracts features from the collected flow-level statistics and flow-information. A network flow has two directions of packet flows, incoming and outgoing, with respect to a client device. We combine the packets from the both directions while keeping the order of packet measurement time and extract two sets of flow features from each combined flow.

The first set of features corresponds to the signaling portion of the flow, consisting of the first 'N' packet sizes as well as server port, server IP, protocol. These packets typically correspond to session initialization and signaling and are unique to each application. For example, the sizes of the first 11 packets of four Twitter flows are all unique. There could be several such unique signatures corresponding to a given application. Hence, we use these features to determine the application name of the flow. However, existing in-network measurement APIs (e.g., OpenFlow protocol, sFlow) do not provide packet size & time information 'per-flow'; typical network switches have no 'flow state'. Thus, previous ML-based solutions use software or hardware packet capture tools (e.g., `pcap`) but this incurs unnecessary overhead for traffic mirroring, capturing and flow processing; and also inadequate for real-time in-switch/AP processing. In this work, we extend Open vSwitch and OpenFlow statistic APIs to provide per-flow packet size (and arrival timestamp, for the second set) with only marginal memory overhead to the *SmartEdge* system.

The second set of features corresponds to the non-signaling (data) portion of the flow,



ML modules	Training data	Real-time data
Application detection	🟦 + 🟨	🟦
New app detection	🟦	🟦
Flow type detection	▨ + 🟨 (+ ▩)	▨ (+ ▩)

FIG. 17: Data requirement for traffic detection.

consisting of packet sizes and inter-packet time (IPT) observed after the first N packets. (Flow-type detection is applied to long flows, with more than N packets, mostly of media or file/screen sharing sessions.) Most existing approaches extract flow statistics over the entire flow or require a certain number of packets over the unlimited amount of time. We collect flow stats over a fixed time window, as short as 200 msec, and determine the flow-type corresponding to each window of the flow. This real-time, per-window detection of flow-type is useful for conferencing apps such as Skype that can switch the flow-type between voice and video(+voice) over time.

In addition to the lower order flow-level statistics (i.e., number of packets, number of bytes, protocol, and mean, median, minimum, maximum, and variance of the packet sizes and IPTs), researcher have also uses higher order statistics such as the Discrete Fourier Transform (DFT), to further improve the detection accuracy. However DFT has two main limitations: 1) It assumes that the signals are stationary, i.e, the periodic patterns do not change over time, but most real world time series signals including the packet size and IPT signals are not stationary. 2) DFT only captures global features thereby losing information on local properties of the signal. Considering these limitations, in this work, we introduce Discrete Wavelet Transform (DWT) [14] that is more robust in capturing both the global and the local variations of the time-series data. DWT ($O(N)$) is also faster than DFT ($O(N \log N)$), where $N = \#$ of data points in the sequence.

In [117], authors have summarized the list of flow features for traffic classification. We use these features to create two features sets; f represents all the features from [117] except DFT/FFT coefficients of packet sizes and IPT sequences, and f_{DFT}, consists of f

time window	low-order features f	f_{DFT}	f_{DWT}
200ms	**75.5 ± 2.6**	**81.5 ± 3.1**	**89 ± 3.8**
2000ms	**83.1 ± 2.3**	**88.0 ± 2.7**	**94.8 ± 2.7**

TABLE 1: Flow-type detection accuracy (% F-measure) of 3 feature sets. [mean ± stdev] over 15 apps with 8 flow types.

combined with the DFT coefficients of packet sizes and IPT sequences. Besides these two feature sets, we create another feature set, f_{DWT} that consists of f and the corresponding DWT coefficients of packet sizes and IPT sequences. Note that, none of the previous traffic classification technique has used DWT coefficients as features. Table 1 compares the flow-type detection accuracy using these three feature sets. Clearly, inclusion of DWT coefficients as features improves the overall accuracy of traffic classification compare to previously used features [117].

5.3.3 GROUND-TRUTH DATA COLLECTION

Fined-grained, accurate classification is only possible with fine-grained and reliable ground truth data. Existing ML-based solution mostly relies on DPI and/or manual inspections [88, 131, 179] for collecting ground-truth data. However, recent reports [**?**] shows 50% detection accuracy using OpenDPI and other opensource tools. Furthermore, Commercial DPI engines still misclassified significant portion of encrypted and HTTPS traffic. DPI requires a manual effort of building the application signature, which makes it not scalable with the large growth of mobile applications. Therefore, in this work, we took the approach of running an "agent" software on user's mobile devices that directly communicate with the SDN controller (e.g. Floodlight) to provide fine-grained application name information. We leverage the agent software to collect standard `netstat` logs, which provides the mapping of every active network socket (TCP, UDP, SSL-encrypted) to the application owning that socket, and we use the mapping to label each flow statistics collected from the wireless edges with the source application name. In many enterprises, employees require deploying device management agent software on their mobile devices. This motivated us to run a software "agent" on selected employee devices, volunteers or dedicated testing device to collect ground-truth data with flow features to train and build ML classifier at the network controller. Such method allows us to build the classifier automatically and efficiently.

Obtaining flow-type ground truth is more challenging as there is no standard API, similar to `netstat` for the app name, that provides clear information of flow type. Therefore, we optionally instrument the software agent to automatically monitor activities of media devices such as the microphone, speaker, and camera. Besides such automation, the user can also voluntarily provide information about the current activity (i.e. playing a video in youtube, skype video chat etc.) using the agent software to the SDN controller. Such activity information, correlated with network flow start/stop information from `netstat`, are used to infer the ground truth flow types of the media applications.

5.3.4 CLASSIFIER

With the help of a set of classification models (database) that we develop, the *classifier* component utilizes the extracted features to identify the application and the flow-type of a flow. The classification model database maintains two models: 1) application classification model, and 2) flow-type classification model. The *classifier* component first applies the application classification model to identify the application, and then it applies the flow-type classification model to identify the flow-types.

Application Detection

Given a large number of mobile applications, it is impractical to have an application classification model to identify all type of applications. This classification model should be developed based on the network activity/environment (e.g., campus, enterprise, home, etc.). For example, based on data collection from our campus, we found that only small set of applications is responsible for the major portion of our campus network traffic. In our implementation, we picked the top 40 popular applications to build the application classification model corresponding to our campus network. In real-world, network administrator often applies policies on the popular applications. Therefore selecting such popular application is a practical approach for building the application classification model.

We use a supervised learning approach for application detection, where a classification rule is learned using the labeled training data. Specifically, we use a C5.0 decision tree classifier [135] due to its overall performance (accuracy and speed) [88]. Each internal (non-leaf) node in the tree denotes a test on a flow feature, each branch represents the outcome of a test, and each leaf node holds a class label (i.e., application name). The training phase is performed by using the device-crowd-sourcing approach to collect the ground truth in a scalable and accurate manner. The training can be done periodically to automatically update or improvise the classifier. This trained classifier is used to identify the application names of network flows in real-time.

However, ability to detect a new/unseen application is a lacking feature of supervised

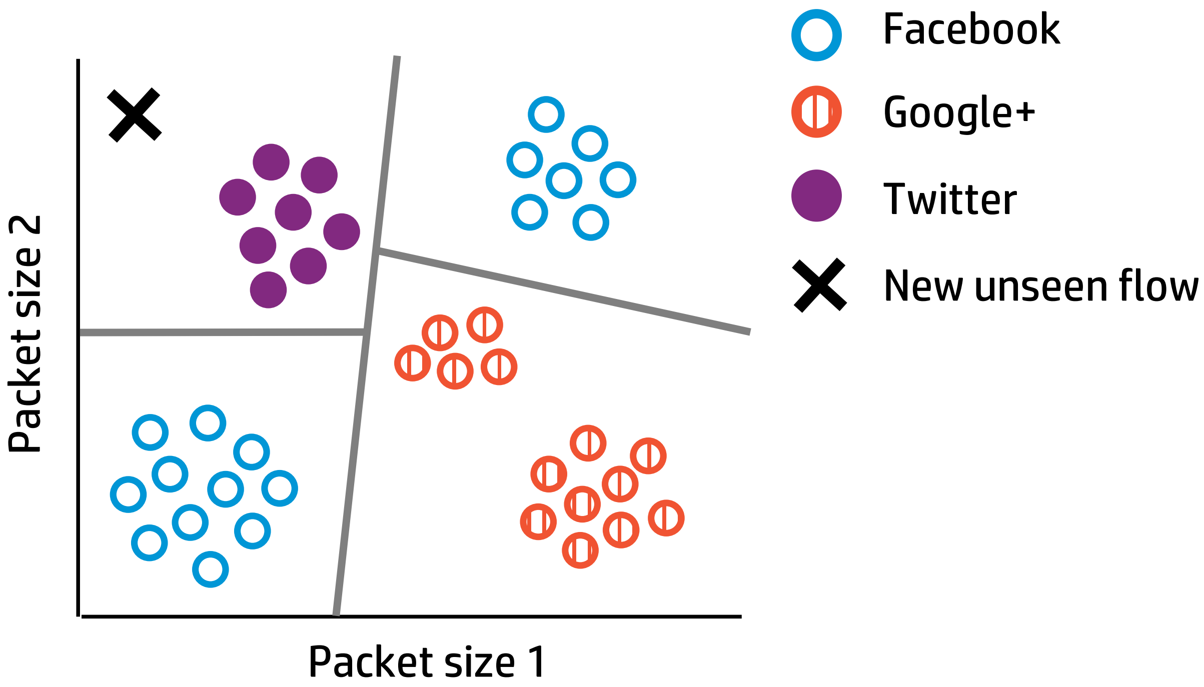


FIG. 18: Limitation of an ML classifier to detect new application.

machine learning (ML) approach like C5.0 decision tree classifier. This is because, in supervised learning, when a flow corresponding to a new/unseen application arrives, it gets classified by the partition it falls into, even if the flow has a very distinct signature compared to the flows in that partition. Identifying such a flow with a significantly different signature is key to solve this problem. Thus, the application classification model in *SmartEdge* will be able to recognize any unknown application that is not one of the selected 40 applications for our campus network.

We take a cluster-based approach illustrated in Figure 5.3.4. In the offline training stage, we cluster the training data (without the ground truth) into 'tight' clusters, each represented with the cluster centroid and the cluster radius. In real-time, we determine if an incoming flow corresponds to a new application or an existing application by checking if the flow lies within the boundary (radius) of any existing clusters. However, building tight clusters from the training data is challenging with most existing clustering techniques such as k-means [29]. These approaches require the user to specify the number of clusters, which is often not known a priori. They also force fit the data into the chosen number of clusters, resulting in bad clusters especially when k is incorrectly estimated.

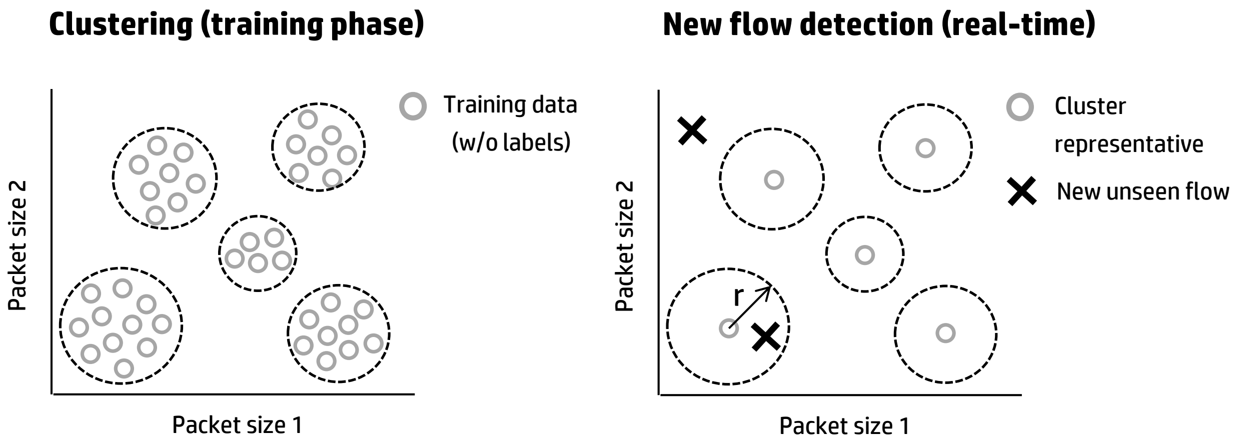


FIG. 19: Proposed cluster based solution for new application detection.

Therefore, we propose a simple, iterative wrapper around *k-means* that selects one tight cluster at a time. The inputs to the wrapper are the training data and a user-defined parameter "*r*" which corresponds to the maximum permissible radius for the desired clusters. This parameter lets the user to control the cluster tightness by setting the flow distance up to which a flow can be treated as an existing flow. Application or network specific information, such as protocol padding size, can be used to derive this parameter. Though each application may use a different cluster radius, we use a system-wide radius "*r*" for simplicity. The proposed scheme is detailed as follows.

- *Data Partition*: The training data is partitioned using the traditional *k-means* clustering with an arbitrarily chosen number of clusters (k). If "k" is less than the actual number of clusters in the dataset, clusters could potentially be merged. However, such merged clusters will be separated in the subsequent steps/iterations. If "k" is greater than the actual number of clusters in the dataset, the *k-means* clustering would return empty clusters.

- *Cluster Cleaning*: The flows residing outside the radius "*r*" from the cluster centroid are removed.

- *Cluster Selection*: The cluster with the largest number of flows within its radius is selected and saved in the *return set.* Flows corresponding to the selected cluster are removed from the original data to create a residual dataset.

- *Iteration*: Steps $1 - 3$ are repeated on the residual dataset until termination criteria is met, e.g., when all flows are clustered or no other cluster that satisfies the tightness requirement is identified.

This approach ensures that the clusters in the *return set* are tight within the radius "r". We determine an incoming flow as 'new-application' flow if the flow lies outside the radius of every cluster in the return set.

Flow-type Detection

We use the k-Nearest Neighbor (k-NN) supervised learning algorithm for flow-type detection. The flow-type classification consists of two different scenarios and consequently, we construct two different flow-type classifiers; 1) aGgregated Flow-Type (GFT) classification model, and 2) Per-Application Flow-Type (PAFT) classification model. While these classifiers use the same k-NN algorithm on the same feature set, different combinations of ground truth and training sets are used for each classifier.

In our implementation, we classified any flow type to one of the eight classes: Audio Stream, Audio/Video Stream, Real-time voice chat, Real-time video/voice chat, File Sharing in cloud, P2P file sharing, Screen sharing, and Background. while we build the GFT classification model using all the flows in our training data set, we build a separate PFT classification model for each individual application using the only the flows in our training data that are corresponding the application. When the application classification model classifies a flow as "novel application" or un-identified application, then the flow-type classification model uses GFT model to identify the flow-type. Otherwise, based on the identified application, the flow-type classification model uses PAFT model to identify the flow-type of the application.

As we will show in evaluation (section 5.6), PAFT model has a higher accuracy than GFT model. Therefore, for all known applications, it is favorable to use the PAFT model instead of the GFT model. Thus maintaining two-level of classification model (application and flow-type), makes *SmartEdge* highly accurate and very practical applicable for a fine-grained identification of the running flows and, consequently, be able to run and enforce and any flow-aware network management policy.

5.3.5 DEPLOYMENT MODE

We have two deployment settings for *SmartEdge* ; 1) Active Client Mode (ACM), where the core components of *SmartEdge* runs on end devices, and 2) Passive Client Mode (PCM), where the core components of *SmartEdge* runs on access devices. We refer the core components of *SmartEdge* to include *agent, classifier, feature extraction,* and *feature engine* as shown in Figure 16. PCM setting is more suitable for network configurations where access devices could be controlled and configured to cooperate with end devices such in enterprise WLAN, campus WLAN, home WLAN, etc. For other network configuration such as Wi-Fi hotspot On the contrary, ACM setting is suitable for ad hoc WiFi hotspots as in the coffee shop, hospital, airport, train station, etc., in which access devices (i.e., APs) are not guaranteed to be configured to cooperate with end devices. Furthermore, ACM setting would be favored by users for privacy issues especially when using sensitive applications. While the *policy controller* (shown in) can run either on access devices (e.g., WiFi access points) or network controller in PCM setting, it could run on cloud as a cloud service for ACM setting.

When *SmartEdge* is running on end devices as in ACM setting, the application classification step of the classifier could be skipped since the actual application package name corresponding to an active flow could be extracted. Therefore, *SmartEdge* could jump directly to use PAFT classification model on the flow features to identify only the flow-type if the extracted application name is one of the applications used in the training data set. Otherwise, *SmartEdge* uses GFT classification model. On the other hand, since *SmartEdge* cannot infer the actual application name when it is deployed on access devices in PCM setting, *SmartEdge* needs to apply the application classification model on the flow first to identify its corresponding application. If the flow is classified as one of the known applications or as "novel application" (i.e., unknown application), then PAFT classification model or GFT classification model will be applied to the flow features respectively to classify the type of the flow.

5.4 ***SMARTEDGE*** IMPLEMENTATION

Figure 20 illustrates the implementation architecture of *SmartEdge* (dark gray) in addition to the regular OVS modules (light gray). In the implementation, *SmartEdge* have three modules i) Flow features engine, ii) Flow classifier, and iii) SmartEdge agent. Among them, the "flow features engine" and the "flow classifier" are two integrated modules in OVS. The "flow features engine" module is in kernel space, and the "flow classifier" module is in user-space. These two modules, in addition to "SmartEdge agent" module, reside either on end-devices or access devices. The "policy controller" is a third-party application that resides either in cloud or access devices. The "policy controller", which is not part of the , only leverages *SmartEdge* to apply application and flow-aware policies on devices of

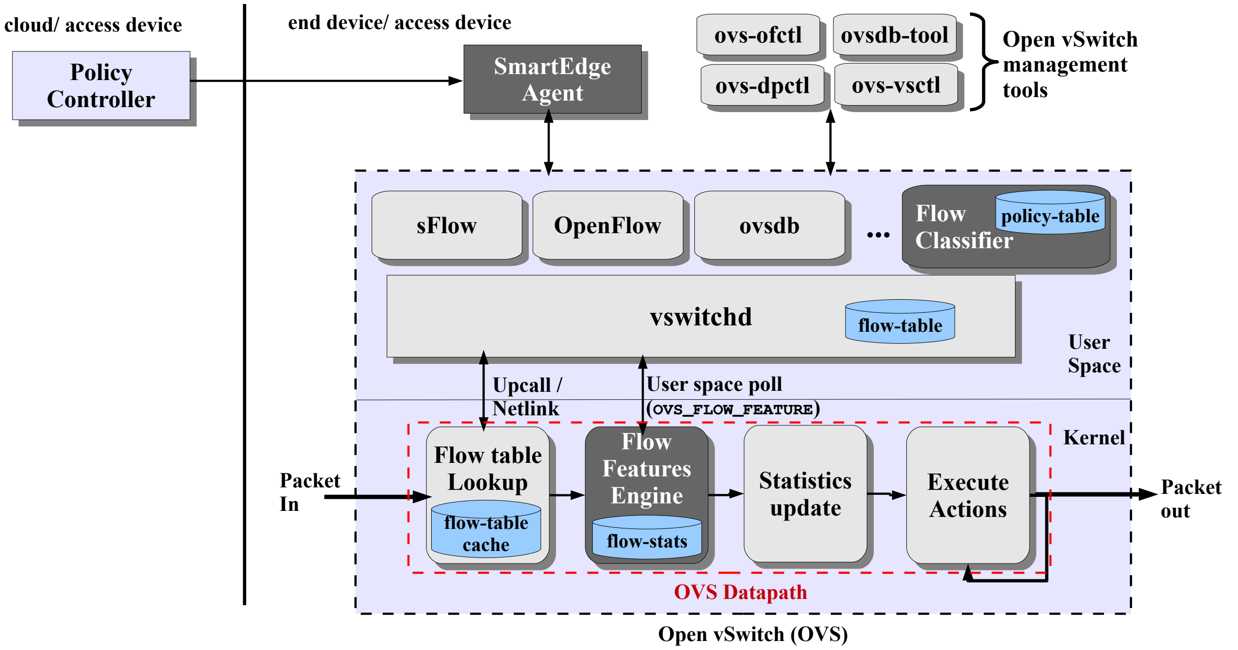


FIG. 20: Implementation architecture of *SmartEdge* .

wireless network edges.

5.4.1 FLOW FEATURES ENGINE

This is an extension in OVS datapath (i.e., OVS kernel module), which maintains a hash-table of flow entries. The hash table is called “flow-stats”[1] that only uses 5-tuple fields of the packet header to calculate the hash value. The main purpose of “flow-stats” table is to store flow information (i.e., port, IP address, QoS priority, protocol etc.) and keep record of flow-level statistics for each flow entry. Note that, “flow-stats” only store flow entries that just have started, and not yet classified.

In OVS, there is a “flow-table” in both user-space and kernel space, following the original OVS implementation [130]. Note that, “flow-table” is a data structure that defines how packets of a flow should be processed. The “flow-table” in kernel space is the cache of the

[1] The “flow feature engine” uses separate lock system for the ”flow-stats” hash table. Therefore, thread operation on the “flow-stats” table do not block any operation of the OVS “flow-table” cache. Furthermore, we use separate `spin_lock` for each bucket of the “flow-stats” hash table, therefore even writing operation in one bucket will not block the whole hash table. With enough number of buckets and per-bucket locking mechanism, we can reduce the chance of blocking for multiple thread operation on the “flow-stats” hash-table. Thus we can reduce the overhead of “flow feature module” on the main thread operation of processing a packet in the datapath.

Algorithm 1 algorithm for processing packet inside "flow features engine".

```
procedure FLOW_FEATURE_PROCESS(flow, skb)
skb, incoming packet for processing.
flow, the matching flow entry from OVS flow-table.
    Calculate Hash value from skb
    if flow is null then                                        ▷ upcall scenario
        Call flow_feature_update to insert new entry
    else
        if flow has application and flow-type information then
            Search entry in the "flow-stats" using hash value
            if Entry found then
                Remove entry from "flow-stats"
            else
                return
            end if
        else
            Call flow_feature_update to update the entry
        end if
    end if
end procedure
```

user space flow-table. In *SmartEdge* we extend "flow-table" to add two new fields, application (i.e., `app_code`) and flow-type (i.e., `fow_code`) information. This extension later allows us to apply application and flow-type aware policies on the traffic flows. Before classifying a flow, these two fields of the flow remains unidentified. Once the flow is classified, we set these two fields with identified application and flow-type information. Therefore, once a flow entry in "flow-stats" is classified, then we require to find the corresponding entry in the "flow-table" to update the application and the flow-type field information. In order to do that, for each flow entry "flow-stats" maintain a field (i.e., `match_mask`) that links to the corresponding entry in the "flow-table".

Figure 20 shows the high-level overview of packet processing pipeline in OVS datapath, where "flow-table lookup" process take place before the "flow feature engine". Note that, in *SmartEdge*, "flow feature engine" is an addition step in the packet processing pipeline of OVS datapath. The responsibility of the "flow-table lookup" process, is to search matching entry from the "flow-table" cache for processing the packet. There can be two outcomes of the searching; In one outcome, no entry is found in the "flow-table" cache that results in an "upcall" (i.e., a system call from kernel space to user-space) action to initiate a user-space thread for searching suitable entry from the user-space "flow-table". Once the thread find the matching entry in the user-space "flow-table", it sends it to the OVS datapath to cache the entry in the "flow-table" cache for processing later packets from the same flow. In the other outcome, a matching entry is found in the "flow-table" cache for processing the packet and consequently there will be no "upcall" action. Regardless of the outcome from 'flow-table lookup" process, `flow_feature_process` process is called (i.e., Algorithm 1) in

Algorithm 2 algorithm for updating the “flow-stats” hash table in Datapath

```
procedure FLOW_FEATURE_UPDATE(flow, skb)
    hash ← calculate the hash value from skb
    entry ← find entry from flow-stats table using hash
    if entry is found then
        entry->timestamps[entry->index]=timestamp
        entry->pkt_sizes[entry->index]=skb->len
        entry->index++
        update the entry->match_mask field to flow
    else
        allocate new entry
        entry->index=0
        entry->timestamps[entry->index]=timestamp
        entry->pkt_sizes[entry->index]=skb->len
        update the entry->match_mask field to flow
        entry->index++
        insert_entry(flow-stats,entry)
        atomic_inc(flow-stats->n_entry)
    end if
    if flow!=null && flow-stats->n_entry>=1 && !poll_thread_enable then
        atomic_inc(poll_thread_enable)
        upcall(OVS_FLOW_FEATURE)
    end if
end procedure
```

“flow feature engine” for further processing of the packet. Note that, in the “upcall” action scenario, process `flow_feature_process` does not wait for the user-space thread to find out the entry from the user-space, rather they operate independently. Thus, the “flow feature engine” module has no blocking overhead on processing the packet in the OVS datapath.

The `flow_feature_process` process has two input parameters; `flow` parameter that represents the entry of “flow-table” cache from the “flow-table lookup” process, and `skb` parameter that represents the packet that is in the packet processing pipeline at OVS datapath. Initially, `flow_feature_process` creates the hash value from `skb`. Then based on `flow` parameter, null value indicates that an “upcall” action had taken place in “flow-table lookup” process, then it calls `flow_feature_update` process (i.e., Algorithm 2) to update the entry in “flow-stats” hash table with the packet information (i.e., `skb`). In the nother case, if `flow` is not null, then we check the application (`app_code`) and flow-type (`flow_code`) information of `flow` parameter that represents the corresponding matching entry from the “flow-table” cache. If the application and flow-type information is available in `flow` parameter, then the flow entry is already classified. In this case, `flow_feature_process` removes the flow entry from “flow-stats” hash-table (step 6-12). Otherwise, if the flow entry is not classified, we continue updating the entry in “flow-stats” with `skb` and `flow` parameters by calling `flow_feature_update` (step 13-14).

Algorithm 2 describes the procedure of creating or updating an entry in “flow-stats” hash-table. Initially, the procedure calculates the hash value from `skb` parameter to search the right entry from “flow-stats” table (step 2-3). If the entry is found, then it keeps record of packet sizes, timestamps, and update `match_mask` field with `flow` parameter (steps 4-8). In this case, if an entry is not found in the “flow-stats”, then we create a new entry, and increase the entry counter of the “flow-stats” (`flow-stats->n_entry`). In algorithm 2, the steps 19-22, describe the action of initiating a “upcall” for starting the “poll thread” in the “flow classifier” module. In later subsection, we will describe details about the “poll thread”. The “upcall’ action for “poll thread” take place when there is at least one entry in the “flow-stats” table that has a corresponding entry in the “flow-table” cache, and is not yet classified. In addition, we also check whether any “poll thread” is running in the user-space(i.e., by checking variable `poll_thread_enable`) to avoid making any unnecessary “upcall” action.

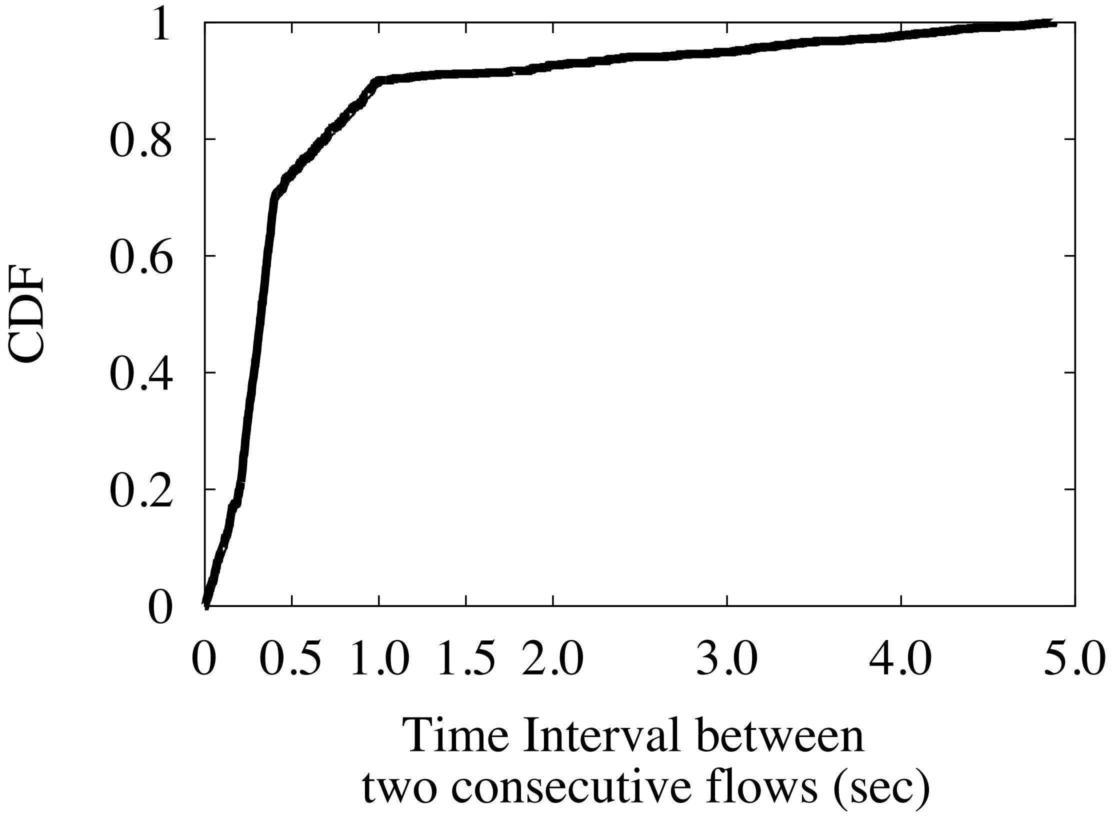


FIG. 21: The cdf plot of the time interval time between two consecutive flows in same app. It shows the characteristics of new flow creation in batch.

