OPENFLOW BASED TRAFFIC ENGINEERING FOR MOBILE DEVICES

A Dissertation by

Radhika Mysore Hirannaiah

Master of Science, Wichita State University, 2006

Bachelor of Engineering, Bangalore University, 2001

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

July 2014

OPENFLOW BASED TRAFFIC ENGINEERING FOR MOBILE DEVICES

The following faculty members have examined the final copy of this dissertation for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy with a major in Electrical Engineering.

_____
John M. Watkins, Committee Chair

_____
Ravi Pendse, Committee Member

_____
M. Edwin Sawan, Committee Member

_____
Rajiv Bagai, Committee Member

_____
Krishna K. Krishnan, Committee Member

Accepted for the College of Engineering

_____
Royce Bowden, Dean

Accepted for the Graduate School

_____
Abu S. M. Masud, Interim Dean

DEDICATION

To my mother and father,
who have always been my strength and inspiration

The only thing greater than the power of the mind is the courage of the heart.

—John Nash

ACKNOWLEDGEMENTS

The journey of obtaining my doctorate degree has taught me three virtues: **p**erseverance in times of difficulty, **h**umility with gain of knowledge and en**d**urance with every challenge. Every test I have faced has made me more motivated and stronger to pursue this dream. Foremost, I would like to thank my parents—my mother late Mrs. Rathna Hirannaiah and my father Mr. M. S. Hirannaiah—for being my strength and inspiration. I am forever indebted and grateful for their sacrifices in support of all my endeavors.

This dissertation would have not been possible without the constant guidance and encouragement of many people. First, I am sincerely thankful to my advisors, Dr. Ravi Pendse and Dr. John M. Watkins, for their direction and assistance. Dr. Pendse always counseled and encouraged me throughout my Ph.D. program, on both professional and personal level. His guidance has been invaluable. Dr. Watkins listened and governed my deliverables with patience. He provided me guidance and encouraged me in my work.

I express my gratitude to Dr. M. Edwin Sawan for his support in my research and publications. Despite his busy schedule, he found time to provide me invaluable insights that enabled me to meet compressed timelines. I also thank my other committee members, Dr. Rajiv Bagai and Dr. Krishna K. Krishnan, for taking time to review this document and provide useful comments and suggestions.

I acknowledge American Portwell Technology Inc. for providing the Network Security Appliance used to conduct my experiments and test my research work. I thank the early researchers and implementers of OpenFlow and related specifications contributed to my research on this open source model (Open vSwitch, FlowForwarding.Org's LINC Switch, and CPqD).

I am thankful to my brother Shivaram and sister Veena for their constant love and encouragement. They have always supported and guided me.

I thank my adopted family in Wichita, Dr. Mark and Mrs. Kelly Huenergardt, who have taken me into their family and have been my support here.

Lastly, I thank all my friends who have supported and encouraged me on this journey.

ABSTRACT

Expansion in mobile devices has led to rapid growth in data traffic. Currently, networks are complex and difficult to manage. The exponential increase in audio/video streaming and continuous demand for network connectivity has necessitated the need for traffic steering and path management. Improving seamless network connectivity with reduced power consumption has been a constant goal in the development of mobile devices. The ability to configure quality of service (QoS) levels for applications on mobile devices is primarily driven by service providers and other players such as users and enterprises who have little or no way of configuring QoS. In the OpenFlow networking approach, having a separate control and data plane offers different levels of service.

This research presents an architectural model to integrate the OpenFlow switch on a mobile device. The work here provides an algorithm design and methodology for using multiple controllers to act on a single OpenFlow switch installed on a mobile device while conforming to the OpenFlow standard. By associating a controller with entities such as a service provider, enterprise, and/or user, the switch on a mobile device is programmed for flows in a mutually exclusive way. The pattern of a pipeline processing scheme is intended to enable different paths driven by flows in various tables. The design solution allows enhanced traffic engineering to accommodate reduced latency, increased QoS, and a flexible model to enable users and enterprises to set their own traffic requirements. By incorporating flows related to network services instead of individual network-related programs on a mobile device, this model may reduce the number of components and thereby power consumption. With the proposed methods, it satisfies an infrastructural need that benefits mobile devices across different players in the process.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

NOMENCLATURE

API	Application Programmer Interface
BDD	Binary Decision Diagram
CDMA	Code Division Multiple Access
CRAN	Comprehensive R Archive Network
DHCP	Domain Host Configuration Protocol
DSP	Digital Signal Processor
DTMC	Discrete Time Markov Chain
FPGA	Field Programmable Gate Array
GPL	General Public License
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IP	Internet Protocol
L2	Layer 2
L3	Layer 3
LTE	Long Term Evolution
MAC	Media Access Control
MLE	Maximum Likelihood Estimator
NAT	Network Address Translation
OF	OpenFlow
PSTN	Public Switched Telephone Network
QoS	Quality of Service

RTSP	Real-Time Streaming Protocol
SDN	Software Defined Network
TCP	Transmission Control Protocol
TLS/SSL	Transport Layer Security/Secure Socket Layer
VoIP	Voice of Internet Protocol
VPN	Virtual Private Network
WAN	Wide Area Network
2G	Second Generation
3G	Third Generation
4G	Fourth Generation

CHAPTER 1

INTRODUCTION

1.1 Overview

Mobile traffic has increased exponentially in the last few years [1]. Mobile devices are untethered by a network connection, such as laptops, smart phones, and tablets. Mobile connectivity options include Wi-Fi, WiMAX, Bluetooth, and cellular (global system for mobile communication [GSM], third generation [3G], fourth generation [4G], long term evolution [LTE], code division multiple access [CDMA] etc.). These options have enhanced functionality to support various scenarios across devices. Sustaining numerous devices while providing connectivity selections challenges service providers to build and maintain an accommodating network infrastructure, in addition to providing enhanced services to end users, at equal or lower costs. With new-generation devices, consumers and enterprises expect superior functionality in the way of applications and connectivity options. Device and application management have increased the complexity with varied security implications. This "management" problem is across the board, from service providers to enterprises to end users.

1.2 Motivation

Mobile devices imply voice, video, and data traffic. The focus of this research is on Internet protocol (IP)-based services using various connectivity mechanisms across all mobile devices. In addition, because the device connects across heterogeneous networks, the network infrastructure to transmit and/or receive may be evaluated for both cost efficiency and quality of service (QoS).

Connectivity refers to linking up the sender and the receiver. Voice connectivity started as a manual function managed by a person working as a switch board operator [2], connecting

the sender and the receiver via a physical "switching" process. Then, connectivity transitioned to an automated electronic device called a "switch," replacing human effort and encouraging seamless connectivity, which was both cost effective and efficient. The technique of connecting two end points via a direct channel is called "circuit switching." Telephones function primarily on this concept.

With the onset of computer technology, communication took place through data exchange. Data communication was not as stringent as voice communication, easily supported by distributing data into multiple packets of similar size, and delivered over the wire as "packets." The Internet enabled these packets to be delivered easily. The Internet uses a "packet-switched network" infrastructure to connect senders and receivers.

Mobile devices use both circuit (voice) - and packet (data)-switching techniques. The use of voice over a packet-switched network has led to the development of the voice over IP (VoIP), dramatically minimizing infrastructure and delivery costs for everyone involved (service providers and end users). IP data networks that have replaced specialized voice, video, and control networks are referred to as "horizontal convergence." Converging-packet and circuit-switched networks, also known as "vertical convergence," benefit a network operator with maximum flexibility by utilizing both technologies in designing and operating their networks. OpenFlow, a software defined network (SDN) approach, combines both types of networks, thus providing interoperability and better performance.

With the rapid growth in data traffic, intelligent and efficient traffic steering and management is needed. The ability to monitor and manage traffic across different technology tools from multiple vendors must be considered. Presently, routers and switches offer limited user programmability. A traditional switched network handles both fast-path forwarding (data

path) and high-level routing (control path). OpenFlow allows the control function to be independent of the hardware it administers, which speeds up the forwarding and routing process [3]. By separating the switch's data-path and control-path activities, data path functionality still resides in the switch, and the controller handles only high-level routing decisions.

The OpenFlow-based SDN enables the optimization of path management based on individual service needs rather than routing options. This is critical in a mobile environment where end users are not location bound. Bandwidth and connectivity options influence content delivery and service quality, which has a direct impact on the mobile device's power consumption. For example, downloading a piece of data over a wireless device can be more power efficient than downloading the same data over a cellular network because the access point is quite a distance away from the mobile device. However, the cellular network can provide better connectivity in comparison to a wireless device, especially when the user is mobile.

Circuit switching is more scalable than packet switching by being able to exchange at higher data rates and consume less power. However, unlike packet switching, circuit switching does not provide statistical multiplexing. This dissertation aims to integrate circuit-switching-based techniques for data delivery as an overlay technique on existing packet-switched network services. This is accomplished by incorporating an OpenFlow switch on a mobile device. The switch installed in access mode on a mobile device is dynamically programmed by the various entities in a mutually exclusive way, thus providing intelligent traffic management. Based on OpenFlow standards, the OpenFlow switch can be programmed through flows and prioritize different types of traffic by developing policies for handling distinct network control functions.

1.3 Thesis Goals

This dissertation focuses on integrating circuit-switching-based techniques for data delivery as an overlay technique on existing packet-switched network services using the OpenFlow protocol. The goals of this dissertation are as follows:

- Develop an architectural model to integrate circuit-switching-based techniques for data delivery as an overlay technique to existing packet-switched network services.
- Identify infrastructure needs for mobile devices and service providers to support an architectural model as delivered by the previous goal.
- Identify the benefits for service providers, enterprises, and consumers using this approach.

1.4 Dissertation Outline

This dissertation is organized into seven chapters. This chapter provided the motivation and the goals for the research work. Chapter 2 provides a literature review. Chapter 3 describes the technology background of OpenFlow. Chapter 4 details the concept of an architecture model. Chapter 5 provides techniques to model multiple controllers with a single OpenFlow switch. Chapter 6 presents the proof of the concept with a demonstration and a simulation. Chapter 7 summarizes the research and highlights opportunities for future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Today, mobile device applications use IP based services over cellular and wireless networks for data. As mentioned previously in section 1.2, these IP based services use a packet-switching technology. The appeal of a portable device is its "mobility." Network support is location-specific, and with the movement of devices across locations, devices need to reestablish network connectivity constantly. Consumers on the go expect consistent, high quality connectivity at all times. To enable seamless service to the end user, the "handoff," i.e., moving the connection from one access point to another must be smooth and non-intrusive to the user. In addition, networks need to make the handoff decision suboptimal, incurring overhead for both the application and the infrastructure when performing necessary handoffs [4].

Not all mobile devices or access points are similar. Each provides varied functionality. For example, mobile devices and likewise may support only second generation (2G), third generation (3G), and/or even fourth generation (4G) connectivity options. When a mobile device enters a new access point zone, a negotiation between the device and access point occurs in order to determine the best connectivity option. For example, if the device supports 2G, 3G, and 4G connectivity but the access point only supports 2G connectivity, then 2G is selected for the connectivity. This selection of connectivity option has a significant impact in choosing the appropriate codec buffers for both data and voice transmission. In the case of voice, these buffers are referred to as "jitter buffers." With the IP running on a packet-switched network, the need to change codecs/jitter buffers can be profound. In summary, every time there is a change in connectivity parameter, there is a corresponding evaluation to possibly change the codec.

2.2 Adaptive Jitter Buffer

To minimize the conflicting aspects of playout buffer time and delayed packet loss, an adaptive jitter buffer model was proposed to enhance the change of audio codecs dynamically [5]. In real time, providing superior voice quality with reduced overhead and enhanced services is a priority on packet networks. During network congestion, voice packets are delayed due to queuing and routing in the intermediate nodes. To attain improved quality of voice connection and to maintain connectivity at all times, the authors proposed a jitter buffer model that changes the audio codecs from a higher bit rate to a lower bit rate during an established call session, thereby reducing packet loss. The performance was evaluated based on voice quality, packet loss, bandwidth utilization, and interarrival delays. The simulation model demonstrated the manipulation of both jitter buffer and codec, which was changed from G.711 to G.723. By changing the codecs, the adaptive nature of the jitter buffer was maintained by checking the size of the buffer when the packet reduced the loss by 30%. Rapidly changing codecs could lead to transmission errors, unexpected end-to-end delays, and poor quality of service, which deteriorates call performance. Adaptive codecs were considered when the buffer reached the threshold, thus not affecting the performance of the call during an established session.

2.3 OpenRadio

In the work of Bansal et al. [6], a novel design for a programmable data plane "OpenRadio" was proposed. The authors proposed the design based on two aspects: (1) a separate wireless protocol from its hardware ensuring the underlying platforms used for implementing the protocols, and (2) the design of a software abstraction layer representing a modular and declarative interface to program those wireless protocols.

Due to the rapid evolution of wireless protocols, updating them and continuously optimizing the network by vendors and operators is extensive and challenging. Dynamically adjusting the spectrum and power allocation to implement management mechanisms at nearby base stations is difficult because it is necessary to ensure that the mobile handset in the intra-cell region is unaffected. To benefit rapidly changing protocols and optimization, the current model was proposed. The main idea here was to reprogram the wireless protocols into processing and decision planes to hide all underlying execution complexity. This paper explored the data plane design essential to provide both performance and flexibility. With an OpenRadio-based programmable wireless network data plane, service providers are able to remotely program the base stations, and upgrade and optimize the network completely through software. The OpenRadio software abstraction layer exposes the modular and declarative interface in order to program the physical and media access control (MAC) layers. It can be implemented on a variety of platforms including digital signal processors (DSPs), field programmable gate arrays (FPGA), and multi-core desktop machines. The processing plane includes data-manipulation on packet data, called “actions.” The decision plane includes “rules” where all operation decisions to transform the data stream are applied. The programmable state model for the design was proposed in this paper.

In this research, it was found that by overlaying circuit-switched network mechanisms over packet-based wireless/cellular networks for mobile devices, it is possible to reduce and simplify the deployment of adaptive jitter buffer software stacks. Also extend the OpenRadio programming model to the end mobile device, and deliver even more flexibility while enhancing the use of a deployed network infrastructure.

2.4 Controllers

In a conventional architecture, a switch or a router decides those actions that need to be executed. The main objective of an OpenFlow controller is to centralize network capability, while maintaining a distributed forwarding plane through OpenFlow switches and routers. A controller allows specific flows to be traced and managed in a flexible approach, based on packet header information. It controls the data plane over a secure channel using the OpenFlow protocol. Due to separation of the control and data plane, network researchers have developed proprietary algorithms to control the data path [7].

The initial approach was to design controller platforms to write controller applications in different programming languages. A few of these applications that provide a distributed controller program for large-scale networks are NOX [9], Onix [10], Maestro [11], and Beacon [12], which have a high-level application programmer interface (API) framework for deploying applications on top of the OpenFlow-enabled substrate. The NOX [9] application was proposed as a framework to coordinate and manage advancing technologies in a centralized network state. It emphasizes controllers built of multiple components that communicate over an asynchronous event system and maintains a database of events containing an abstract view of available network components. NOX applications are written using C and Python programming languages. The Onix [10] application presents control platforms as distributed state management APIs at the controller, which is feasible in nature. Control plane implementation allows them to make tradeoffs among consistency, durability, and scalability. Maestro [11] and Beacon [12] provide frameworks for building OpenFlow controllers in production environments. They are developed in Java and aim to provide high performance by exploiting parallelization in modern server architectures. However, all these OpenFlow solutions are based on OpenFlow standard v.1.0 [13]

Another approach was considered to test OpenFlow controllers as an application. NICE [14] is an efficient systematic technique to test unmodified controller programs and performs symbolic execution of event handlers. This controller-verification tool uses model checking and automates the testing of OpenFlow applications. It checks Python applications on the popular NOX platform. NICE attempts to explore the state space of the entire network and find invalid system states. It uses correctness properties provided by developers or testers to specify the correct behavior for the tested software.

