INNOVATION IN SOFTWARE DEFINED NETWORKING AND THROUGHPUT OPTIMUM SCHEDULING ALGORITHM

A Thesis by

Fahad Naeem Khan

Bachelor of Engineering, NUST Pakistan, 2012

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
Master of Science

December 2014

INNOVATION IN SOFTWARE DEFINED NETWORKING AND THROUGHPUT STABLE SCHEDULING ALGORITHM

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Networking.

_____

Pu Wang, Committee Chair

_____

Murtuza Jadliwala, Committee Member

_____

Jibo He, Committee Member

To my friends and family

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my thesis advisor Dr. Pu Wang, who has had the attitude and the substance of a genius. His continues help, covering attitude toward research and handwork, and the excitement he possesses in teaching helped me to come up with results of my thesis. I believe that without his guidance and persistent help this thesis would not have been a possibility thereby I thank him for his efforts and help throughout my research. I would also like to thank my committee members Murtuza Jadliwala and Jibo He for taking out time from their busy schedule in order to review my research work and thesis.

Moreover my deepest regards to ON.LAB for offering me the CoOp where I was able to learn and gain extensive experience in the intense research environment related to Software Defined Networks (SDN), OpenVirtex (OVX) and Open Networking Operating System (ONOS) specifically Guru Parulkar and Bill Snow for giving me the wonderful opportunity.

I must thank my team members at ON.LAB with whom I worked and got the opportunity to learn a lot, most importantly Ayaka for being my mentor and for helping me learning OpenVirteX, Ali Al-Shabibi, Marc De Leenheer for guidance during the designing and development OVX Testing Framework, Saurav Das for believing in me and making a part of ONOS-Segment Routing team, Tom Tofigh as he brought me over to the packet optical use-case of ONOS and motivated me to go out of the way and also for encourage me in writing this thesis.

Last but not least it is my utmost duty to thank my friends and family who have been of great help, without their love, affection, prayers and motivation it would not have been possible for me to try, attempt and succeed in working on my research.

Abstract

The networking industry, which has not undergone a substantial change within the past 20 or so years, is being disrupted by a technology known as Software Defined Networks (SDN). Even though SDN has taken the networking world by a storm, but consensus on a specific standard has caused the industry to be wary in embracing the technology. This thesis addresses this issue in a four faceted manner. By collaborating with Open Networking Laboratory (ON.LAB) we developed three major OpenFlow [12] based SDN tools. Moreover, we proposed maximum weight α scheduling algorithms for network switches, whose performance is evaluated in a SDN environment.

The first tool is the OVX Testing Framework. OpenVirteX (OVX) [9] is a network hypervisor that creates virtual networks on top physical network by virtualizing OpenFlow. OVX just like FlowVisor [13] acts as a controller to data-plane and to make sure isolation between different virtual networks, OVX rewrite packet header. This very tough and complex task and language specific testing tools and technique like JUnit test etc. are not enough to makes sure OVX correct implementation. To solve this problem we designed and developed OVX Testing Framework, a tool to test OVX end to end delivery.

The second tool is based on Open Networking Operating System (ONOS) [10]. ONOS is a distributed SDN controller. It is best known for scalability, higher availability and high performance. It provides useful network abstractions like network graph at northbound and plugins mechanism at southbound to allow different southbound protocols like OpenFlow, NetConf. ONOS has many applications and use cases. One of its use case involves Segment Routing. Segment Routing is the latest routing mechanism to route traffic effectively. We develop Command Line Interface (CLI) for ONOS Segment Router application thats allows the routing application to retrieve switch statistic using OpenFlow v1.3 messages. In addition CLI also allows application to create tunnels and policies. To develop CLI we develop new APIs for ONOS as existing ONOS API does not support OpenFlow v1.3. We use sdncon [16] for formatting and displaying the data.

The third tool we propose aims to achieve convergence of packet and optical networks. The functionality this tool implements is amalgamation of both optical and logical layers to optimize the traffic flow, based on a cross layer control infrastructure. The SDN controller acts as a maintainer of Performance Management, which handles TE tools like Path Computation Element. This provides flexibility in access and control of traffic-flow forwarding and bandwidth assignment on both of the layers. In this project we create an emulator for both packet and optical switches. We used mininet [8] for packet switches and Linc-oe, an open source optical switch emulator for optical switches and ONOS as controller. Using this emulator, one can emulate real world scenarios, in packet-optical hybrid setting without any expense.

We propose MWA scheduling algorithm provides the throughput optimality to input queued switch under the presence of hybrid data traffic for all admissible arrival processes. We are simulating our results in SDN environment which is constituted by mininet, OpenVirteX, OpenvSwitch [11], OpenFlow protocol and two POX [15] controllers.

TABLE OF CONTENTS

Chapter **Page**

LIST OF FIGURES

Figure **Page**

Chapter 1

INTRODUCTION

SDN is about transformation of current practices to new way of network services and management. SDN provides the hierarchical control that next generation networks need. It will enable programable forwarding plane at minimum, and will put forth opportunities to mash up network functions with the data flows. This thesis brings about how SDN is transforming the way networking is done, slowly but surely. In particular, we designed and developed the OpenVirteX testing framework, an essential tool to test the end to end delivery of packets. Moreover we integrate packet-optical convergence use case of ONOS. We also designed and developed a CLI for a segment router in a SDN setting; last but not least we proposed a throughput optimization scheduling algorithm.