5.4.2 FLOW CLASSIFIER

The “flow classifier” module has two main tasks; i) Collect flow-level statistics (i.e.,

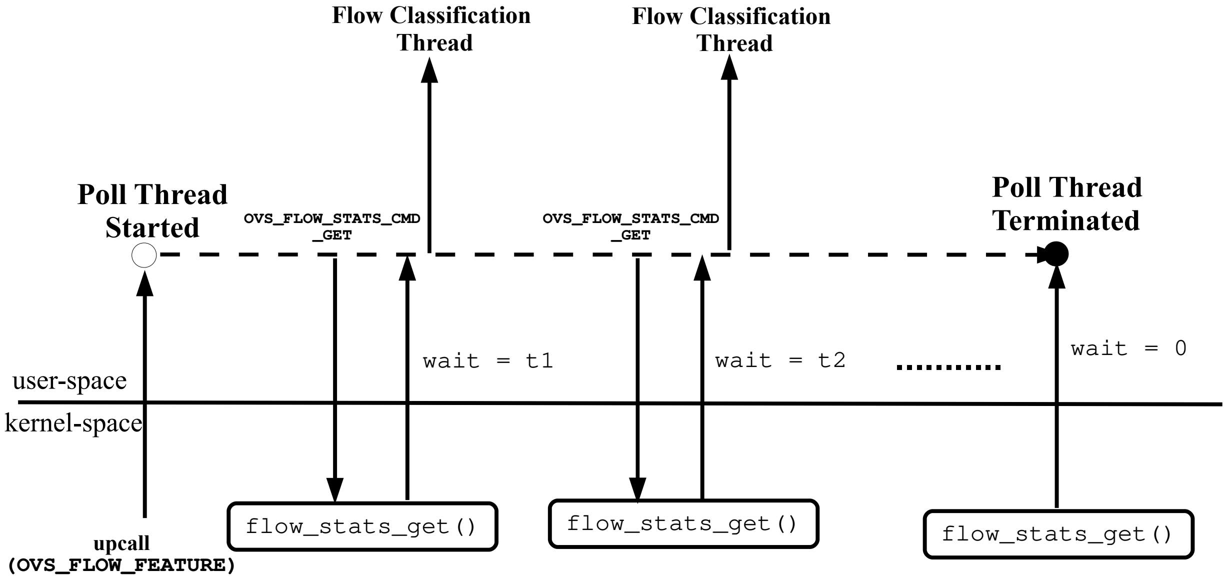


FIG. 22: Polling thread for collecting statistics from the Datapath.

packet sizes and arrival timestamp) and flow information (i.e., source port, destination port, IPs, protocol, TOS bit, MPLS) of the flow entries from "flow-stats", and ii) Extract features and apply classification on the collected flow entries from "flow-stats". In this section, we describe both of these tasks in details.

Collection process

In collecting flow-level statistics and flow information, we took the polling approach from the user-space to reduce the overhead on the packet processing pipeline of the OVS datapath. In OVS user-space, the "flow classifier" module runs a thread, "poll thread", which is responsible for the repetitive action of polling flow-level statistics and flow information of flow entries from "flow-stats". Figure 22, shows the life cycle of "poll thread" that initiate a function `flow_stats_get()` in the datapath, which is responsible for replying back with the available flow-level statistics and flow information. In addition, this function also returns `wait` time, which is used for scheduling the next polling action of the "poll thread". The "poll thread" terminates when function `flow_stats_get()` returns `wait` of 0 value, which means there is no entry in "flow-stats".

Algorithm 3 provides the overview of function `flow_stats_get()` that iterates over

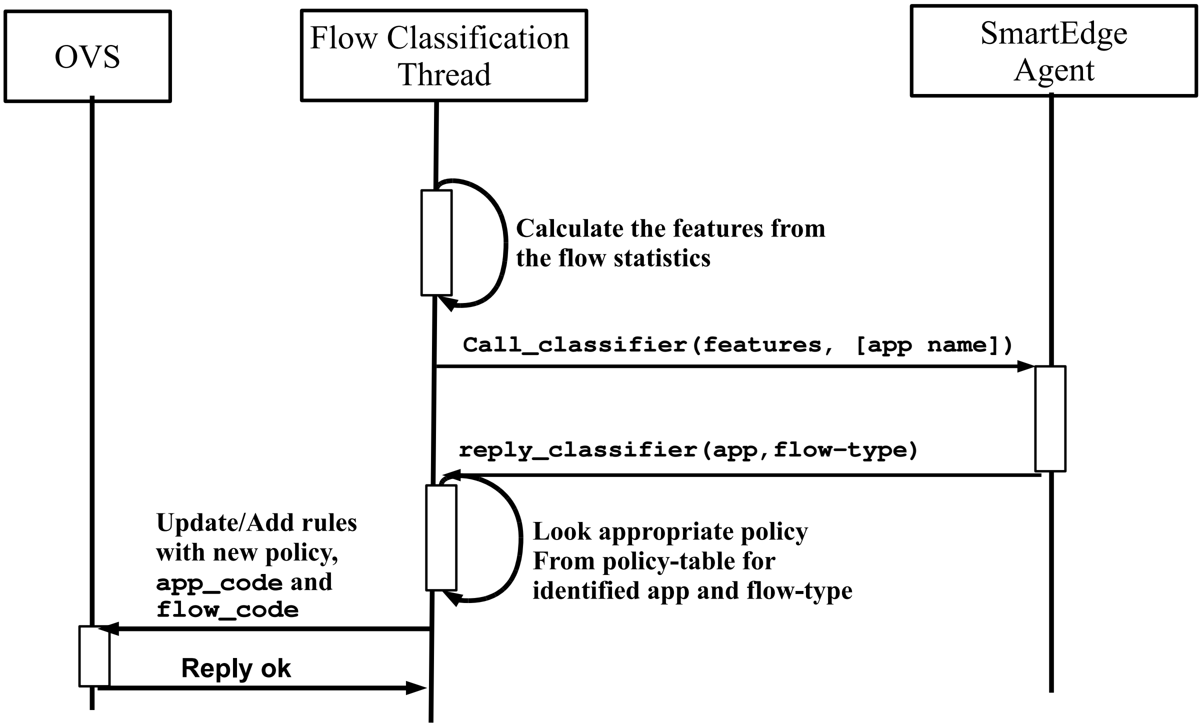

OVS
Flow Classification Thread
SmartEdge Agent
Calculate the features from the flow statistics
Call_classifier(features, [app name])
reply_classifier(app,flow-type)
Look appropriate policy From policy-table for identified app and flow-type
Update/Add rules with new policy, app_code and flow_code
Reply ok

FIG. 23: The sequence diagram between different modules of *SmartEdge* for extracting features, classifying, and applying policies per flow entry.

"flow-stats" to select flow entries that have sufficient amount of flow-level statistics with non-null matching_maskWe mean by sufficient that the flow entries have flow-level statistics collected during `time_window` period at the beginning of flow life-time. In addition, the search process also checks the value of `matching_mask` field. If an entry with null `matching_mask` field, it indicates that the "flow-stats" entry has no link to the corresponding entry in "flow-table" cache. In that case, the application and flow-type information of that entry are not known. Consequently, we will not be sure whether the flow entry needs to be polled for classification. Hence, we wait until `matching_mask` field get updated to be considered in the next polling action.

In designing poll-based flow-statistics collection process, we focus on two important performance criteria. The first is to maximize the number of entries collected per polling operation. The second is to minimize the delay of polling an entry that has already meet the criteria. Considering these two performance criteria, we focus on finding an optimal schedule of the polling operation during the life cycle of the "poll thread". In order to do that, first, we have conducted an experimental study, where we have collected large

Algorithm 3 Algorithm for collecting flow statistics from `flow-stats` table

```
 1: procedure FLOW_STATS_GET
 2:     time_window=2000                                          ▷ This can be configurable
 3:     if flow_stats->n_entry==0 then
 4:         wait=0
 5:         atomic_dec(poll_thread_enable)
 6:     else
 7:         wait=time_window
 8:         current_time=getTime()
 9:         for each entry from flow_stats do
10:             t = first packet's timestamp of entry
11:             w=current_time-t
12:             if w>time_window && entry->match_mask!=null then
13:                 Collect all timestamp and packet size statistics of entry
14:                 Send the statistics to the user-space
15:             else
16:                 wait=min(wait,w)+t
17:             end if
18:         end for
19:     end if
20:     Send the wait value to user-space
21: end procedure
```

scale of flow-level information for different categories of mobile applications[2]. From the study, Figure 21 shows the cdf plot of the time interval between the starting time of two consecutive flows from the same application. The cdf plot shows 75% of the time interval between the starting time of the two consecutive flows are less than 400ms. Therefore, in the context of mobile applications, it seems flows are often started in batch. This observation is used in function `flow_stats_get()` to figure out the polling time schedule.

In `flow_stats_get()`, for calculating the next time schedule for polling action(i.e. `wait`), first we search the entries which have not yet meet the time window criteria yet. Then we take the measurement of how far those entries are from meeting the `time_window` requirement. Among those measurements, first, we select the *min* value. Then we set the next polling schedule, `wait=min+t` , where `t={100,200,400}` (step 28 in algorithm 3). We select the maximum value of `t` to 400 based on the previous study at figure 21. Figure 5.4.2 shows the statistics (i.e., mean and variance) of flow entries per polling action from "flow-stats" for different values of `t`. Figure 5.4.2 clearly shows the trend that the number of flow entries per poll action increases with the increase of polling schedule time. Figure 5.4.2 shows the time delay statistics of polling the flow entries that have already meet the `time_window` criteria. Here, we calculate the delay by taking the time difference between the polling time and the time when it meet the `time_window` requirement. Figure 5.4.2

[2] 40+ different mobile applications of following categories; social network App, Real-time video/voice chat app, Real-time screen sharing app, browsing app, email, video streaming app, voice streaming app, gaming app, calendar app etc

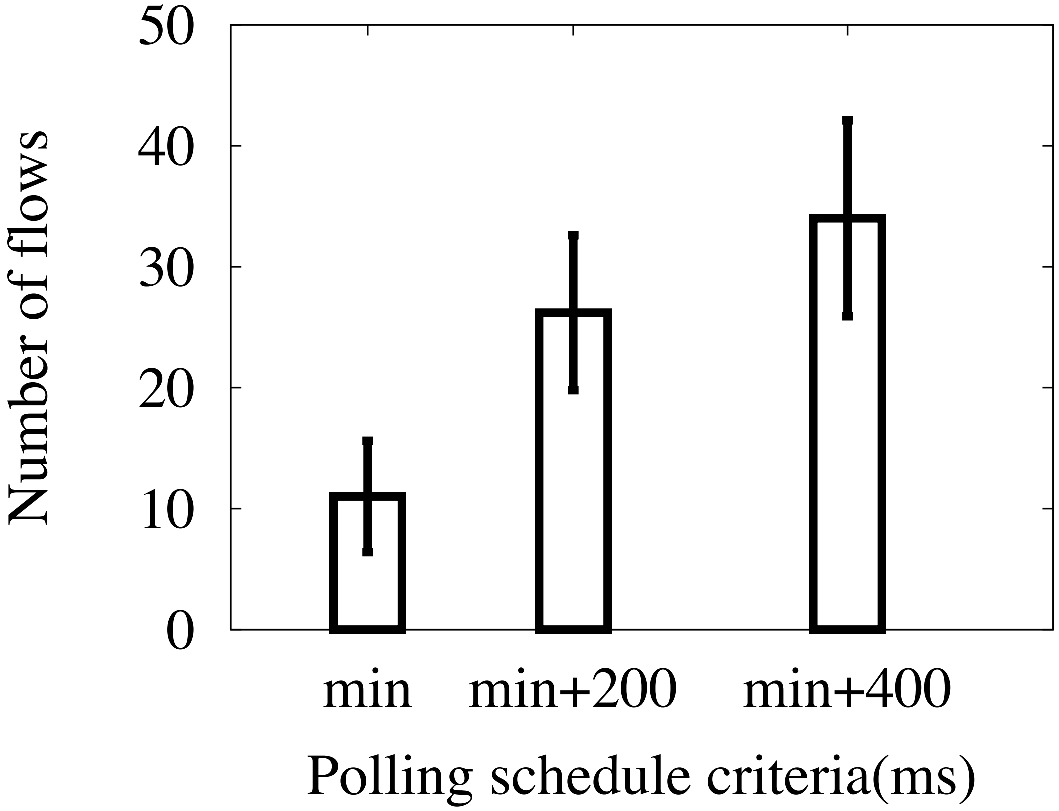


FIG. 24: The statistics (i.e., mean and variance) of the number of polled entries for different values of `wait`.

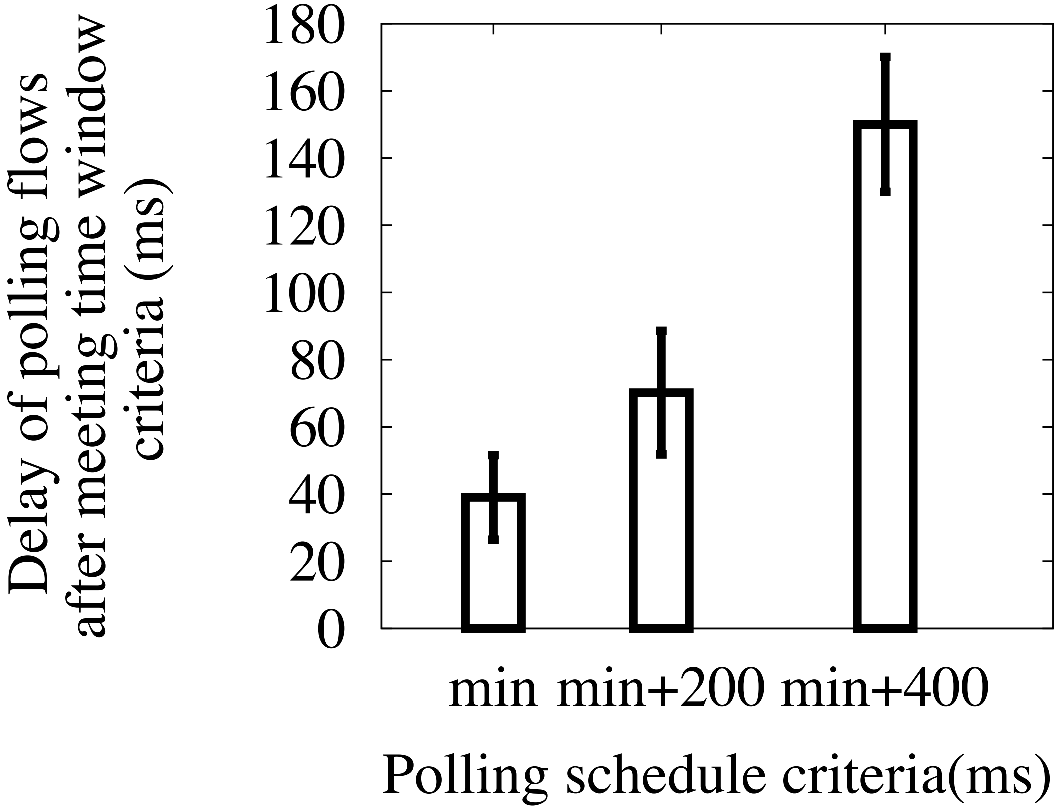
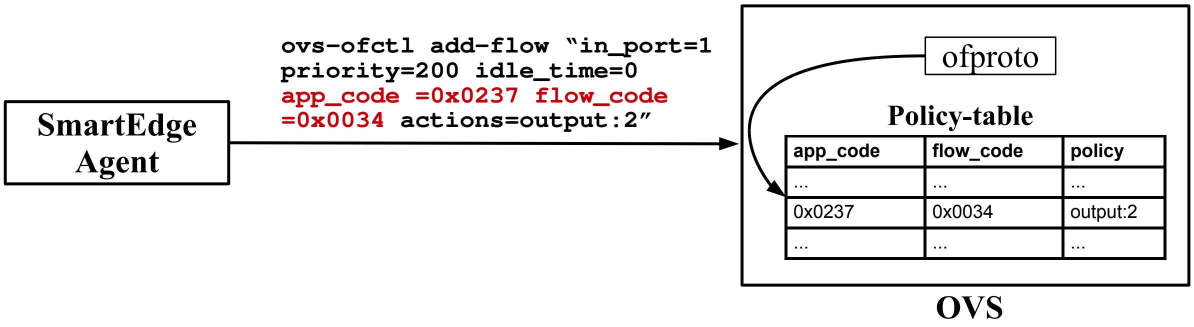


FIG. 25: The statistics of the delay of polling the flow entries after meeting the time window requirement for different values of `wait`.



```
ovs-ofctl add-flow "in_port=1
priority=200 idle_time=0
app_code =0x0237 flow_code
=0x0034 actions=output:2"
```

app_code	flow_code	policy
...	...	...
0x0237	0x0034	output:2
...	...	...

FIG. 26: The interaction between different different modules of *SmartEdge* for setting policies in policy table

shows the time delay increases with the increase of polling schedule time. These two figures (Fig 5.4.2 and 5.4.2) show the time delay and the number of flow entries per poll action for different values of `t`. In the optimal schedule, our objective is to minimize the time delay and increase the number of flow entries per poll operation. Considering this tradeoff we select `t=200` for `time_window=2000`, which we found to provide near optimal performance for scheduling the polling action. In addition, instead of polling continuously, we initiate the polling action when there is a flow entry in "flow-stats". We terminate the polling when there are no flow entries in "flow-stats". Thus, we reduce the overhead of unnecessary polling action.

Classification process

Once the "poll thread" receives the flow-level statistics and flow information of flow entries from "flow-stats", it initiates "flow classification thread". The "flow classification thread" is responsible for extracting the features from each collected flow entries, and initiate the interaction with the local "SmartEdge agent" for classifying the flow. In the sequence diagram of Figure 23, "flow classification thread" sends the calculated features to the "SmartEdge Agent" to apply classification model (i.e. `call_classifier`). In return, after applying the appropriate classifiers models "SmartEdge agent" returns with application (i.e., `app_code`) and flow-type (i.e., `flow_code`) information. Finally, based on the identified application and it is flow-type, "flow classification thread" apply policies on the corresponding flow entry of the OVS "flow-table" according to the "policy-table".

In OVS user-space, “flow classifier module” maintains a “policy-table” where each entry consists of application code (i.e., `app_code`), flow code (i.e., `flow_code`), and policies. In “policy-table”, policies of each entry is basically the supported action commands of OpenFlow protocol. “Policy-table” allow *SmartEdge* to apply application and flow-type aware policies on the classified network flows. In “policy table”, we represent the application as 16 bit application code, called `app_code`. Similarly, we represent the flow type as 16bit flow code, called `flow_code`. The “flow classification thread”, matches the returning `app_code` and `flow_code` from “SmartEdge agent” with “policy-table” to identify the right policies to apply on the classified flow. In addition, “flow classification thread” makes appropriate changes in “flow-table” to update application and flow-type information of the classified flow entry. Furthermore, it removes the classified flow entries from “flow-stats” (in algorithm 1, step 6-12).

5.4.3 SMARTEDGE AGENT

“SmartEdge agent” is an agent in the client devices or in the access device that act as an interface for the “policy controller” to apply application and flow-aware policies on the wireless network edges. This module maintains a database of “classification models” for mobile applications and it is flow-types. It also provide APIs to apply application and flow-aware policies on the traffic. We extend the standard `add-flow` OpenFlow command that allows “SmartEdge agent” to setup policies in the “policy-table” at OVS (figure 26).

5.5 APPLICATION AND FLOW-TYPE AWARE POLICY EXAMPLES

5.5.1 SCENARIO 1

In this example, we develop another application that would take advantage of deploying *SmartEdge* at a home network. Suppose that John, the householder, and his family members have several smartphones and a tablet. They usually use their devices to watch movies and other videos from Netflix, YouTube, and other popular video providers. As the tablet has a bigger screen, he would like to have a better video quality streamed to the tablet whenever there are other devices streamed videos at the same time. Having this requirement, we develop a control application to allocate more bandwidth to the video flow streamed by the tablet to enhance the quality. This application allocates more bandwidth to the tablet by limiting the traffic rates of other devices (smartphones). Consequently,

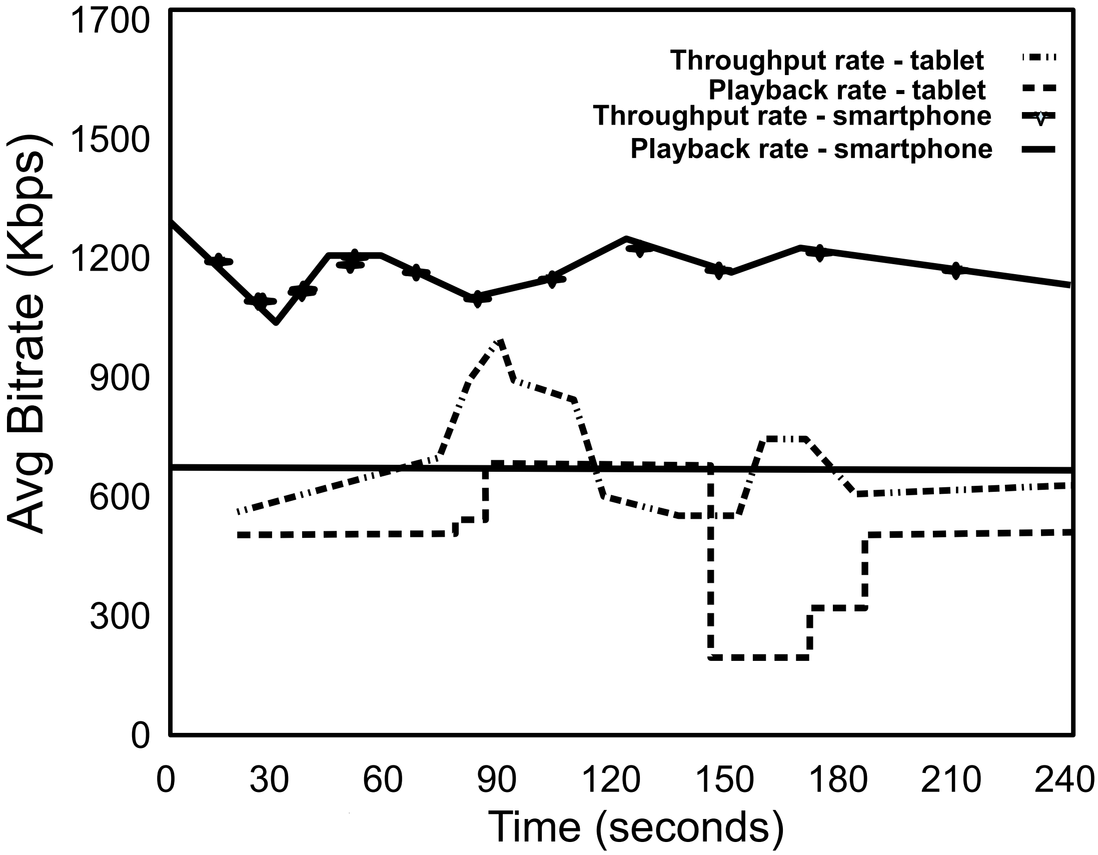


FIG. 27: Throughputs and video rates for one smartphone and one tablet without activating *SmartEdge*.

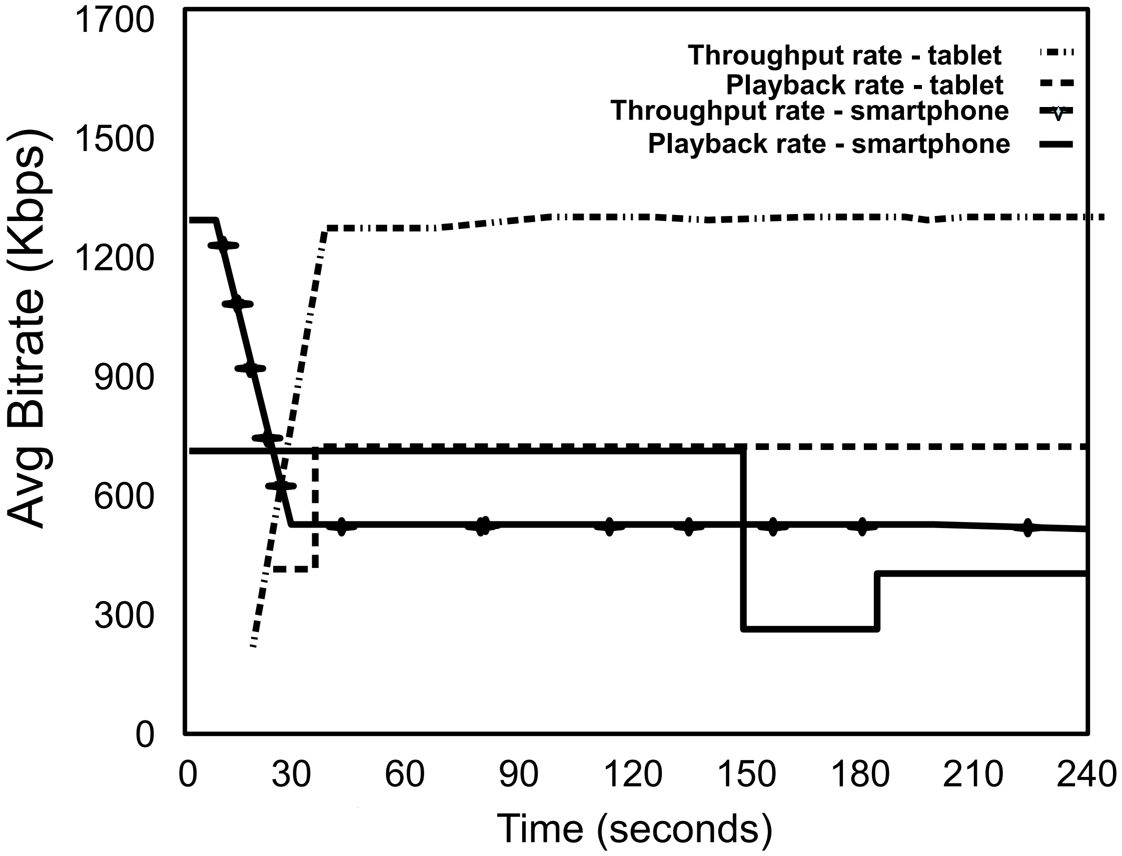


FIG. 28: Throughputs and video rates for one smartphone and one tablet with *SmartEdge* is being activated.

the video player on the tablet would experience a high throughput and thus can request a higher quality profile. Intuitively, this policy of device-based rate limiting is only applied when there is a video stream initiated by the tablet and automatically disabled as soon as the player finished downloading all chunks.

We use a prototype similar to the one used in the first example but with one smartphone and one tablet. To evaluate our prototype, we use the YouTube app on both devices to stream and watch the same video. The selected video is encoded with five bit rates ranges from 245kbps to 730kbps corresponding to the resolutions 144p to 480p. We start streaming the video on the smartphone first and after a couple of seconds we start playing the other video. We measure the throughput and the playback rates on both devices while *SmartEdge* is not being activated, and then when *SmartEdge* is activated. We can see from Figure 27 that without *SmartEdge* the player on the tablet not only suffers from poor viewing quality, but also has large instability in the quality. Figure 28 on the other hand shows the performance when we activate *SmartEdge*. It is apparent from the figure that the video player on the tablet achieves better viewing quality (730kbps/480p) in addition to better stability due to the bandwidth reallocation performed by *SmartEdge*. These great results reveals the necessity of deploying *SmartEdge* at the network edge to such important applications.

5.5.2 SCENARIO 2

As a proof of concept, we develop a simple application aware traffic management policy that leverages *SmartEdge* framework. We consider a scenario, where in enterprise WLAN setting, company has a policy to use their proprietary video chat application other then Skype, therefore they want to offload the skype video chat traffic to cellular during office hour. Considering this scenario, we develop a “control application” that force the user’s device to run the Skype Video chat over cellular interface. Where in the evening, the control application allows the Skype Video chat to run over the company’s enterprise WLAN. However such policy of offloading is not applied on the Skype voice chat traffic at any time of the day. In this enterprise scenario, we assume that the client devices are running an agent software of the company that switches the interface after receiving the command directly/indirectly from our traffic management policy application that are running at access device.

In the prototype setup, we used 8 android phones and one laptop. The ethernet interface of the laptop is connected with the LAN and the Wi-Fi interface is used as AP

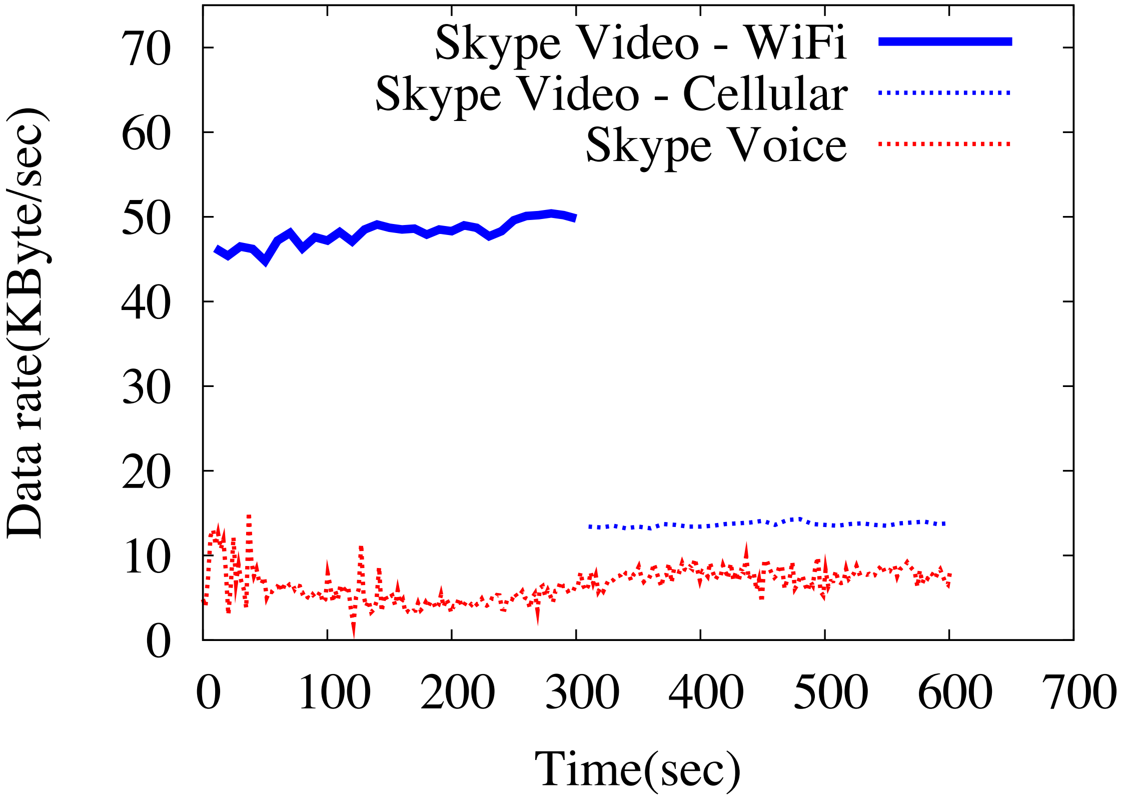


FIG. 29: Traffic Management of offloading skype video chat traffic.

for the smartphones. In addition, we run our traffic management policy application and *SmartEdge* in the laptop. In the client devices, we run an “agent” software that received direct command from the “policy application” about switching the interface. In the evaluation of our prototype, we run skype video chat in one android smartphone and skype voice chat in another smartphone. The rest of the smartphone generate background traffic using `iperf`. In laptop the “policy application” set a policy to offload the skype video chat flow to cellular after certain time. In Figure 29 shows how skype video chat traffic flow has offloaded to cellular interface after about 5 minutes, where in another smartphone skype voice chat traffic remain over the Wi-Fi interface. We see in the plot 29 that the data rate drop when skype video chat flow offloaded to cellular, which is HSPA+. This prototype application shows the potentiality of the real-time, and more fine-grained policy making capability of *SmartEdge* framework.