Lithium [15] is an event-driven network control that can implement policies expressed in different domains. It supports real time network policies with a wider class of applications to be programmed in a declarative language. It handles changes to network conditions that arise from temporal conditions and changes in network state. It also allows packet processing rules to change over time and in reaction to internal or external network events. Lithium is a high-level design with a finite-state machine model to express and enforce network policies. It supports reactive dynamic event handler processes of network events that arrive at the controller.

2.5 Controllers in Multi-Domain Networks

HyperFlow [16] provides a distributed control plane for OpenFlow in a multi-domain environment. Here, a domain can run multiple controllers simultaneously, and each controller handles only local area OpenFlow switches (multiple switches). HyperFlow interconnects independently managed OpenFlow networks. All controllers have the same consistent network-wide view and locally serve requests without actively contacting any remote node, thus minimizing flow setup times. This control plane adopts a distributed file system to share global information among controllers. The HyperFlow application is implemented on top of the NOX

platform, which synchronizes controllers by propagating events affecting the controller state. It is adaptable to network partitioning and component failures.

The routing in multi-domain OpenFlow networks (RMOF) [17] model also supports distributed controllers in different domains for managing large-scale OpenFlow networks concurrently. This model proposes a mechanism to collaborate with different network domains efficiently by routing via an effective-path computing algorithm. Here, a collaborator is used as a communication channel for controllers from different network domains to cooperate with each other. The RMOF collaborator maintains a global view of the entire network and processes the collected information, providing helpful instructions or networking services for the controllers. Based on OpenFlow standard v.1.3 [18], it provides a routing method to compute shortest-path routes over multiple domains.

2.6 Flow Visor

FlowVisor [19] is a distinct approach using the concept of multiple controllers in an OpenFlow network. It adopts switch virtualization to share multiple controllers with a definite forwarding logic. It provides a vendor independent layer by slicing the network of OpenFlow switches and allowing isolated operation access control of controllers. Each slice is managed by a controller. It uses the concept of “flowspaces” to create network slices so that it can process multiple OpenFlow controllers independently. The FlowVisor layer is completely transparent to both controllers and switches. Also, isolating the slices ensures that flowspaces do not overlap anywhere in the topology. The controllers are completely independent and do not affect each other. The idea of network virtualization is explored here for the effective usage of network resources.

2.7 Alloy Model

Behavior of the OpenFlow switch is studied with the Alloy [20] analyzer, which uses software abstraction to describe the internal state of a network. It is considered in a scenario where multiple switches are managed by only one controller. An Alloy analyzer was proposed using lightweight verification language for OpenFlow switches to analyze its network properties. It depicts the static structure and dynamic behavior of the OpenFlow switch by matching rules and actions in table entries as part of a real size network. It models the state of one switch and its interaction with other nodes in the network. The Alloy analyzer checks against network constraints and translates the model into a Boolean expression, returning with a finite number of objects.

2.8 OpenFlow Switches

All the models proposed regarding OpenFlow switches are based on OpenFlow standard v.1.1.0 [13]. SOFT [21], an OpenFlow agent software, was proposed to test the interoperability testing of OpenFlow switches by finding inconsistencies among the implementations of OpenFlow agents that run in the switches. This software enables remote programmatic access to its forwarding tables. It symbolically executes each agent under a test in isolation to derive the set of inputs that cause the behavior. It automates the task of identifying the deviations in behavior among different switches.

Using a different outlook, FlowChecker [22] and VeriFlow [23] were designed to check for network invariants in multiple OpenFlow switches. FlowChecker [22] tests for any intra-switch misconfigurations within a single flow table. It verifies any OpenFlow configuration conflicts and inconsistencies within a path of multiple OpenFlow switches. It translates flow

table configurations into Boolean expressions using binary decision diagrams (BDDs) and then uses the model-checker technique to interconnect the network of OpenFlow switches.

VeriFlow [23], designed as a layer between the OpenFlow controller and network devices, checks for network-wide violations addressed in the OpenFlow switch. Conflicting rules that cause anomalous network behavior and that are dynamically inserted in each forwarding rule are detected and prevented by debugging. VeriFlow also processes forwarding table updates and adapts network slicing, like FlowVisor, to a set of equivalence classes of packets to confine the verification action. However, VeriFlow is not suitable for a multi-controller scenario. It uses graph search techniques to verify network-wide invariants and handle dynamic changes in real time.

FlowAdapter [24] intends to address the incompatibility of the flow table pipeline process based on OpenFlow standard v.1.1.0 [13]. This hardware abstraction layer (HAL) converts flow entry rules from the controller flow table to the switch-hardware flow table. It can adapt multi-table rules from the controller to different switch-hardware capabilities. It is designed to achieve the conversion of M-multiple flow tables to a one-stage flow table, and a one-stage flow table to N-multiple flow tables via an N-tree data structure representation. The application of the controller can construct and issue rules depending on its own requirement, despite various capabilities of the flow table pipeline in the switch hardware.

This chapter reviewed the adaptive jitter buffer algorithm and OpenRadio design as literature background for this research idea. In summary, using adaptive jitter buffer technology, packet loss can be minimized, but there is a cost of software stacking and tuning. The adaptive jitter buffer software stack needs to be deployed throughout the network, including the end device. The OpenRadio methodology clearly defines a model wherein service providers can

program the network infrastructure dynamically, in order to deliver programmable base stations with flexibility in delivering QoS for applications. Various models designed for OpenFlow controllers and switches were also reviewed in this chapter. Based on prior research work discussed here, the idea of multiple controller methodology is utilized to compliment the OpenRadio concept in the work here. This methodology focuses on programming the service provider network to the last mile but does not include the end-user device. Extending this concept, it is proposed that an OpenFlow switch could be installed on a mobile device to increase network programmability.

CHAPTER 3

TECHNOLOGY BACKGROUND: OPENFLOW

3.1 Introduction

Currently deployed network equipment such as switches, routers, and firewalls perform high-speed packet processing. Packet processing refers to the inspection of packet headers and, in a few cases, payload when a packet enters the device. They are matched up against a "look-up" table located in these devices that provides match-action rules. This look-up table is pre-programmed by the network equipment administrator and is rarely modified. Most network equipment manufactured as such has "static" (i.e., limited to no change of features) functionality, for example, the layer 2 (L2) switch, layer 3 (L3) switch, network firewall, and application firewall.

3.2 OpenFlow

OpenFlow is a protocol used to enable dynamic look-up table programmability wherein match-action rules are constantly updated based on various network requirements, and the look-up table is constantly referred to as the "flow table." A switch that supports OpenFlow is called an OpenFlow switch. During switching, the packet processing function is normally associated with the "data" plane, and the ability to modify the flow table is handled by the "control" plane. In a traditional switch, the control and data plane elements are on the same device, as shown in Figure 1. In an OpenFlow switch, the control and data planes are separated, with the switch only having the data plane. Programmability is offered by the "controller" (or OpenFlow controller) corresponding to the control plane element of the switch. The protocol between the OpenFlow switch and the controller is referred to as the OpenFlow Protocol—a standard defined by the

Open Network Foundation [18]. OpenFlow architecture was introduced and best described in the work of McKeown et al. [25]. Figure 2 shows OpenFlow switch components.

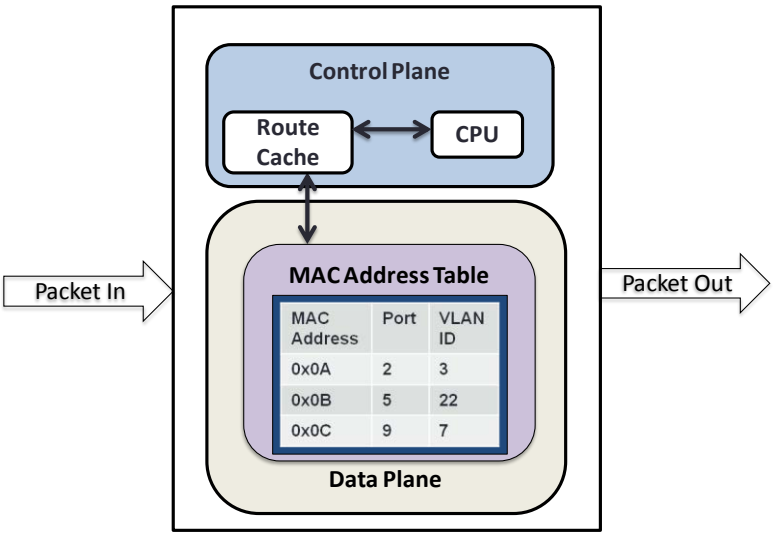

Figure 1. Traditional Switch

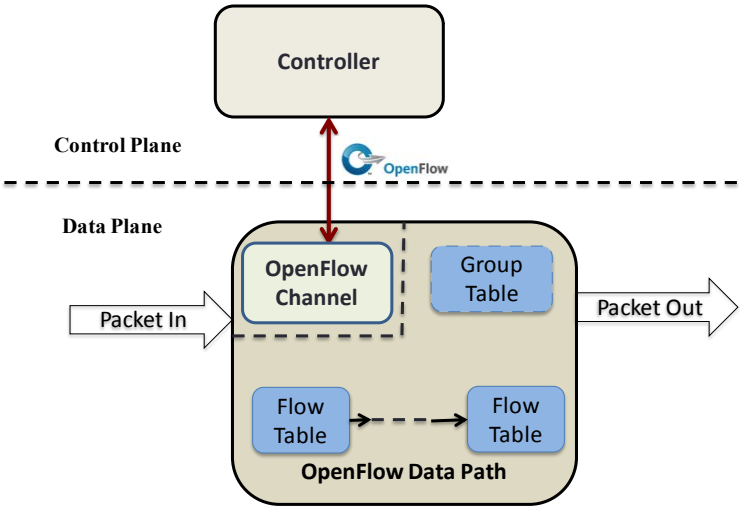

Figure 2. OpenFlow Switch Components

Due to separation of the data and control planes in OpenFlow, the same switch can be dynamically programmed to act as a switch, firewall, or any kind of network control function. This means that, if programmed appropriately, the switch can process any or all (L2, L3 headers, etc.) parts of the packet that flow through it.

3.3 Flow Tables and Group Tables

A traditional switch uses only one look-up table or flow table to make forwarding decisions on an incoming packet. In OpenFlow (version 1.2+)-based switches, multiple flow tables and group tables are supported. Here, the first table (root table), called Table 0, is normally used as a basic match on the packet so that it can be processed in subsequent tables.

OpenFlow switches perform an exact match by inspecting specific fields of an incoming packet. For every incoming packet, the switch goes through its flow table to find a matching flow entry. The flow table entry is identified by its match fields and priority, where it is determined by a unique flow entry identifier in the flow table. If the entry exists, then the switch performs the action associated with that particular flow entry on that packet. If the entry does not exist, then a "table-miss" response is sent to the controller. Subsequently, the process is repeated for all incoming packets. Instructions associated with each flow entry include actions that describe packet forwarding, packet modification, and group table processing. A flow entry can only direct a packet to a flow table number, which is greater than its current flow table number. This introduces the pipeline processing of packets, which can only go in the forward direction. Instructions may explicitly direct the packet to another flow table where the process of matching flow entry is repeated. When the matching flow entry does not direct packets to another flow table, then the pipeline processing is stopped at that table, and the packet is processed with its associated action set. The flow entry that wildcards all the match fields and has a priority equal to zero is known as a table-miss flow entry. Figure 3 shows flow table fields in an OpenFlow switch and are explained below.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Figure 3. Flow Table in OpenFlow Switch

- Match Fields—consists of ingress port and packet header to match against incoming packets.
- Priority—matching precedence of the flow entry.
- Counters—updated when packets are matched; these counters are used to track the number of packets and bytes for each flow, and the time elapsed since the flow initiation.
- Instructions—used to modify the action set or pipeline processing; actions are applied to matching packets, specify the packet flow processing, and can be one of the following:
  - Forward the packet to a given port(s) after optionally rewriting some header fields.
  - Drop the packet.
  - Forward the packet to the controller.
- Timeouts—maximum amount of time, or idle time, before flow is expired by the switch.
- Cookie—opaque data value chosen by the controller to filter flow statistics, flow modifications, and flow deletions; this is not used when processing packets.

Figure 4 explains packet processing with the OpenFlow flow table. Actions associated with flow entries may also direct packets to a group, which can specify additional processing. Additional methods of forwarding in OpenFlow enable the ability for flow entry to be directed to a group. The group table consists of group entries and is identified by its group identifier. The components of a group table, shown in Figure 5 are as follows:

- Group identifier—32-bit unsigned integer uniquely identifying the group.
- Group type—used to determine group semantics.
- Counters—updated when packets are processed by a group.
- Action buckets—ordered list of buckets, where each action bucket contains a set of actions to execute with associated parameters.

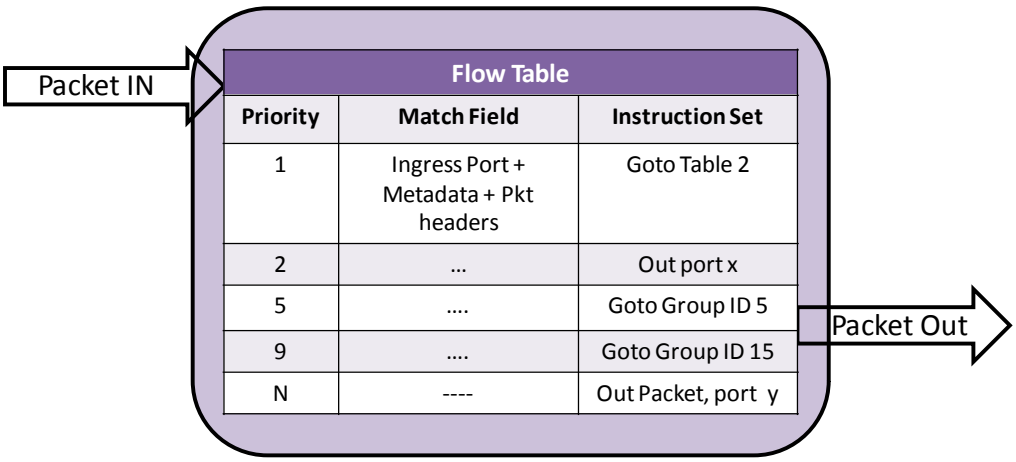

Figure 4. Packet Processing with OpenFlow Flow Table

Group Identifier	Group Type	Counters	Action Buckets

Figure 5. Components of Group Table

A switch supports group types that are "required" or "optional." The group identification (ID) specifies the group used to process the packet. The set of buckets applied to the packet process is driven by the group type. Groups may consist of either zero or additional action buckets. A group with no buckets (zero) will not alter the action associated with a packet. A group may include buckets that forward to other groups, if the switch can support it. The group type enables the switch to change the forwarding without involving a round trip to the controller to acquire the required feedback. If buckets are not live, then packets are dropped.