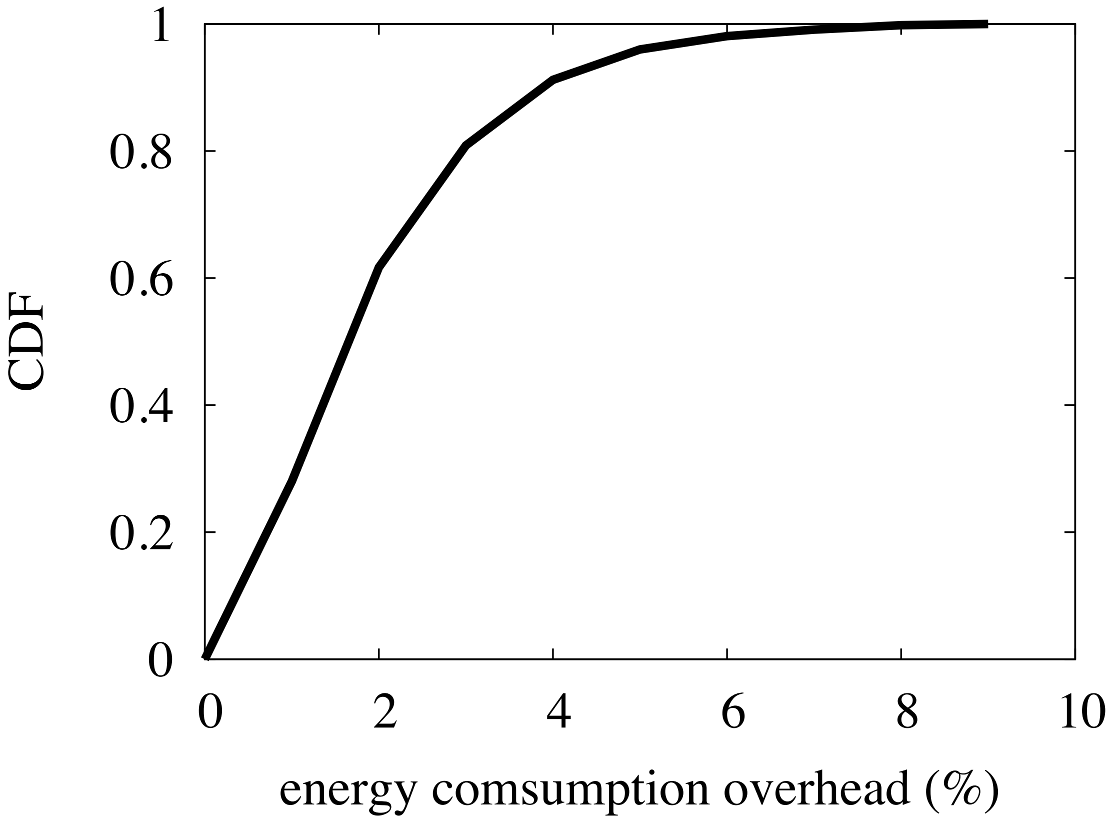


FIG. 30: In ACM, energy consumption overhead (in percentage) of running the *SmartEdge* modules in the Nexus 4 mobile device.

5.6 EVALUATION

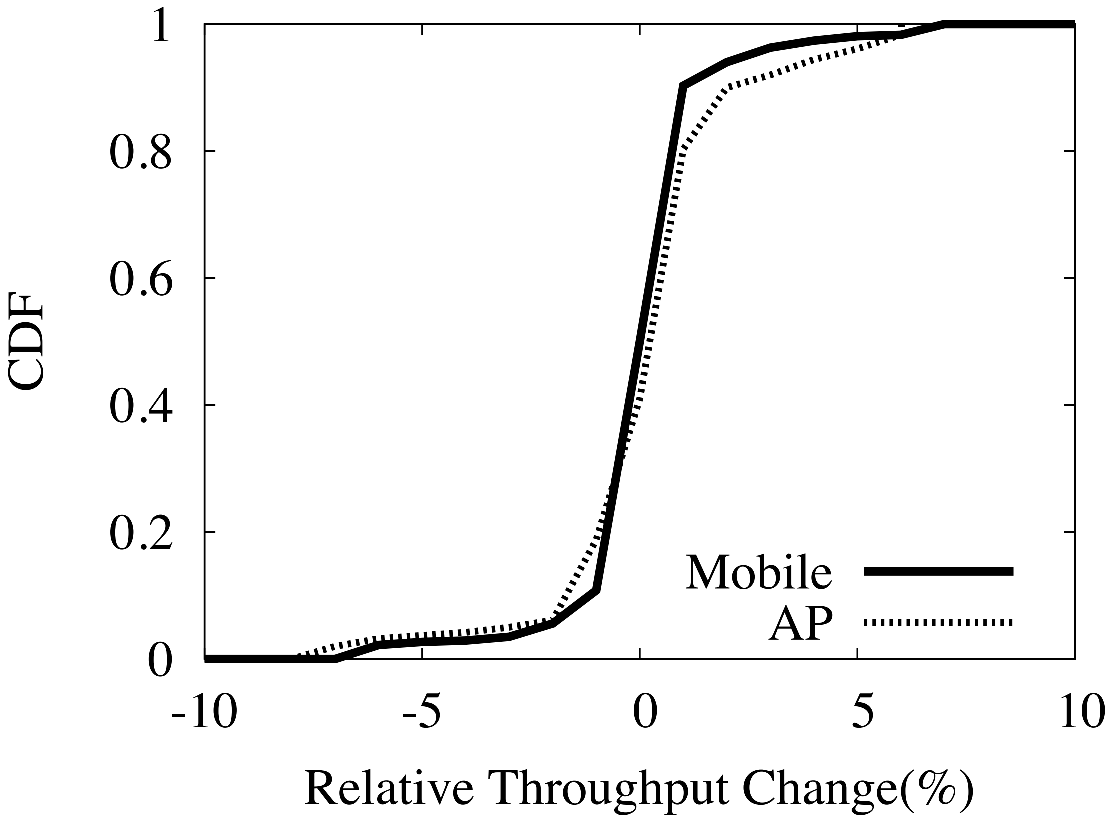
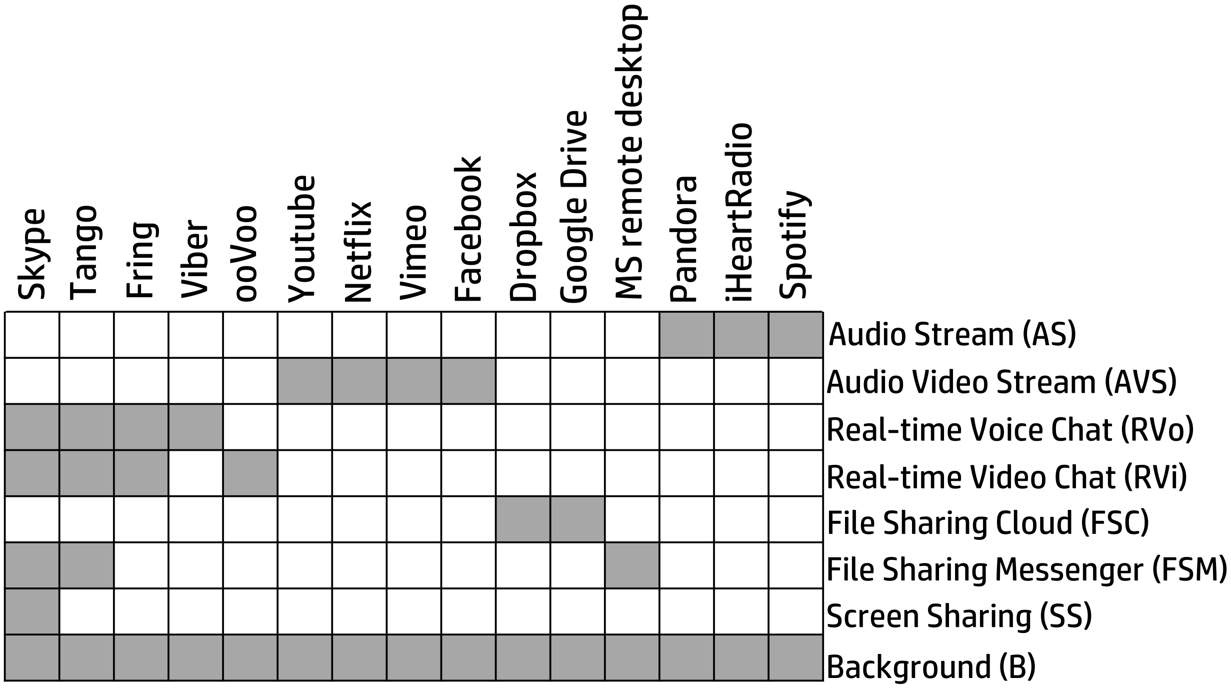


FIG. 31: Relative throughput changes, while using *SmartEdge* for both ACM and PCM.

In this section, we evaluate *SmartEdge* for both "Active Client Mode" (ACM) and "Passive Client Mode" (PCM) in respect to energy efficiency, classification accuracy, system performance, and network performance. We have tested *SmartEdge* on the following dataset,

Dataset 1 was collected from a production enterprise WLAN and used for application detection and new-app detection experiments. The agent has been deployed on eight volunteer Android phones, as well as two dedicated testing phones for manual collection. The manual collection was needed to collect a reasonable sample size for the applications of interest. Over 100K flow samples of 89 different applications were collected, along with their application name during the period of 4 weeks. Among them, 36 are the most popular applications according to Google play, and the remaining 53 applications are labeled as "Other 53 apps" class.

Dataset 2 was collected from a University campus network and used for the flow-type detection module. In term of user choice [56], we choose 40 applications that are the most



Skype	Tango	Fring	Viber	ooVoo	Youtube	Netflix	Vimeo	Facebook	Dropbox	Google Drive	MS remote desktop	Pandora	iHeartRadio	Spotify	
												▨	▨	▨	Audio Stream (AS)
					▨	▨	▨	▨							Audio Video Stream (AVS)
▨	▨	▨	▨												Real-time Voice Chat (RVo)
▨	▨	▨		▨											Real-time Video Chat (RVi)
									▨	▨					File Sharing Cloud (FSC)
▨	▨										▨				File Sharing Messenger (FSM)
▨															Screen Sharing (SS)
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Background (B)

FIG. 32: Selected popular apps and the corresponding flow types in the dataset .

popular application among different mobile app category (e.g. a video stream, video/audio chat, audio stream, social networking etc.). The flows from these applications were categorized into seven major flow types as shown in Figure 32. We categorized all other flows that do not fit into any of the seven flow types as *Background*. Each application consisted of flows corresponding to at least two different flow types including Background. In total, up to 3 million samples (windows of minimum 200 msec) corresponding to 1200 flows with a net flow duration of over 10, 000 minutes was collected during these experiments.

5.6.1 ENERGY EFFICIENCY

The energy efficiency evaluation is more relevant in ACM, where *SmartEdge* runs on the mobile device. Because in mobile devices energy is an issue of scarcity. In order to evaluate the energy efficiency, we measure the increase in energy usage (in percentage) while running *SmartEdge* in the mobile device for different categories of applications. Figure 30 shows the cdf plot of energy usage increment (in percentage) for running *SmartEdge* in the mobile device. The plot shows that there is 1.8% increase of energy usage on average or different categories of applications. Note that, among the applications, the social app has the highest energy consumption, where video streaming has the lowest. Furthermore, in an idle scenario with no application running, *SmartEdge* shows 0.2% energy usage increase in the mobile device. Despite the increase of energy usage, ACM has the privilege of knowing ground-truth application package name of a flow. Thus in *SmartEdge*, ACM provide higher classification accuracy compare to PCM for both application and flow-type classification.

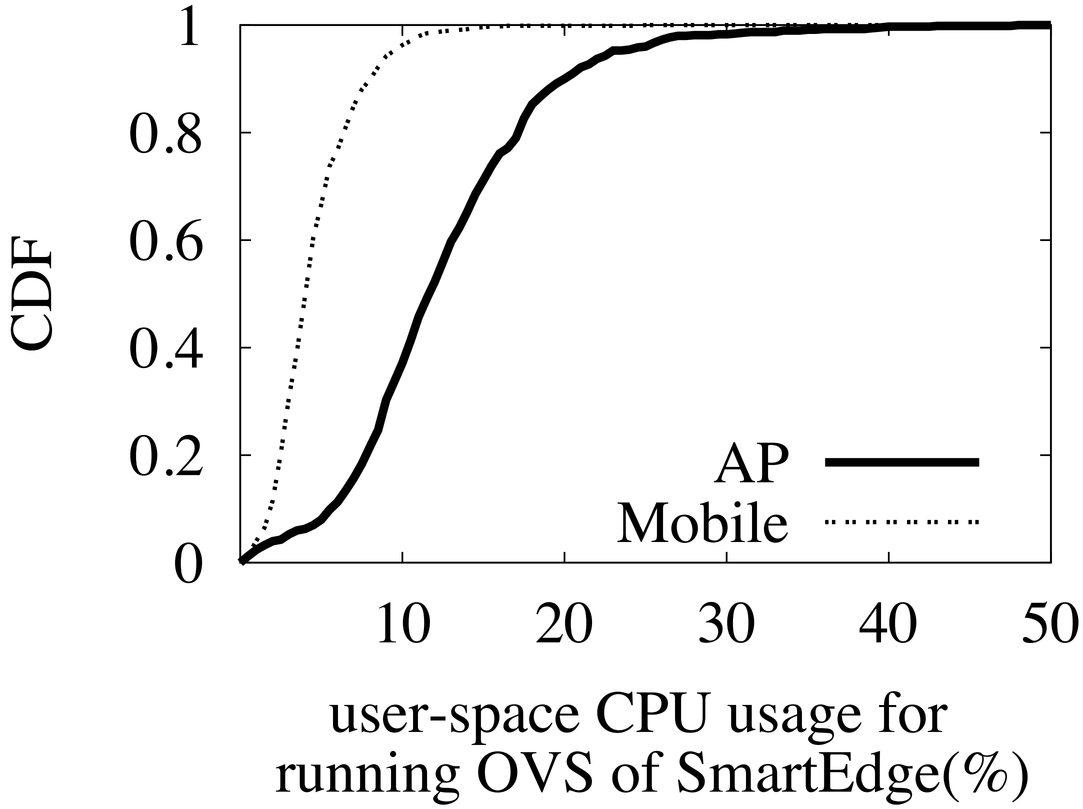


FIG. 33: Comparison of user-space CPU usage distribution of running *SmartEdge* OVS modules between ACM and PCM.

5.6.2 SYSTEM EVALUATION

In this subsection, we evaluate *SmartEdge* for both ACM and PCM context in respect to CPU usage overhead, system response, and effect on the throughput. Note that, we use Open vSwitch version 2.3 in our implementation. In ACM, we use Nexus 4 smartphone (2 core) to run the *SmartEdge*. In PCM, we use a Linksys E3000 (450MHz) as our Wi-Fi AP, where we run *SmartEdge*. We also associate one mobile device with the Wi-Fi AP for PCM. In order to evaluate the system for both PCM and ACM, we try our best to have same/similar traffic from the same applications and it is flow type. Furthermore, we repeat the experiment with the similar setup for reducing the random nature impact of the traffic

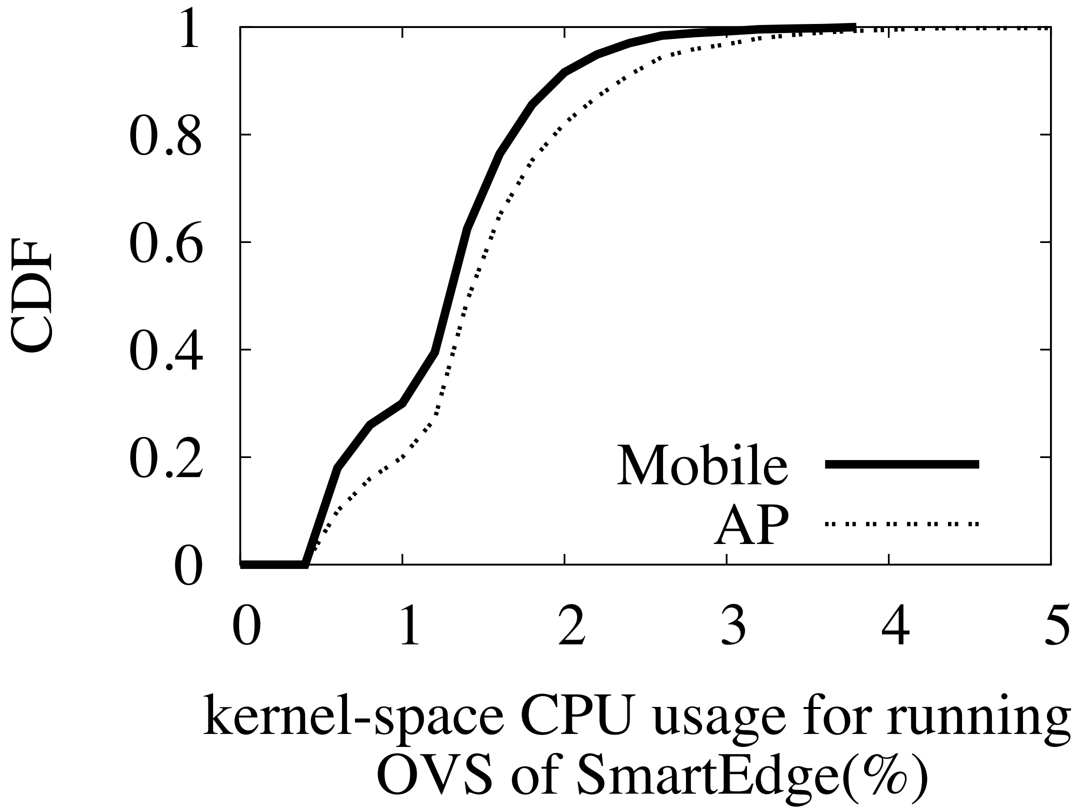


FIG. 34: Comparison of kernel-space CPU usage distribution of running *SmartEdge* OVS modules between ACM and PCM.

flows.

We observe that the user-space component of the *SmartEdge* has 2% CPU usage on average in mobile device (i.e. Nexus 4 smartphone). However, the user-space CPU usage increases with the frequency of "upcall" operation. In *SmartEdge* we have extended the OVS user-space to handle new class of "upcall" operation, and to run an additional thread of polling entries from the kernel space. This extension increase the CPU usage of the user-space. Figure 33 shows the user-space CPU usage of running *SmartEdge* in both mobile device and AP. Note that the AP has very slow processing speed (450MHz) compare to the smartphone (1.5GHz quad-core Snapdragon S4). In addition, in PCM Wi-Fi AP has the additional task of application classification. Therefore we see that the mobile devices has much less CPU overhead compare to the AP. Note that in PCM the CPU usage measurement is for just one mobile client. In that case, running multiple mobile devices might have significantly large CPU usage overhead. Thus, we recommend to use computationally powerful AP for running the *SmartEdge* system.

The standard kernel-space component of the OVS, Datapath has 1.2% CPU usage on average in the mobile device. In *SmartEdge* we have extended the Datapath to have an extra step of processing a packet in "flow feature engine". However, this extra step is equivalent to a single hash-table access operation (i.e. `O(1)`). Therefore, we observe negligible overhead of the extended Datapath both in the mobile device and in the AP. The figure 34 shows the statistics of the CPU usage for running the extended Datapath both in the mobile device and in the AP. Almost 80% of the cases extended Datapath shows 1.5% CPU usage or less in the mobile device. Similarly, in the AP extended Datapath shows 1.9% CPU usage or less for 80% of the cases.

Figure 31, shows the relative throughput changes of running *SmartEdge* both in the mobile device and in the Wi-Fi AP. Note that, in both cases *SmartEdge* has no noticeable relative changes on throughput. Typically, OVS is designed to operate on very high line-speed such as 10Gbps, which is much higher than what we see at the wireless network edges. Moreover, In *SmartEdge* the collection process runs as a separate "poll thread" from the main thread of handling an incoming packet in the kernel space. Furthermore, we use a separate lock for each bucket of the "flow-stats" hash table. Therefore, the task of "poll thread" has a negligible effect on the main packet processing thread in kernel-space. Thus, we see no relative changes of throughput regardless of running in the AP or in the mobile device. Similar to throughput, we also do not see any significant changes in the RTT value compare to normal case.

The figure 35 shows the distribution of total time require for a flow to get classified using *SmartEdge* for both in mobile device (ACM) and in Wi-Fi AP (PCM). Note that Wi-Fi AP and mobile device has different clock frequency speed. Therefore absolute time measurement can not provide accurate comparison between them. Hence, we use equation from [57] to convert the actual time measurement in Wi-Fi AP to the equivalent measurement of mobile device. Figure 35 shows comparison between the converted time measurement of Wi-Fi AP, and the actual measurement of mobile device. Note that, in figure 35 Wi-Fi AP shows more time on average compare to mobile device. Because, unlike in mobile device, Wi-Fi AP has the additional task of application classification on the flow.

5.6.3 APPLICATION DETECTION

Figure 36 shows the application detection accuracy for the 37 application classes in our dataset. This figure shows the average precision and recall over the 10-fold cross-validation,

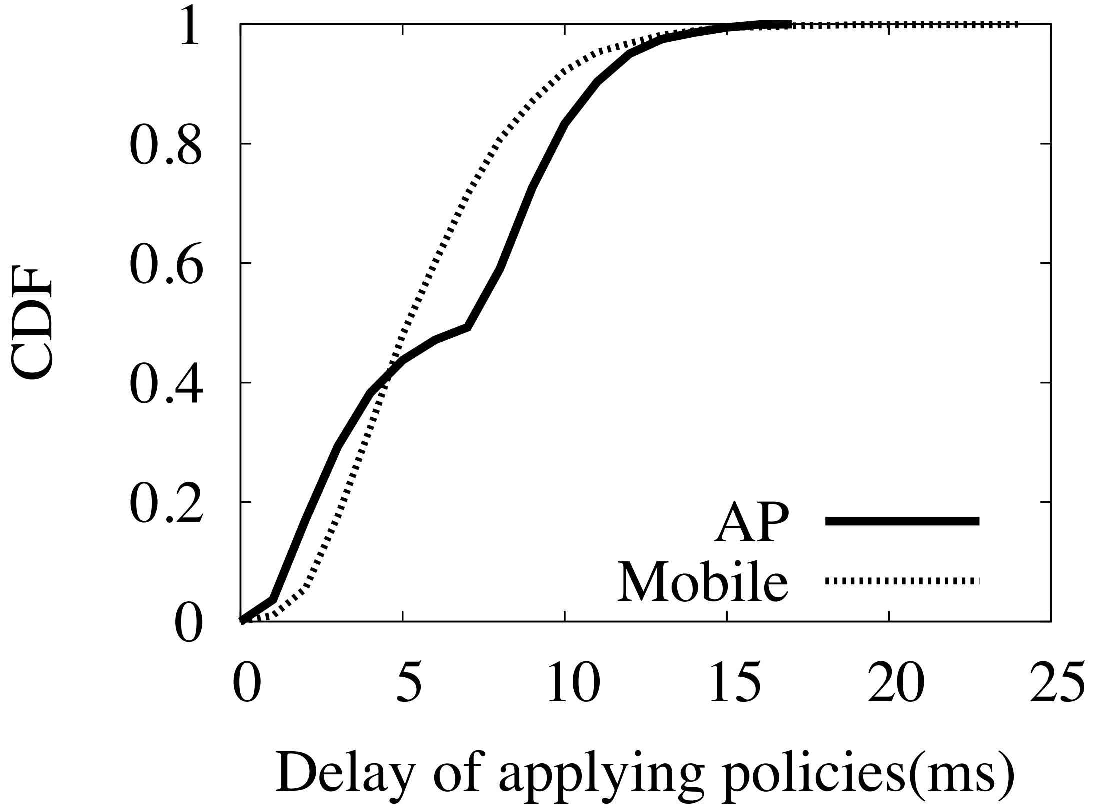


FIG. 35: The distribution of total time spend on collecting the flow statistics from kernel-space, extracting features, applying classification, and finally updating application and flow type information in the "flow-table" for a flow entry from "flow-stats" hash-table.

where the applications are ordered in a decreasing order of their size. We achieved an accuracy greater than 90% for most applications, with an overall accuracy of 95.5%. In addition, eight popular applications (including Microsoft Exchange service, Facebook, and Google+) achieved an accuracy of 100%. These eight applications constituted around 40% of the flow dataset. Moreover, 60% of the flows in our dataset were over HTTPS, which demonstrates the ability of the ML based approach compare to DPI-based classifiers. Note that the average application detection accuracy does not vary significantly beyond the top 7 packets.

5.6.4 NEW APPLICATION DETECTION

We tested the new application detection algorithm on Dataset 1, where we used sizes of the first 7 packets as flow features. We clustered the training data using the proposed iterative clustering technique, where the cluster radius r corresponds to a value of 10 bytes, as we observed the deviations in the signaling packet sizes to be within 10 bytes for most apps[3]. The clustering algorithm terminated after identifying 121 clusters, which constituted around 80% of the training data. The remaining 20% flows were quite diverse with no coherent pattern, and hence, could not be clustered.

We performed real-time new application detection on this data subset X, constituting 80% of the training data. In order to simulate a realistic detection scenario, we conducted two experiments with each flow in the dataset. In the first experiment, we compare flow $i \in X$ with the residual dataset that consists of all flows except for i (i.e., $X \setminus \{i\}$). In the second experiment, we compare flow $i \in X$ with the residual dataset that consists of all flows except for the flows belonging to the same cluster as i (i.e., $X \setminus \{j : y_j = y_i\}$, where y_i corresponds to the cluster label for flow i). The first set of experiments corresponds to a scenario where similar flows are still part of the residual data while none of the similar flows are part of the residual data in the second set of experiments. Together these experiments determine the effectiveness of the proposed approach to distinguish a new application flow versus an existing flow. We quantified the accuracy of our new application detection algorithm using F-measure. We achieved an overall accuracy greater than 99%.

Finally, we demonstrate in Figure 37 the effectiveness of the proposed clustering technique over traditional k-means in finding tight clusters. This figure compares the coverage (i.e., fraction of flows that lie within a radius r of the resulting clusters) using these two approaches. Note from this figure that the proposed technique achieves a significantly better

[3] Using a radius optimal for each app could further improve the accuracy, which is our future work.

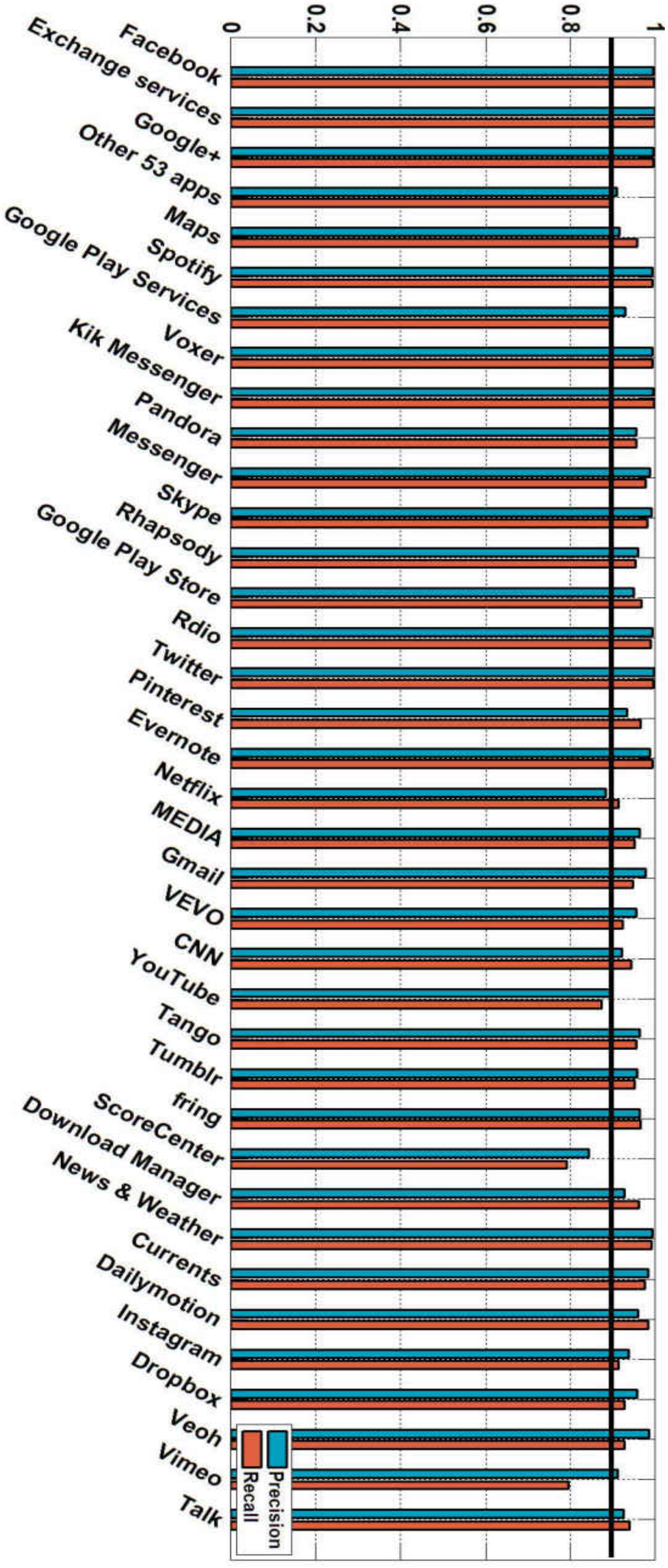


FIG. 36: Application Detection Accuracy Using the top 7 packet sizes in the flow features (the applications are ordered in a decreasing order of their dataset size)

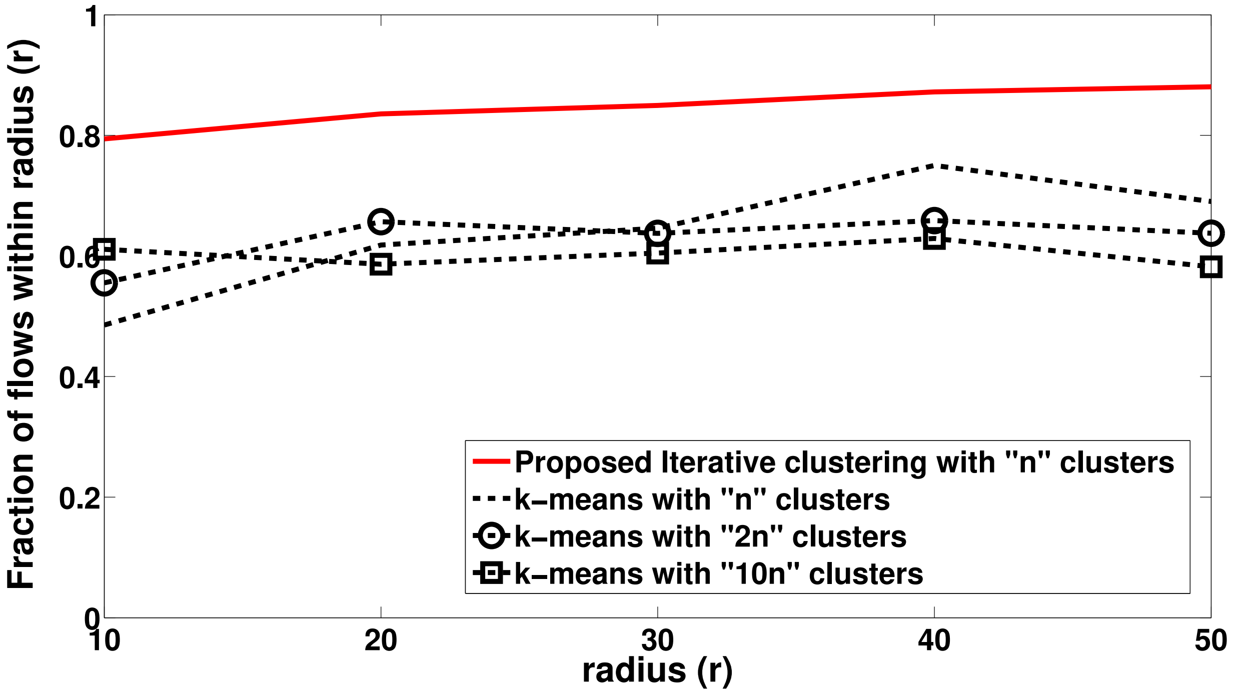


FIG. 37: Demonstrates the effectiveness of the proposed iterative clustering technique in generating tight clusters over traditional k-means using much fewer clusters.

coverage over traditional k-means.

5.6.5 FLOW-TYPE DETECTION

We test the 2 classifiers of the flow-type detection module on Dataset 2 using feature sets f_{DFT} and f_{DWT}. We used the *weka* [66] machine learning tool to build the GFT, and PAFT supervised *K-NN* classifiers, with $k = 3$, which we found to be optimal.

GFT detection

The *aGgregated Flow-type (GFT)* classifier is used to identify the flow-type corresponding to a flow from a new application, as identified by the new application detection module. To evaluate GFT, over the 15 tested applications, we built a classifier using the training data that contains all flows except for that application. The GFT classifier has eight classes corresponding to the tested flow types. Due to space restriction, we present the confusion matrix only for a subset of the applications in Table 2, using the f_{DWT} feature set and a window size of 200 msec. The accuracy of the GFT classifier is high (more than 90%) for most media flow types, except for the 'file sharing' and 'screen sharing' flow types (from

		Pandora		Skype					Youtube		Tango				MRD	
		AS	B	RVo	RVi	FSM	SS	B	AVS	B	Rvo	RVi	FSM	B	SS	B
Audio Stream	GFT	**86.9**	2.6	0	0	5	4.7	0	5.5	0	0	0	0	0	0	0
(AS)	PAFT	**97.3**	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0
Audio Video Stream	GFT	8.7	0	0	0	0	24.1	0	**89.1**	0	0	0	30.0	0	0	0
(AVS)	PAFT	0	0	0	0	0	0	0	**95.6**	1.3	0	0	0	0	0	0
Real-time Voice Chat	GFT	0	0	**94**	0	3	3.4	0	0	0	**95.4**	3.9	0	0	0	0
(RVo)	PAFT	0	0	**94.6**	0	0	0	0	0	0	**93.5**	2	0	0	0	0
Real-time Video Chat	GFT	0	0	0	**90.3**	15	0	0	0	0	2.6	**92.1**	0	0	0	0
(RVi)	PAFT	0	0	0	**95.7**	4	3	0	0	0	3.5	**94.8**	11.3	0	0	0
File Sharing - Cloud	GFT	2.4	5.8	0	5.8	0	0	7.9	0	6	0	0	38.7	10.3	7.5	1.8
(FSC)	PAFT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
File Sharing - Messenger	GFT	0	0	0	3.9	**37.5**	0	2	0	4	2	5	**27.8**	3.2	5.2	8.2
(FSM)	PAFT	0	0	0	0	**70**	4.3	4.2	0	0	0	2	**78.7**	11.8	0	0
Screen Sharing	GFT	0	0	3.5	0	39.5	**67.8**	0	5.4	0	0	0	3.5	0	**87.3**	0
(SS)	PAFT	0	0	2.3	4.3	26	**92.7**	3.5	0	0	0	0	0	0	**98.5**	1.4
Background	GFT	2	**90.6**	2.5	0	15	0	**90.1**	0	**94.0**	0	0	0	**86.5**	0	**90**
(B)	PAFT	2.3	**99.1**	3.1	0	0	0	**92.3**	4.4	**98.7**	3	1.2	10	**88.2**	1.5	**98.6**

TABLE 2: GFT and PAFT classifier accuracy using the f_{DWT} feature set for a 200 msec time window.

Skype and Tango), which are misclassified with ‘audio-video stream’ type flows. We relate this low accuracy to the very similar continuous TCP transfer patterns of these three flow types, which is hard to differentiate in 200 msec window; the accuracy increases to 98% when using a window size of 2000 msec (not shown in the table).

PAFT detection

The *Per-Application Flow-Type* (PAFT) classifier is used when the application name is already identified using app-detection of Section 5.6.3. In this case, we build and use a per-application classifier to identify the flow-type. We built a PAFT classifier for each of the 15 applications using only the flows corresponding to that application. We present the confusion matrix of PAFT for the subset of applications (same with the GFT evaluation) in Table 2. It clearly shows that the PAFT classifier outperforms the two other classifiers. PAFT yields more than 90% accuracy for most media flow types, except for the file sharing and screen sharing types that exhibit similar TCP patterns; the accuracy increases to 98% using a window size of 2000 msec (not shown).

5.7 SUMMARY

SmartEdge is based on extended SDN framework to support fine-grained, lightweight, and real-time traffic awareness at the wireless network edges. We believe, *SmartEdge* is the first system of its category that is designed, developed, and tested for the wireless edge devices. Specially in the context of *edge computing*, we require to have greater control and visibility over the traffic to enable smart network management and policies at wireless network edges. Therefore, it essential to have framework such as *SmartEdge* to provide fine-grained application and flow-type awareness. In *SmartEdge* we use the ML-based approach for classifying both mobile applications and its flow types in real-time. In this chapter, we evaluate SmartEdge for number of popular applications and its flow-types. We found that SmartEdge perform relative poorly in applications that generate lot of mice flows. For example, we have seen cases where facebook mobile app generate almost 270 flows in just 5 second. Among them many of them are mice flows that slow down the SmartEdge system. On the other hand video streaming, video/voice chat app generate few mice flow, which make SmartEdge perform efficiently.

CHAPTER 6

SAFEEND: AN APPLICATION-AWARE PROGRAMMABLE NETWORK SECURITY SOLUTION FOR MOBILE DEVICES

6.1 INTRODUCTION

In recent years, there is a significant growth of running *sensitive apps* (i.e., applications that communicate sensitive data, relative to the user, over internet) on mobile devices. This trend is prominent in different domains such as, physicians remotely/locally consult with colleagues and nurses in real-time to diagnose a patient more accurately and efficiently, retailers use tablets for processing electronic payments directly on the sales floor, and law enforcement officers use commercial phones to interact with their peer officers in ad-hoc manner during mission-critical operation. The motivations for using mobile devices for sensitive applications are diverse, including: cost reduction, minimizing carried equipment, and maintaining device/user interface familiarity. In addition, recent studies shows that over half of the connections made by mobile apps are insecure, except the established developer (i.e. Google, Facebook) [53].

Mobile devices mostly use wireless LANs (WLANs) (i.e., WiFi networks) as the prominent network interface to connect with the Internet. A recent research found that 64% of the smartphone users hit WiFi hot spots at least once a day outside their home or office (e.g., at cafe, hotel, school campus, train/bus stations, etc.) [163] and that 74% of smartphones data goes through WiFi link [64], which is broadcast in nature. Given that sensitive applications are running on mobile device, their traffics are exposed to any eavesdropping adversary and hence several potential attacks such as man-in-the middle, Denial of Service (DoS), etc. Although existing wireless security standards (e.g., 802.11i) provide certain level of security such as user authenticity, data confidentiality, and data integrity, not all wireless networks enforce such security level. For example, imagine that a person is consulting with his/her doctor about certain condition/advice through a specific mobile application while s/he is at a coffee shop . If the coffee shop uses an open Wi-Fi network, then the person has no control on the security level of the wireless network and, consequently, his sensitive data (e.g., type of decease, medication) could be compromised and utilized by any nearby attacker.

Several previous works show that, even with WiFi encryption (data and control frames), statistical analyzes of WLANs traffic such as frame size, data rate, ratio of incoming to outgoing frames, inter-arrival time of the frame, etc. could infer several user-related information such as user identity [123] and user's online activities [198]. For example, Zhang et al. use sophisticated Machine Learning (ML) algorithms to analyze WLAN traffic to recognize, with high accuracy (8̃0%), user's online activities such as browsing, chatting, gaming, downloading/uploading, video streaming and P2P file sharing. In this chapter, we present a new type of WLAN attack, *App-spoof* that allows the adversary to identify the applications currently running on user's smart device thru only eavesdropping and analyzing WiFi traffics.

In addressing the above security concern of sensitive applications on mobile devices, we propose a fine-grained programmable security solution, called *SafeEnd*, for mobile devices based weSDN framework. This solution is inspired by our vision of pushing the Software Define Network (SDN)-like paradigm all the way to wireless network edge, by deploying a software switch (i.e. Open vSwitch [171]) on mobile devices to have greater visibility and fine-grained control over the traffic generated from these devices. In SafeEnd, we extend and leverage Open vSwitch (OVS), a programmable software switch used in SDN framework, to virtualize the network between both sensitive and non-sensitive applications running on mobile device. With this virtualization, SafeEnd provides programmable and flexible APIs for applying fine-grained network security policies on the sensitive/non-sensitive application's flows.

Previously proposed mobile devices management (MDM) schemes [8, 31, 47, 61, 100, 114, 121, 143, 144, 177, 182] focus on protecting devices and applications' data but not user's sensitive information. Moreover, existing network security solutions for mobile devices are mostly enterprise solutions that requires infrastructure support in applying security policies [174]. Clearly, these solutions are not well-suited for mobile devices that keep moving between different infrastructure domains. Moreover, infrastructure based security solutions are typically use static and coarse-grained policies. Unlike these previous network security solutions, SafeEnd is independent of infrastructure and supports users' mobility. In order to guarantee end-to-end enforcement of network security policies, SafeEnd needs to be deployed at both ends of the application. In other words, while SafeEnd should be deployed at both mobile devices (e.g., smartphone, tablet, smart TV, Amazon Echo) for peer-to-peer mobile applications, SafeEnd should be deployed at both the mobile device and the cloud server for client-to-server/cloud applications.

In this chapter, we develop two types of network security policies for SafeEnd that protects the sensitive applications network flows. In the first policy, SafeEnd applies traffic shaping policy on flow traffic to obfuscate the eavesdropping attack (i.e., App-spoof attack). Previously, researchers have proposed mainly three types of traffic obfuscation approaches: randomization, mimicry, and tunneling to avoid censorship over any application or application protocol. These approaches often use a fixed strategy, and incapable of changing their traffic obfuscation approach without system re-engineering or redeployment of code. Furthermore, unlike the previous obfuscating approach of making an application to avoid censorship, our objective is to make all sensitive application's traffic unidentifiable to the attacker. Therefore, SafeEnd requires more flexible traffic obfuscation approach. In order to do that, SafeEnd extends and leverages the OVS to apply programmable and flexible traffic shaping policies on the sensitive application flows.

In the second policy, SafeEnd applies end-to-end secure communication policy on sensitive application flows. As part of the end-to-end secure communication, SafeEnd uses IPSec tunneling protocol to accomplish end-to-end secure network communication that provides packet-level source authentication, data integrity, and confidentiality. Other security solutions in transport and application layers such as SSL/TLS and PGP have the following disadvantages suffers when compared to IPSec tunneling scheme: (1) Unlike IPSec, other security protocols require to have well-known and static service port numbers; (2) Unlike IPSec in which we can specify the security requirements, other security protocols don't support this flexibility; and finally (3) the security and performance of IPSec outperform significantly any other security mechanism [196].

Although IPSec combined with Internet Key Exchange (IKE) [69] protocols have been the standard part of the Linux-based OS for years, their usage has been limited to only Virtual Private Networks (VPN) [196]. We believe that the lack of a user-friendly APIs of IPSec and IKE protocols is one of the reasons. Therefore, SafeEnd leverages existing IPSec framework and IKE protocol and provides user-friendly APIs to enable ensure programmable and application transparent end-to-end IPSec tunneling for the sensitive application's network flows.

In addition, SafeEnd leverages the MOBIKE [91] addendum of IKE protocol to enable a seamless vertically handover of IPSec tunneling over different wireless interfaces, without breaking any active connection session.Since Wi-Fi channel is more prone to DoS attack scenarios compared to cellular, SafeEnd could seamlessly switch from the Wi-FI interface to the cellular interface in case of a DoS attack to guarantee end-to-end secured flows.

We implement a prototype of the proposed SafeEnd solution on android mobile devices, where we deploy and extend the OVS at both the user-space and the kernel-space. We implement both the traffic shaping policy and the end-to-end secure communication policy in the prototype solution. Although in our prototype we implemented randomize padding as traffic shaping policy [11, 46] to address App-Spoof attack, SafeEnd is flexible and could adapt any other traffic shaping policy. In addition, we integrate the SafeEnd with the IKEv2 daemon (i.e., `charon`) to provide programmable network policy of IPsec tunneling and MOBIKE protocol for vertical handover. We analyze the performance of SafeEnd on actual mobile devices to evaluate its energy consumption, throughput overhead, and CPU usage. Results show that SafeEnd has negligible performance overhead and minor impact to battery life while achieving desired flexibility of applying fine-grained network security policy on sensitive apps. Finally, we evaluate the performance of applying MOBIKE protocol for seamless vertical handover between Wi-Fi interface and cellular interface.

The rest of the chapter is organized as follow; section 6.2 describes the problem context and security threat model. In section 6.3, we describe the *App-spoof* attack and the privacy issues related to that attack. Section 6.4 describe the design objective of SafeEnd, and in section 6.5, we describe the architecture details of SafeEnd. While section 6.6, we describe the implementation details, and finally, we evaluate the performance of SafeEnd in section 6.7.

6.2 SECURITY THREAT MODEL

In this paper, primarily, we consider the following three security threat models for WiFi channels. First, the *DoS attack* by exhausting the wireless MAC or the wireless PHY layer resources. For example, attacker with the right tools can easily jam the 2.4 GHz frequency in a way to corrupt any ongoing traffic or to make the channel looks busy and not available for any traffic for any authorized user. Second, the *man-in-the-middle attack*, where any device on the communication path can modify, intercept or drop the traffic packets. For example, in ad-hoc wireless communication scenarios, any device with right software and on the route of an ongoing end-to-end communication between two smart devices could easily conduct a man-in-the-middle attack. Third, the *App-spoof attack* based on eavesdropping, where the adversary can track the user' application activities. In *App-spoof* attack, the intruder typically uses a Wi-Fi sniffing tool like wireshark [122] to passively capture and analyze encrypted wireless traffic from different smart devices to identify the running applications and when the user uses these applications. In the following section, we explain App-spoof attack in more details.

6.3 APP-SPOOF ATTACK

In this section, we describe the process of how an adversary could identify the running applications and their launching events from analyzing the encrypted/unencrypted Wi-Fi traffic.

6.3.1 FLOW DNA SEQUENCE AND APPLICATION GENOME

Different applications are rapidly introduced and deployed on smart devices in which each generates various flows. For example, once the Facebook android application starts, 20-30 flows are created within the first 10 seconds. We define a flow as an ordered sequence of IP packets with identical values of source and destination IP addresses, source and destination ports, protocol type. From experiments, we found that *sequences of traffic features* (e.g., sequence of frame sizes) of the beginning *packet sequences* of few of these flows are unique per application regardless of the user or the application contexts such as user's login information, physical location, device configuration, etc. We refer to a feature's sequence of an individual flow as the *application DNA sequence* while the total set of the features' sequences as the *application genome*. Although WiFi data encryption does not allow any adversary to observe the TCP/IP header information and hence can not identify the different flows that belong to an application, these unique packet sequences (application genome) are enough to be utilized as an application fingerprint and be used to detect and identify the application once it is launched. Moreover, these unique packet sequences would be able to identify the application even if they are mixed with background flows of other applications.

Knowing the application genome, an adversary with a very little computational effort can detect and track when and what different applications are launched and used by the user's smart device. For a successful detection of applications, the adversary needs to maintain a ground truth database with genomes of different applications. Note that, this database could be built offline by extracting the features of unencrypted packet sequences of different applications at their launching times. WiFi encryption (i.e., 802.11i WPA2) is based on CCMP [94] that encrypts only the data part of the WiFi frame and not WiFi header and adds extra field of 16 bytes to the unencrypted original WiFi data frame (8 bytes is appended to the data and 8 bytes of CCMP header is added to the WiFi frame header). Giving that these additional constant number of bytes is the main difference in sizes between encrypted and non-encrypted frames, it is easy for an adversary to still detect the application DNA sequences regardless of the WiFi encryption. With the application genome database, the adversary can match the database records with the passively extracted features from the WiFi encrypted frames. We found, as we will discuss later, that by applying local sequence matching technique [127] using only frame sizes, the adversary will be able to detect with very high accuracy the application once it is launched.

6.3.2 APPLICATION LAUNCHING DETECTION PROTOTYPE

In this section, we demonstrate the process of building the application genome database and then using it in detecting the application launching events for the most common applications of our campus network as shown in Table 3 and Table 4. In building the ground truth data for extracting application features, we capture unencrypted WiFi frames of each of the applications by running each of these applications on Android smartphones on an unencrypted WiFi network. After we filter out the WiFi management frames, TCP retransmissions, and TCP ACK frames from the unencrypted captured frames, we generated the *sequences of traffic features* for each application. To generate the testing data, we sniffed encrypted WiFi frames from seven different users using both Android and iPhone smartphones, which are instrumented to log all application activities. In collecting the encrypted testing data, users had the freedom to launch any number of the interesting set of applications at any time. In addition, users were allowed to run any number of other applications in the background while they are launching any new application.

We use Smith-Waterman Algorithm [127] to calculate the Maximum Local Sequences (MLS) between the frame size sequence of the ground truth data and the encrypted frame sizes from the testing data. Smith-Waterman Algorithm is a dynamic algorithm that takes two strings of different lengths and calculates the MLS between them. In our scheme, we use the sequence of the captured frame sizes as an input string to the algorithm. Thus, we have two strings; one corresponding to the ground truth data and the other one corresponding to the testing/encrypted data. Table 3 and Table 4 show the mean and the standard deviation (std) of the MLS between the ground truth data and the testing data (std values are shown between brackets). We calculate MLS values as percentages of the sequence length of the ground truth data. Assuming the sequence length of the ground truth data is X and the sequence length of the testing data is Y, we calculate the MLS percentage value as $(SW(X,Y)/length(X)) \times 100$, where $SW(X,Y)$ is the MLS outcome of Smith-Waterman Algorithm when applied over strings X and Y. In MLS calculations, we limit the maximum sequence length of the ground truth data to the first 100 frames.

Tables 3 and 4 show the MLS confusion matrix between the ground truth data (shown

	FB Messenger	**Youtube**	**Vimeo**	**Skype**	**Twitter**	**Google Plus**	**Google Maps**	**Hangout**	**Facebook**	**Gmail**	**Fring**
FB Messenger	**44.3(1.7)**	34.1(2.1)	23.3(2.6)	34.6(2.3)	**37.7(2.1)**	33.7(2.27)	32.1(3.14)	36.3(3.6)	**39(2.23)**	36.5(2.6)	37.3(3.7)
Youtube	32.7(2.7)	**51.3(1.3)**	21.1(2.4)	31.1(2.6)	31.7(2.7)	**37.8(2.4)**	28.4(5.4)	34.5(3.5)	30.3(1.4)	**38.3(4.3)**	30.5(2.5)
Vimeo	22.3(2.1)	22.1(1.4)	**46.3(4.5)**	22(2.4)	22.2(1.9)	19.3(2.3)	17.6(2.6)	20.6(3.2)	22(2.5)	33.0(4.5)	33.9(4.4)
Skype	33.6(2.4)	32.1(2.6)	24(3.4)	**40(4.6)**	32.2(1.9)	28(1.5)	24(5.4)	29.5(2.7)	34(3)	32(2.3)	34.7(1.6)
twitter	37.1(1.9)	32.7(1.7)	23.5(2.9)	32.7(2.9)	**43(2.4)**	29.7(2.7)	28.3(3.5)	33.3(4.7)	35.5(2.8)	34.8(2.6)	33.3(2)
Google plus	33.2(2.3)	36.7(2.5)	19.9(2.3)	28.3(2.5)	30.7(1.7)	**52.6(2.06)**	30.8(2.6)	34.5(2.2)	31.4(2.5)	36.4(3.4)	27.6(2)
Google maps	31.3(3.0)	28.8(4.4)	18.1(2.6)	25.1(3.4)	28.9(2.5)	31.8(3.6)	**53.8(8.2)**	**36.4(3)**	28.4(4.87)	35.9(1.4)	25.4(5.4)
Hangout	35.3(2.6)	34.1(2.5)	22.6(2.2)	30.5(2.7)	33.8(3.7)	33.5(2.3)	**37.5(3.1)**	**50.3(3.14)**	33.2(3.2)	35.4(3.4)	32.1(4.4)
Facebook	**39(2.23)**	31.3(1.2)	24(2.5)	34.2(2.3)	35.1(2.8)	33.4(2.5)	28.5(4.7)	32.7(3.1)	**41.2(1.1)**	33.8(2)	36.4(3.0)
Gmail	35.6(2.2)	**38.1(2.3)**	32.2(3.5)	32.6(2.5)	34.7(3.6)	35.4(2.4)	36.1(3.4)	33.8(2.4)	32.8(2.2)	**42(4.4)**	33.4(2.1)
Fring	37.1(2.7)	31.5(2.5)	33.3(4.1)	34.4(2.6)	33.3(2)	28.6(2.4)	25.9(4.4)	32.4(4.2)	34.4(2.6)	34.1(1.8)	**46.3(2.06)**
Dropbox	37.2(1.5)	32.3(2.1)	33(3.3)	35.6(3.3)	34.3(2.8)	27.7(2.7)	26.9(4.1)	31.5(2.6)	34.8(3.1)	36.1(1.4)	**38.0(2.4)**
Tango	31.4(2.2)	29.9(2.7)	34.1(2.5)	32.6(2.8)	30(3.4)	25.6(2.6)	23.8(4.1)	29.9(2.4)	32.7(3.8)	32.4(2.8)	30.1(2.3)
Instagram	37.5(2.9)	33.8(2.1)	**36.9(3.2)**	**39.4(2.1)**	36.6(2.4)	27.8(2.3)	28.7(3.8)	32.2(3.2)	36.1(2.9)	36.9(3.7)	36.2(1.8)
Pandora	33.3(3.6)	28.2(3.8)	32.6(2.7)	30.5(2.9)	32.7(3.8)	25.3(3.5)	23.8(3.5)	28.3(3.6)	32.4(3.4)	32(2.6)	32.7(2.5)

TABLE 3: Mean (std) MLS value between the ground truth data(left most column) and the testing data (top most row). Please see Table 3 for the remaining of columns of the table

	Dropbox	**Tango**	**Instagram**	**Pandora**
FB Messenger	38.2(2.5)	32.4(2.3)	37.2(3.9)	34.3(4.6)
Youtube	32.5(2)	29.6(1.7)	34(2.1)	28.8(2.8)
Vimeo	35(3)	32(3.5)	36.8(3.3)	32.8(3.7)
Skype	35.5(2.3)	31.6(1.8)	39.3(2.5)	31.5(2.9)
twitter	34.3(2.8)	30(3.4)	36.6(2.4)	32.7(3.8))
Google plus	28.7(2.07)	26.6(1.6)	27.8(2.3)	24.8(2.5)
Google maps	26.7(6.1)	23.2(4.5)	26.8(6.7)	24(4.6)
Hangout	32.5(3.06)	29.5(3.1)	31.6(4.9)	28(4.9)
Facebook	35.6(2.7)	30.7(2.8)	37.2(3.9)	34.4(3.4)
Gmail	35.4(1.4)	31.3(2.1)	37.5(4.3)	30(3.6)
Fring	38.1(1.4)	31.5(2.3)	36.7(2.8)	32.3(3.5)
Dropbox	**46.6(1.8)**	32.1(2.5)	**40.4(2.2)**	33.4(3.6)
Tango	33.1(2.1)	**39(1.2)**	32.7(3.3)	29.7(3.1)
Instagram	**41.2(2.2)**	**32.9(2.3)**	**56.3(6.0)**	**34.7(4.6)**
Pandora	34.4(2.6)	30.1(2.7)	33.6(4.1)	**45.6(6.5)**

TABLE 4: Mean (std) MLS value between the ground truth data(left most column) and the testing data (top most row).

in the left most column) and the testing data (shown as the top row) for the interested set of applications. Note that, diagonal elements in Tables 3 and 4 happen to have the highest mean MLS values (shown in red) in comparison to the other elements of the corresponding columns. In another word, Smith-Waterman algorithm, on average, gives the highest MLS when the testing data of a specific application are tested agains the ground truth data of the same application. These results show that an adversary, with very high accuracy, can detect the application once it is launched by selecting the application from it genome application database that is corresponding to the highest MLS value. Tables 3 and 4 also show that the second highest MLS value (shown in blue) of each column is corresponding to applications from the same vendor. For example, YouTube application has second highest MLS value for GMail application testing data, where both of YouTube and GMail are products of the same vendor (i.e., Google). We see similar behaviors for Hangout, Google Plus, and Google Map applications. Similarly, Facebook Messenger application and Facebook application show the second highest MLS value when compared to others.

6.3.3 APP-SPOOF THREAT MODEL

The leaked information from app-spoof attack would lead to inferring many other information about the user identity. For example, studies show that by analyzing application usage information, we could infer demographic information of the user like gender and age [148]. In addition, an adversary with a cheap hardware and free software tools such as

WireShark could capture passively the WiFi traffic of house members over a long period such as a week. By analyzing the traffic and the corresponding application usage activities such as what application start when (as we will discuss later), and giving that different individuals have different application interests [183], the adversary will be able to correlate the detected application activities to different members of the house (e.g., husband, wife, kid). In addition, the adversary could infer the occupancy periods of each member over the week and when the house is empty. This example demonstrates that tracking online activities and application usages of individuals are not just a privacy issue, but also could be a serious threat to the individual safety. Moreover, nowadays, individuals use more personalized applications such as health applications on their smart devices. Consequently, by knowing the usage of specific apps, we can infer many personal health related information, which is a very serious privacy threat to individuals.

6.4 SAFEEND DESIGN OBJECTIVES

In this section, we discuss several design objectives of SafeEnd. In SafeEnd, one of the key design objectives is to providing fine-grained, application-aware, and programmable network security policies on the application's network flow.

Supporting fine-grained application-aware network security policies. Studies show that data sensitivity of an application varies from a user to another [53]. Therefore, it is implausible to tag an application as sensitive (or not sensitive) for all users. Therefore, in designing SafeEnd, we focus on enabling individual users to be in charge of setting the application's network security policy. Although users could tag all their applications as sensitive ones, users are encouraged to select only the real sensitive applications since applying security policies on application's network flows imposes an additional overhead as well as degradation in overall network performance [166]. SafeEnd leverages the OVS in the network stack of mobile devices to virtualize the WiFi networks to support both sensitive applications and non-sensitive applications. This virtualization permits the application of fine-grained network security policies on the sensitive application's network flows only in order to minimize overhead and enhance network efficiency.