Groups represent a set of actions for flooding, along with more complex forwarding methods, including multipath, fast re-route, and link aggregation. Groups also enable multiple flow entries to forward to a single identifier. The group table contains group entries, each containing a list of action buckets, which are dependent on group type. The actions in one or more action buckets are applied to packets sent to the group.

3.4 Pipeline Packet Processing

The OpenFlow pipeline of every OpenFlow switch contains multiple flow tables and entries. The pipeline processing defines packet interaction with these flow tables. By having multiple flow tables, the switch exposes hardware with greater efficiency and flexibility. Network deployments combine orthogonal processing of packets, which indicates that all those processing's are combined in a single table, thus creating large rule sets. Using multiple tables properly decouples this processing. Multiple tables can separate the wildcard match table and the exact match table, and still act logically as a single table.

Pipeline processing instructions permit packets to be sent to subsequent tables for further processing and to authorize information in the form of metadata to be communicated between tables. Since flow tables are sequentially numbered starting from zero, pipeline processing always starts from first flow table (Table 0). The packet is matched against the entries of the flow table to select the entry. If the flow entry is found, then it is executed based on the pre-defined instruction set. This instruction (Goto instruction) directs the packet to another flow table where a similar process is executed. A flow entry can direct a packet to a flow table number that is greater than its own flow table number. This stipulates that the pipeline processing can only go forward and not backward. The table pipeline processing ends when the instruction set associated with a matching flow entry does not specify a subsequent table for the packet to be forwarded and modified.

A flow table entry is identified by its match fields and priority. Match fields and priority together represent a unique entry in the flow table. A table ID is assigned such that it can choose the table to which the configuration change should be applied. Flow tables are numbered from zero followed by any subsequent number. The switch needs to check for any overlapping flow

entries in the requested table. If a single packet matches two flow entries, then both entries have the same priority. If there is an overlap conflict in an existing flow entry, then the switch refuses the addition entry and responds with an error message. For any non-overlapping flow entries, the switch must insert the flow entry into the requested table with identical match fields and priority. Figure 6 represents the pipeline processing of a packet.

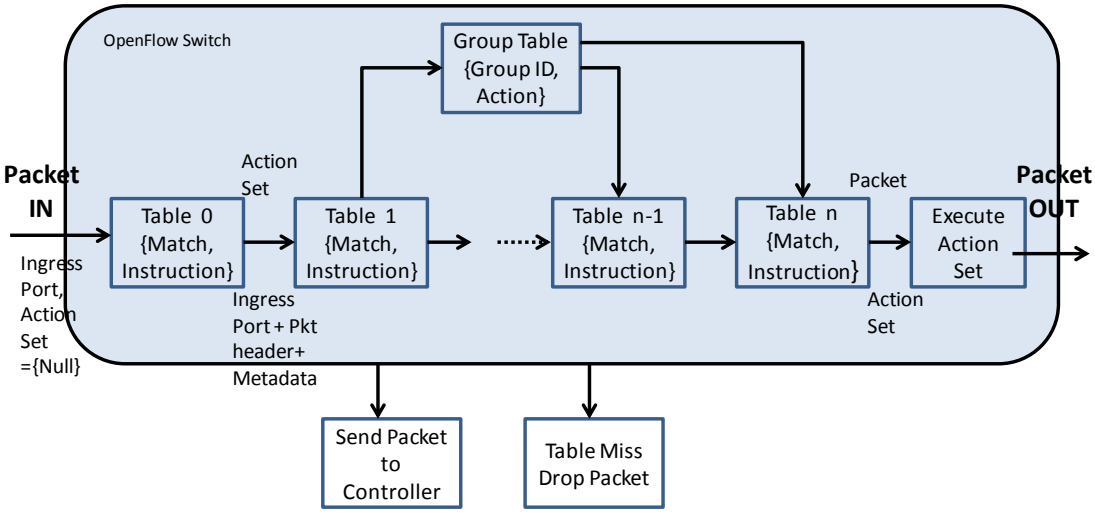

Figure 6. Pipeline Processing of a Packet

If a packet does not match a flow entry in a flow table, this is called table-miss. A table-miss flow entry in the flow table can state how to process the unmatched packets, such as dropping a packet, passing a packet on to another table, or sending a packet to the controller over a control channel. A table-miss flow entry is identified by its match and priority. It wildcards all match fields and has the lowest priority (0). The table-miss flow entry must support at least sending packets to the controller using the controller reserved port and dropping packets using the Clear-Actions instruction.

When the packet is received, the switch performs a table look-up in the first flow table. The packet match fields depend on packet type and various packet header fields (tuples). Using pipeline processing, the OpenFlow switch performs table look-ups in other flow tables. A packet

matches a flow table entry if the values in the packet matching the fields used for the look-up match the pre-defined flow table entry. The selection of the packet match against the table is based on the order of the highest priority flow entry match. Then the counters associated with the selected flow entry along with the instruction will be updated and applied. If there are multiple matching flow entries with the same priority, then the selected flow entry is explicitly undefined. Figure 7 shows a schematic representation of packet processing in the OpenFlow switch.

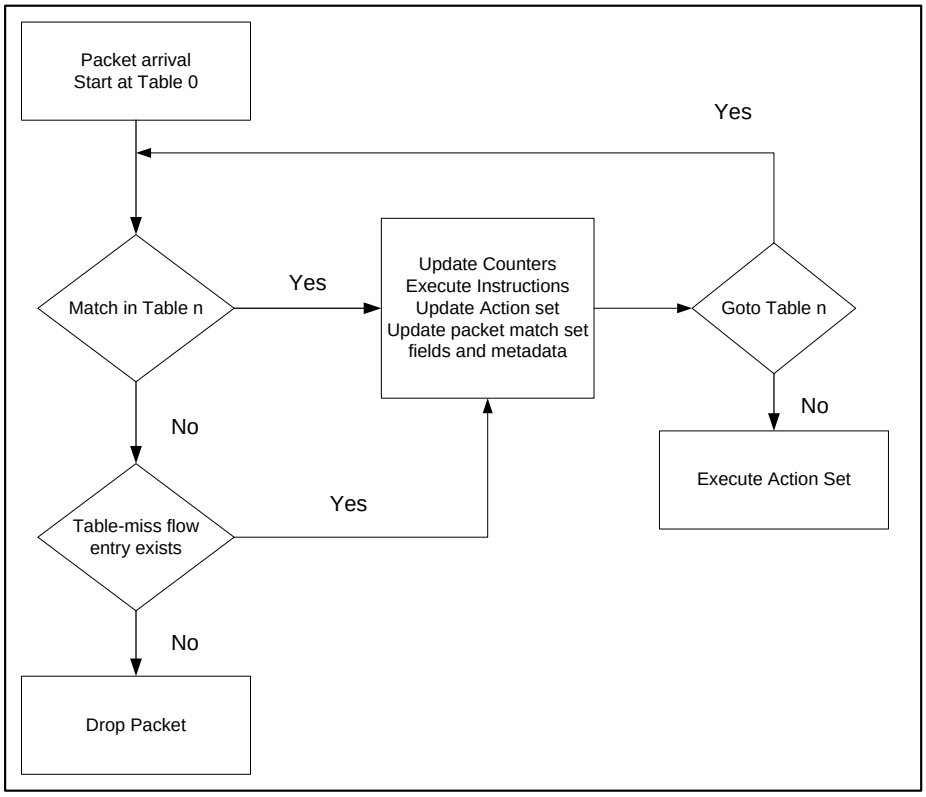

Figure 7. Packet Processing in OpenFlow Switch

3.5 OpenFlow Controller

An OpenFlow control plane is referred to as a controller. The OpenFlow channel is the interface that connects an OpenFlow switch to a controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out to the switch (adds, updates, and deletes flow entries in flow tables). All OpenFlow channel messages

must be formatted according to the OpenFlow protocol, which is usually encrypted using TLS/SSL.

An OpenFlow controller manages multiple OpenFlow channels, one for each OpenFlow switch. The OpenFlow switch must have the ability to create an OpenFlow channel by initiating a connection to the controller. An OpenFlow switch may connect to one or multiple controllers. A controller may manage an OpenFlow switch remotely over TCP/IP connectivity. The OpenFlow switch must have the ability to create an OpenFlow channel by initiating a connection to an OpenFlow controller. The controller is notified about the supported capabilities of each connecting switch during the initial handshake sequence.

The OpenFlow controller maintains a one-to-one relationship with each switch within its domain (similar to management systems in IP and the transport network). The controller performs functions such as switch configurations, statistics queries, performance monitoring, etc. Packets that do not match any flow entry are encapsulated and sent to the controller, which evaluates how to process the packet and then caches its decision in the data plane so that similar future packets in the flow are processed in a like manner.

The OpenFlow protocol enables communication between the separated data and control plane. It aids the controller in summarizing each data plane switch as a flow table. The control plane decides the flow of packet forwarding and stores the packet in the data plane's flow table. The data plane is oblivious to routing decisions that are made. The control plane determines the necessary packet access and forwarding actions such as access control, routing, and multicast.

When an OpenFlow operation is initiated, the switch must connect to all controllers to which it is configured and try to maintain connectivity with all of them concurrently. Controllers may send commands to the switch on the associated connection. Using asynchronous messages,

different controllers can receive different messages that need to be enabled or filtered out. The switch does not do any arbitration or resource sharing between controllers, but it can operate in OpenFlow mode if one controller or controller connection fails. Having multiple controllers can improve reliability, enable fast recovery from failure, and support load balancing.

The controller sends commands to modify the state of the switch using the OpenFlow protocol. It can connect to the switch in three different roles/states as specified by the standard [18]: (1) Master, (2) Slave, and (3) Equal. The Master/Slave roles are used in cases like high availability or load balancing. In the Master role, the controller has exclusive control over the switch. In the Slave role, the controller has read-only access and does not receive asynchronous messages. The Slave becomes the Master, if the Master fails. The switch cannot change the state of the controller on its own. It can be changed only if any one of the controller makes a request. The Equal role is similar to Master role where it has full access to the switch and is the only controller role available. The controller in the Master state (if any) and all controllers in the Equal state can completely alter the switch state. There is no mechanism to enforce partitioning of the switch among the controllers. No controller should be in an Equal state, if the controller in the Master role requires being the only controller that has the ability to make changes on the switch. All controllers, other than the Master, should be in a Slave state. The default role of a controller is the Equal state. Here, the controller has full access to the switch and is equal to other controllers in the same role. In this role, the switch can simultaneously connect to multiple controllers, and they all have the ability to modify the flow table. It becomes challenging to track or select the modification determined by each controller.

The OpenFlow protocol exploits the concept of look-up tables in modern Ethernet switches and routers. These flow tables run at a line rate to implement firewalls, network address

translation (NAT), or QoS, and to collect statistics. OpenFlow provides a standardized way of manipulating flow tables that can enable a flow-based network traffic partition. Network traffic can be organized into various flows that can be grouped and isolated in order to be processed, routed, or controlled as preferred. Flows can be created and maintained by a centralized entity called the controller. The controller can be extended in order to perform additional tasks such as routing and network-access decisions. In order to implement OpenFlow, switches are modified to support it and a centralized controller process that contains network logic. Thereby, the control and data plane no longer exists on one single network device, but rather resides as a separate and dynamic link to one another. The separation of control and data plane functions and the adoption of a centrally controlled network model are concepts that have been discussed and approached by researchers [26]. Researchers have proposed architectures for enabling the decoupling of the control and data plane functionality of network devices, aimed at providing more efficient packet forwarding and greater flexibility in control functions.

3.6 OpenFlow Network

The main components of a controller-based OpenFlow network [27] are as follows:

- OpenFlow-enabled switches
- Server(s) running the controller process
- Database containing the network

The controller is the core, central entity that gathers the control plane functionality of the OpenFlow network. It provides an interface for creating, modifying, and controlling the switch's flow tables. Running typically on a network-attached server, there can be one controller for the each set of OpenFlow switches on the network, or there can be one controller for each switch. Therefore, the control functionality of the network can be completely or locally centralized based

on the performance of delegation of switch management to the controllers. The network view includes the switch-level topology, as well as the location of users, hosts, middle boxes, and other network elements and services.

The OpenFlow switch does not perform L3 routing. There is no longest-prefix-match or any other complicated calculation that takes place on the switch. In fact, the protocol does not define how the forwarding decisions for the specific header fields are made. The decisions are determined by the programmable controller and simply installed in the switches' flow tables. OpenFlow switches address the flow tables and match the incoming packets' header fields to pre-computed forwarding decisions, and they simply follow these decisions [3].

The secure channel is the interface that connects each OpenFlow switch to a controller. Through this interface, the controller exchanges messages with the switches in order to configure and manage them. OpenFlow provides a protocol for communication between the controller process and OpenFlow switches.

Three types of messages are supported by the OpenFlow protocol: controller-to-switch, asynchronous, and symmetric. Controller-to-switch messages are initiated by the controller and may not always require a response from the switch. Through these messages, the controller configures the switch, manages the switch's flow table, and acquires information about the flow table state or the capabilities supported by the switch at any given time.

Asynchronous messages are sent without solicitation from the switch to the controller and denote a change in the switch or network state. When the packet enters and reaches a switch without finding a matching flow entry, the packet message is sent to the controller. The information sent to the controller contains the entire packet or a fraction of the packet in order

for the controller to examine it and determine the kind of flow that should be established for it. Other events include flow entry expiration, port status change, or other error events.

Symmetric messages are sent without any solicitation in either direction. They are used to assist or diagnose problems in the controller-switch connection. The controller in OpenFlow-enabled network architecture is responsible for all forwarding and routing decisions by managing and manipulating the flow tables on the OpenFlow switches. Once the traffic is logically organized in the flows, its manipulation becomes easier and is much more straight-forward.

3.7 OpenFlow in Mobile Devices

Cellular networks use IP to provide universal data connectivity. The IP architecture is fixed, thus challenging the addition of new capabilities, and has poor support for mobility and security. Radio technology connects to several networks at once for robustness and increased signal quality. The main elements of a cellular network are the base stations, a switching network, land lines, the Public Switched Telephone Network (PSTN), and mobile users. A simple cellular system has only one base station per cell. Users make a call by entering the phone number and activating the "send" button on the phone device. Base stations reserve a few control channels through which the user initially communicates with the base station. All base stations within the cellular network constantly monitor these control channels, and when a "send" command is requested, the base station with the strongest signal takes control. It then routes the user request to the switching network. If a channel is available, the switching network allocates this channel to the user and sends the channel assignment back to the user's current base station. The base station then sends this assignment to the user and facilitates conversation. When the conversation ends, the user terminal sends a "disconnect" signal to its base station, which in

turn relays the signal to the switching network. The network can then free the previously assigned channel and reallocate it to another user.

In this research, the architectural model proposed has the control plane in network switching, and the routing equipment is aligned to a separate controller. This is envisaged by the OpenFlow SDN by separating the data and control plane activities of the switch. The control plane determines the necessary packet access and forwarding actions, such as access control, routing, and multicast. The controller programs the switch with flow specifications that control the packet and routes throughout the network. For routing purposes, the switches only need to run an OpenFlow control plane, which simplifies the implementation. The flow table provided by the controller routes the traffic and empowers it to prioritize applications. This addresses the switching infrastructure capacity to offer the level of service. Traffic that is segregated based on flows and flow prioritization reduces latency, increases quality, and utilizes better bandwidth.