Choosing the right type of traffic obfuscating technique to address the App-Spoof attack. In SafeEnd, we apply a randomize traffic shaping policy to obfuscate the network flows of the sensitive apps to prevent the eavesdropping attack (e.g., App-spoof attack). The basic idea of the randomize traffic shaping techniques or policy is to randomize the packet sizes or the packet inter-arrival timestamps [178, 180]. The randomize traffic shaping techniques have less overhead and are faster that make it more suitable for mobile devices when compared to other traffic obfuscation or shaping techniques including mimicry [107, 176] and tunneling [21, 75, 97, 102, 172, 201]. However, unlike the main objective of previous traffic obfuscating approaches in bypassing any application/protocol censorship policy, SafeEnd objective is to prevent any eavesdropping attacker from recognizing and tracking user's sensitive applications. In doing so, we leverage the randomize traffic shaping approach in SafeEnd to jeopardize the active flow/application genome pattern. Thus, we confuse the eavesdropping attackers from detecting the active flows/applications.

Applying traffic obfuscating techniques in a seamless and transparent way to applications. Many of the previous traffic obfuscation techniques are application-based end-to-end solutions that require redesigning both the client side and the server side of each application in order to be able to reshape packet flows. However, such application-based end-to-end solution requires great effort for redesigning each application at both client side and server side. In SafeEnd, we address this challenge by leveraging the OVS in the network stack to apply randomize traffic shaping policy on the sensitive application's network flows. Thus, SafeEnd ensures end-to-end traffic shaping solution that is transparent to applications, and consequently, doesn't require any application modification.

Providing programmable and user-friendly end-to-end secure communication policy. Besides traffic shaping policy, SafeEnd also leverages the existing IPsec framework (i.e. `XFRM` framework) [90] and IKE protocol [69, 103] to guarantee an end-to-end secure communication policy for the sensitive application's network flows. Unfortunately, in Linux OS, the existing implementation of the IPsec framework and IKE protocol don't have any user-friendly APIs that would enable developers to utilize and adapt these protocols. SafeEnd, on the contrary, supports user-friendly APIs that facilitate the use of an IPsec tunneling based end-to-end secure communication policy. SafeEnd, at the back, is responsible for handling the underlying interactions with the IPSec framework and IKE daemon (e.g., racoon [101], StrongSwan [158]) to set up the policy.

Converting application-aware security policies to flow-level policies. In order to apply application specific network security policies both in OVS and IPSec framework, SafeEnd should be able to transform application-level policies to flow-level policies. Therefore, SafeEnd needs an efficient mechanism to map active network flows to their corresponding applications. Furthermore, SafeEnd has to be able to apply flow-level policies even before capturing any of the flow packets. In order to do that, SafeEnd needs an efficient socket connection listener to be able to extract the details about the socket connection (i.e. port number, protocol, IP address) once an application initiates a socket connection.

Instead, if using the existing Linux-based tool `netstat` that is not efficient enough to provide the network socket information, SafeEnd deploys and leverages another Linux based tool `ss` on Android OS in order to have an efficient socket connection listener to extract information about any newly created socket connection.

Applying the traffic shaping policy before the IPsec tunneling policy on a network flow. In order to apply randomize traffic shaping techniques, SafeEnd extends and leverages the OVS to manipulate IP packets of sensitive application's flows by either pad packets with additional bytes, split packets into multiple sub-packets, or to modify packet transmission times. Since the receiving side of flow needs reversed the traffic shaping at the destination to retrieve original packets, we utilize the unused IP option header of IP packets to carry out the needed information about the randomize traffic shaping technique that enables to reserve the traffic shaping process. Unfortunately, many intermediate network switches/routers are configured to drop any IP packet with unknown IP option header [51]. In order to address this end-to-end unreliability issue, SafeEnd needs to apply IPSec tunneling after the randomize traffic shaping to be able to hide the IP option header. Typically, OVS operates below the IP/IPsec layer in which IPsec tunneling is applied before the traffic shaping process. In SafeEnd, we configure both the routing table and the OVS in the kernel space to create an additional loop in the packet traversing path in the kernel network stack between OVS and IP network layer. Thus, SafeEnd ensures that IP packets traverse through OVS for traffic shaping before the application of IPsec tunneling.

Making SafeEnd extendable. SafeEnd is self-sufficient to allow users to apply network security policies for mobile applications. However, SafeEnd can be extended to provide APIs that allows third-party control programs to apply any additional network security policies on mobile devices. This third-party control program could reside either locally on wireless APs or remotely in the cloud. Providing such APIs should be a straightforward client-server interaction between the third-party control application and SafeEnd service.

6.5 SAFEEND ARCHITECTURE

Figure 38 shows the overall architecture of SafeEnd solution, which is a combination of both kernel and user space components. In SafeEnd, we extend both OVS kernel module (i.e., datapath) and OVS user space component (i.e., OVS client `vswitchd`). OVS kernel module (i.e., datapath) is responsible for applying QoS and access control policies on network flows based on Layers 2-4 packet header fields. On the other hand, OVS client is responsible for interacting with the SafeEnd controller to setup network policy rules in the

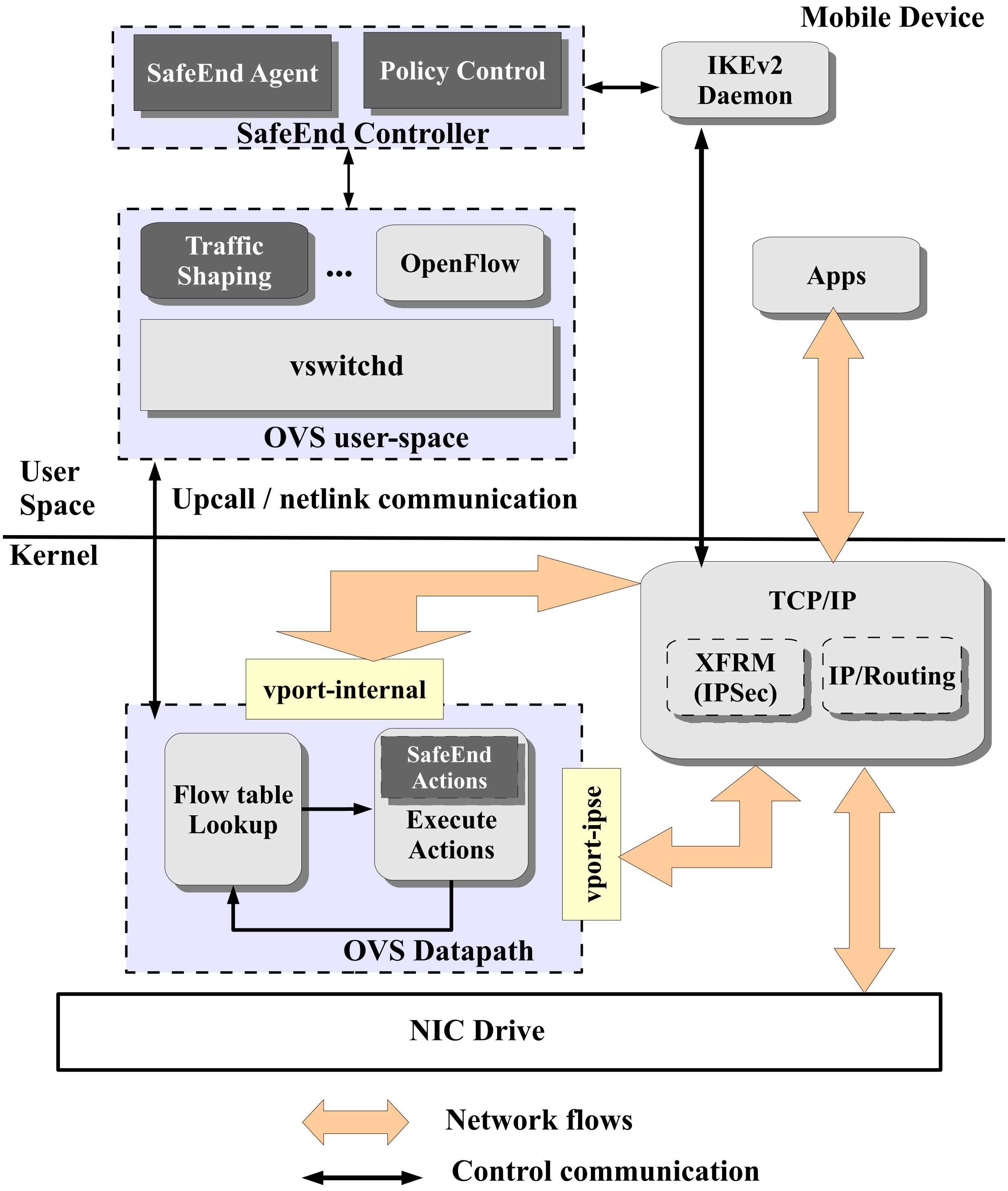


FIG. 38: Architecture of SafeEnd solution.

datapath [130].

In SafeEnd, we have three main components. The first component is the *SafeEnd* controller, which is responsible for setting up application-aware traffic shaping network security policies in OVS client. In addition, it interacts with the IKE daemon (i.e., charon [158], an open source IKE daemon) to provide application-aware IPsec policies. Note that,*SafeEnd controller* converts the application-aware network security policies to the flow-level policies and then appliess IPSec tunneling policy on those flows using IKE daemon. Finally, *SafeEnd controller* is responsible for configuring the routing-table and the OVS datapath to orchestrate the traversing path of a packet within kernel network stack. The second component of SafeEnd is the OVS client, which is responsible for applying traffic shipping policies on the network flows. This component also supports flexible configuration scheme to apply different traffic shipping policies for different network flows. The final component of SafeEnd is within the kernel space that extends the OVS datapath to introduce new actions for applying randomize traffic technique on sensitive application packets. In addition, it introduces new actions to be able to reverse the randomize traffic shaping technique to retrieve the original packets at the receiver side. This components also changes the IP header, to steer the packet, to ensure that traffic shaping get applied before the IPsec tunneling. In the following subsections, we describe different components of SafeEnd in more details.

6.5.1 SAFEEND CONTROLLER

SafeEnd controller consists of two modules: *SafeEnd agent* and *policy control*. The *SafeEnd agent* module provides a GUI that allows users to select, from the list of installed application packages on their mobile devices, a subset of these applications as sensitive applications in order to secure all their network communication. The *SafeEnd agent* component also generates "flow-to-application mappings" by monitoring active network sockets and their `pid/uid` bindings that are available on Android OS and major Linux-based operating systems. Since this component runs on the end device, it can map active sockets/flows to their corresponding applications and, consequently, transfers the corresponding flow information (i.e., source IP address, source port, destination IP address, destination port, and protocol) of all flows generated by sensitive applications to the *policy control* component. Note that, in the remaining of this chapter we use the term "secure channel" flow to refer to any flow generated by a sensitive application.

The *policy control* module is responsible for setting up the flow-level policies for the network flows that pass through OVS. More specifically, *policy control* uses OpenFlow protocol [104] to add the corresponding policy rules to OVS. In order to support randomize traffic shaping policy rules, we introduce new actions to OVS. Furthermore, we extend both OpenFlow protocol and OVS kernel module (i.e, datapath) to randomize traffic shaping policy rules for "secure channel" flows. In the following subsection (Subsection 6.5.2), we discuss more details about the new actions for randomize traffic shaping policy.

In addition, *policy control* communicates with *traffic shaping* module that resides in OVS user-space (i.e., `vswitchd`) to specifically define the randomize traffic shaping technique to use and the corresponding configuration parameters for the "secure channel" flow. In Subsection 6.5.2, we describe more details about the *traffic shaping* module. In existing SafeEnd solution, we support two types of randomize traffic shaping techniques; *padding* and *splitting*. The *padding* technique adds a random number of bytes to the end of IP packets. The *splitting* technique, on the other hand, fragments an IP packet at a random position to create two IP Packets. Note that, in addition to these two techniques, SafeEnd is extensible, and could support any other traffic shaping technique.

The *policy control* module also interacts with the *IKEv2* daemon (like racoon or charon in StrongSwan etc.) to initiate the negotiation process for the "secure channel" flows to set up their IPsec configuration parameters, which includes symmetric key (or more then one key), authentication (i.e., MD5, SHA etc.), encryption (i.e., 3DES, AES, ect.), data integrity and key exchange algorithm (i.e. Diffie-Hellman). After the negotiation process, the IKEv2 daemon ultimately interacts with the `XFRM` framework [68] to setup the IPSec parameters of the "secure channel" flow in the Security Policy Database (SPD) and the Security Association Database (SAD). Note that, SafeEnd uses the same IPSec policy parameters for all the flows that belong to the same application.

Given mobile devices are typically equipped with both Wi-Fi and cellular data connectivity, each device has two different IP addresses. Also, given Wi-Fi networks are more prone to DoS attack than the cellular networks, mobile devices should be able to vertically handover the network traffic from Wi-Fi to cellular whenever a DoS attack is detected. In doing this, SafeEnd leverages the MOBIKE protocol [91] in the IKEv2 daemon to be able to change the IP address associated with the IKE and IPSec SAs (Security Association) of the "secure channel" flows from the Wi-Fi interface to the cellular interface without the need for disconnecting the existing SAs and establishing new ones. Thus, MOBIKE provides a seamless vertical handover for the "secure channel" flows (figure 39).

6.5.2 SAFEEND IN OVS USER-SPACE

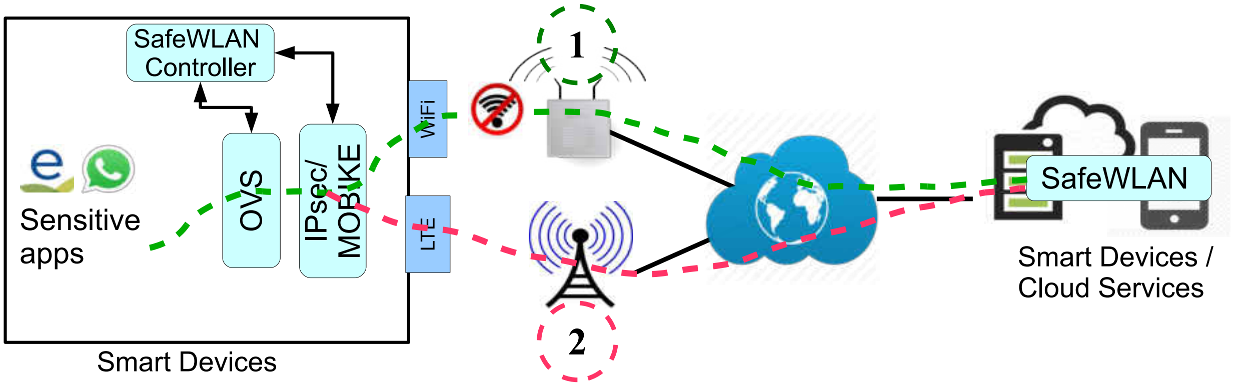


FIG. 39: Vertical handover of the "secure channel" flow from Wi-Fi to cellular interface in SafeEnd.

In this component, we build a new module *traffic shaping* in OVS user-space or OVS client (i.e., `vswitchd`). The responsibility of the *traffic shaping* module is to keep records of the randomize traffic shaping policy to apply for each "secure channel" flows. Furthermore, it provides the flexibility to configure the randomize traffic shaping policy for each "secure channel" flows. In order to keep the record, *traffic shaping* maintains a table in which each entry consists of the "secure channel" flow information (i.e., 5-tuple flow identifier information), the applied traffic shaping technique (e.g., padding and splitting), and the corresponding parameters (e.g., min-max range, probability distribution). *Traffic shaping* module provides a set of APIs that is utilized by *policy control* to create an entry for each "secure channel" flow. In order to use a randomize traffic shaping technique, it is required to pick up a random number for each table entry. For generating random numbers, we define two parameters for each entry; the minimum/maximum range of integer number, and the discrete probability distribution function to be used on this range. Note that, typically *traffic shaping* module use default values for the randomize traffic shaping parameters. However, the user has the flexibility to modify these parameters based on his previous experience and evaluations.

Traffic shaping module also runs a user-space thread that process the "upcall" action, corresponding to the "adaptive sampling" action which we will describe in details in Section 6.5.3. Once receiving the "upcal" action for a packet, the thread extracts the 5-tuple flow information from the packet header, and search in the table to find a matching entry. After finding the entry, the thread picks up a random number according to the defined configuration parameters (e.g., min-max range, probability distribution) of the entry. The random number could be used as the amount of padding bytes or as the byte position index to split the packet at, based on whatever randomize traffic shaping technique is used in the matching entry.

Note that, in each entry, *policy control* uses the IP address of the `vport-internal` port to define the “secure channel” flow in the *traffic shaping* module. Thus, it makes sure that the traffic shaping technique gets applied on the packets of the “secure channel” flow. An example of an entry in the table of *traffic shaping* module is as follows:

srcIP=’A’,srcPort=’a’,dstIP=’B’,dstPort=’b’,
type=*padding,* min=packet_size,max=MTU,P(x)=Poisson(λ)

where the traffic shaping technique is padding and the range is between the current packet size and the `MTU`. Finally, the discrete probability distribution function, $P(x)$ is *poisson* with parameter λ. Based on these parameters, *traffic shaping* module uses the algorithm 4 to generate a random integer number. In order to do that, *traffic shaping* module examines at the egress packet header to match an entry from the table. If an entry is found, it calls 4 algorithm with the parameters from the matching entry. In case, if no matching entry is found, then no traffic shaping technique is applied.

Algorithm 4 Pseudocode for generating a random integer number for padding.

```
procedure DISCRETE_RANDOM(min,max,P(x))
    generate the CDF function F(x) from P(x)
    min_a ← F(min)
    max_b ← F(Max)
    k ← min_a + rand(max_b-min_a)    ▷ generate uniform random number between min_a,max_b
    return $F^{-1}(k)$    ▷ return the inverse value of the cdf function, $F(x) = k$
end procedure
```

6.5.3 SAFEEND IN KERNEL-SPACE

In SafeEnd, every packet passes through OVS datapath, where the randomize traffic shaping takes place only on the packets that belong to the “secure channel” flows and have a corresponding entry in the *traffic shaing* module. In order to apply the randomize traffic shaping, we create three actions “adaptive sampling”, “shaping” and “reverse shaping” in OVS datapath. Among them, OVS uses both “adaptive sampling” and “shaping” action to apply traffic shaping to the egress packet of the “secure channel” flows. Initially, “adaptive sampling” action samples a packet of a “secure channel” flow with the certain probability, and then sends it to *traffic shaping* module in OVS user-space through *upcall* operation.

Once *traffic shaping* module decides on the random number and the type randomize traffic shaping to apply, it sends this information to the "shaping" action in the datapath. The responsibility of the "shaping" action is to act according to on the input from *traffic shaping* module. If the randomize traffic shaping technique is *padding*, then it pads the selected random number of bytes at the end of the IP packet and add IP option header to convey the information of padding with the packet. Otherwise, if the randomize traffic shaping technique is *splitting*, then it splits the IP packet at selected position and create two consecutive IP packet with IP option header to convey the information of splitting.

The "adaptive sampling" action also allows changing the probability of sampling over time in a flow. For example, in 'App-spoof" attack, the sizes of the early beginning packets of an application's flow shows more noticeable signature pattern than the later ones. Therefore, in "adaptive sampling" action, picking the beginning packets of a flow with higher probability optimize the overhead of randomizing traffic shaping approach to obfuscating the "app-spoof" attack.

Note that, In randomize traffic shaping approach, not necessarily every packet of the "secure channel" flow gets picked up. Therefore, we use the unused reserve bit 6-7 of the ToS field of the IPv4 header to tag the picked up packets for the randomize traffic shaping technique. In order to decide on the type of randomize traffic shaping technique to apply on the picked up packet, we use the variable option field in the IP header. We use the option class 1, which is currently reserved for future use. We also define different option numbers for representing different traffic shaping techniques. In addition to option number, option length and option value in the IP header contain the parameters of the traffic shaping technique (i.e., padding, splitting). On the receiving end, datapath checks every ingress packet header, especially IP option header, to identify whether this packet was picked up for traffic shaping. In case a packet was picketed up for traffic shaping, datapath applies "reverse shaping" action that uses the IP option header information to recover the original IP packet.

Figure 38 shows that the OVS datapath has two `vports`, one is the internal `vport` (i.e., `vport-internal`) and the other one is IPsec `vport` (i.e., `sport-ipsec`). The existing OVS source code provides several tunneling ports, like *vport-gre*, *vport-vxlan*, which are used for IPsec tunneling protocol. However, such tunneling ports have the extra overhead of GRE protocol or VLAN protocol header in addition to the IPsec protocol header. Therefore, in order to avoid these additional overhead, we introduce a new class of `vport` in the OVS source code, called `vport-ipsec` that provides IPSec tunneling protocol without the overhead of GRE and VLAN tunneling. Note that, in SafeEnd, we apply traffic shaping before applying IPSec tunneling protocol. In that way, IP option header remains inside the IPSec tunneling header and, hence, avoid any disturbance to legacy network routers (many routers may drop packets having the IP header option field for security purposes).

SafeEnd binds both the internal and the IPsec `vport` with IP addresses. Among them, `vport-ipsec` is bind to the IP address of a physical interface (i.e., Wi-Fi, LTE). For example, If the device is using WiFi for the internet connection, then `vport-ipsec` is bind to the WiFi IP address. Once the device starts using the cellular interface for internet connectivity, it is the responsibility of the *SafeEnd controller* to bind the `vport-ipsec` to the cellular IP address. On the other hand, `vport-internal` is bound to an IP address, which is setup by the SafeEnd to use it as the default source IP address for all the running applications in the mobile device. Thus, SafeEnd makes sure that all application's network flows pass through the OVS datapath using `vport-internal` port. Note that SafeEnd uses the IP address of the `vport-internal` port as the source address to setup the flow table rules in OVS.

Every egress packet that passes through the OVS datapath gets forwarded to the `vport-ipsec`. Before, sending it to the `vport-ipsec` port, OVS datapath changes the source IP address of every egress packet to the IP address of the `vport-ipsec` port. The egress packet that passes through the port, *vport-ipsec*, get forwarded to `XFRM` framework, which basically responsible for handling IPsec protocol. The `XFRM` framework have the IPsec configuration parameters, in SAD and SPD for the "secure channel" flows. The IPsec protocol only applies to the egress packet that belongs to the "secure channel" flows. Note that, SafeEnd *policy controller* uses the IP address of the `vport-ipsec` as the source IP address for defining all the "secure channel" flow in the IKEv2 daemon, and eventually in `XFRM` framework. Thus, we make sure that `XFRM` framework applies IPsec protocol on the egress packet of the "secure channel" flow after it leaves the OVS `vport-ipsec` port.

Figure 40 shows the pipeline of processing egress packets in the kernel space by SafeEnd. Note that, SafeEnd kernel components change the IP header two times when handling egress packets. Initially, before entering OVS datapath, every egress packet have the IP header with source IP address A (i.e., IP_A), which is the IP address of the `vport-internal` port. When an egress packet is selected for traffic shaping inside the OVS datapath, the traffic shaping action adds the IP header option field in the packet header (i.e., IP'_A). Then, before every egress packet get forwarded to the `vport-ipsec` port, OVS changes the source IP address to B (i.e., IP'_B), which is the IP address of the `vport-ipsec` port. Finally, XFRM framework apply IPSec tunneling on the egress packet.

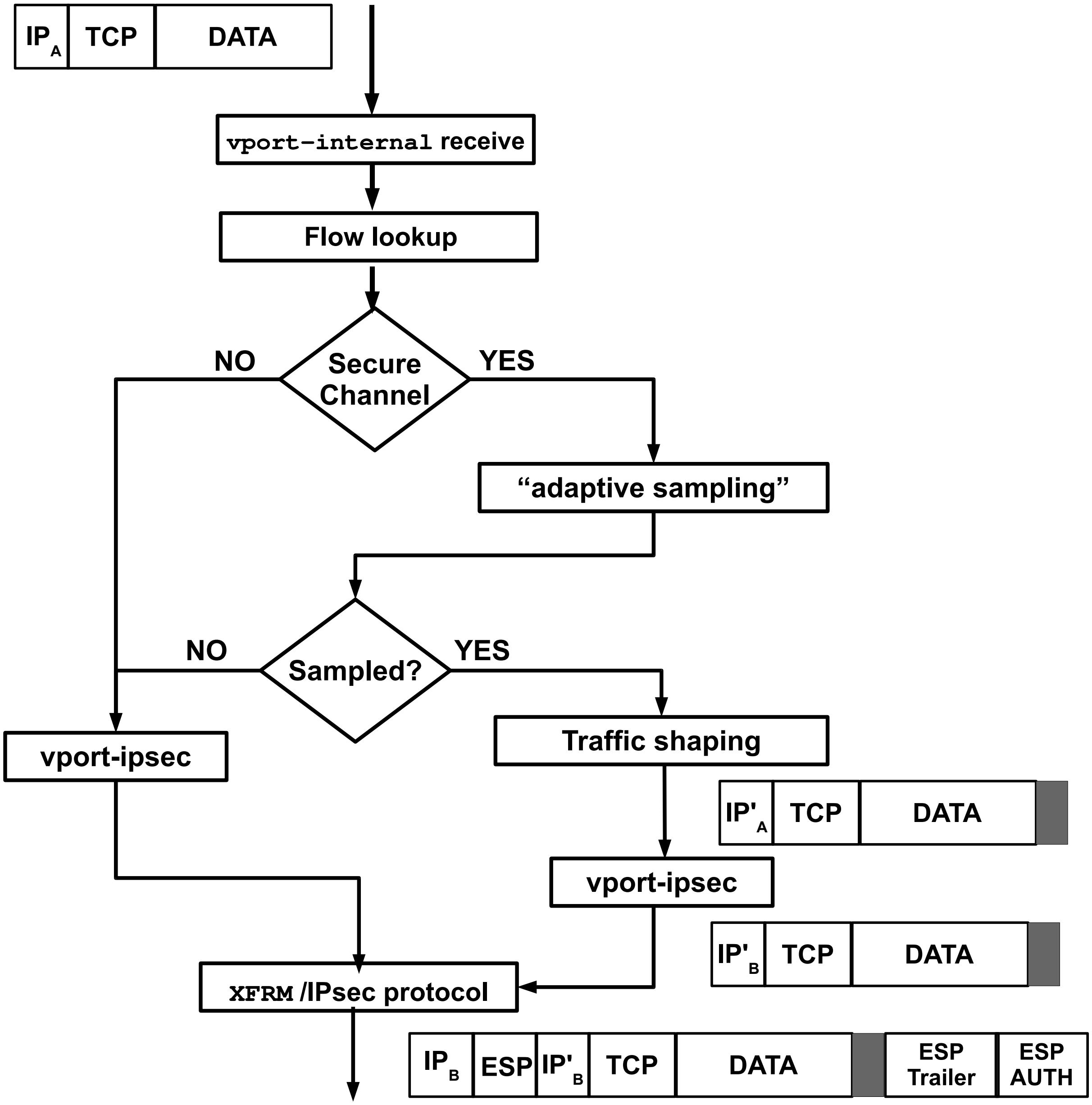


FIG. 40: Pipeline of processing egress packet in kernel space by SafeEnd solution.
]

6.6 IMPLEMENTATION

We implement *SafeEnd agent* on Android OS. In *SafeEnd agent*, we use the standard Android UI APIs and application package APIs(`android.app`) to show the list of installed applications on the mobile device. The UI in *SafeEnd agent* allows the users to setup per-application security policies. In addition, *SafeEnd agent* uses the `ss` tool to collect the App-flow mapping information. Based on the App-flow mapping information, *policy control* coverts application-specific policies to flow-level policies, and add corresponding flow rules to OVS using `OpenFlow` protocol [104] for applying the security policies. In addition, *policy control* interacts with IKEv2 daemon to setup up IPsec policies using App-flow mapping information. In SafeEnd, we extend IKEv2 daemon, `charon` to provide a set of APIs to interact with `XFRM` framework in the kernel to update the SPD and the SAD. In SafeEnd, we leverage the standard implementation of the IPsec in the kernel space.

We use `charon` tool to run IKE protocol, MOBIKE protocol, and negotiate symmetric keys between two peers. Note that, during the negotiation, the device that initiates the IKE/MOBIKE protocol is called the *initiator*, and the other end is called the *responder*. In MOBIKE protocol, `charon` negotiates with the peer device, which can be either a server or a mobile device with SafeEnd. During the negotiation, `charon` reports two IP addresses of both the Wi-Fi interface and the cellular interface. Thus, `charon` makes sure that the other end device is aware of the possible different IP address of ingress packets during handover scenario. The following are the steps between the *initiator* and the *responder* during handover scenario;

- The *initiator* bind the `vport-ipsec` port to the IP address of cellular interface.
- The *initiator* update the IKE SA with the IP address of cellular interface using internal APIs that provide access to the SAD and SPD in `XFRM` framework.
- The *initiator* start the *MOBIKE* protocol to notify the *responder*.
- The *responder* update IKE SA with the new IP address from the *initiator*.
- The *responder* acknowledge the update of the new IP address to the *initiator*.
- Both the *initiator* and *responder* update the IPSec SAs stored in SADs with new IP address.

In *traffic shaping* module at OVS user-space, we create a hash-table that keeps the record of the randomize traffic shaping information per-flow. The hash-table uses the 5-tuple flow information as a hash value of the record. In our prototype implementation, we only use the randomize *padding* as the traffic shaping technique to apply on the "secure channel" flow. In the prototype, we extend OVS kernel module and datapath to support three actions "adaptive sample", "shaping", and "reverse shaping". In order to add flow rules with corresponding new actions in OVS, we also have extended `OpenFlow` protocol. In "shaping" and "reverse shaping" action, we only have implemented the randomize *padding* technique, which adds a random number of bytes at the end of the packets. We set the 6th bits of `TOS` field in the IP header struct `iphdr` to indicate the polluted packets.