Wireless or cellular deployment includes both control and data planes. By utilizing the OpenFlow-based architecture, it is possible to increase data plane performance and enable dynamic programmability, thus enabling intelligent traffic steering and path management. Traffic engineering can include classification based on packet headers (L2 or L3) and setting the flow-based prioritization (e.g., set highest priority for VoIP traffic and lowest for all others).

In traditional networks, depending on latency, in order to provide high capacity and uniform distribution, necessary protocols must be updated. Also, the transition between packet switching to circuit switching requires enhanced functions to perform at a higher granularity and at a faster time scale. In this dissertation, a novel and unique OpenFlow architectural model is proposed, where the OpenFlow switching paradigm is placed on top of the packet-based switching model on mobile phones with a centralized controller. The switching fabric residing on

the phone is programmed in such a way that it can communicate to route packets similarly to the service provider end. The OpenFlow switch installed in access mode on the mobile device increases network programmability. It is also designed to be programmed by multiple controllers, and associating them to different entities such as an end user, service provider, and/or enterprise.

This chapter detailed SDN-based OpenFlow architecture, providing a brief background of the functioning of control and data plane features, and the components of OpenFlow. It also provided an overview of implementing OpenFlow switching on mobile devices.

CHAPTER 4

THE CONCEPT

4.1 Introduction

With consumers' thirst for information at their finger tips, mobile devices with advanced functionality are consumed at a rapid rate. At the same time, consumers prefer not to pay high price for carrier charges. The use of advanced functionality has led to an explosion in mobile traffic, and has necessitated large-scale infrastructure upgrades and additions to currently deployed service provider networks. Service providers are constantly looking for technology that can reduce their infrastructure costs, while enabling new and interesting features for their customers. In trying to reduce their capital costs, enterprises want employees/consumers to use their personal mobile devices for enterprise activity. While reducing capital costs, enterprises now have more information leakage exposure because the data resides on a device they do not own. Consumers prefer to carry only one device that can be used for both personal and enterprise use.

4.2 Solution Architecture

This research proposes a novel architecture wherein high quality application services can be provided to end devices by a service provider at a relatively low cost. Additionally, both consumers and enterprises should benefit from this model due to its simplistic nature.

Circuit-switching technology enables low-latency communication but is expensive to deploy and maintain. Packet-switching networks use IP and are cheaper to deploy but cannot guarantee low or specific latency. Real-time or live video broadcasts, audio and video conferencing, and parallel processing are some examples of applications requiring low-latency requirements. This dissertation focuses on deploying a cost-effective circuit-switching network

on a packet-switching network to improve delivery of applications requiring low latency. To deploy circuit-switching network topology over a packet-switching network, the standards-based OpenFlow protocol is used for switching. OpenFlow-based switching enables a dynamically programmable layer-2 switch fabric. The fabric itself is split into control and data plane layers. Instructions sent to the control plane (aka OpenFlow controller) are sent as OpenFlow instructions to corresponding OpenFlow switches that are controlled by the same. Instructions sent to the switches are stored in flow tables, which were discussed previously in Chapter 3.

In this novel approach, it is assumed that each mobile device has an OpenFlow-based switch. It should be noted that most commercially available OpenFlow switches are hybrid in nature, i.e., they support both traditional networking and OpenFlow-based switching functionality. With mobile devices becoming powerful with enhanced capabilities, it is not difficult to install a software stack that can provide OpenFlow-based switching on the device.

In the current OpenFlow standards [18], a switch can have multiple flow tables, some of which can be grouped. Grouping allows for actions after processing the corresponding table rules. In this novel approach, there is a minimum of three tables on the switch running on the mobile device, which are exclusively controlled by corresponding controllers:

- Service Provider Flow Table
- Enterprise Flow Table
- Consumer Flow Table

While there are several ways to constrain the policies around overriding rules in flow tables, the approach considered here is the “everyone follows the line, or pipeline” method.

For a particular service, the service provider, in addition to programming the network of OpenFlow switches (for example an OpenRadio [6] style network), would also program the

mobile device OpenFlow switch. The programming or flow installation happens on the service provider flow table. The user with the system preferences application may be able to easily program "flow" preferences (as appropriate and allowed) to its consumer flow table. For certain corporate applications installed on the mobile device, the enterprise will be able to program flow entries into its enterprise flow table. The OpenFlow standardized pipeline processing is followed to deliver QoS.

4.3 Assumptions

When using multiple controllers, the following is assumed, per the OpenFlow switch specification v. 1.3.2 [18], in order to comply:

1. "The controllers coordinate the management of the switch amongst themselves via mechanisms outside the scope of the present specification, and the goal of the multiple controller functionality is only to help synchronize controller handoffs performed by the controllers." The work in this dissertation utilizes this idea, and it is understood that there is an out-of-band agreement to act on specific predetermined flow tables.
2. "The controller can send controller-to-switch commands to modify the state of the switch. The switch does not do any arbitration or resource sharing between controllers." In the work here, the switch specification complies in the context that "controller-to-switch commands change the state of the switch." The switch does not explicitly share resources among the controllers but may implicitly accomplish the same due to the nature of controller interaction with the switch. This act does not break the switch compliance.
3. "The controller in the Master state (if any) and all controllers in the Equal state can fully change the switch state; there is no mechanism to enforce partitioning of the switch between those controllers. If the controller in the Master role needs to be the only

controller able to make changes on the switch, then no controllers should be in the Equal state, and all other controllers should be in the Slave state." In the work here, the Equal role concept of the controllers and work within the limitations of the switch compliance is adopted. All controllers have full visibility to entire flow tables and their actions. Controllers act only on those tables to which they are constrained (by out-of-band mechanism). All controllers receive the entire set of switch-to-controller messages, but only those relevant to it are used, and the rest are disregarded.

4.4 Architecture Model

In this section, an architecture to model an OpenFlow switch with multiple controllers is proposed. Figure 8 shows a representation of the architecture model.

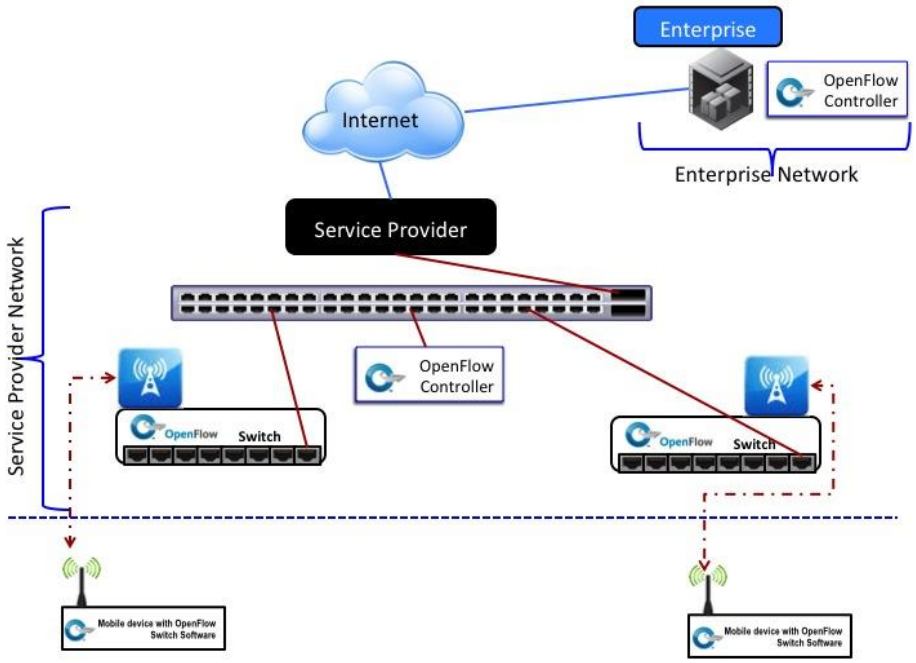

Figure 8. Architecture Model

4.4.1 Use Case: Walk-Through of Architecture

Problem Statement: The service provider wants to deliver a live broadcast of a sports game to its mobile customers. Modes of delivery include "free" and "paid subscription."

Solution Architecture: Free services have reasonable QoS and are delivered over packet-switched IP network currently deployed. The new "paid" model relies on programming only those OpenFlow switches deployed on the network to which these specific mobile devices are connected. If the service does not include a "free" component, then this "paid" network would be super efficient because it would not clog the network at all due to its programmability. On the mobile device side, the following flows may be set on the corresponding tables:

1. User Table—allows a live broadcast view over both Wi-Fi and 3G (switch to 3G if Wi-Fi is not available).
2. Enterprise Table—does not allow viewing of a live broadcast over the corporate Wi-Fi.
3. Service Provider Table—allows a live broadcast by connecting to Access Points of Type "OF," and if not available, then connect to Access Points of Type "ANY."

The order of pipeline processing is programmed via a flow table. Depending on the application flow, the order could be user, enterprise, service provider or enterprise, service provider, user, etc. All combinations are pre-programmed in the flow table. When a packet is received, a generic flow application entry is matched and programmed to use the appropriate flow table.

4.5 User-Driven Network Connectivity Architecture Model

This section discusses a user's ability to control the network connectivity when using a specific application. The goal of this architecture model is to remove the limitation of pre-set algorithm selection of network connectivity. Users and applications will be empowered to control and implement custom network connectivity algorithm logic. This architecture is developed by incorporating a network bridge to support legacy of currently deployed applications and a programmable OpenFlow switch such that applications can directly get

connected. The concept proposed here focuses on a model to enable users to control network connectivity on a mobile device with the applications while continuing to allow the current working mobile device model. Figure 9 illustrates the architecture of using a standard network bridge and programmable OpenFlow switching on the mobile device.

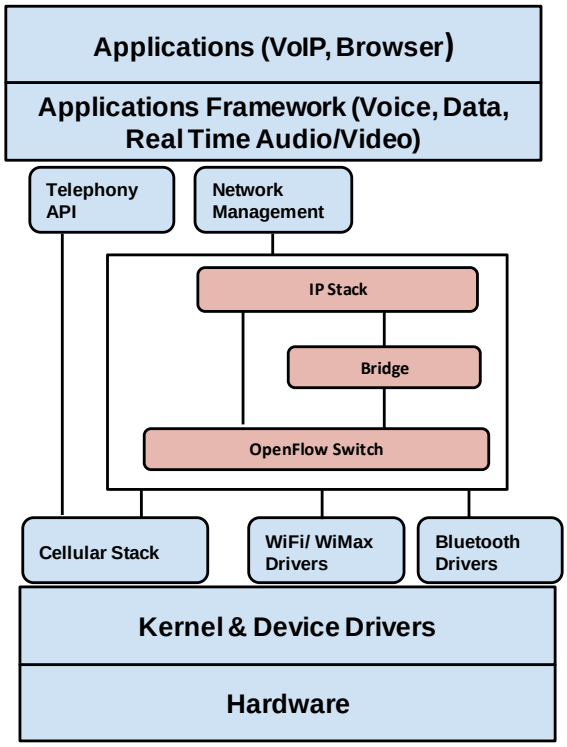

Figure 9. OpenFlow Switch on Mobile Device

Mobile devices use various radios for network connectivity solutions. The main focus of this proposed concept is only on the data connectivity part. For data services, with today's infrastructure regardless of the type of radio interface, IP is relied upon. The service provider provides the IP address when the radio successfully connects to the service access point for data services. With various radios available on the mobile device, those providing data connectivity services are hidden behind an OpenFlow switch (WAN/north side). A bridge is setup on the other (south) side of the switch where currently deployed applications get connected. The

combination of OpenFlow switch and the bridge will continue to provide services for current applications without changes.

Utilizing the OpenFlow switch on the mobile device, an application needing to use a specific network connectivity option directly can be programmed. For example, consider a scenario where a mobile device may have cellular or Wi-Fi connectivity options. Assuming that a user needs to make a VoIP call, then by default, Wi-Fi connectivity is automatically selected. By employing this architecture model, the user can opt to select either cellular or Wi-Fi network connectivity.

On the radio side, after power is turned on, the data service is activated and tries to get the IP address from service provider via DHCP automatically. Alternatively, it may be is manually configured (static) which is rare. If static IP address is selected, then the Internet connectivity is checked (e.g., Ping of 1 byte). If the connectivity fails, then the registration fails to happen. On successful connectivity, the corresponding routable IP address is associated to a port on the OpenFlow switch for the radio channel and the switch routing table is updated. The association (IP, port and radio connectivity channel type) is stored on the switch in routing table with additional information such as connectivity speed. The port is also marked as a wide area network (WAN) side port. When a radio is powered down or connectivity drops or changes (e.g., 3G to LTE, or Wi-Fi 802.11n to 802.11b), then a new data connectivity session is setup that results in getting a new IP address. There is a corresponding switch action occurring to update the route table on teardown and setup of connection. Figure 10 shows the routing table on WAN side. For simplicity and ease of understanding, it is mentioned that routing table is stored on the OpenFlow switch. But in reality, they are programmed as flows. The controller could be running on the same device acting on the switch.

Switch Routing Table				
Destination	**Gateway**	**GenMask**	**Class of Service**	**Action Set**
Default	10.10.10.1	0.0.0.0	WAN-WiFi	OutPort 4
169.254.0.0	169.254.0.22	255.255.0.0	USIM1-3G	OutPort 7
192.168.2.0	0.0.0.0	255.255.255.0	USIM2-LTE	OutPort 1
10.10.10.0	0.0.0.0	255.255.255.0	WAN-WiFi	OutPort 5
172.168.3.0	172.168.3.25	255.255.0.0	WiMax	OutPort 3

Figure 10. Routing Table on WAN Side

Applications use API to connect to southbound network services. With the proposed architecture, the IP stack layer is continuously maintained to translate to the existing conditions. A network bridge is introduced between the IP stack and OpenFlow switch such that the connectivity between the applications and various radios providing data services are enabled. Figure 11 shows the routing table on bridge side.

Iface (port)	MAC Address
WAN-WiFi	Mac A
USIM1-3G	Mac B
USIM2 - LTE	Mac C
WAN-WiFi	Mac D

Figure 11. Bridge Routing Table

Additionally, programmability is directly facilitated to the OpenFlow switch. When the applications utilize this programming interface, they will be able to allow user selection for network connectivity. Programming an OpenFlow switch implies installing flows on the switch. The process of checking for flow selection path using the user-driven architecture model is illustrated in Figure 12.