In addition, we extend the IP header to add an option field to include the corresponding of the traffic shaping parameters. In order to define different traffic shaping technique, we use two fields in the IP option field; `option class` and `option type`. Typically this two fields of the IP option field are used together to represent different protocols. The value of `option class` represent different class of protocols. Among those class, we have different values for `option type` to represent different protocols. Note that, `option class` is a 2 bit field. The value of 0 and 2 is already occupied to represent a different class of protocols, and value 3 is illegal to use. In that case, we use the unused value of 1 for the `option class` field to represent traffic shaping class. However, we might have multiple traffic shaping technique like padding, splitting etc. In order to represent different traffic shaping technique, we use the `option type`, which is a 5-bit field. For instance, in our implementation, we use the value 4 for the `option type` filed, in addition with value 1 for `option class` field to represent the padding technique. In addition, we set *option length* field to "2" as the length of our `option data` field. We use `skb_put` and `skb_trim` functions on `skb_buff` in order to add or remove the padding bytes respectively.

6.7 EVALUATION

6.7.1 APP-SPOOF ATTACK PREVENTION

In this section, we evaluate the impact of randomizing traffic shaping technique of *padding* on preventing *App-spoof* attack for different mobile apps. In addition, we also evaluate the impact of network overhead for using the *padding* technique. The *padding* technique we used for our evaluation is the following; First, we set the "adaptive sampling" action to sample a packet, if it is below 800 bytes in size. Then, we pick-up a value between

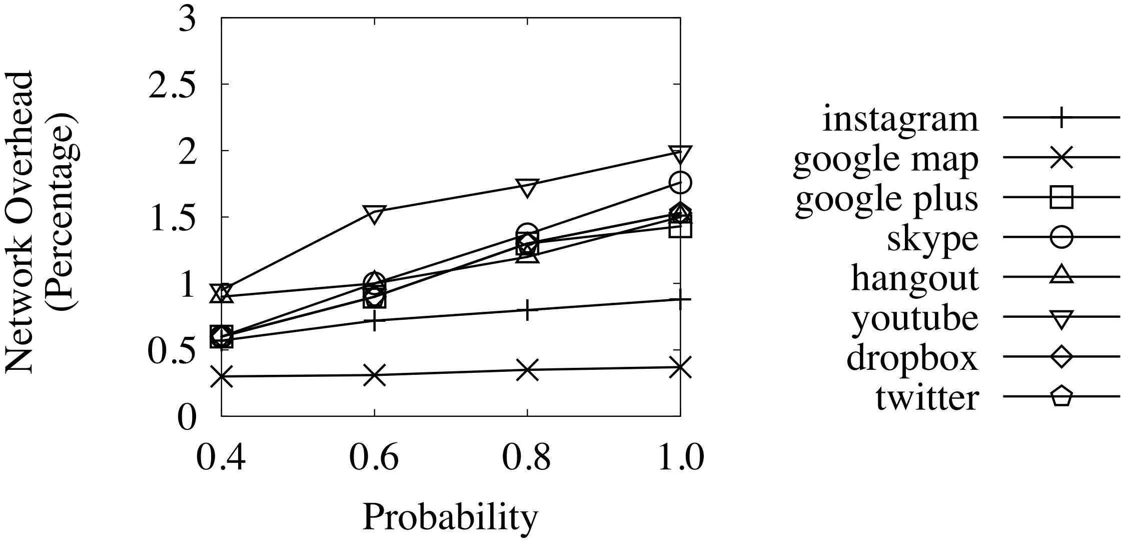


FIG. 41: The relation between network overhead and the probability (p) of selecting packets for *padding* technique for different mobile apps. We use normal distribution for picking up the random number for padding.

0 to 1 uniformly, if the value is below a certain threshold p then we select the packet for padding. We called the p as the probability in the plots. We use different value of p in the plots (i.e. 0.4, 0.6, 0.8, 1.0). Finally, for selecting the number of bytes to pad, we pick a discrete random number between 20 to 100. Between this range, we try two different probability distribution function, normal distribution and uniform distribution, for picking up the random number to pad at the end of the packet.

Figure 41 shows the network overhead of additional bytes added in percentage compare to the original bytes transferred for different mobile apps. Both figures show how the network overhead increases with the increase of probability (p) for different mobile apps. Note that, in Figure 41, we use a normal distribution for picking up the random of bytes to pad at the end of the packet. Besides the network overhead, Figure 42 shows the degradation of accuracy for *app-spoof* attack for the increase of p.

Note that the *google map* app shows no changes of network overhead as well as no degradation of accuracy with the increase of p, which means most of the unique sequence of frame sizes for the *google map* are close to MTU. This shows a good example that not all app can be hidden using the padding scheme. On the other hand, *instagram* shows flat network overhead but the accuracy degrades with the increase of p. It means, there are a lot of large frames in launching *instagram* app, but the frames that are important in identifying

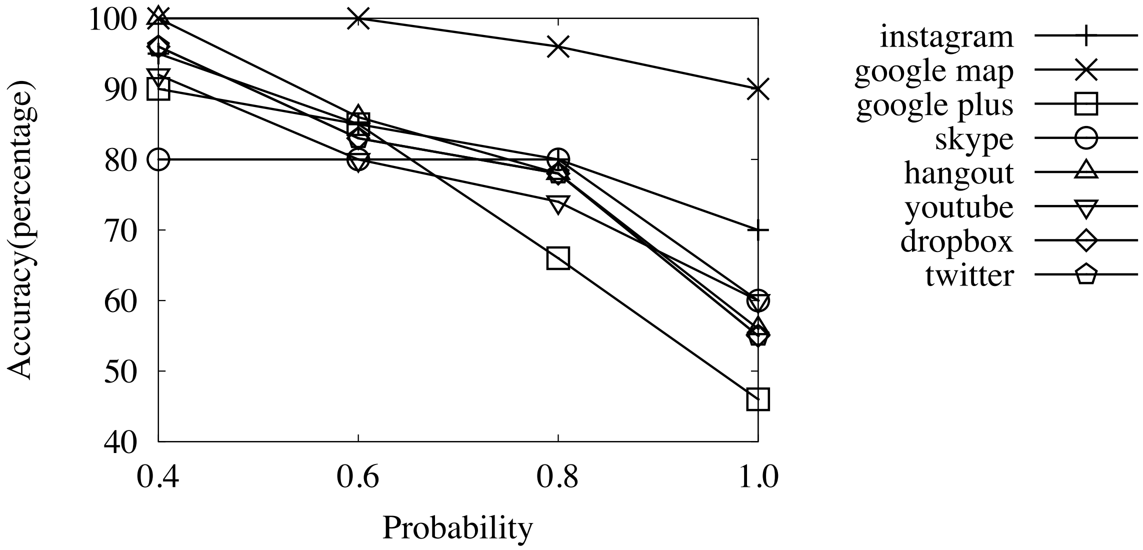


FIG. 42: The relation between the accuracy of *app-spoof* attack for identifying different mobile apps, and the probability (p) of selecting packets for *padding* technique. We use normal distribution for picking up the random number for padding.

the app are below 1500 bytes. Among the apps, *youtube* shows a large increment of network overhead, that means most of the packets in *youtube* launching are less in size. We found that applying padding on a small number of packets (small p value) in *skype* degrade its detection accuracy instantly to 80%. After that; accuracy remains almost constant even if we increase the value of p until we select to pad to every packet.

6.7.2 PERFORMANCE EVALUATION

The energy efficiency evaluation is more relevant in the context of the mobile device. Because in mobile devices energy is an issue of scarcity. In order to evaluate the energy efficiency, we measure the increase in energy usage (in percentage) while running the SafeEnd in the mobile device. Figure 43 show the cdf plot of energy usage for SafeEnd. The plot shows that there is 1.8% increase of energy usage on average. Among the applications, the social app has the highest energy consumption, where video streaming has the lowest. Because social app produces large number flows, where video streaming app produces a small number of flows. Furthermore, in the idle scenario, SafeEnd components in the mobile device shows 0.2% energy usage overhead on average. Despite the increase of energy usage, SafeEnd provides us for applying network security policy on sensitive application's flow. Figure 44, shows the relative throughput changes of running the SafeEnd in the mobile

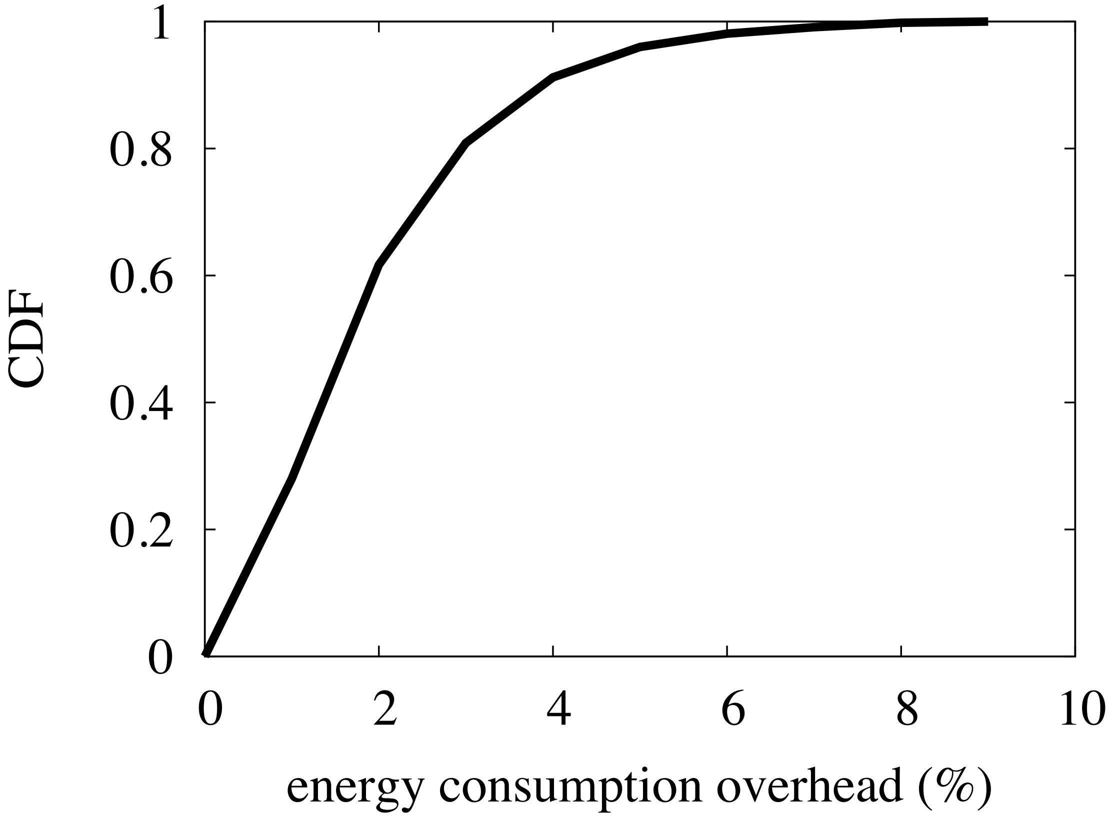


FIG. 43: Energy consumption overhead (in percentage) of running the SafeEnd modules in the Nexus 4 Android mobile device.

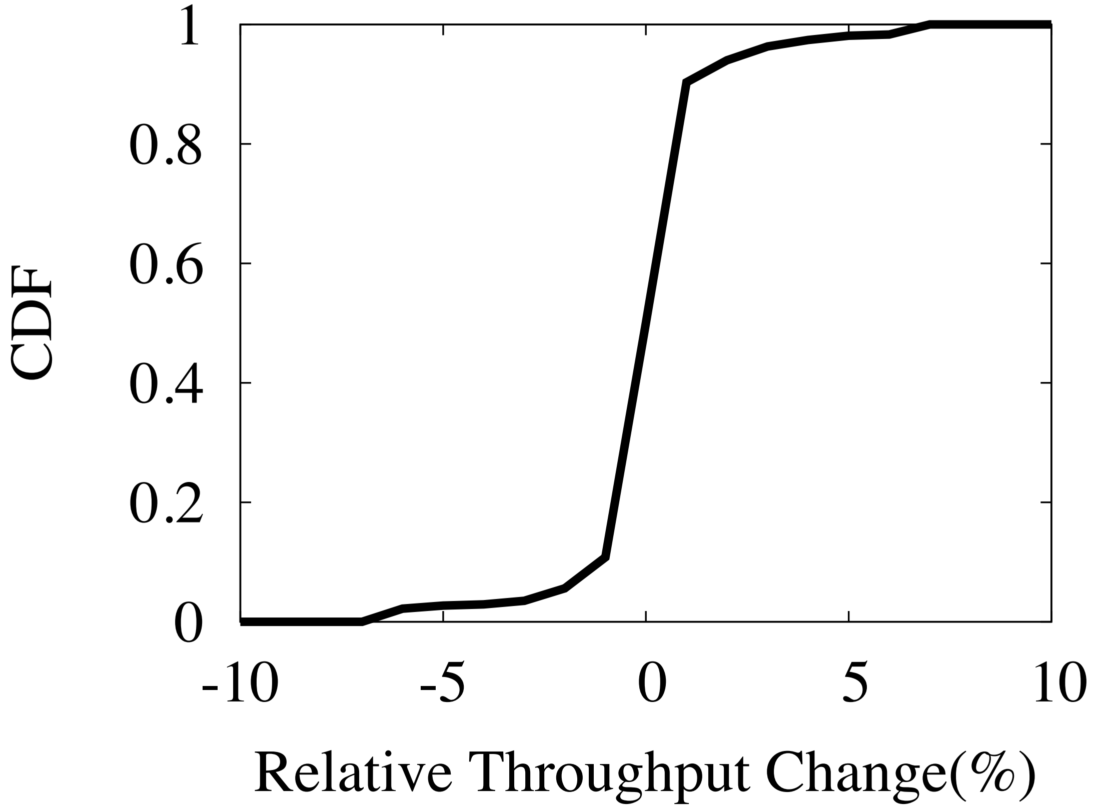


FIG. 44: Overall relative throughput changes, while using SafeEnd for the Dataset.

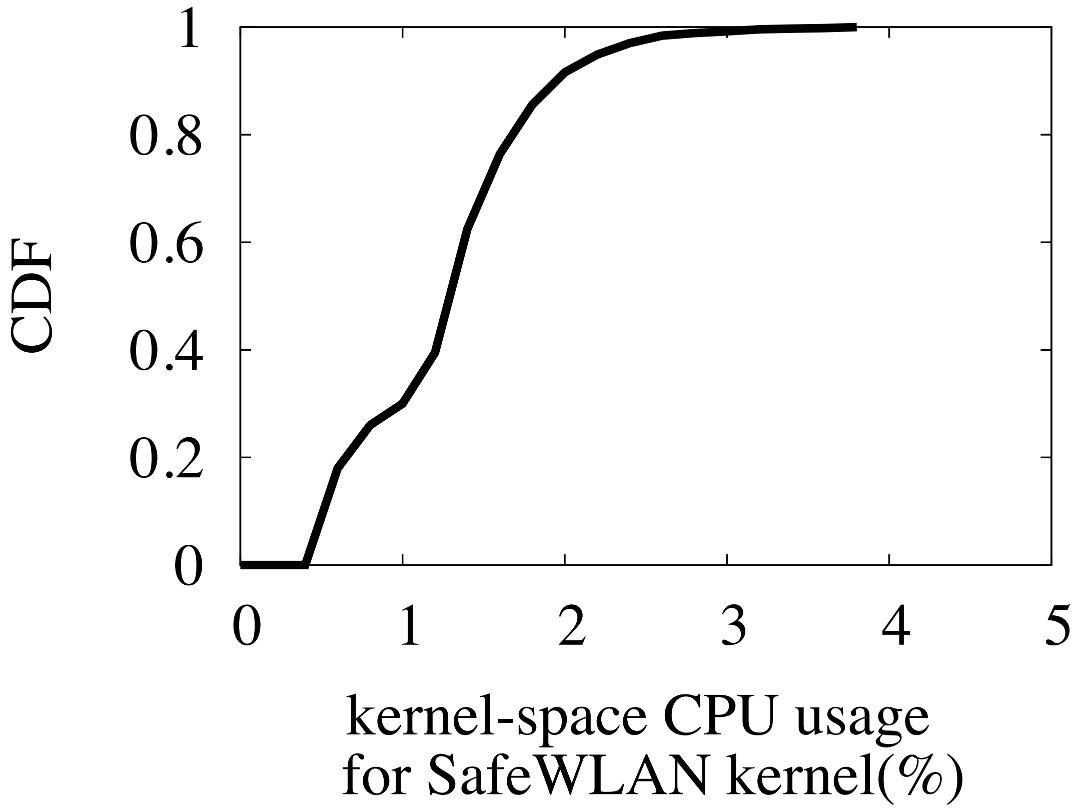


FIG. 45: Kernel-space CPU usage distribution of running SafeEnd kernel modules in Nexus 4.

device. Note that, SafeEnd has no noticeable relative changes on throughput.

The standard kernel-space components of the SafeEnd has 1.2% CPU usage on average in the mobile device. In SafeEnd, we have extended the Datapath to have a new action of padding random bytes at the end of IPsec packet. Despite that new action, we observe negligible overhead of the extended Datapath in a mobile device. The figure 45 shows the cdf plot of the CPU usage for running the extended Datapath and IPsec protocol in the mobile device. Almost 80% cases the SafeEnd kernel components show 1.5% CPU usage or less on the mobile device. We observe that the user-space component of the SafeEnd has 2% CPU usage on average in the mobile device (i.e. Nexus 4 smartphone). However, the user-space CPU usage increases with the frequency of flows per-app. Figure 46 shows the CPU usage of running the SafeEnd agent in the mobile device.

6.7.3 VERTICAL HANDOVER

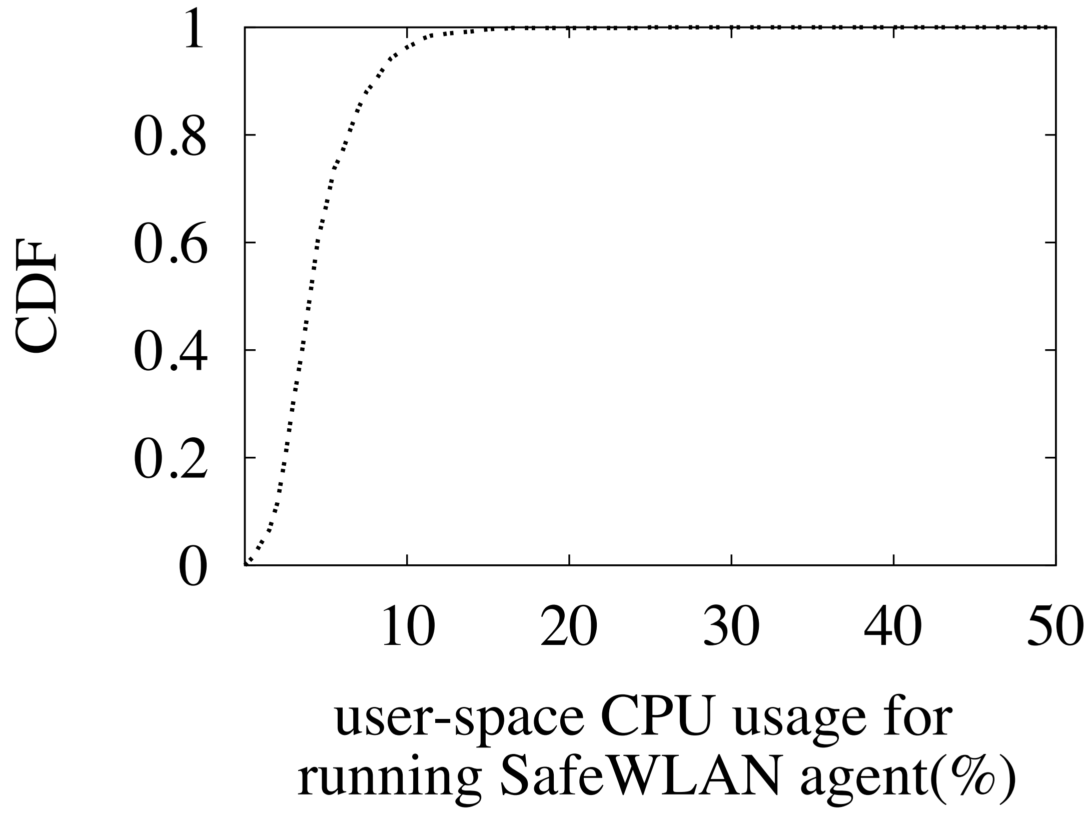


FIG. 46: User-space CPU usage distribution of running SafeEnd agent modules in Nexus 4.

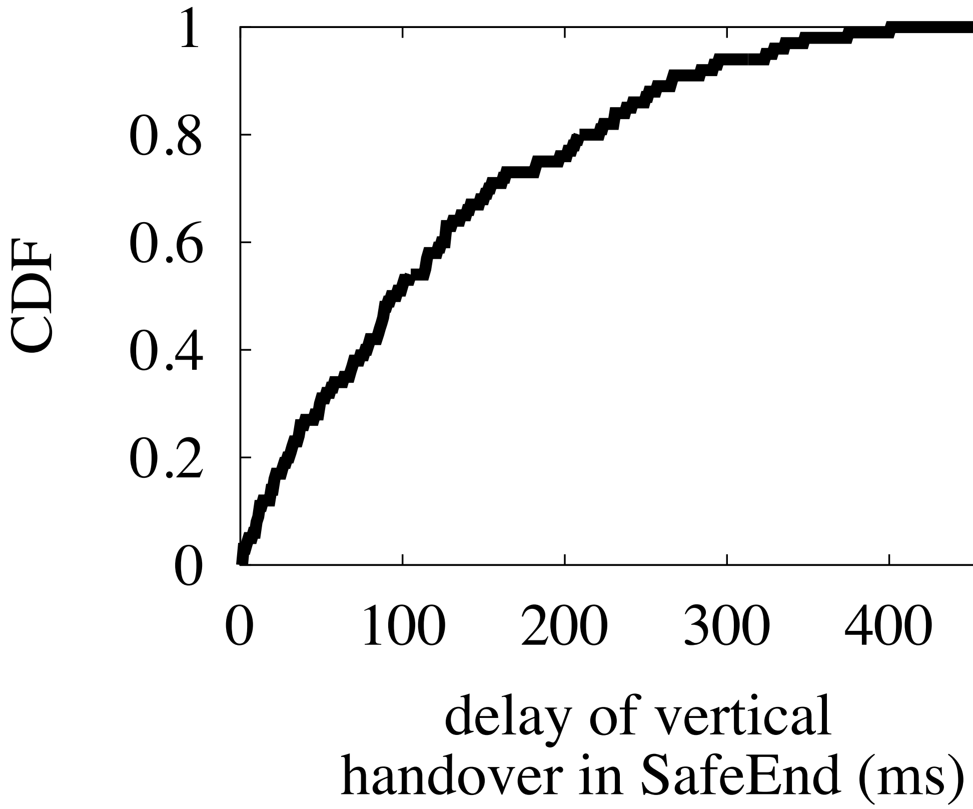


FIG. 47: Statistical distribution of the delay for vertical handover in SafeEnd.

In order to analyze the MOBIKE handover in SafeEnd, we evaluate the delay and the packet loss of vertical handover from Wi-Fi to cellular (i.e., HSPA+) for “secure channel” flows. In handover, the delay represents the time it takes for the *initiator* to start the MOBIKE protocol with the *responder*, update the IPSec SAs stored in SADs with new IP address, bind the `vport-ipsec port` to the IP address of cellular interface, and finally getting the acknowledgement from the *responder* about the update of new IP address. Figure 47 shows the statistical distribution of delay for doing the vertical handover in SafeEnd for “secure channel” flows. Typically real-time interactive application such as VOIP has a strict requirement on delay. For instance, voice and interactive video require delay is not exceeding 150ms. However, Figure 47 shows over 65% cases delay is below 150ms. This result indicates that the delay is not small enough for real-time application in many scenarios. On the other hand, video streaming applications are not delay sensitive due to the buffering. In that case, vertical handover mechanism in SafeEnd has a unnoticeable impact on the video streaming applications.

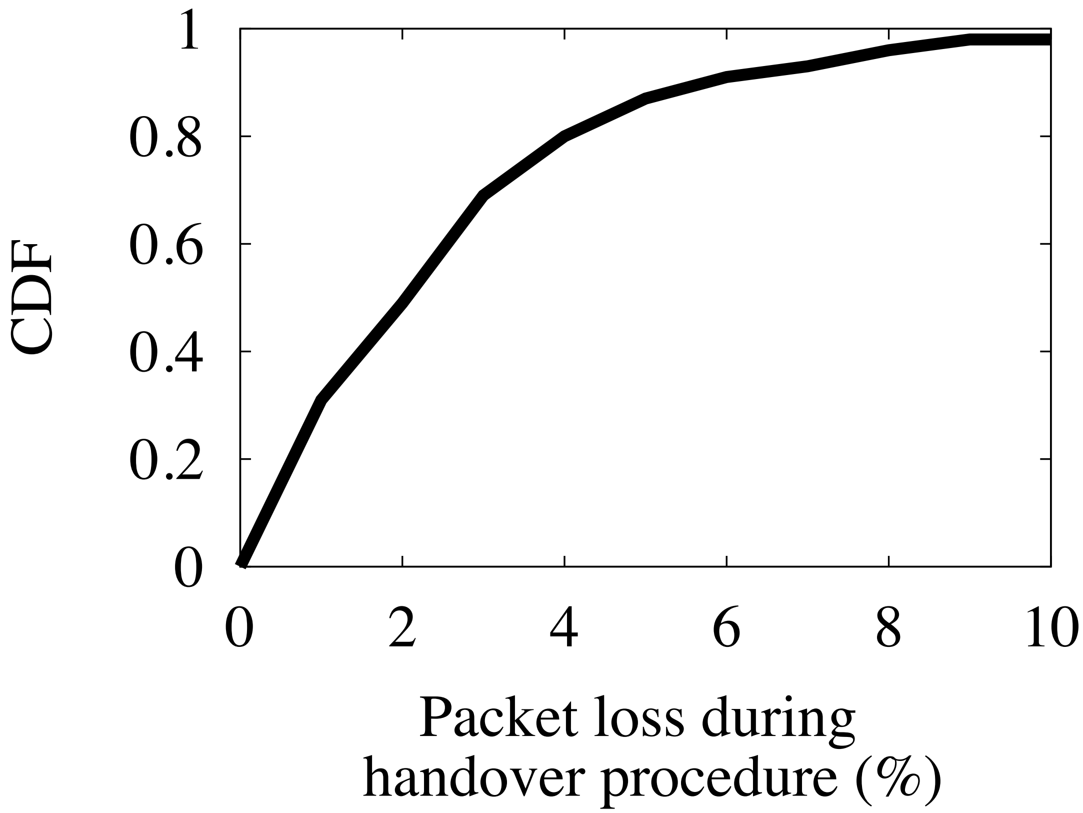


FIG. 48: Statistical distribution of the packet loss in percentage during vertical handover procedure in SafeEnd.

Regarding packet loss, the greater the throughput of the flow, the higher the amount of packet losses. During the delay of vertical handover in SafeEnd, the "secure channel" flow continues to send the packets. Since there is no method to hold these data packets during the handover procedure, so these sent out packets may not reach the destination. Hence, the packet loss increases if the throughput rate is high, assuming handover delay is constant. Note that, different percentage of packet losses is acceptable for the different type of applications. For instance, packet losses above 5%-10% have a significant effect on the quality of the video streaming application. Figure 48 shows the distribution of packet losses in percentage for the different type of mobile applications (video streaming, VOIP app, web browsing, social networking etc.). From the figure, we observe, SafeEnd suffers from 2% packet loss on average.

6.8 RELATED WORK

In the case of Bring Your Own Device (BYOD), several solution allows to separate corporate data from personal data on mobile devices [8, 31, 47, 61, 100, 114, 121, 143, 144, 177, 182]. Most of these solutions have coarse-grained and static policies on the applications. Furthermore, theses solutions mainly focus on protecting the devices or application data, and nothing to do with the network security. In order to protect the application's network data, there have been many proposed solution of managing network-wide mobile devices from network infrastructure [174]. However, such remote network management solutions are not well-suited for dynamic network devices like mobile devices. Therefore, recently researchers are focusing client-side network security solutions [73, 85]. Among them, very few of them have fine-grained and programmable network security policies on the applications. For instance solution such as [73], provide application specific and device-context-aware network access policies. In another work [85], the author has used network vitalization technique similar as SafeEnd to isolate the network traffic between sensitive (i.e., medical apps) and non-sensitive apps. However, unlike SafeEnd, none of the work have focused on the network security concern of running sensitive apps over the unlicensed wireless channel.

In SafeEnd, we have used traffic shaping or traffic obfuscating technique, which are heavily used in the area of application censorship. However, very few work have actually proposed and validate to use the traffic shaping technique to address eavesdropping attack [11, 32, 199]. Unlike SafeEnd, none of them have addressed mobile application traffic before. Besides traffic shaping, similar to [196], SafeEnd provides APIs for applying application-aware IPsec security policy. However, the work in [196], was mostly focusing on developing a programmable IPSec framework for the user.

6.9 SUMMARY

With the use of more and more sensitive applications running on mobile devices, we believe there is a great need for in-device network security solutions. Towards that effort, in this chapter, we have proposed an application-aware network security solution, SafeEnd. In the chapter, we have addressed few of the security attacks of a wireless network in SafeEnd. We also implemented and evaluated a prototype of the security solution in Android device. We believe, there is the lack of user-friendly and flexible security tools for mobile devices. Therefore, In addition to the performance and the energy efficiency, we need an in-device security solution that is easy to use. In future work, we like to explore the usability issues of our proposed SafeEnd.

CHAPTER 7

FUTURE WORK

In this chapter, we discuss the future work direction of weSDN framework. More specifically, we focus on two important future work directions of our framework. In first part, we lay out a design plan to integrate and bring the Wi-Fi MAC layer under the control of weSDN framework. Thus, we can have full programmable network network stack for end-devices and wireless access devices. In second part, we discuss the potentiality of migrating weSDN framework in cellular network, and the possibility of new services in that context.

Following is the outline of this chapter. In section 7.1, we propose the design overview of integrating Wi-Fi MAC layer with weSDN framework, and the technique of making the MAC layer programmable. Finally, in section 7.3, we discuss the idea of utilizing weSDN framework in cellular network context.