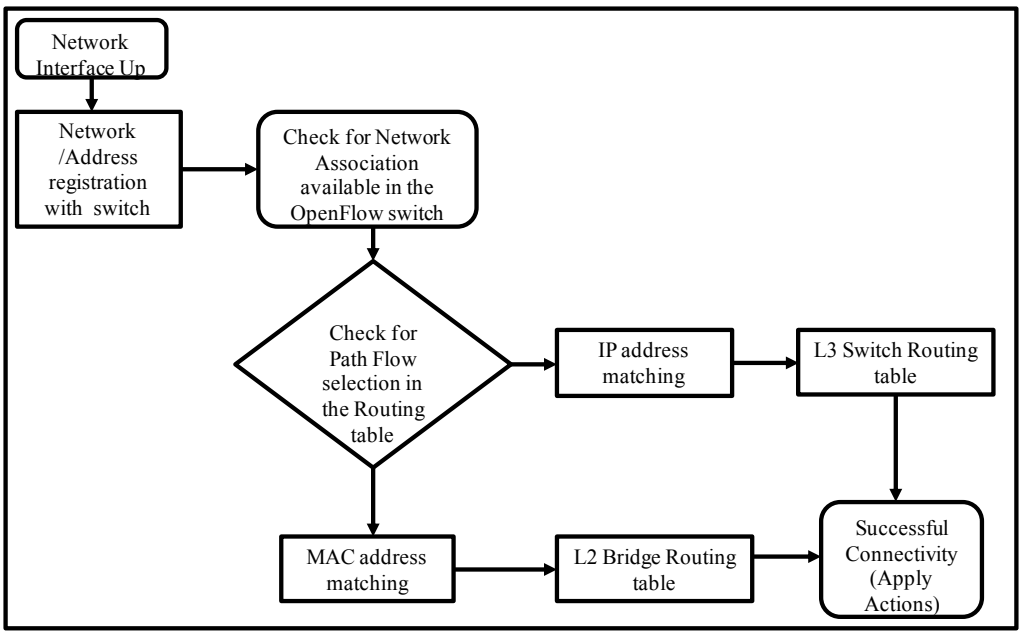

Figure 12. Process for Checking Path Flow

In summary, by adding an OpenFlow switch on a mobile device, inexpensive and programmable circuit switching is deployed to provide supplementary value-added services. This chapter introduced the novel architectural model and the concept of the proposed research work. It also projected user-driven architecture model that expands to application driven technology. The ability of selecting network connectivity option for a specific application on-demand by the user was focused. Hiding radio services behind an OpenFlow switch on the mobile device can lead in providing better and interesting network solutions.

CHAPTER 5

MODELING MULTIPLE CONTROLLER ON A SINGLE OPENFLOW SWITCH

5.1 Introduction

OpenFlow based switching architecture provides an end-to-end programmable dynamic network. It supports multiple tables that facilitate the flows to be programmed in a pipeline process. Utilizing the concept of multiple controllers, which are user, enterprise, and/or service provider entities, the OpenFlow switch on a mobile device can be programmed mutually and exclusively with flows.

This chapter proposes different mathematical techniques to model programming the OpenFlow switch with multiple controllers. One method is via an ordered subset of arrangement, known as the permutation method, which supports the flow table assignment for multiple controllers. Another method is to model flow table actions and controller interactions using computational finite-state automata on a single OpenFlow switch. These methods are described in later sections of this chapter.

5.2 Permutation Method

Using the assumptions in section 4.3, this theory is extended to mathematically model multiple controllers with multiple flow tables. Consider a set of three controllers A, B, and C that installs flows onto different tables on the switch. Assume that pre-defined scenarios dictate which controller gets to act on the packet first, even though all controllers are equally competent to act on the packet. The order in which the scenarios are applied and processed is determined by various processing rules. With the assumed out-of-band mechanism, the controllers are aware of which set of tables they are capable to modify. The order of arrangement of these three controllers results in a number of ordered sets, known as permutations [30].

For a set of three controllers—A, B, and C—the arrangement can be as follows: A-B-C; A-C-B, B-C-A, B-A-C, C-A-B, and C-B-A. This is mathematically represented as

$$3! = 3 \cdot 2 \cdot 1 = 6$$

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120 \quad (5.1)$$

If there are four controllers, then there are 24 possibilities; if there are five controllers, there are 120 possibilities; and so on.

The number of permutations in the set of (1… n) controllers is given by n!. This results in a factorial time of $O(n!)$. The unordered subsets containing k elements are known as k-subsets of a given set. The number of ways obtained by an ordered subset of k elements from a set of n controllers is given by

$$nPk = \frac{n!}{(n-k)!} \quad (5.2)$$

In this model, it is assumed that k = n, where n controllers are chosen from k elements with no repetition, and the order is considered. The controller will act on the switch by associating through flow tables. There will be a minimum of (n + 1) set of table associations for a single controller. For more than one controller, the number of table association corresponds to (2n + 1), (3n + 1), (4n + 1) etc., in order to accommodate all sets of permutations.

For "m" sets of tables, the number of sets of table associations for "n" controllers is given as

$$n(n-1) + 1 \; for \; n \geq 1 \quad (5.3)$$

This is demonstrated as follows:

$$\{ 1 \times 0 + 1 = 1$$

$$2 \times 1 + 1 = 3$$

$$3 \times 2 + 1 = 7$$

$$4 \times 3 + 1 = 13$$

$$5 \times 4 + 1 = 21 \ldots\} \tag{5.4}$$

The relation between the controllers and flow tables, obtained by equations (5.3) and (5.4), are depicted in Figure 13. The values for only ten controllers are considered in this graph. With this model, multiple controllers can be programmed with multiple flow tables on an OpenFlow switch effectively by having an ordered arrangement set.

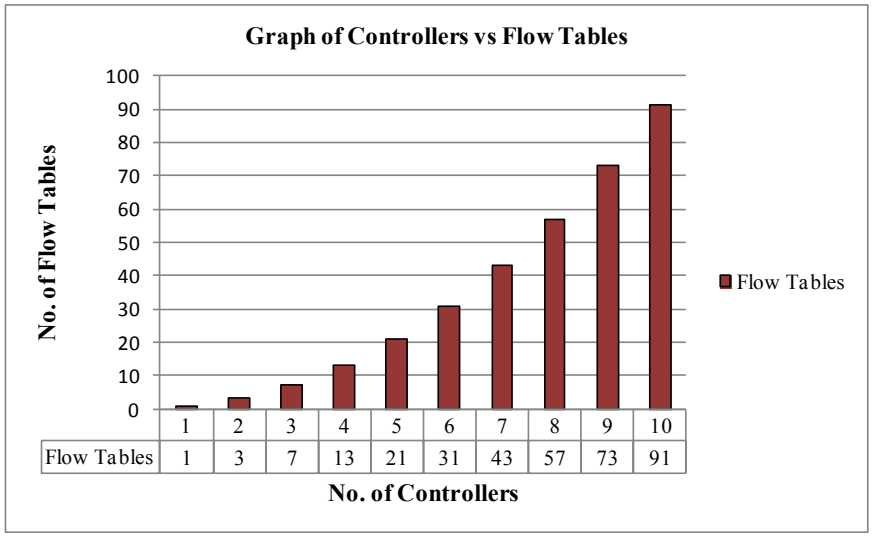

Figure 13. Relation between Controllers and Flow Tables

5.3 Finite-State Model

The finite-state model captures actions across flow tables with a controller and actions by multiple controllers via the concept of finite-state Markov chains [31, 32]. All flow tables exist within the switch. A switch is externally managed by a controller. First, a scenario is laid out in which only one controller is considered and multiple flows tables within are in use. The actions across flow tables are referred to as steady state. Then, multiple controllers acting simultaneously but only on specific flow tables, referred to as mutually exclusive action, are introduced. In the analysis, certain conditions are treated as endogenously given, such as ability to move from flow

A to flow B within a flow table. Figure 14 is a representation of a switch environment with multiple controllers and flow tables.

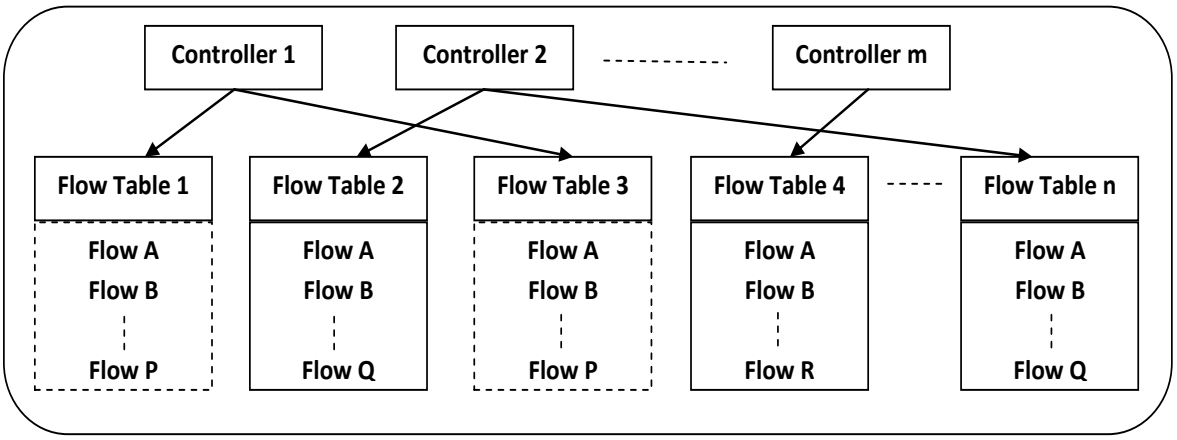

Figure 14. Multiple Controllers Acting on Multiple Flow Tables

5.3.1 Steady State

Each controller manages one or more flow tables. Each flow table F hosts numerous flows that carry action instructions. Movement within a flow table is inherent and explicit. The focus of the model is to depict conditions that prevail and drive interactions across flow tables.

For the purpose of this model, the reference of flows indicates movement across various flow tables with the controller. It is assumed that i is the initial state of the flow, and j is the final state of the flow. Flows are driven by predefined actions. The movement of the flow from i to j can occur only in the forward direction, as defined by flow tables. Flow from i to j is defined as transition (change of state) movements. The transition probability from i to j is denoted by p_{ij}. The characteristics of the flow are modeled as a stationary Markov chain.

If state j is accessible from state i (indicated by i ➔ j), then i has a non-zero probability of transitioning into state j. Since transitions are independent of each other and are not driven by previous states, the probability of state transition can be represented as

$$p_{ij} = P_r\left(F_j = j \middle| F_i = i\right) > 0, \forall\, i \leq j \tag{5.5}$$

where i and j are independent states.

Flow actions can move to tables of the same or a higher number. So, flow movements are represented via a transition matrix (stochastic matrix) [33] where the conditional probability indicates the change in states. The stochastic matrix consists of nonnegative values, where each row sum equals one.

Consider flow tables with i states. For all i, there is an equal probability of i transitioning to the j state, where $i \leq j$. For all the mutually exclusive events causing transition, probability can be denoted as

$$p_{ij} \geq 0, \ \sum_j p_{ij} = 1, \text{for all i} \tag{5.6}$$

The transition matrix satisfying all flow tables in a controller is given by

$$p = \begin{pmatrix} p_{11} & \cdots & p_{1j} \\ \vdots & \ddots & \vdots \\ p_{i1} & \cdots & p_{ij} \end{pmatrix} \tag{5.7}$$

5.3.1.1 Illustration

Consider the transition matrix p for a controller that manages three flow tables: 1, 2 and 3. Based on the model in section 5.3.1, the state diagram can be drawn as shown in Figure 15. Note that the direction of the arrows indicates flow directions.

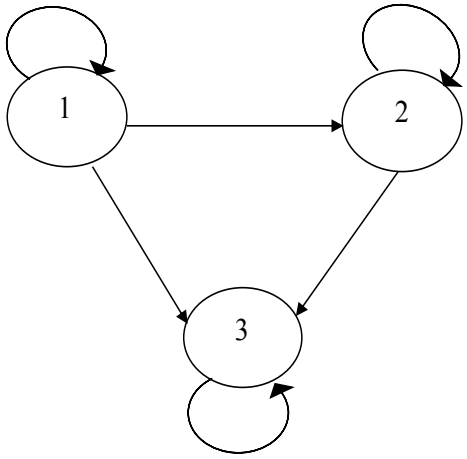

Figure 15. Finite-State Model for Single Controller

The transition matrix p is given as

$$p = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} \tag{5.8}$$

The transition probabilities p_{11} is the probability computed for the movement of flow table from 1 to 1. Similarly, p_{12} and p_{13} are the probabilities for the movement of flows from table 1 to 2 and 1 to 3 respectively. Because flows cannot transition to tables lower than their current state, the probabilities of occurrence of p_{21}, p_{31}, and p_{32} are all zero. Thus, the probability matrix is denoted as

$$p = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.9}$$

5.3.2 Introducing Mutually Exclusive Actions

Now the concept of multiple controllers acting on different flow tables at the same time, referred to as mutually exclusion actions, is introduced. During switching, all controllers assume an Equal role. They are independent and share action information across the system. As a connectivity part, controllers interact with a set of predetermined flow tables. No two controllers can drive the action of the same flow table.

Consider m controllers acting on a switch. At any point in time, one or more controllers can be actively managing flows tables. However, the same flow table F will not be managed by two controllers. This action implies that the controllers are aware (memory) within the switch (finite space). There is finite number of n instruction actions within flow tables. Thus, action on flow table n by controller m is dependent on all the other (m–1) controller actions.

Thus, the probability of action of a controller managing specific flow tables is represented by

$$P_r(F_m = p_m \mid F_{m-1} = p_{m-1}, F_{m-2} = p_{m-2}, \ldots, F_{m-n} = p_{m-n}) = \sum_{r=1}^{m} f(p_m, p_{m-n,} r) \text{ for } n \geq m, \quad (5.10)$$

where, r is an integer. Note: It is assumed that a controller has at least one flow table to manage, and n is greater than or equal to m.

The behavior of flow table actions and multiple controllers can be mapped to the concept of finite-state machines. The probabilities of actions of flow table movements to attain steady state and the assignment of flow tables across multiple controllers can be estimated using transition matrix probabilities, as indicated.

In conclusion, this chapter presented some mathematical modeling techniques to establish the interaction of multiple controllers with multiple flow tables on a single OpenFlow switch installed on a mobile device. The key purpose of this section was to showcase the application of theoretical modeling techniques for the proposed idea of using multiple controllers in an OpenFlow switch.

CHAPTER 6

PROOF OF CONCEPT

6.1 Introduction

OpenFlow approach provides a unified interface to programming network switches across network devices of different vendors in a simple and effective manner. Working at the flow level, OpenFlow offers granularity of traffic processing. Its architecture can accommodate all types of header fields and packet actions to off-load complex processing tasks to the switch hardware. Multiple control planes can interact with a single data plane, thus facilitating multiple flows to be programmed in a pipeline process. The packet that enters into the data plane can take different paths driven by flows in various tables, as per pipeline processing rules.

6.2 Flow Path Process

This section characterizes the design of the model depicted in section 5.2. The design is enhanced by analyzing the flow path that the packets need to follow. In this model, the following assumptions are made:

- Packets need to pass through all tables managed by the controllers.
- There must be out-of-band agreement between controllers on which of the tables would be modified with flows.
- All designated table modifications are mutually exclusive.
- Each controller may modify one or more tables as assigned.
- Design continues to support standard pipeline processing, where it moves only in the forward direction.

It is implied that at any instance, each controller can modify its designated table with entries for processing. To perform flow processing, the ordered combinations of controllers are

stored in the first set of table entries. The order in which the controller gets to act on the packet is stored in the "Goto" table instruction. For an invalid arrangement, the packet may be dropped or sent to the controller to define actions.