7.1 PROGRAMMABLE MAC LAYER - INTEGRATING WI-FI MAC LAYER WITH WESDN FRAMEWORK

In this thesis work, so far we have discussed several enhancing techniques of the OVS and the scheduler in the network stack of end-devices and wireless APs. However, in this thesis work, we have not discussed how to make the layer 2 open and programmable. More specially, how to make 802.11 MAC layer programmable, and bring it under the control of SDN framework. Therefore, in this chapter we have discussed the outline of making the MAC layer integrated with the SDN framework. This extension in the network stack will provide flexible and programable APIs to control full network stack of the end-device. Furthermore, through such APIs, users or mobile applications or network infrastructure can make context-aware network management to increase energy efficiency, throughput, truly end-to-end QoS and QoE. We believe, enhancing and leveraging the OVS with 802.11 MAC layer in mobile devices could have use-case scenarios, such as network diagnosis, application-aware security settings, and many-more. In this section, we propose the design overview of integrating OVS with the WiFi MAC layer, and the technique of making the MAC layer programmable. Finally, we also discuss the potential prospect of integrating OVS with the WiFi MAC layer.

Figure 7.1 shows the programmable full network stack (L1-L4) of a wireless Node (i.e., end-device, wireless APs etc.). The figure shows, we use the OpenFlow switch, Open vSwitch (OVS) kernel module in the network stack below the TCP/IP layer. The OVS kernel module and it's user space client provide us the programmability for the Network layer L2-L4. However, OVS is specially designed for ethernet layer, which can handle many of the functionalities of ethernet layer 2, for example mac learning, LLDP etc. Similar to ethernet layer 2;, WiFi (802.11) layer 2 or MAC layer has many of the functionalities (i.e., rate control, power saving control, RTS-CTS, frame aggregation etc.), which are not supported by the existing OVS. Therefore, we propose to extend the OVS and build a programmable MAC layer below the OVS.

Figure 7.1 shows different components of the programmable MAC layer. Similar to OVS datapath "flow-table" cache the MAC layer contains an "MAC table". The "MAC table contains number of "mac-rules" that allows the network operator to define different 802.11 MAC layer functionalities per-device/per-flow/per-frame. In addition, we extend the OpenFlow protocol and OVS to append "mac-rules" with the standard flow-rules. The "Wireless ext." in the OVS datapath is responsible for interacting with the "MAC table" for inserting/updating the "mac-rules".

We have categorized the MAC layer functionalities in three groups, i) Management Functionality, ii) Atomic Functionality, and iii) Compound Functionality. The Management functionality correspond to Association, Authentication, Active/Passive channel scanning, and Power Saving Management (PSM). These functionalities are not directly related to handling the TX and RX of 802.11 frames. Often management functionality is set by the cfg80211 through client applications like wpa_supplicant, hostap.

The Atomic functionality consists of Backoff configuration, RTS/CTS, ACK/no-ACK, HT-mode, Band, WMM setting etc. This functionality often considered as one-time setting of the Wi-Fi driver. However, in this framework, we make the setting of these functionalities programmable based on the matching rules in the MAC table. The Compound functionalities consist of rate control, aggregation, and fragmentation. These functionalities often require having some sort of algorithmic computation. Also, such functionalities can also depend on each other, for example, aggregation can depend the on the output of the rate control algorithm.

The Tx Handler handles the egress frame based on the matching "mac-rules" in the "MAC table". According to the matching "mac-rules" Tx handler interact with MAC layer functionalities to update the mac layer configuration and frame transmission configuration. On the other hand, Rx Handler, handles the ingress frame to update any statistic in the "MAC table" matching "mac-rules". In addition, it also update the MAC layer functionalities according to the matching "mac-rules". The Queue Management, handle

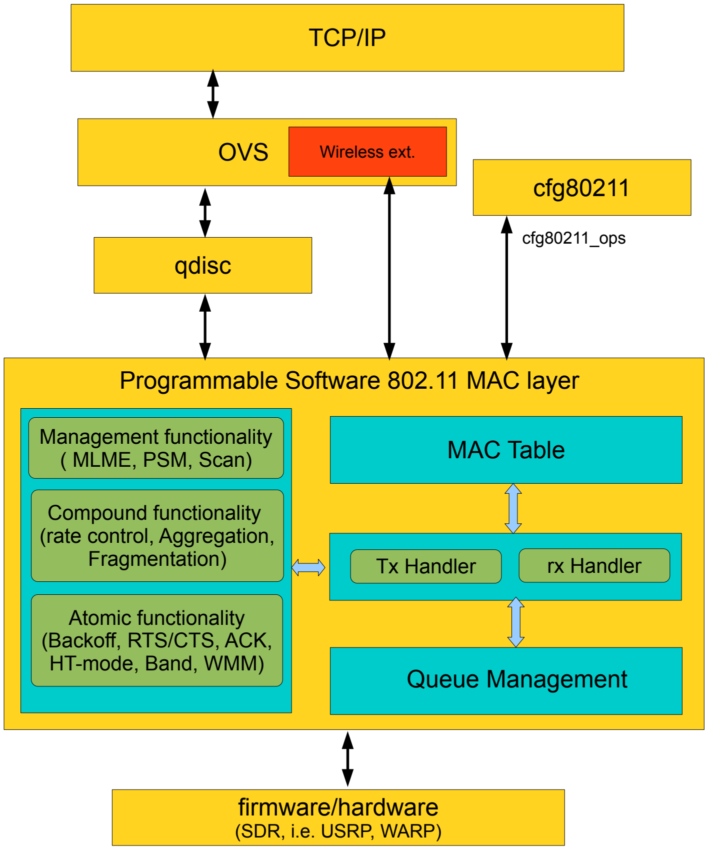


FIG. 49: Architecture of the programmable network stack for the Wi-Fi Devices.

different numbers of queues, and their queuing and dequeuing event according to the define "mac-rules" in the "MAC table".

We can build the programmable MAC layer on top of the Software Defined Radio (SDR). The SDR allow us to have programmable PHY layer and the lower 802.11 MAC layer functionalities such as Retransmission, DCF, RTS/CTS transmission, ACK transmission, Contention window counting, carrier sense, and packet recognition. Furthermore, we also rely on the SDR to collect PHY layer statistics such as frequency band, receive signal strength indicator (RSSI), Tx power, Tx rate etc.

In following subsections, we describe the design consideration and implementation challenges for different components of the programmable framework for 802.11 MAC layer. More specially, the following four main tasks: i) Design and implementation of the MAC table, ii) Tx/Rx Handling, iiI) Design and Implementation of the Queue Management, iv) Integration with the SDR.

7.1.1 TASK 1: DEFINING THE "MAC-RULES" AND DESIGNING THE MAC-TABLE

We define the "mac-rule" as part of the "flow-rule" in the OVS datapath. The "mac-rules" allow the network operator to define different configuration and MAC functionalities of handling a frame transmission. However, defining "mac-rules" for covering diverse type of MAC layer functionalities is one of the key design challenge of this task. Specially designing the "mac-rule" for the compound functionalities is challenging. Often the computation of one compound functionality depends on the computation of another compound functionality. Such dependency make "mac-rule" defining more challenging.

Following is an example of a simple "mac-rule" as part of the "flow-rule" in the a*add-flow* command. Note that the existence of starting keyword "-wlan0:mac" represent a "mac-.rule" portion in the "flow-rule".

```
"add-flow br0 in_port=1 udp action=output:2 -wlan0:mac
mac_action=RTS_CTS(0),ACK(0)"
```

The above rule disable the RTS/CTS and apply no-ACK policy for the frame that is UDP flow. There can be three kinds of rule in the framework, i) rule without mac-rule, ii) rule with both mac-rule and flow-rule, and iii) rule with only mac-rule. The MAC-table is responsible for storing only the portion of "mac-rule" in the hash-table. The Wireless ext. in OVS is responsible for splitting the rule, if require to get the "mac-rule" portion, and send it to the MAC table, and the rest is stored in the flow-table.

Now splitting the rule in two part can create a scenario where single flow-rule can link with multiple mac-rule, and vice-versa. In such circumstance linking between flow-rule and mac-rule can be challenging. In order to address this challenge, we require careful designing and implementation of the MAC-table.

7.1.2 TASK 2: TX/RX HANDLING

Tx handling applies matching "mac-rules" on the egress frame. In order to find the matching "mac-rules" from MAC-table, the Tx handler needs to link the matching "flow-rule" with the corresponding "mac-rule" . Note that there can be multiple matching "mac-rules" that will define several MAC layer functionalities, which eventually will be applied on the frame transmission. In MAC-table, there can be “mac-rules” which works on every egress frame. However, operating the matching“mac-rules” at certain order is crucial for successful frame Tx handling. For example, rate control “mac-rule” should operate before frame fragmentation rule. Tx handling also receive statistics about transmitting the frame, such as the rate that is used for transmitting the frame, retry count, Tx power, antenna, band etc. Such statistic is feed to the MAC layer functionality to update according to the “mac-rules”.

Rx handling is responsible for updating the statistics for the MAC layer functionalities according to the “mac-rules”. For example, the RSSI of the received frame can be send to the rate control or aggregation functionalities which can update or adapt according to “mac-rules”. In summary, In receiving a single frame in the Rx handler, initiate the process of adapting multiple MAC layer functionalities based on given multiple “mac-rules”. However the order of executing “mac-rules” can be important for MAC layer functionalities, specially compound functionalities. In this task, we like to address this challenge. Furthermore, we like to explore efficient way of executing MAC layer functionalities.

7.1.3 TASK 3: DESIGN AND IMPLEMENTATION OF THE QUEUE MANAGEMENT (QM)

We found wireless driver queue management implementation often varies from vendor to vendor. There is no fixed standard defined scheme for the wireless driver queue management. Therefore, both in academic and industry we have seen a plethora of work on Wi-Fi driver queue management. Furthermore, according to the literature the queue management in the ethernet NIC card is not applicable for Wi-Fi NIC card.

Also, the number of queues maintained in the wireless driver also varies among different implementation. Typical practice is that wireless driver maintains four queues for four Access category (AC). Some wireless driver maintains additional queues for frame aggregation. Providing flexible number of queues make wireless driver implementation complex. Therefore, we propose to stick with fixed size of queues in QM. Defining a suitable number of queues for QM is a challenging task. Therefore, we like to explore the performance of the QM for different number queue with different queue size.

In the enqueue event, first an egress frame is mapped to certain queue based on either priority or Access Category or `skb` marking. Then the enqueue action take place on the selected queue. The successful enqueue event also update the statistics in the QM. Note that the enqueue event is not defined by the “mac-rules”. Unlike that, dequeue event is defined by the “mac-rules”. Following is an example of dequeue scheme in the QM,

“`-wlan0:mac --dequeue=`$(p_1, p_2, p_3 ... p_n)$”

where p_i, represents the probability of picking up a frame from queue index i, and n is the total number of queues in the Queue Management. The dequeue event pick up the queue that has the maximum (i.e. *max*) probability value. In case of equality the *max* function pick up the smallest queue index. The network operator can defined this probability dynamically or fixed. In the dynamic scenario, the network operator can leverage statistics of r_i number of frames in the queue i, and T represents the total number of frames in all queues. Note that, the MAC table only contains one such rule that defines the QM. In the “mac-rule” syntax, the network operator can use the symbol r_i and T for having a mathematical expression for calculating the p_i. For example following is an example of “mac-rule” for weighted queue management,

“`-wlan0:mac --dequeue=`$(r_1/T, r_2/T, r_3/T, ..., r_n/T)$”

Such “mac-rule” get an update after every enqueue and dequeue event in QM. Thus, it maintains the updated probability or weigh of picking up the queues for dequeue event. However, designing such programmable and flexible rule based QM has many open questions, for example, *What will happen if the QM “mac-rule” get updated in the middle?*

7.1.4 TASK 4: INTEGRATION WITH THE SDR

Existing implementation of 802.11 MAC layer on top of SDR often focus on the basic functionality like DCF, ACK Tx, Frequency etc. Such MAC layer implementation is considered as thin MAC layer. Because the objective of those project is to leverage PHY layer programmable functionality of the SDR to implement different wireless technology, like 802.11. In this task, we focus on integrating the programmable 802.11 MAC layer with the PHY layer or thin MAC layer running on SDR. One of the key challenge of such integration is the communication between the programmable MAC layer and the SDR. For example, the MAC layer rate control functionalities defined the rates of transmitting a frame. Now the question is, *how the SDR will know the rates of actually transmitting the frame*? One of the typical way is to add the rate information in the *Control Block (CB)* of the linux packet buffer (i.e. `skb_buff`).

There are other MAC layer functionalities, like RTS/CTS, no-ACK, Backoff which are often configure statically in the Wi-Fi driver. However in our framework, we allow these functionalities to configure programmatically for handling the transmission of a frame. Therefore, similar to rate control setting, we also need to convey the configuration of these functionalities to the PHY layer in SDR. One logical approach can be to extend the `struct ieee80211_tx_info`, to add new fields for these functionalities. Note that, `struct ieee80211_tx_info` is also part of the *Control Block (CB)* in `skb_buff`. However such approach require further investigation and evaluation.

On the other hand, the PHY layer or the thin MAC layer in SDR also needs to forward the frame Rx/Tx statistics to the programmable MAC layer for updating the MAC layer functionalities according to the "mac-rules". However defining the statistics from SDR, and the process of forwarding and updating the MAC layer functionalities, is one of the open design question we like to address in this task. In overall, we like to explore different possible design consideration of interacting the programmable MAC layer with the PHY layer in SDR.

7.1.5 PROSPECT OF INTEGRATING OVS WITH WIFI INTERFACE

Recently we have seen several effort of implementing cross-layer network scheme to improve the performance of application and services for smart devices. The programmable full-network stack provide complete flexibility of implementing different cross-layer scheme. Thus we can provide innovation in developing more adaptive network stack for the smart devices based on the network stack. Furthermore, we can develop end-to-end system that adjust the MAC layer functionalities based on the network measurement.

In addition, integrating OVS with the WiFi interface will allow us to monitor the wireless traffic for different diagnosis purpose. There are number of reasons, when a user can face network communication problem which can be due to the weak signal, traffic congestion, interference, bottleneck, and handover. Different network communication problem require different remedy or troubleshooting technique. We believe OVS with WiFi interface at end-device could provide unique capability to diagnosis the network communication.

7.2 DESIGNING OPEN NETWORK APIS FOR WESDN FRAMEWORK

Our objective is to utilize weSDN framework in collecting and providing information on the current network conditions to both users and application developers, gathering users feedback, developing novel wireless protocols based on users feedback. weSDN framework is capable of collecting various fine-grained real-time network properties on wireless links (interference level, bit-rate, congestion level, etc.) via weSDN OpenWireless API (as above) as well as applications/flows (packet size, jitter time, end-2-end delay, flow type, etc.) from weSDN Flow Manager.

In order to have such API development, we need to address several issues such as how to translate these network properties to users level information? For example, user will relate more to time duration, amount of battery consumption, and the cost to upload/download a file using Wi-Fi interface in comparison to Cellular interface rather than knowing the available bandwidth. Another issue is how and when this information should be presented to user? Should it be presented on periodical notifications, with a start of new application, when a significant change in network condition, or as a continuous widget? In addition to smart device users, we like to explore how to translate this information into APIs (part of Southbound API in weSDN framework that could be integrated with Android SDK in future) that could be utilized by application developers to query users feedback and/or adapt the behavior of their application. This can enable future innovation where application providers can optimize the network for their specific needs in order to better serve the users.

Once network information is communicated to users, it is important to capture user feedback and intent. One way to express user's feedback is the user high-level preferences that are translated to low-level semantics and used to control the network. Another way of user feedback is the change in user behavior ("wireless behavior") that could be encouraged by introducing incentives, e.g., differentiated pricing. Our design address questions such as: what is the best place to capture user feedback/intents (e.g. application, device, or network-wide)? What types of intents (change his location for better link quality, reschedule the starting time of his session for a better battery consumption, switching between different AP for better transmission time, etc.) are appropriate for users and what types of incentives (and eventually penalties) to be used? Should we collect user feedback implicitly that could be inferred from device sensors and the user activities on the device, or explicitly in form of suggestions with incentives on the graphical user interface, e.g., a map and directions towards a better location?

One main operation of weSDN control plane, once the user intents are collected, is to do a network-wide coordination, resolve conflicts and mediate feedback and intents from different users or even from the same user through the collaboration between local and global user in-the-loop modules residing local controller and weSDN Controller respectively. This is necessary to ensure the request is valid and the network is not overloaded to maintain a functional network at all times. In addition, the role should include providing feedback on whatever the intent is admitted or denied, and possibly suggest alternatives to a denied intent. Moreover, one of the key future work will be to develop sets of Northbound APIs and Southbound APIs for exploring different management services for both Home and enterprise Wi-Fi networks.

7.3 WESDN FRAMEWORK IN CELLULAR NETWORK CONTEXT

In this thesis work, so far we have focused on Wi-Fi network. Besides Wi-Fi, one important direction of our research will be to explore extending the weSDN framework to end devices in the context of the cellular network (smartphones, tablets, home gateways, Wi-Fi hotspots over cellular backhaul, etc). In particular, in cellular network, some questions of interest include adding new control and management features, scalability and overhead, control granularity refinement to per-device level, opening APIs to content providers and end users, multi-interface management, etc. Similar to our current implementation, we consider to apply OpenFlow, vSwitch and similar technologies on the end devices to improve cellular network performance and enhancing cellular user experience.

In this thesis work, we have developed three services based on our framework. In chapter 3, we have discussed about other possible services based on weSDN framework. In each of these services, we require to provide new set of APIs from end-devices or wireless access device. One of the key challenge is to design and develop these APIs for different services based on weSDN framework.

There are many new services that could benefit from our weSDN framework in cellular network context. For example, video streaming application is expected to grow significantly and most of the bandwidth consumed by end devices will be from multi-media. Studies shows, more than half of YouTube watch time happens on mobile devices, with a large and rapidly increasing fraction of this time spent on cellular networks. Therefore, we believe, video streaming applications such as video on demand and live streaming apps is one type of applications that would take a great advantage of our framework. One of the well-known shortcomings of the current wireless network systems is that they can not treat end devices based on their conditions or capabilities. Consequently, users, for instance, streaming videos while they are distant from the access point and having weak wireless signals are likely to suffer from poor viewing experience particularly when they compete over bandwidth with other devices with good links. Therefore, to guarantee a satisfactory level of viewing experience for all users in the network, it is necessary to have a mechanism to treat users based on their current situations and conditions.

In order to develop such mechanism, first we have to conduct extensive analysis of the video traffic from one of the most popular HTTP adaptive video player to answer the following two questions: is there still a way to assist and optimize the performance of the commercial video players even with an encrypted traffic? And, how this can we effectively assist the video players? In our studies, we highlighted some performance issues with commercial players regarding concurrent video stream over the shared wireless network. As confirmed by our experiments, most of these issues are mainly caused by the aggressive competition between the players for the available network resources. Therefore, any future effort that aims to enhance the performance of video players and the viewing quality for all clients, should concentrate on mitigating this competition in such away that ensures the fairness among the player and, maximizes the utilization of the available bandwidth. In addition, we beleive that through a deep understanding and careful analysis of the HTTPs video traffic, valuable information about the competing streams can be obtained and utilized in developing a network based solution that can significantly improve the video QoE and assist the video players to perform much better.

CHAPTER 8

CONCLUSION

Existing SDN framework is incapable of addressing the current complexity of the wireless network edges. Therefore, in this thesis work, our effort is to make wireless network edge (i.e., access devices and end devices) more open and flexible with respect to networking in order to facilitate better monitoring and controlling capabilities. More specifically, we believe that an SDN-like paradigm needs to be pushed to mobile clients to provide optimal network performance for the wirelessly-connected clients. Therefore, in this thesis work, we have designed and developed a framework, called weSDN that pushes the SDN capability all the way to end-devices to bring the last-hop under the control of SDN framework by creating new components and APIs. Thus, end-devices with an SDN-like paradigm, can efficiently interact with the SDN-based wireless network infrastructure to optimize and enhance the overall performance of wireless network management. Furthermore, SDN at end-device allows the end-users to have programmable control and monitoring capabilities over the network stacks of the end-devices. Thus, the end-users can apply their own network policies, diagnose network connectivity, and select appropriate interfaces for their network flows.

In this thesis work, we have described detail architecture of weSDN framework specially designed for end-devices and wireless APs. In the design, we have focused on having similar components for both wire (e.g., switches, routers) and wireless (e.g., wireless APs, end-devices) network devices. However, we have extended those components with new sets of APIs to support wireless network connectivity. Finally, based on this architecture, we have developed and evaluated three new services: WLAN virtualization, application-awareness networking and securing wireless network edges. In each of these services, we elaborate the design and the implementation of the components of weSDN framework. In this thesis, we have three separate chapters for describing each of this services in details.

In chapter 4, we use weSDN framework to design, implement and evaluate a WLAN virtualization service that slices end devices using a Time Division Multiple Access (TDMA) like airtime scheduling, named pseudo TDMA (pTDMA), that runs on top of 802.11 MAC. By using a modified Linux Qdisc on end devices, pTDMA virtualizes (separates) airtime resource between network slices while minimizing contention between clients within a slice.

pTDMA also allows client WiFi interfaces to more efficiently utilize their active time and to sleep longer outside of the given transmission windows. We also have evaluated network virtualization capability and its improved power efficiency from our prototyping on Android phones.

In chapter 5, we design, develop and evaluate a distributed lightweight traffic classification system, SmartEdge based on weSDN framework. In realizing SmartEdge, we envision to extend and push SDN paradigm all the way to mobile devices. More specifically, we leverage and extend SDN tools (i.e, Open vSwitch, OpenFlow etc.) to wireless edge devices (i.e., smartphone, tablet, Wireless AP, Base Station etc.). Hence, SmartEdge can efficiently recognize individual mobile applications (e.g., facebook, skype, tango, fringe etc.) and its various traffic flows (e.g., video chat, voice chat, video stream, etc.) on fly. We evaluate the performance of our system using several metrics such as CPU utilization, energy efficiency and network throughput, which are critical to provide QoE to mobile users.

In chapter 6, we design network security solution SafeEnd based on weSDN framework to address the security threat that can happen over unlicensed wireless medium. For instance, we show a new type of attack, *App-spoof* based on eavesdropping on wireless channel, and the security policies of SafeEnd to address this attack. Typically, existing network security solutions for mobile devices are mostly enterprise solution that depends on the network infrastructure to apply security policies. This network infrastructure based security solutions are often static and coarse-grained. Therefore, we design and develop in-device application-aware network security solution, *SafeEnd* for mobile devices. We evaluate a prototype of SafeEnd framework in Android OS. The result show, SafeEnd introduce a negligible performance overhead and minor impact to battery life, while providing fine-grained network security policies to protect the wireless network communication of sensitive applications.

Finally, in chapter 7, we have discussed the possible extension of weSDN framework, and new services based on this framework. As a future work, we have laid out the design of integrating the Wi-Fi MAC layer with weSDN framework to provide full programmable network stack for end-devices and wireless access device. We also discuss about applying weSDN framework in cellular network context. In addition, we discuss about a service, based on weSDN framework, that could benefit concurrent video streaming applications over cellular network .

With the emerging trend of new technologies like IoT and edge computing, we are at the dawn of an era in networking that is defined by the connection of everything and everyone with the goal of automating much of our daily life, and optimizes decision-making of our everyday routines and processes. In this regard, we believe, weSDN is a timely solution to provide such intelligence and dynamic approach to networks. In future, we envision to build many more intelligent services in network based on weSDN platform.

In summary, following is the contribution of this thesis:

- We design weSDN framework, where we extend the SDN framework to network wireless-edges and end devices.
- We implement weSDN framework on real-testbed in linux platform.
- We implemented open APIs to provide extensible and programmable abstraction of the wireless network edges.
- We utilize weSDN framework to develop and evaluate new services: i) Application-awareness networking, ii) WLAN virtualization, and iii) Security at network edges.

REFERENCES

[1] Linux Kernel Version 4.3. HOWTO for multiqueue network device support. `http://www.mjmwired.net/kernel/Documentation/networking/multiqueue.txt`, November 2015.

[2] Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Shobha Venkataraman, and He Yan. Prometheus: Toward Quality-of-Experience Estimation for Mobile Apps from Passive Network Measurements. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, page 18. ACM, 2014.

[3] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 19–24. ACM, 2013.

[4] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio La Oliva, Martin Draxler, Rajesh Gupta, Vincenzo Mancuso, Laurent Roullet, and Vincenzo Sciancalepore. CROWD: an SDN approach for DenseNets. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 25–31. IEEE, 2013.

[5] Safa Alkateb. 5 Things You Need to Know About Deep Packet Inspection (DPI). `http://www.cavium.com/pdfFiles/CSS-DPI-White-Paper.pdf`, 2011. Online; accessed 29 January 2015.

[6] Wi-Fi Alliance. WMM^{TM} Power Save for Mobile and Portable Wi-Fi®CERTIFIED Devices. `https://www.wi-fi.org/file/wi-fi-certified-for-wmm-power-save-support-for-advanced-power-save-for-mobile-and-portable`, 2005.

[7] Scott Amyx. Why the internet of things will disrupt everything. `http://innovationinsights.wired.com/insights/2014/07/internet-things-will-disrupt-everything/`, July 2014.

[8] Android. Device Administration. `http://developer.android.com/guide/topics/admin/device-admin.html`. Online; accessed 29 December 2015.

[9] Aruba Networks. Configuring Adaptive Radio Management (ARM) Profiles and Settings. `http://community.arubanetworks.com/aruba/`

`attachments/aruba/unified-wired-wireless-access/7592/1/ARM%20Doc%20Supplement.pdf`, 2008.

[10] AT&T. ECOMP (Enhanced Control, Orchestration, Management and Policy) Architecture White Paper. `http://about.att.com/content/dam/snrdocs/ecomp.pdf`, February 2016.

[11] Michael Backes, Goran Doychev, and Boris Köpf. Preventing Side-Channel Leaks in Web Traffic: A Formal Approach. In *Proceedings of the 20th Network and Distributed Security Symposium*, 2013.

[12] Marc Balon and Bernard Liau. Mobile virtual network operator. In *Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2012 XVth International*, pages 1–6. IEEE, 2012.

[13] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[14] Iyad Batal and Milos Hauskrecht. A Supervised Time Series Feature Extraction Technique Using DCT and DWT. In *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*, pages 735–739. IEEE, 2009.

[15] Michael Till Beck, Martin Werner, Sebastian Feld, and S Schimper. Mobile edge computing: A taxonomy. In *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014.

[16] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62. ACM, 2009.

[17] Gautam Bhanage, Dipti Vete, Ivan Seskar, and Dipankar Raychaudhuri. SplitAP: leveraging wireless network virtualization for flexible sharing of WLANs. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[18] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype Traffic: When Randomness Plays with You. *ACM SIGCOMM Computer Communication Review*, 37(4):37–48, 2007.

[19] Sem Borst. User-level performance of channel-aware scheduling algorithms in wireless data networks. *IEEE/ACM Transactions on Networking (TON)*, 13(3):636–647, 2005.

[20] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 271–282. ACM, 2014.

[21] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. Cloudtransport: Using cloud storage for censorship-resistant networking. In *Privacy Enhancing Technologies*, pages 1–20. Springer, 2014.

[22] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5. 0 Machine Learning Algorithm. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000882–000887. IEEE, 2012.

[23] Kan Cai, Junfang Wang, Reza Lotun, Michael J Feeley, Michael Blackstock, and Charles Krasic. A wired router can eliminate 802.11 unfairness, but it's hard. In *Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 49–54. ACM, 2008.

[24] Arthur Callado, Carlos Kamienski, Géza Szabó, Balázs Péter Gerö, Judith Kelner, Stênio Fernandes, and Djamel Sadok. A survey on internet traffic identification. *Communications Surveys & Tutorials, IEEE*, 11(3):37–52, 2009.

[25] Valentın Carela-Espanol, Pere Barlet-Ros, and Josep Solé-Pareta. Traffic classification with sampled netflow. *traffic*, 33:34, 2009.

[26] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[27] Martin Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Communications of the ACM*, 57(10):86–95, 2014.

[28] Jeffery Case, Mark Fedor, Martin Schoffstall, and C. Davin. A simple network management protocol (SNMP). *Network Information Center, SRI International*, 1989.

[29] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[30] Chris Chase. The internet of things as the next big thing. http://www.directive.com/blog/item/the-internet-of-things-as-the-next-big-thing.html, June 2013.

[31] Kevin Zhijie Chen, Noah M Johnson, Vijay D'Silva, Shuaifu Dai, Kyle MacNamara, Thomas R Magrino, Edward XueJun Wu, Martin Rinard, and Dawn Xiaodong Song. Contextual policy enforcement in android applications with permission event graphs. In *In Proceedings of the 20th Network and Distributed Security Symposium*, 2013.

[32] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 191–206. IEEE, 2010.

[33] Tao Chen, Marja Matinmikko, Xianfu Chen, Xuan Zhou, and Petri Ahokangas. Software defined mobile networks: concept, survey, and research directions. *Communications Magazine, IEEE*, 53(11):126–133, 2015.

[34] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic Execution Between Mobile Device and Cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[35] Cisco. Cisco Compatible Extensions Program for Wi-Fi Tags Guidelines. https://www.cisco.com/web/partners/downloads /765/other/ccx_program_for_wi-fitags_qa.pdf, 2007.

[36] Cisco. Radio Resource Management under Unified Wireless Networks. http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/71113-rrm-new.html, May 2010.

[37] Cisco. Meraki White Paper: Meraki Hosted Architecture. http://www.voyager.net.uk/wp-content/uploads/downloads/2014/02/WP-Meraki-Hosted-Architecture.pdf, February 2011.

[38] ONF Market Education Committee. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.

[39] Jonathan Corbet. Network transmit queue limits. https://lwn.net/Articles/454390/, 2011.

[40] Mike Coward. Encryption: will it be the death of DPI? `http://telecoms.com/39718/encryption-will-it-be-the-death-of-dpi/`, February 2012.

[41] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[42] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *Network, IEEE*, 26(1):35–40, 2012.

[43] Ivan Delchev. Linux traffic control. In *Networks and Distributed Systems Seminar, International University Bremen, Spring*, 2006.

[44] Shuo Deng, Anirudh Sivaraman, and Hari Balakrishnan. All your network are belong to us: A transport framework for mobile network selection. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, page 19. ACM, 2014.

[45] Avri Doria, J Hadi Salim, Robert Haas, Horzmud Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. Forwarding and control element separation (ForCES) protocol specification. *Internet Requests for Comments, RFC Editor, RFC*, 5810, 2010.

[46] Kevin P Dyer, Scott E Coull, and Thomas Shrimpton. Marionette: A programmable network traffic obfuscation system. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 367–382, 2015.

[47] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.

[48] A Ericsson. Ericsson mobility report, on the pulse of the networked society. *Ericsson, Sweden, Tech. Rep. EAB-14*, 61078, 2014.

[49] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9):1194–1213, 2007.

[50] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support vector machines for TCP traffic classification. *Computer Networks*, 53(14):2476–2490, 2009.

[51] Francois Faucheur. IP Router Alert Considerations and Usage. *Internet Engineering Task Force (IETF)*, 2011.

[52] Andrew D Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: An API for application control of SDNs. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 327–338. ACM, 2013.

[53] Denzil Ferreira, Vassilis Kostakos, Alastair R Beresford, Janne Lindqvist, and Anind K Dey. Securacy: an empirical investigation of Android applications' network usage, privacy and security. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 1–11. ACM, 2015.

[54] Graham Finnie. DPI and Traffic Analysis in Networks Based on NFV and SDN). `http://www.qosmos.com/wp-content/uploads/2014/01/Heavy-Reading_Qosmos_DPI-SDN-NFV_White-Paper_Jan2014.pdf`, January 2014.

[55] Open Networking Foundation. OpenFlow-Enabled Mobile and Wireless Networks. `https://www.opennetworking.org/solution-brief-openflow-enabled-mobile-and-wireless-networks`, 2013.

[56] Dan Frommer. 25 most popular mobile apps. `http://qz.com/253527/these-are-the-25-most-popular-mobile-apps-in-america/`, August 2014.

[57] Gary Shute. The Performance Equation. `https://www.d.umn.edu/ gshute/arch/performance-e` Online; accessed 29 January 2016.

[58] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward software-defined middlebox networking. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 7–12. ACM, 2012.

[59] Dennis Gessner, Joao Girao, Ghassan Karame, and Wenting Li. Towards a user-friendly security-enhancing BYOD solution. *NEC Technical Journal*, 7(3):113–116, 2013.

[60] Jim Gettys and Kathleen Nichols. Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55:57–65, 2012.

[61] Google. Android for Work. `https://www.google.com/work/android/`. Online; accessed 29 December 2015.

[62] Fabrizio Granelli, Anteneh A. Gebremariam, Muhammad Usman, Filippo Cugini, Veroniki Stamati, Marios Alitska, and Periklis Chatzimisios. Software defined and virtualized wireless access in future wireless networks: scenarios and standards. *Communications Magazine, IEEE*, 53(6):26–34, 2015.

[63] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. Softran: Software defined radio access network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2013.

[64] Victor H. Android smartphone users download just 870 MB over cellular per month. `http://www.fiercewireless.com/story/npd-android-smartphone-users-downloadjust-870-mb-over-cellular-month/2012-09-27`, September 2012.

[65] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, and Slim Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. *Communications Magazine, IEEE*, 53(9):48–54, 2015.

[66] Mark Hall et al. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explor. Newsl.*, 2009.

[67] Nikhil Handigol, Srinivasan Seetharaman, Mario Flajslik, Nick McKeown, and Ramesh Johari. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, 2009.

[68] Jiang HanPing and Yan Jun. Research and Design for IPSec Architecture on Kernel. In *Knowledge Discovery and Data Mining, 2008. WKDD 2008. First International Workshop on*, pages 509–512. IEEE, 2008.

[69] Dan Harkins, Dave Carrel, et al. The internet key exchange (IKE). *RFC 2409, november*, 1998.

[70] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: saving energy in data

center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 17–17. USENIX Association, 2010.

[71] Stacey Higginbotham. OpenRadio changes what it means to be an ISP. http://gigaom.com/2012/04/19/openradio-changes-what-it-means-to-be-an-isp/, April 2012.

[72] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 15–26. ACM, 2013.

[73] Sungmin Hong, Robert Baykov, Lei Xu, Srinath Nadimpalli, and Guofei Gu. Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security. In *Proceedings of the 23th Network and Distributed Security Symposium*, 2016.

[74] Mohammad Ashraful Hoque, Matti Siekkinen, Jukka K Nurminen, Mika Aalto, and Sasu Tarkoma. Mobile multimedia streaming techniques: QoE and energy saving perspective. *Pervasive and Mobile Computing*, 16:96–114, 2015.

[75] Amir Houmansadr, Thomas J. Riedl, Nikita Borisov, and Andrew C Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *In Proceedings of the 20th Network and Distributed Security Symposium*, 2013.

[76] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 363–374. ACM, 2013.

[77] Cisco IOS. Netflow white papers. http://www.cisco.com/en/US/products/ps6601/prod-white-papers-list.html, 2006.

[78] Nachikethas A Jagadeesan and Bhaskar Krishnamachari. Software-defined networking paradigms in wireless networks: a survey. *ACM Computing Surveys (CSUR)*, 47 (2):27, 2015.

[79] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, and Min Zhu. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[80] Michael Jarschel, Florian Wamser, Thomas Hohn, Thomas Zinner, and Phuoc Tran-Gia. SDN-based Application-Aware Networking on the Example of YouTube Video Streaming. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 87–92. IEEE, 2013.

[81] Hongbo Jiang, Andrew W Moore, Zihui Ge, Shudong Jin, and Jia Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 299–304. ACM, 2007.

[82] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108. ACM, 2012.

[83] 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. ACM, 2013.

[84] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proc. of the USENIX HotICE workshop*, 2011.

[85] Makito Kano. SeaCat: an SDN end-to-end containment architecture. Master's thesis, University of Utah, 2015.

[86] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[87] Hoon Kim, Keunyoung Kim, Youngnam Han, and Sangboh Yun. A Proportional Fair Scheduling for Multicarrier Transmission Systems. In *Proceedings of the 60th IEEE conference on Vehicular Technology Conference*, volume 1, pages 409–413. IEEE, 2004.

[88] Hyunchul Kim, Kimberly C Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 11. ACM, 2008.

[89] Wonho Kim, Puneet Sharma, Jeongkeun Lee, Sujata Banerjee, Jean Tourrilhes, Sung-Ju Lee, and Praveen Yalagandula. Automated and scalable QoS control for network

convergence. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 1–1. USENIX Association, 2010.

[90] Csaba Kiraly, Simone Teofili, Giuseppe Bianchi, Renato Lo Cigno, Matteo Nardelli, and Emanuele Delzeri. Traffic flow confidentiality in IPsec: Protocol and implementation. In *The Future of Identity in the Information Society*, pages 311–324. Springer, 2007.

[91] Tero Kivinen and Hannes Tschofenig. Design of the IKEv2 mobility and multihoming (MOBIKE) protocol. *RFC4621*, 2006.

[92] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[93] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[94] Arash Habibi Lashkari, Mir Mohammad Seyed Danesh, and Behrang Samadi. A survey on wireless security protocols (WEP, WPA and WPA2/802.11 i). In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. IEEE.

[95] Jeongkeun Lee, Mostafa Uddin, Jean Tourrilhes, Souvik Sen, Sujata Banerjee, Manfred Arndt, Kyu-Han Kim, and Tamer Nadeem. meSDN: mobile extension of SDN. In *Proceedings of the fifth international workshop on Mobile cloud computing & services*, pages 7–14. ACM, 2014.

[96] Li Erran Li, Zhuoqing Morley Mao, and Jennifer Rexford. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 7–12. IEEE, 2012.

[97] Shuai Li, Mike Schliep, and Nick Hopper. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 163–172. ACM, 2014.

[98] Xin Jin1 Li Erran Li, Laurent Vanbever, and Jennifer Rexford. Cellsdn: Software-defined cellular core networks. *Open Networking Summit*, 2013.

[99] Wei Liu, Daoli Huang, and Leyuan Zhang. Analysis of network user behavior. In *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on*, pages 126–129. IEEE, 2010.

[100] Kasia Lorenc and Fritz Nelson. Mobile Device Management: Vendors and Comparison Guide. http://www.tomsitpro.com/articles/mdm-vendorcomparison,2-681.html, June 2014.

[101] Linux man pages. Racoon IKEv2 Daemon. http://linux.die.net/man/8/racoon. Online; accessed 12 November 2015.

[102] Jeroen Massar, Ian Mason, Linda Briesemeister, and Vinod Yegneswaran. JumpBox–A Seamless Browser Proxy for Tor Pluggable Transports. In *International Conference on Security and Privacy in Communication Networks*, pages 563–581. Springer, 2014.

[103] Douglas Maughan and Mark Schneider. Internet security association and key management protocol (ISAKMP). *RFC2408*, 1998.

[104] 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.

[105] Deepankar Medhi. *Network routing: algorithms, protocols, and architectures*. Morgan Kaufmann, 2010.

[106] Meraki. Meraki enterprise cloud management. https://meraki.cisco.com/products/wireless/enterprise-license. Online; accessed 29 June 2014.

[107] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012.

[108] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Network Measurement*, pages 41–54. Springer, 2005.

[109] Henrique Moura, Gabriel V.C. Bessa, Marcos A.M. Vieira, and Daniel F. Macedo. Ethanol: Software Defined Networking for 802.11 Wireless Networks. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 388–396. IEEE, 2015.

[110] Rohan Murty, Jitendra Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill. Designing high performance enterprise Wi-Fi networks. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 73–88. USENIX Association, 2008.

[111] Rohan Murty, Jitendra Padhye, Alec Wolman, and Matt Welsh. Dyson: An Architecture for Extensible Wireless LANs. In *USENIX Annual Technical Conference*, 2010.

[112] Thomas D. Nadeau and Ken Gray. *SDN: software defined networks*. O'Reilly Media, Inc., 2013.

[113] Kiyohide Nakauchi, Yozo Shoji, and Nozomu Nishinaga. Airtime-based resource control in wireless LANs for wireless network virtualization. In *Ubiquitous and Future Networks (ICUFN), 2012 Fourth International Conference on*, pages 166–169. IEEE, 2012.

[114] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332. ACM, 2010.

[115] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 11–18. ACM, 2009.

[116] Thuy TT Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world IP networks. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 369–376. IEEE, 2006.

[117] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008.

[118] Shahriar Nirjon, Angela Nicoara, Cheng-Hsin Hsu, Jatinder Singh, and John Stankovic. MultiNets: Policy Oriented Real-Time Switching of Wireless Interfaces on Mobile Devices. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 251–260. IEEE Computer Society, 2012.

[119] Bruno AA Nunes, Manoel Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3): 1617–1634, 2014.

[120] Nathalie Omnes, Marc Bouillon, Gael Fromentoux, and Olivier Le Grand. A programmable and virtualized network & it infrastructure for the internet of things: How can nfv & sdn help for facing the upcoming challenges. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 64–69. IEEE, 2015.

[121] Machigar Ongtang, Stephen McLaughlin, William Enck, and Patrick McDaniel. Semantically rich application-centric security in android. *Security and Communication Networks*, 5(6):658–673, 2012.

[122] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[123] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 99–110. ACM, 2007.

[124] Byungchul Park, James Won-Ki Hong, and Young J Won. Toward fine-grained traffic classification. *Communications Magazine, IEEE*, 49(7):104–111, 2011.

[125] Byungchul Park, Youngjoon Won, JaeYoon Chung, Myung-sup Kim, and James Won-Ki Hong. Fine-grained traffic classification based on functional separation. *International Journal of Network Management*, 23(5):350–381, 2013.

[126] Ashish Patro and Suman Banerjee. COAP: a software-defined approach for home WLAN management through an open API. *ACM SIGMOBILE Mobile Computing and Communications Review*, 18(3):32–40, 2015.

[127] William R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11 (3):635–650, 1991.

[128] Kostas Pentikousis, Yan Wang, and Weihua Hu. Mobileflow: Toward software-defined mobile networks. *Communications Magazine, IEEE*, 51(7):44–53, 2013.

[129] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[130] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, and Pravin Shelar. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, 2015.

[131] Gajen Piraisoody, Changcheng Huang, Biswajit Nandy, and Nabil Seddigh. Classification of applications in HTTP tunnels. In *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*, pages 67–74. IEEE, 2013.

[132] Zafar Ayyub Qazi, Jeongkeun Lee, Tao Jin, Gowtham Bellala, Manfred Arndt, and Guevara Noubir. Application-awareness in SDN. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 487–488. ACM, 2013.

[133] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[134] Shravan Rayanchu, Ashish Patro, and Suman Banerjee. Catching whales and minnows using WiFiNet: deconstructing non-WiFi interference using WiFi hardware. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 5–5. USENIX Association, 2012.

[135] RuleQuest Research. C5.0 Open Source Tool. `http://www.rulequest.com/see5-info.html`, 2008.

[136] research2guidance. Research 2 Guidance. `http://research2guidance.com/`. Online; accessed 28 February 2014.

[137] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 190–201. ACM, 2006.

[138] Injong Rhee, Ajit Warrier, Mahesh Aia, Jeongki Min, and Mihail L Sichitiu. Z-MAC: a hybrid MAC for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(3):511–524, 2008.

[139] Dario Rossi and Silvio Valenti. Fine-grained traffic classification with netflow data. In *Proceedings of the 6th international wireless communications and mobile computing conference*, pages 479–483. ACM, 2010.

[140] Eric Rozner, Yogita Mehta, Aditya Akella, and Lili Qiu. Traffic-aware channel assignment in wireless LANs. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(2):43–44, 2007.

[141] Kshira Sagar Sahoo, Bibhudatta Sahoo, and Abinas Panda. A secured SDN framework for IoT. *International Conference on Man and Machine Interfacing (MAMI)*, December 2015.

[142] Ola Salman, Imad Elhajj, Ayman Kayssi, and Ali Chehab. Edge computing enabling the internet of things. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 603–608. IEEE, 2015.

[143] Samsung. Samsung Enterprise. `http://www.samsung.com/us/business/samsungfor-enterprise/index.html`, . Online; accessed 12 February 2016.

[144] Samsung. Samsung KNOX. `http://www.samsung.com/global/business/mobile/platform/mobile-platform/knox/`, . Online; accessed 12 February 2016.

[145] Mahadev Satyanarayanan. Cloudlets: At the leading edge of cloud-mobile convergence. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 1–2. ACM, 2013.

[146] Julius Schulz-Zander, Lalith Suresh, Nadi Sarrar, Anja Feldmann, Thomas Hühn, and Ruben Merz. Programmatic orchestration of WiFi networks. In *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, pages 347–358. USENIX Association, 2014.

[147] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann. OpenSDWN: Programmatic control over home and enterprise WiFi. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 16. ACM, 2015.

[148] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. Your Installed Apps Reveal Your Gender and More! In *Proceedings of the ACM MobiCom Workshop on Security and Privacy in Mobile Environments*, SPME '14. ACM.

[149] Sakir Sezer, Sandra Scott-Hayward, Pushpinder-Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for SDN? Implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

[150] Scott Shenker, M. Casado, Teemu Koponen, and N. McKeown. The future of networking, and the past of protocols. *Open Networking Summit*, 2011.

[151] Cong Shi, Kaustubh Joshi, Rajesh K Panta, Mostafa H Ammar, and Ellen W Zegura. CoAST: collaborative application-aware scheduling of last-mile cellular traffic. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 245–258. ACM, 2014.

[152] Vivek Shrivastava, Nabeel Ahmed, Shravan Rayanchu, Suman Banerjee, Srinivasan Keshav, Konstantina Papagiannaki, and Arunesh Mishra. CENTAUR: realizing the full potential of centralized wlans through a hybrid data path. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 297–308. ACM, 2009.

[153] Vivek Shrivastava, Nabeel Ahmed, Shravan Rayanchu, Suman Banerjee, Srinivasan Keshav, Konstantina Papagiannaki, and Arunesh Mishra. CENTAUR: realizing the full potential of centralized wlans through a hybrid data path. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 297–308. ACM, 2009.

[154] Vivek Shrivastava, Shravan Rayanchu, Suman Banerjee, and Konstantina Papagiannaki. PIE in the sky: online passive interference estimation for enterprise WLANs. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 337–350. USENIX Association, 2011.

[155] Paul Skelton. OpenFlow-enabling the wireless LAN can bring new levels of agility. *Electrical Connection*, 2014.

[156] Gregory Smith, Anmol Chaturvedi, Arunesh Mishra, and Suman Banerjee. Wireless virtualization on commodity 802.11 hardware. In *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 75–82. ACM, 2007.

[157] The statistics portal. Number of newly developed applications/games submitted for release to the iTunes App Store from 2012 to 2016. `http://www.statista.com/statistics/258160/number-of-new-apps-submitted-to-the-itunes-store-per-month/`, 2016.

[158] StrongSwan. StrongSwan: the OpenSource IPsec-based VPN Solution. `https://www.strongswan.org/`. Online; accessed 12 November 2015.

[159] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, and Anja Feldmann. Demo: programming enterprise WLANs with odin. *ACM SIGCOMM Computer Communication Review*, 42(4):279–280, 2012.

[160] 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. ACM, 2012.

[161] 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. ACM, 2012.

[162] Riverbed Technology. Riverbed Steelhead Mobile. `http://media-cms.riverbed.com/documents/ProductBrief-Riverbed-SHM.pdf`, 2012.

[163] Technology News. Facing data caps, consumers keep turning to Wi-Fi. `http://msoftnews.com/google/facing-data-caps-consumers-keep-turning-to-wi-fi/`, June 2011.

[164] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The TCP/UDP bandwidth measurement tool. *http://dast.nlanr.net/Projects*, 2005.

[165] University of California Irvine Forum. UC SDN Use Case – Automating QoS. http://www.ucif.org/Portals/0/documents/2014_02_27_Use_Case.pdf, February 2014.

[166] Alexander V. Uskov. Information security of ipsec-based mobile vpn: authentication and encryption algorithms performance. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1042–1048. IEEE, 2012.

[167] Ángel Leonardo Valdivieso Caraguay, Alberto Benito Peral, Lorena Isabel Barona López, and Luis Javier García Villalba. SDN: evolution and opportunities in the development IoT applications. *International Journal of Distributed Sensor Networks*, 2014, 2014.

[168] C.P Vandana. Security improvement in IoT based on Software Defined Networking (SDN). *International Journal of Science, Engineering and Technology Research*, 5 (1), January 2016.

[169] Luis M. Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[170] Jonathan Vestin, Peter Dely, Andreas Kassler, Nico Bayer, Hans Einsiedler, and Christoph Peylo. Cloudmac: towards software defined wlans. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(4):42–45, 2013.

[171] Open vSwitch. Production Quality, Multilayer Open Virtual Switch. http://openvswitch.org/. Online; accessed 29 July 2013.

[172] Qiyan Wang, Xun Gong, Giang TK Nguyen, Amir Houmansadr, and Nikita Borisov. CensorSpoofer: Asymmetric Communication with IP Spoofing for Censorship-Resistant Web Browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 121–132. ACM, 2012.

[173] Richard Wang, Dana Butnariu, Jennifer Rexford, et al. Openflow-based server load balancing gone wild. *Hot-ICE*, 11:12–12, 2011.

[174] Xueqiang Wang, Kun Sun, Yuewu Wang, and Jiwu Jing. DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices. In *Proceedings of the 22th Network and Distributed Security Symposium*, 2015.

[175] Yu Wang and Shun-Zheng Yu. Machine learned real-time traffic classifiers. In *Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on*, volume 3, pages 449–454. IEEE, 2008.

[176] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120. ACM, 2012.

[177] Wikipedia. Mobile Device Management. `https://en.wikipedia.org/wiki/Mobile_device_manag` Online; accessed 29 December 2015.

[178] Brandon Wiley. Dust: A blocking-resistant internet transport protocol. *Technical report*, 2011.

[179] Nigel Williams and Sebastian Zander. Real time traffic classification and prioritisation on a home router using DIFFUSE. Technical report, Swinburne University of Technology, Australia, 2011.

[180] Philipp Winter, Tobias Pulls, and Juergen Fuss. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224. ACM, 2013.

[181] Nielsen Wire. Americas new mobile majority: A look at smartphone owners in the US. `http://www.nielsen.com/us/en/insights/news/2012/who-owns-smartphones-in-the-us.html`, 2012.

[182] Chiachih Wu, Yajin Zhou, Kunal Patel, Zhenkai Liang, and Xuxian Jiang. AirBag: Boosting Smartphone Resistance to Malware Infection. In *Proceedings of the 21th Network and Distributed Security Symposium*, 2014.

[183] Bo Yan and Guanling Chen. AppJoy: personalized mobile application discovery. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2011.

[184] Lily Yang, Petros Zerfos, and Emek Sadot. Architecture Taxonomy for Control and Provisioning of Wireless Access Points (CAPWAP), RFC 4118. 2005.

[185] Mao Yang, Yong Li, Depeng Jin, Li Su, Shaowu Ma, and Lieguang Zeng. Openran: a software-defined ran architecture via virtualization. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 549–550. ACM, 2013.

[186] 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. *Mobile Networks and Applications*, 20(1):4–18, 2015.

[187] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. Openroads: Empowering research in mobile networks. *ACM SIGCOMM Computer Communication Review*, 40(1):125–126, 2010.

[188] Kok-Kiong Yap, Rob Sherwood, Masayoshi Kobayashi, Te-Yuan Huang, Michael Chan, Nikhil Handigol, Nick McKeown, and Guru Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 25–32. ACM, 2010.

[189] Kok-Kiong Yap, Yiannis Yiakoumis, Masayoshi Kobayashi, Sachin Katti, Guru Parulkar, and Nick McKeown. Separating authentication, access and accounting: A case study with OpenWiFi. *Open Networking Foundation, Tech. Rep*, 2011.

[190] Kok-Kiong Yap, Te-Yuan Huang, Masayoshi Kobayashi, Yiannis Yiakoumis, Nick McKeown, Sachin Katti, and Guru Parulkar. Making use of all the networks around us: a case study in android. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 19–24. ACM, 2012.

[191] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE, 2015.

[192] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM, 2015.

[193] Shanhe Yi, Zhengrui Qin, and Qun Li. Security and privacy issues of fog computing: A survey. In *Wireless Algorithms, Systems, and Applications*, pages 685–695. Springer, 2015.

[194] Yiannis Yiakoumis, Kok-Kiong Yap, Sachin Katti, Guru Parulkar, and Nick McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, pages 1–6. ACM, 2011.

[195] Yiannis Yiakoumis, Sachin Katti, Te-Yuan Huang, Nick McKeown, Kok-Kiong Yap, and Ramesh Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1114–1119. ACM, 2012.

[196] Heng Yin and Haining Wang. Building an application-aware IPsec policy system. *IEEE/ACM Transactions on Networking (TON)*, 15(6):1502–1513, 2007.

[197] Sung-Ho Yoon, Jun-Sang Park, and Myung-Sup Kim. Behavior signature for fine-grained traffic identification. *Appl. Math*, 9(2L):523–534, 2015.

[198] Fan Zhang, Wenbo He, Xue Liu, and Patrick G Bridges. Inferring users' online activities through traffic analysis. In *Proceedings of the fourth ACM conference on Wireless network security*, pages 59–70. ACM, 2011.

[199] Fan Zhang, Wenbo He, Yangyi Chen, Zhou Li, XiaoFeng Wang, Shuo Chen, and Xue Liu. Thwarting Wi-Fi Side-Channel Analysis through Traffic Demultiplexing. *IEEE Transactions on Wireless Communications*, 1(13):86–98, 2014.

[200] Min Zhang, Wolfgang John, KC Claffy, and Nevil Brownlee. State of the art in traffic classification: A research review. In *PAM Student Workshop*, pages 3–4, 2009.

[201] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. Sweet: Serving the web by exploiting email tunnels. *arXiv preprint arXiv:1211.3191*, 2012.

APPENDIX A

PUBLISHED/UNDER REVIEW WORK

A.1 PAPERS

1. *Mostafa Uddin, and Tamer Nadeem*, "SafeEnd: An Application-Aware Programmable Network Security Solution for Mobile Devices." (In review)

2. *Mostafa Uddin, and Tamer Nadeem*, "SmartEdge: Toward Making Wireless Network Edges Traffic-Aware." (in review)

3. *Mostafa Uddin, Gowtham Bellala, Jeongkeun Lee, and Tamer Nadeem*, "TrafficVision: A Case Scenario of Pushing SDN to Wireless Edges." (in review)

4. *Ibrahim Ben Mustafa, Mostafa Uddin, and Tamer Nadeem*, "Understanding the Intermittent Traffic Pattern of HTTP Video Streaming over Wireless Networks." WINMEE 2016

5. *Mostafa Uddin, Ahmed Salem, Ilho Nam, and Tamer Nadeem*, "Wearable Sensing Framework for Human Activity Monitoring." ACM WearSys'15

6. *Mostafa Uddin, and Tamer Nadeem*, "Harmony: Content Resolution using Acoustic Channel." IEEE INFOCOM 2015.

7. *Jeongkeun Lee, Mostafa Uddin, JeanTourrilhes, Souvik Sen, Sujata Banerjee, Manfred Arndt, Kyu-Han Kim, Tamer Nadeem*, "meSDN: mobile extension of SDN." ACM MCS 2014.

8. *Mostafa Uddin, and Tamer Nadeem*, "SpyLoc: A Light Weight Localization System for Smartphones." IEEE SECON 2014.

9. *Mostafa Uddin, Ajay Gupta, Kurt Maly, Tamer Nadeem, Sandip Godambe, Arno Zaritsky,* "SmartSpaghetti: Accurate and Robust Tracking of Human's Location." IEEE-EMBS International Conferences on Biomedical and Health Informatics, 2014.

10. *Mostafa Uddin, Ajay Gupta, Kurt Maly, Tamer Nadeem, Sandip Godambe, Arno Zaritsky,* ” SmartSpaghetti: Use of Smart Devices to Solve Health Care Problems.” International Workshop on Biomedical and Health Informatics, 2013.

11. *Mostafa Uddin, and Tamer Nadeem*, ”RF-Beep: A light ranging scheme for smart devices.” IEEE PerCom 2013.

12. *Mostafa Uddin, and Tamer Nadeem*, ”A2PSM: Audio Assisted Wi-Fi Power Saving Mechanism for Smart Devices.” ACM HotMobile 2013.

13. *Mostafa Uddin, and Tamer Nadeem*, ”MagnoTricorder: What You Need To Do Before Leaving Home.” ACM HomeSys, UbiComp 2012.

14. *Mostafa Uddin, and Tamer Nadeem*, ”EnergySniffer: Home Energy Monitoring System using Smart Phones.” IEEE IWCMC, 2012.

A.2 ARTICLE

1. *Igor Pernek, Mostafa Uddin and Jack Fernando Bravo Torres,* ”Report of HotMobile 2012” IEEE Pervasive Computing.

2. *Mostafa Uddin and Tamer Nadeem,* ”HotMobile 2012 Poster: MachineSense: Detecting and Monitoring Active Machines using Smart Phone.” ACM SIGMOBILE MC2R.

3. *Mostafa Uddin and Tamer Nadeem,* ”HotMobile 2012 Poster: Audio-WiFi: Audio Channel Assisted WiFi Network for Smart Phones.” ACM SIGMOBILE MC2R.

A.3 POSTER/DEMOS

1. *Mostafa Uddin, Ashish Kshirsagar, and Tamer Nadeem*, ”SafeWLAN: A WLAN-based SDN Approach for Securing WLAN Traffic.” ACM HotMobile 2015.

2. *Mostafa Uddin and Tamer Nadeem*, ”SpyLoc: a Light Weight Localization System for Smartphones.” In Proceedings of MobiCom’13.

3. *Jeongkeun Lee, Mostafa Uddin, Jean Tourrilhes, Souvik Sen, Sujata Banerjee,Manfred Arndt, Tamer Nadeem,* ”Extending SDN for mobile device.” ACM HotMobile, 2014 .

4. *Mostafa Uddin and Tamer Nadeem*, ”Audio-WiFi: Audio Channel Assisted WiFi Network for Smart Phones.” IEEE INFOCOM, 2012 .

5. *Mostafa Uddin and Tamer Nadeem*, ”EnergySniffer: Home Energy Monitoring System using Smart Phones.” IEEE INFOCOM, 2012 .

6. *Mostafa Uddin and Tamer Nadeem*, ”MachineSense: Detecting and Monitoring Active Machines using Smart Phones.” ACM HotMobile, 2012 .

VITA

Mostafa Uddin
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Education:

Old Dominion University, Norfolk, VA, USA
PhD in Computer Science (2011 - present)
Advisor: Dr. Tamer Nadeem (nadeem@cs.odu.edu)
Dissertation Topic: Toward Open and Programmable Wireless Network Edge
Dissertation Committee: Prof. Mahadev Satyanarayanan (CMU), Prof. Kurt Maly (ODU), Prof. ChunSheng Xin and Prof. Michele Weigle (ODU)
CGPA 3.98/4.0
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
B.S. in Computer Science and Engineering, 2006
CGPA 3.72/4.0

Industrial Experiences:

Bell Lab Researcher - intern, Bell Labs, Murray Hill, NJ **Fall 2015-present**
Advisor: **Randeem Bhatia**
Research Associate Intern, HP Labs, Palo Alto, CA **July,2014 - August,2014**
Mentor: **Kyu-Han KIM**, Senior Researcher and Research Manager
Research Associate Intern, HP Labs, Palo Alto, CA **May,2013 - August,2013**
Mentor: **Jeongkeun Lee**, Senior Research Scientist

Patents and Invention Disclosures:

Jung Gun Lee, Mostafa Abdulla Zahid Uddin, Jean Tourrilhes, Souvik Sen, Manfred R Arndt. ”Wireless Software-Defined Networking”, Publication number WO2015065422 A1, Publication date May 7, 2015.
Mostafa Uddin, Tamer Nadeem. ”SMILE – Towards Smarter Network Edges for Next Generation Networks”, Filling Date Oct, 2015.