The number of designated tables in the switch for a controller may not be the same for all controllers acting on that switch. But at least one table is assigned completely for each controller. Controllers acting on a switch may or may not know the existence of each other, with no expectation of communication among them. This implies that the controller(s) can be removed with corresponding modifications to flow tables.

When a packet enters the switch, it first performs a table look-up according to the packet match fields to find the appropriate ordered set of controllers to process the packet. The ordered combinations are listed in the first set of table entries (say Table 0). Subsequently, the switch performs table look-ups in other flow tables and routes the packet appropriately. The flow table entry with a higher number (priority) is selected when the packet is matched. Flow paths of a packet processed among different controllers are illustrated in Figure 16.

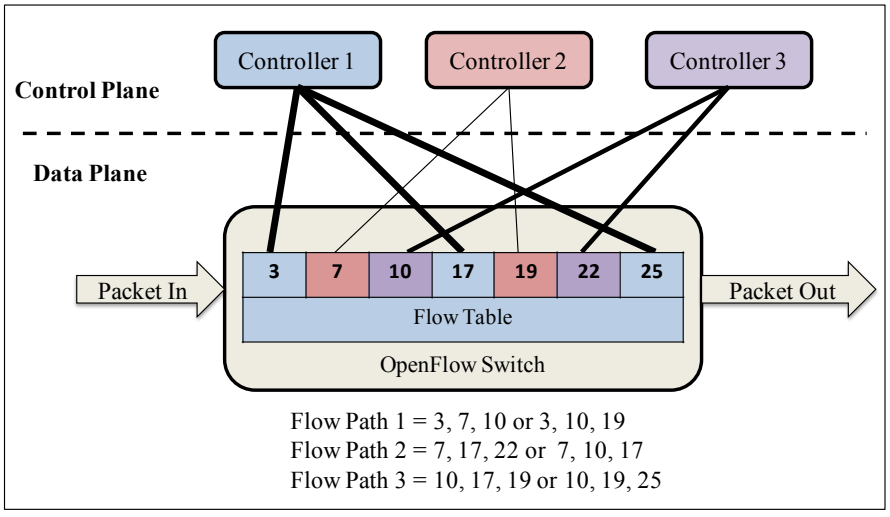

Figure 16. Flow Path Process

Consider an example of three controllers C1, C2, and C3. Based on the illustration, there are six possible arrangements: 1-2-3; 1-3-2; 2-1-3; 2-3-1; 3-2-1; and 3-1-2. This set of ordered arrangement numbers is obtained by permuting the number of controllers considered as taken from equation (5.2) in section 5.2. When a packet enters, it performs a table look-up and matches the order of controllers it needs to process. According to the example,

- Controller 1(A) is managing flow tables 3, 17, and 25.
- Controller 2(B) is managing flow tables 7 and 19.
- Controller 3(C) is managing flow tables 10 and 22.

The content of tables to which the controllers are connected is identical, except for the Goto table entry at the end of the instruction set. This means that entries in tables 3 and 17 of controller 1 are identical, except for the Goto table entry in the instruction set. If the order is set to start with controller 1, then there would be two possibilities: 1-2-3 or 1-3-2. Then, the flow table for the packet that follows this order can be 3-7-10 or 3-10-19, respectively. This method of the flow path process is adapted to comply with pipeline processing in OpenFlow switches. Once the flow processing is completed, the associated action set is applied and the packet is forwarded.

Similarly, if the order of the controller is 2-1-3 or 2-3-1, then the flow table path that the packet can take is 7-17-22 or 7-10-17. And if it is 3-1-2 or 3-2-1, then the flow table path is 10-17-19 or 10-19-25. In the model, this process of flow table path that a packet can take is known as table associations. To satisfy the possibility of controllers 3-2-1, flow table 25 must be associated with controller 1, and because the packet cannot go backwards, one more flow table is needed. This method of arrangement fulfills flow table associations with controllers. Thus, there are seven flow table associations for three controller scenarios. This situation can be extended for

m sets of flow table associations with n sets of controllers by using the model proposed in section 5.2 as represented by equation (5.3).

6.3 Experimental Setup

To validate the concept of flow table movements and controller assignments, an experiment using a laboratory test bed was executed, as shown in Figure 17. The test bed consisted of an Intel Atom N450 processor Network Security appliance and two laptops as hosts. The LINC [34] OpenFlow switch software that supports OpenFlow Protocol v1.3 was installed. This software switch can host multiple logical switches having its own controller, ports, and queues. The switch is assumed to reside on the mobile device. The software is written in Erlang programming language. Switches are identified as switch 0, switch 1, switch 2, and so on. The controller connects to switch over an IPv4 network to the port where the controller is listening. By default, the controller port is 6633 (6653 moving forward).

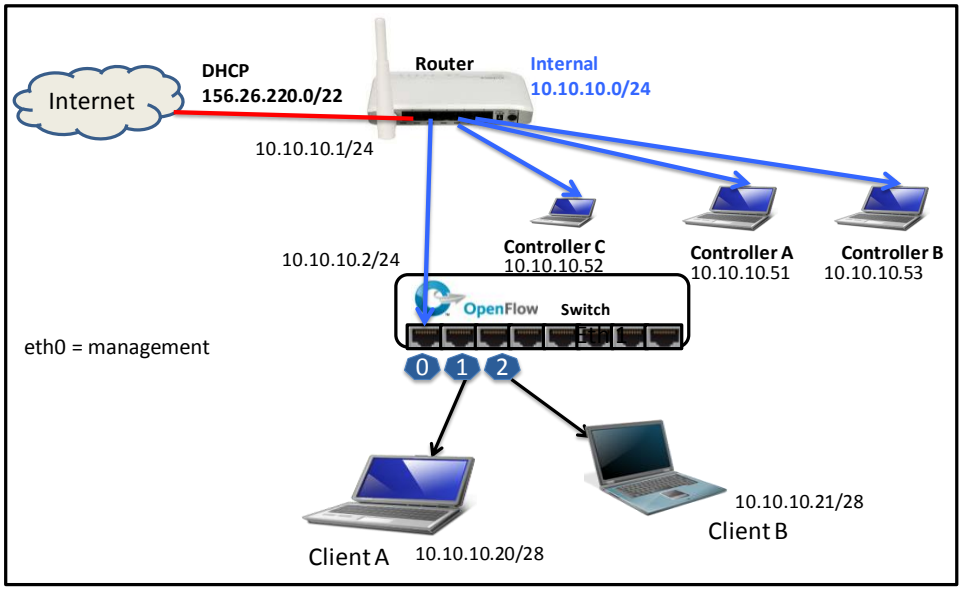

Figure 17. Experimental Test Bed

A scenario of a single controller acting on a switch with multiple flow tables, as shown in Figure 18, was first considered. Based on simple layer-2 learning switch logic, flows were

inserted by the controller on the switch into multiple tables. Flow movements were noted and were analogous to the switch behavior as discussed in section 3.4 and modeled in Chapter 5 (section 5.3.1).

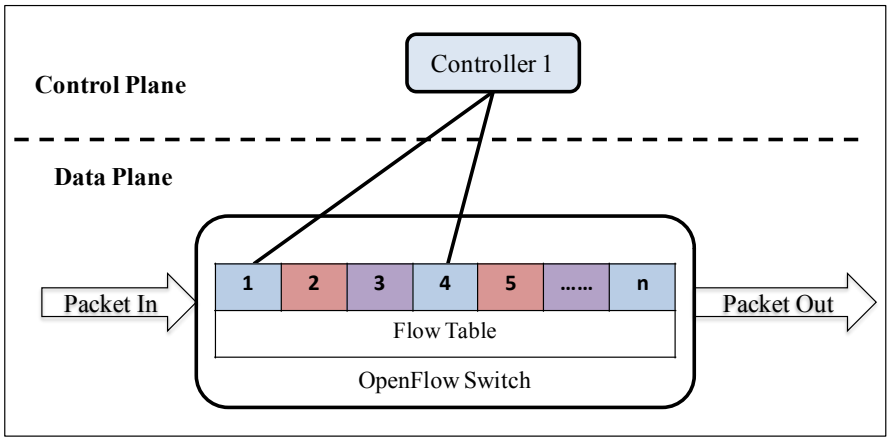

Figure 18. Single Controller Acting on Single OpenFlow Switch

The next scenario was testing with multiple controllers acting on a single OpenFlow switch. Three controllers were considered to demonstrate a multiple controller scenario, as shown in Figure 19. All controllers must pair with the switch resolving to addresses and ports to which they listen. Hence, communication between the controllers and the switch is established.

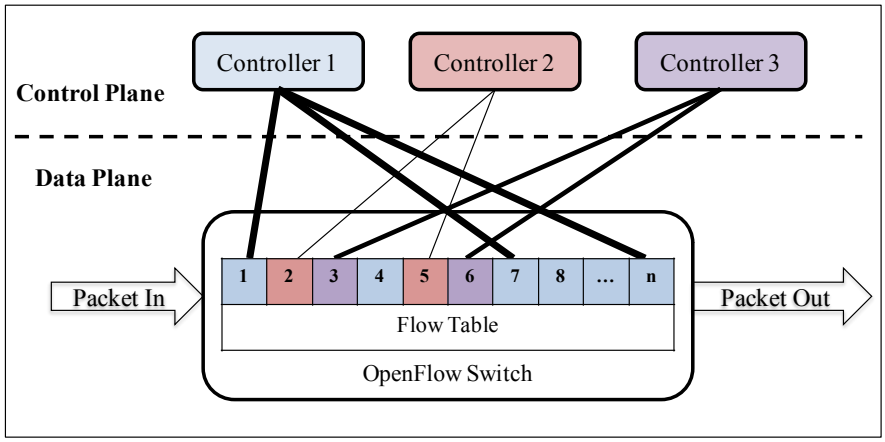

Figure 19. Multiple Controllers Acting on Single OpenFlow Switch

Flows were inserted through the controllers on the switch. A specific set of tables was assigned to the controllers, which could only act on them. The mechanism of the behavior of flows validated the logic described. The result (output of flow entries) obtained is documented in Appendix A.

6.4 Validating Finite-State Model: Simulation Using R Software

6.4.1 Introduction to R

R [36, 37] is an open source statistical programming language and environment that supports statistical computation and graphics. The R project was started by Robert Gentleman and Ross Ihaka of Statistics Department of the University of Auckland in 1995. It is a GNU general public license (GPL) modeled after S and S-plus. R is an object oriented programming language and available for all major operating systems such as Windows, UNIX and Mac. It combines the aspects of functional and object-oriented programming. It is a tool developed for data manipulation, modeling and computation, and visualizing (graphics) fit of models. It can deal with a huge variety of mathematical and statistical tasks.

In addition to base R packages, contributed packages also called repositories, can be accessed from comprehensive R archive network (CRAN [38]). R offers the ability to classify data as quantitative, qualitative, logical and numeric data [39]. It has a special data structure for categorical data, called factors. Factors are closely related to characters because any character vector can be represented by a factor. R classes are polymorphic, which means that the same function call leads to different operations for objects of different classes. It allows inheritance, which allows extending a given class to a more specialized class. The graphical capabilities in R are quite extensive, and the plots and other graphs it produces are highly customizable. Many of the functions in R are generic, which means that they encapsulate a generic concept such as

a plot, the mean, residuals, and summaries. The implementation of a generic function for a particular class of object is called a method. Every object in R has a class, and every class may have one or more methods.

The R editor is useful for building and then executing the script. The instructions or scripts are run on the console. The code will be saved in the R history so that it can be rebuilt or reloaded. R interprets the code you provide directly and converts it into lower-level calls to pre-compiled code/functions thereby providing extensibility. R graphical user interface (RGui) offers some tools to manage in R environment, most importantly, a console window. A third party RGui software that provides a consistent user interface regardless of the operating system is RStudio. RStudio is a cross-platform application with some features that support R. RStudio has four work areas –

1. Source editor - enter multiple lines of code, save R script(s) file to disk, and perform other tasks on the script
2. Console – type commands to see the output
3. Workspace and History-inspects the variables created and show all active objects
4. Files, plots, package and help – shows files and folders in the default workspace, shows graphs, lists add-on packages and additional information.

The simulations using R were run to verify and evaluate the models discussed in chapter 5.

6.4.2 Steady State Model Using R

In this section, the simulation for the probability of an application of finite-state Markov chain model designed in section 5.3 is presented. The movement of flow tables was observed and flow table resolution was resolved by computing the steady state transition probability matrix. For each flow table, the absorbing steady state was noted. The actions across the flow tables and

their movements within the switch are based on the switch's behavior detailed in section 3.4. The simulated model behavior for different samples was as expected in the theoretical model. The simulation was executed for different data sets and repeating for multiple trials. The program code is presented in Appendix B.

6.4.2.1 Observation1: Transition Matrix

A scenario with three flow tables was considered. The probability transition matrix of interaction with three flow tables was generated. The program was executed and repeated through 10 trials. For the transition probability matrix generated in each trial, the results obtained were observed and compared to ensure that it satisfied the equation (5.5).

The probability transition matrix satisfied the conditions discussed in section 5.3.1. For each trial it was noted that the probabilities were consistent thus validating the mathematical model. The transition matrix *p* computes the probability of flow table movement. For example in trial 1, probability of flow movement from 2 to 3 is 0.514. The process is robust and can be executed for n sample sizes and m replications, where n and m are any integers.

```
Trial 1
     [,1]      [,2]      [,3]
[1,] 0.3127036 0.3469055 0.3403909
[2,] 0.0000000 0.4852941 0.5147059
[3,] 0.0000000 0.0000000 1.0000000

Trial 2
     [,1]      [,2]      [,3]
[1,] 0.2993631 0.3949045 0.3057325
[2,] 0.0000000 0.4857820 0.5142180
[3,] 0.0000000 0.0000000 1.0000000

Trial 3
     [,1]      [,2]      [,3]
[1,] 0.3570325 0.3029366 0.3400309
[2,] 0.0000000 0.4790123 0.5209877
[3,] 0.0000000 0.0000000 1.0000000
```

```
Trial 4
     [,1]      [,2]      [,3]
[1,] 0.3322981 0.3416149 0.3260870
[2,] 0.0000000 0.4449878 0.5550122
[3,] 0.0000000 0.0000000 1.0000000

Trial 5
     [,1]      [,2]      [,3]
[1,] 0.3640625 0.3109375 0.3250000
[2,] 0.0000000 0.5247059 0.4752941
[3,] 0.0000000 0.0000000 1.0000000

Trial 6
     [,1]      [,2]      [,3]
[1,] 0.3147541 0.3459016 0.3393443
[2,] 0.0000000 0.4541284 0.5458716
[3,] 0.0000000 0.0000000 1.0000000

Trial 7
     [,1]      [,2]      [,3]
[1,] 0.3243671 0.3243671 0.3512658
[2,] 0.0000000 0.4668246 0.5331754
[3,] 0.0000000 0.0000000 1.0000000

Trial 8
     [,1]      [,2]      [,3]
[1,] 0.321875 0.3593750 0.3187500
[2,] 0.000000 0.4848485 0.5151515
[3,] 0.000000 0.0000000 1.0000000

Trial 9
     [,1]      [,2]     [,3]
[1,] 0.3281972 0.3143297 0.357473
[2,] 0.0000000 0.5000000 0.500000
[3,] 0.0000000 0.0000000 1.000000

Trial 10
     [,1]      [,2]      [,3]
[1,] 0.3386328 0.3116057 0.3497615
[2,] 0.0000000 0.4711779 0.5288221
[3,] 0.0000000 0.0000000 1.0000000
```

6.4.2.2 Observation2: Actions Managing Specific Flow Tables

In this section, the probability for movement of specific flow tables acted by a controller is discussed. The transition probability for the movement from state i to j across different flow

tables given by p_{ij} was determined. This indicates that the probability of being in a particular state and there is no chance of being in other state. For example, the probability of a packet moving from flow table 1 to flow table 2 or flow table 3 were calculated. This signifies the prediction of controller actions managing specific flow tables at different instances as compared to equation (5.10). By Markov property, the Markov chain starts over again and is independent of the past every time it enters state j. An example for five flow tables is considered for demonstrating the output.

```
P, transition probability matrix (considered for one trial)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.1895262 0.1745636 0.2044888 0.2319202 0.1995012
[2,] 0.0000000 0.2056738 0.2836879 0.2553191 0.2553191
[3,] 0.0000000 0.0000000 0.3287037 0.3518519 0.3194444
[4,] 0.0000000 0.0000000 0.0000000 0.3653846 0.6346154
[5,] 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000

rowSums(P)
[1] 1 1 1 1 1

colSums(P)
[1] 0.1895262 0.3802373 0.8168804 1.2044758 2.4088802
```

Probabilities for managing specific flow table movements for multiple steps were computed. It was assumed that the system had 100% certainty in the initial position in state 1 with no chance of being in any other state. Similarly, the initial position with 100% certainty being in state 2, then state 3 and so on was considered. The probabilities of flow movements (p_{11}, p_{12}, p_{13}….. p_{22}, p_{23}…..) across different flow tables for each row vector from the transition matrix were enumerated five times. The result for each row vector yielded F_1, F_2, F_3, F_4 and F_5. F_1 indicates the probabilities calculated to transition from flow table 1 to other flow tables in forward direction. Likewise, the outputs for other row vectors F_2, F_3, F_4 and F_5 were estimated.

The outputs F_1, F_2, F_3, F_4 and F_5 depict the behavior across each flow table movement. The plot for behavior of flow table movement for F_1 is shown in Figure 20. The probabilistic

nature of the model designed was inferred from the graph. For 100% certainty being in state 1, the initial position was assumed as (1, 0, 0, 0, 0) in trial 0. The probability of transition from one to other tables was noted in trials 1 through 5. The output of probabilities for F_1 is shown below.

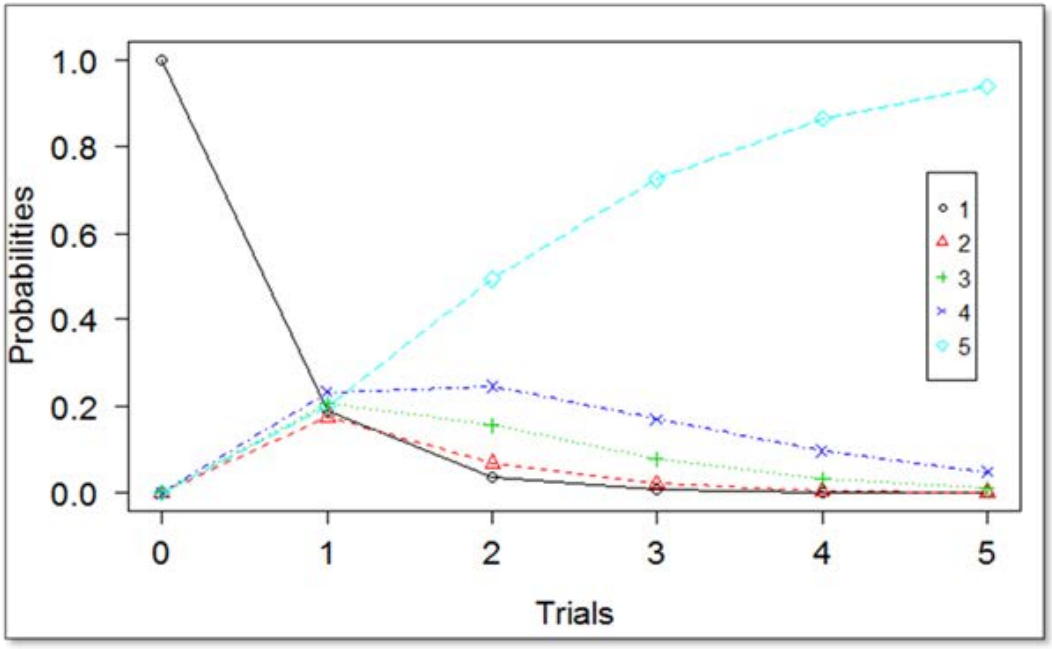

Figure 20. Behavior of Flow Table Movement for F_1

F_1

	[,1]	[,2]	[,3]	[,4]	[,5]
Trial0:	1.0000000000	0.000000000	0.00000000	0.00000000	0.0000000
Trial1:	0.1895261845	0.174563591	0.20448878	0.23192020	0.1995012
Trial2:	0.0359201746	0.068987521	0.15549378	0.24521421	0.4943843
Trial3:	0.0068078136	0.020459277	0.07802758	0.17025272	0.7244526
Trial4:	0.0012902589	0.005396333	0.03284413	0.09646439	0.8640049
Trial5:	0.0002445379	0.001335116	0.01259070	0.04847990	0.9373497

Figure 21 shows the graph of flow table movement behavior for F_2. In F_2, the initial position is considered 100% certainty to be in flow table 2. As flow cannot move backward, transition from 2 to 1 is zero. Flow movements will occur from 2 through 5 as depicted in the plot.

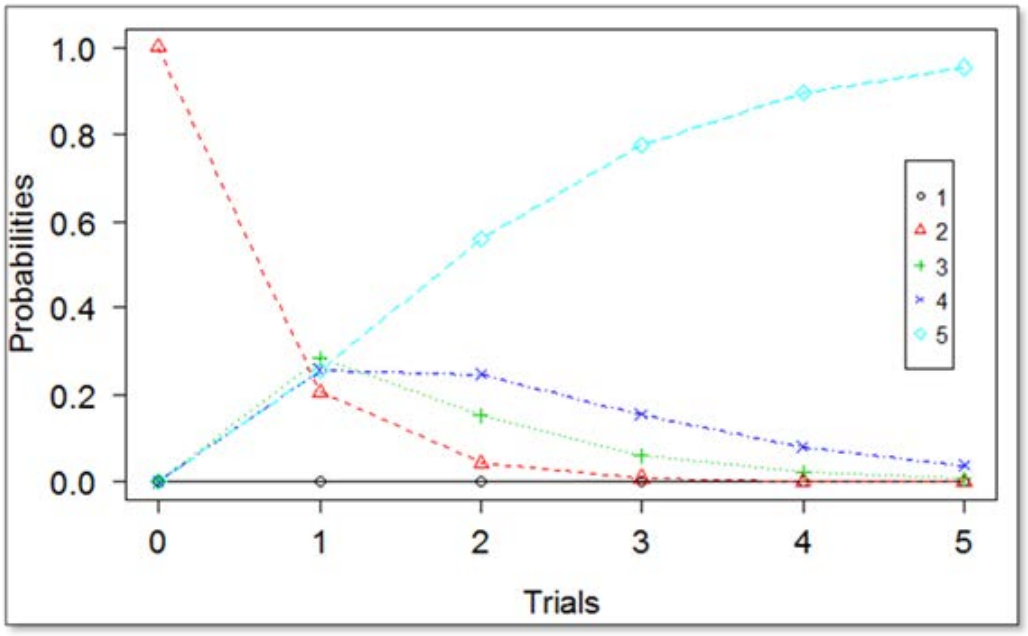

Figure 21. Behavior of Flow Table Movement for F_2

Similarly in F_3, the probability of flow movements from 3 to 1 and 3 to 2 is zero. Transitions occur in state 3, 4 and 5 as shown in Figure 22. Figure 23 shows the behavior for F_4 where the flow table movement is between 4 and 5. In F_5, the movement of flow table 5 is within the same state. Thus, there are no transitions occurring across the flow tables. Results for F_2, F_3, F_4 and F_5 are presented in Appendix B.

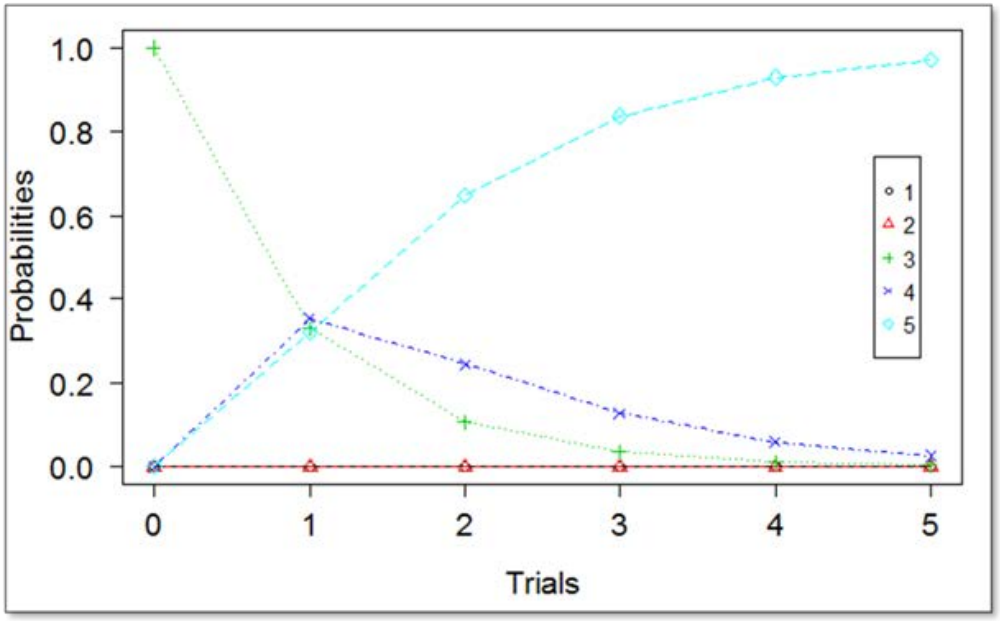

Figure 22. Behavior of Flow Table Movement for F_3

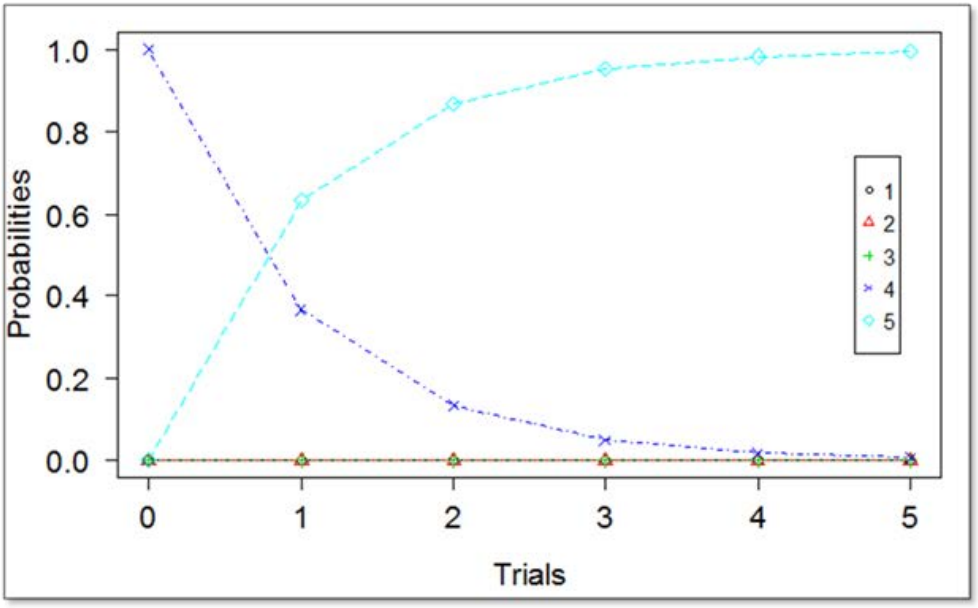

Figure 23. Behavior of Flow Table Movement for F_4

6.5 Implications

Currently networks are very static. Network entities such as firewalls, load balancers, and routers are all connected in a deterministic pattern. Policies are programmed into them to do

specific actions for the traffic coming from various entry points, such as an external web, virtual private network (VPN), and intranets. If a scenario is envisioned such that the firewall rules can be programmed into one or more tables, then the functionality is assigned for those tables with controllers that may be performing only one type of service (i.e., firewall). Similarly, this case can be applied for load balancers, routers, and so on. This model is difficult to implement in today's static networks; however, it can be easily accomplished with the proposed model for multiple controllers.

Similarly, this technique can be applied in broader aspects such as service providers, users, and enterprises. If an OpenFlow switch exists on an end device, such as a mobile phone or laptop, then user preferences and corporate policies can be programmed into the switch tables with the proposed model for multiple controllers, thus obtaining higher efficiency and flexibility. By using multiple controllers on a single switch with multiple tables, it is possible to provide interesting network services.

This chapter dealt with providing the packet process for the flow path modeled in Chapter 5. It represented a simulation model for finite-state proven mathematically in Chapter 5. It illustrated the behavior of flow table movement thereby validating the simulation. It also supported the proof of concept with a demonstration executing an experiment using a laboratory test bed.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

Traffic engineering is an art and a science. Exponential growth of connected and mobile devices forces us to efficiently use network services. OpenFlow based architecture helps to dynamically program network data paths. Implementing OpenFlow technology enables traffic to be dynamically programmed with flows to provide various network services at the data path layer. This research utilized OpenFlow architecture to model network service programmability for mobile devices. The model included incorporating an OpenFlow data path on a mobile device and enabling end-to-end programmability from the service provider to a user's mobile device. The model was enhanced to incorporate additional players, such as an enterprise that would want to control certain aspects on the user's mobile device. With this kind of approach, expensive mobile device management solutions (such as device space utilization, memory, power consumption, and network access) can be significantly simplified and value-added network connectivity services can be provided with higher service quality. OpenFlow controllers were utilized to represent various network players with the goal of controlling the data path on a mobile device. It is believed that this is the beginning of a new emerging class of models for dynamic traffic engineering.

7.2 Future Work

In this research, due to infrastructure limitations, only wireless-based mobile devices were used to model successfully. The use of a cellular network with various connectivity options such as 3G, LTE, and 4G could be used to further validate this model. Also, the use of OpenFlow controllers to represent various network elements such as firewall, load balancer, and

routers are yet another way to validate this research model. Mobile devices today provide various network radio connectivity channels such as Wi-Fi, WiMax, Bluetooth, and cellular, which, on a switch, can be on the WAN side with a seamless TCP/IP on the switch port side. This heterogeneity model enables advanced service quality to be provided over connectivity options (such as call connectivity, call handoff, and application QoS—voice, data, video). All of these combined is simply the beginning of things to come in the new area of software defined networks.

REFERENCES

REFERENCES

[1] Cisco Systems Inc., "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html (cited February 15, 2014).

[2] IEEE Global History Network, "Telephone Operators," http://www.ieeeghn.org/wiki/index.php/Telephone_Operators (cited February 15, 2014).

[3] Das, S., Parulkar, G., and McKeown, N., "Unifying Packet and Circuit Switched Networks," *IEEE Globecom Workshop*, 2009.

[4] Javed, U., Han, D., Caceres, R., Pang, J., Seshan, S., and Varshavsky, A., "Predicting Handoff in 3G Networks," *ACM MobiHeld*, vol. 45, issue 3, December 2011.

[5] Hirannaiah, R., Jasti, A., and Pendse, R., "Influence of Codecs on Adaptive Jitter Buffer Algorithm," *IEEE Vehicular Technology Conference (VTC)*, Fall 2007.

[6] Bansal, M., Katti, S., Mehlman, J., and Levis, P., "OpenRadio: A Programmable Wireless Dataplane," *Proceedings of First Workshop on Hot Topics in Software Defined Networks*, ACM, HotSDN 2012.

[7] Fernandez, M. P., "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," *Proceedings of IEEE 27th International Conference on Advanced Information Networking and Applications*, AINA 2013.

[8] Pahlavan, K., Prashant Krishnamurthy, and Hatami. A., "Handoff in Hybrid Mobile Data Networks," *IEEE Journal on Personal Communications*, vol. 7, issue 2, 2000.

[9] Gude, N., Koponen, T., Pettit, J., and Pfaff, B., Casado, M., McKeown, N., Shenker, S., "NOX: Towards an Operating System for Networks," *In SIGCOMM CCR*, 38(3), 2008.

[10] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S., "Onix: A Distributed Control Platform for Large-scale Production Networks," in *OSDI*, October 2010.

[11] Cai, Z., Cox, A., and Ng, T., "Maestro: A System for Scalable OpenFlow Control," Technical Report TR10-08, Rice University, December 2010.

[12] Erickson, D., "Beacon: A Java-based OpenFlow Controller," *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13.

[13] Open Networking Foundation, *OpenFlow Switch Specification, v.1.1.0.* http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf (cited February 15, 2014).

[14] Canini, M., Venzano, D., Peresini, P., Kostic, D., and Rexford, J., "A NICE way to Test OpenFlow Applications," in *NSDI*, 2012.

[15] Kim, H., Voellmy, A., Burnett, S., Feamster, N., and Clark, R., "Lithium: Event-Driven Network Control," Technical Report GT-CS-12-03, Georgia Institute of Technology, 2012.

[16] Tootoonchian, A., and Ganjali, Y., "HyperFlow: A Distributed Control Plane for OpenFlow," *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN 2010.

[17] Pham, X. T., Thoai, N., and Kuonen, P., "A Collaborative Model for Routing in Multi-domains OpenFlow Networks (RMOF)," *International Conference on Computing Management and Telecommunications* (ComManTel), 2013.

[18] Open Networking Foundation, *OpenFlow Protocol Specification, v.1.3.2*, https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.2.pdf (cited February 15, 2014).

[19] Sherwood, R., Gibb, G., Yap, K-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G., "Flow Visor: A Network Virtualization Layer," OPENFLOW-TR-2009-1, Deutsche Telekom Inc. R&D Lab, Stanford University, Nicira Networks, October 2009.

[20] Mirzaei, S., Bahargam, S., Skowyra, R., Kfoury, A., and Bestavros, A., "Using Alloy to Formally Model and Reason about an OpenFlow Network Switch," Boston University's Technical Reports, July 2013.

[21] Kuzniar, M., Peresini, P., Canini, M., Venzano, D., and Kostic, D., "A SOFT Way for OpenFlow Switch Interoperability Testing," *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12.

[22] Al-Shaer, E., and Al-Haj, S., "FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures," in *ACM SafeConfig*, October 2010.

[23] Khurshid, A., Zou, X., Zhou, W., Caesar, M., and Godfrey, P. B., "Veriflow: Verifying Network-wide Invariants in Real Time," *In Proceedings of the first workshop on Hot Topics in Software Defined Networks*, HotSDN '12.

[24] Pan, H., Guan, H., Liu, J., Ding, W., Lin, C., and Xie, G., "The FlowAdapter: Enable Flexible Multi-table Processing on Legacy Hardware," *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in SDN*, HotSDN'13.

[25] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., "Openflow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, April 2008.

[26] Vaughan-Nichols, S. J., "OpenFlow: The Next Generation of the Network," *IEEE Computer Society*, vol. 44, issue 8, 2011.

[27] Kontesidou, G., and Zarifis, K., *Openflow Virtual Networking: A Flow-Based Network Virtualization Architecture*, M.S. Thesis, School of Information and Communication Technology (ICT), Royal Institute of Technology, Stockholm, Sweden 2009.

[28] Kim, B., and Henry, P., "Directions for Future Cellular Mobile Network Architecture," Tech. Rep. TD:101033, AT&T Labs – Research, December 2012, http://www.research.att.com/techdocs/TD_101033.pdf.

[29] Wang, H., Katz, R., and Giese, J., "Policy-Enabled Handoffs across Heterogeneous Wireless Networks," *Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[30] Weisstein, E. W., "Permutation," from Wolfram MathWorld http://mathworld.wolfram.com/Permutation.html (cited February 15, 2014).

[31] Cassandras, C. G., and Lafortune, S., *Introduction to Discrete Event Systems*, Springer Publications 2008.

[32] Grinstead, C.M., and Laurie Snell, J., "Markov Chains," chapter 11 in *Introduction to Probability*, American Mathematical Society 2006.

[33] Weisstein, E.W., "Stochastic Matrix," from Wolfram MathWorld http://mathworld.wolfram.com/StochasticMatrix.html (cited February 15, 2014).

[34] LINC-Switch, http://flowforwarding.github.io/LINC-Switch/ (cited February 15, 2014).

[35] American Portwell, http://www.portwell.com/openflow/ (cited February 15, 2014).

[36] Pace, L., *Beginning R: An Introduction to Statistical Programming*, Apress Publications 2012.

[37] Gardener, M., *Beginning R: The Statistical Programming Language*, Wrox Press 2012.

[38] R Project, *The Comprehensive R Archive Network (CRAN)*, http://cran.r-project.org (cited June 6, 2014).

[39] Jay Kerns, G., *Introduction to Probability and Statistics Using R (IPSUR)*, First Edition, 2011.

[40] Crawley, M. J., *The R Book*, John Wiley & Sons 2007.

APPENDICES

APPENDIX A

OUTPUTS OBTAINED FROM DEMONSTRATION

Connection of controller with the switch output –

```
15:47:53.874 [info] Connected to controller 127.0.0.1:6633/0 using OFP v4
```

On Controller side

A flow added is indicated by the following output:

```
15:48:35.059 [info] Adding new entry:   1 | :0:11:11:D3:89:7E |
<<0,17,17,211,137,126>>
15:48:46.517 [info] Adding new entry:   2 | :F0:DE:F1:12:9E:6E |
<<240,222,241,18,158,110>>
```

Switch side outputs

The details of flow entry that matched the packet flow are listed in the flow table. Once the flow entries are matched according to the packet headers, the instructions are applied and packet is processed accordingly.

```
 [{flow_entry,
     {1,#Ref<0.0.0.505>},
     1,
     {ofp_match,
         [{ofp_field,openflow_basic,eth_type,false,<<8,0>>,undefined},
          {ofp_field,openflow_basic,ip_proto,false,<<6>>,undefined},
          {ofp_field,openflow_basic,ipv4_src,false,
              <<192,168,0,79>>,
              undefined}]},
     <<0,0,0,0,0,0,0,0>>,
     [],
```

```
    {1392,335274,457711},
    {infinity,0,0},
    {infinity,0,0},
    [{ofp_instruction_apply_actions,2,
         [{ofp_action_set_field,13,
              {ofp_field,openflow_basic,tcp_dst,false,
                  <<31,43>>,
                  undefined}},
          {ofp_action_output,16,2,no_buffer}]}]},
{flow_entry,
    {1,#Ref<0.0.0.500>},
    1,
    {ofp_match,
        [{ofp_field,openflow_basic,eth_type,false,<<8,0>>,undefined},
         {ofp_field,openflow_basic,ipv4_src,false,
             <<192,168,0,68>>,
             undefined}]},
    <<0,0,0,0,0,0,0,0>>,
    [],
    {1392,335274,257089},
    {infinity,0,0},
    {infinity,0,0},
    [{ofp_instruction_apply_actions,2,
         [{ofp_action_set_field,13,
              {ofp_field,openflow_basic,ipv4_dst,false,
                  <<10,0,0,68>>,
                  undefined}},
          {ofp_action_output,16,2,no_buffer}]}]},
{flow_entry,
```

```
    {0,#Ref<0.0.0.629>},
    0,
    {ofp_match,
        [{ofp_field,openflow_basic,eth_dst,false,
             <<240,222,241,18,158,110>>,
             undefined}]},
    <<0,0,0,0,0,0,0,0>>,
    [],
    {1392,335326,519400},
    {infinity,0,0},
    {infinity,0,0},
    [{ofp_instruction_write_actions,4,
         [{ofp_action_output,16,2,no_buffer}]}]},
{flow_entry,
    {0,#Ref<0.0.0.620>},
    0,
    {ofp_match,
        [{ofp_field,openflow_basic,eth_dst,false,
             <<0,17,17,211,137,126>>,
             undefined}]},
    <<0,0,0,0,0,0,0,0>>,
    [],
    {1392,335315,61238},
    {infinity,0,0},
    {infinity,0,0},
    [{ofp_instruction_write_actions,4,
         [{ofp_action_output,16,1,no_buffer}]}]},
{flow_entry,
    {0,#Ref<0.0.0.495>},
```

```
0,
{ofp_match,[]},
<<0,0,0,0,0,0,0,0>>,
[],
{1392,335274,52373},
{infinity,0,0},
{infinity,0,0},
[{ofp_instruction_apply_actions,2,
     [{ofp_action_output,16,controller,no_buffer}]}]}]
```

APPENDIX B

SIMULATION CODE FOR STEADY STATE MODEL USING R

R Program Code:

```
CalculateTransitionMatrix <- function(replications, n)  {

  # Computes probability transition matrix P[i,j] for flow table
  # movement.
  # Args:
  #   replications: No. of replications.
  #   n: Sample size.
  # Returns:
  #   Probability transition matrix of 5x5.
  #
  # Trial:
  #   datatset: Takes data set (or, flow tables).
  #   Here 5 flow tables A, B, C, D, E are selected from the
  #   sample size to calculate the first order transition matrix.
  #   replace: If TRUE, print Transition Matrix else no. Default
  #  is TRUE.
  #

  dataset <-data.frame(replicate(replications, sample(c("A", "B",
"C","D","E"), size = n, replace=TRUE)))

  Compute1stOrderMarkovChain <- function(X, TransProb=T)  {

    # Function to compute first-order Markov chain
    # Calling the function based on the data sets assigned
    # Args:
    #   X: sample value to be chosen
    #   TransProb: Produces estimated transition probability (prob =
TRUE)
    # Returns:
    #  Each row corresponding to the Markov chain computation
    #
    # Trial: Output of transition matrix executed for different trials

    tt <- table( c(X[,-ncol(X)]), c(X[,-1]) )
    B <- round(upper.tri(matrix(1,5,5),diag=TRUE))
    mat <- as.vector(tt)*B
    if(TransProb) tt <- mat/ rowSums(mat)
    tt
  }
  mat1 <- Compute1stOrderMarkovChain(as.matrix(dataset))
}

Trial1 <- CalculateTransitionMatrix(20,100)
```

```
Trial2 <- CalculateTransitionMatrix(20,100)

# Assigning one of the transition probability matrix got from the
# trial runs to P. The entry P[i,j] gives the probability of moving
# from state i to state j. Here, Trial1 run was considered.
# Calculating the row of P sum and column sum.

P <- Trial1
rowSums(P)
colSums(P)

# Probability of a controller managing specific flow table actions

ntransition.P <- function(x, P, n) {

  # function ntransition.P to calculate
  # n-step transition probability of moving from state i to state
  # j in n steps. This refers to, the probability of the action of
  # moving from flow table 1 to flow table 3 in 6th or 10th step.
  # Args:
  #   x: state vector, x[i] is the probability of being in state i
  #   P: Transition probability matrix
  #   n: number of steps advancing the system.

  res <- matrix(NA, n+1, length(x))
  res[1,] <- x
  for (i in seq_len(n))
    res[i+1,] <- x <- x %*% P
  res
}

# Calculating the transition probabilities for each flow table
# movement
# Vector x is [1 0 0 0 0] multiplying with transition
# probability matrix P and number of steps specified.

n <- 5
F1 <- ntransition.P(c(1, 0, 0,0,0), P, n)
F2 <- ntransition.P(c(0, 1, 0,0,0), P, n)
F3 <- ntransition.P(c(0, 0, 1,0,0), P, n)
F4 <- ntransition.P(c(0, 0, 0,1,0), P, n)
F5 <- ntransition.P(c(0, 0, 0,0,1), P, n)

matplot (0:n, cbind (F1), type="b",
xlab="Trials",ylab="Probabilities",las=1, pch=c(1,2,3,4,5),
col=c(1,2,3,4,5), lwd = 1.5 )
matlines(0:n, cbind(F2), lwd=1.5)
matlines(0:n, cbind(F3), lwd= 1.5)
matlines(0:n, cbind(F4), lwd=1.5)
matlines(0:n, cbind(F5), lwd=1.5)
legend("right", inset=.05, legend=c("1","2","3","4","5"),
pch=c(1,2,3,4,5) col=c(1,2,3,4,5), horiz=FALSE, cex = 0.7)
```

Results of F_2, F_3, F_4 and F_5:

F_2

```
     [,1]       [,2]       [,3]       [,4]      [,5]
Trial0  0 1.0000000000 0.00000000 0.00000000 0.0000000
Trial1  0 0.2056737589 0.28368794 0.25531915 0.2553191
Trial2  0 0.0423016951 0.15159644 0.24561827 0.5604836
Trial3  0 0.0087003486 0.06183079 0.15388506 0.7755838
Trial4  0 0.0017894334 0.02279219 0.08020388 0.8952145
Trial5  0 0.0003680395 0.00799952 0.03778162 0.9538508
```

F_3

```
       [,1] [,2]       ,3]       [,4]      [,5]
Trial0    0    0 1.000000000 0.00000000 0.0000000
Trial1    0    0 0.328703704 0.35185185 0.3194444
Trial2    0    0 0.108046125 0.24421626 0.6477376
Trial3    0    0 0.035515161 0.12724909 0.8372357
Trial4    0    0 0.011673965 0.05899094 0.9293351
Trial5    0    0 0.003837276 0.02566189 0.9705008
```

F_4

```
        [,1] [,2] [,3]     [,4]      [,5]
Trial0     0    0    0 1.000000000 0.0000000
Trial1     0    0    0 0.365384615 0.6346154
Trial2     0    0    0 0.133505917 0.8664941
Trial3     0    0    0 0.048781008 0.9512190
Trial4     0    0    0 0.017823830 0.9821762
Trial5     0    0    0 0.006512553 0.9934874
```

F_5

```
        [,1] [,2] [,3] [,4] [,5]
Trial0     0    0    0    0    1
Trial1     0    0    0    0    1
Trial2     0    0    0    0    1
Trial3     0    0    0    0    1
Trial4     0    0    0    0    1
Trial5     0    0    0    0    1
```