It is obvious that next generation networks are all about enabling the users and applications through open interfaces. These interface and application examples are focused in this thesis. The building blocks of SDN span across many features such as graph abstractions, network partitioning, and programming of the network. Specifically OVX which is one of the core projects of this effort is highly critical in the future networks. OVX is a tool which provides every tenant with their own virtual SDN. It enables multi-tenancy and allows security by isolation of the user's traffic. For the data path, it acts as a controller, while from the controller it acts as a data path between the communicating nodes. OVX rewrites the packets it receives and could operate in more efficient way, and for this purpose we developed unit testing framework for OVX. It operates by sending packets to OVX as a controller and also as a switch, and using this framework, we can compare the response given by OVX with the expected response that OVX should give. This ensures end to end delivery.

ONOS is an open network operating system developed at ON.LAB. It is a scalable, multi instance distributed network operating system. It is known for its high performance and high scalability. The architecture provides north bound interfaces to applications very much like a typical OS, and on the South bound, it provides plugins for communicating to open flow switches and perhaps other interfaces in the future. One of the use cases for ONOS is segment routing which is a technique by which the path followed by a packet is encoded in the packet itself. It uses a specific kind of shortest path first algorithm for forked routes, which optimizes routing efficiency. We designed and developed a CLI for segment routing app of ONOS, to pass

control commands into it and to fetch the statistics of the router using Open 1.3 statistic messages. We also developed a new Application Programming Interface (API) to achieve these goals. Last but not the least we use SDNcon, an extension of our API to format the ingress responses from the router. Another use case for ONOS is the convergences of Packet-Optical Networks. This is a multi-layer SDN control app for optimization across packet and optical networks. The issue this use case solved was creating an interaction between packet and optical layers. The intelligence of ONOS, coupled with the cross-layer interaction increases the efficiency of routing in a Wide Area Network greatly.

Last but not the least we proposed throughput optimal scheduling algorithm for input queued switch under the presence of hydride data traffic. It is well known [3] that maximum weight matching (MWM) algorithms perform poorly under the presence of heavy tailed (HT) traffic. Lots of work [2],[3],[4] has been done in wireless domain to solve delay and throughput stability of the queuing network under the presence of HT traffic. It has also been proven [3] that maximum weight-α (MWA) scheduling algorithm is delay stable in the presence of HT for input queued switch. But there is no or very little amount of work that has been done for the throughput optimality of input queued switch under the presence of hybrid (combination light tailed (LT) and HT) data traffic. In the last section we show that MWA scheduling algorithm provides throughput optimality for all admissible arrival rates in input queued switch under the presence of hybrid traffic if we choose the value of α parameter according to tail coefficient. We prove throughput optimality using Lyapunov optimization, a well-known technique to prove system stability [7], [1] and we simulate our results in Software Defined Network (SDN) settings using *mininet* [8], *OpenvSwitch* [11], *OpenVirteX*(OVX) [9] two *POX* controllers.

Remaining of this thesis is as following. Chapter 2 describes all the SDN projects that we accomplish by collaborating with ON.LAB. Specifically section 2.1 describes OVX, testing OVX and the design and implementation of OVXTesting Framework, in section 2.2.3 we describe Segment Routing application of ONOS with design and implementation of Command Line Interface (CLI) for Segment Routing application. Section 2.3 describes convergence of packet optical network use case, multi-layer SDN controller, emulation of packet and optical networks and its end to end integration. In chapter 3 we propose the maximum weight α scheduling algorithm, which can achieve throughput optimality in the challenged heavy-tailed environment. Then, we evaluate its performance in the SDN settings. In chapter 4 we conclude this thesis.

Chapter 2

SDN-ENABLED TRAFFIC ENGINEERING SOLUTIONS

2.1 OVX TESTING FRAMEWORK

2.1.1 Introduction

In this section we give a brief introduction about OpenVirteX (OVX), current techniques to test programs and problems of testing OVX using current techniques. We also explains the design and implementing of OVX Testing Framework. In the end we explain why OVX Testing Framework is the solution to those problem and some tests written in this framework.

2.1.2 Motivation

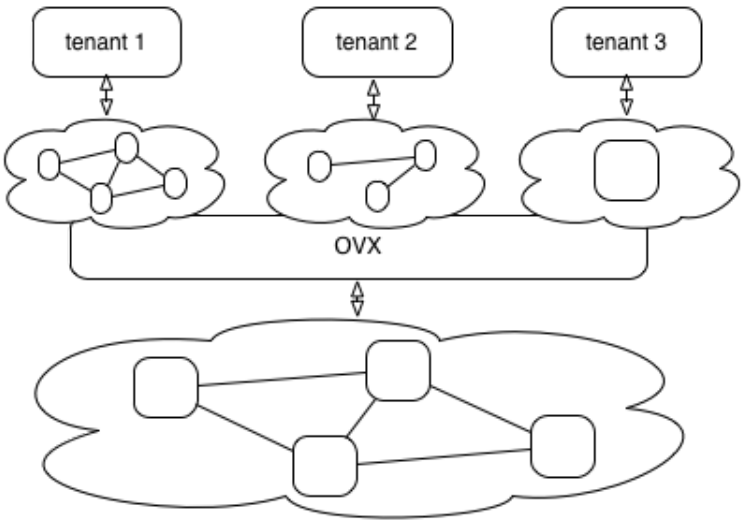

Figure 2.1: Overview of OVX

OVX is a network hypervisor that can create multiple virtual and programmable networks on top of a

single physical infrastructure. Each tenant can use the full addressing space, specify their own topology, and deploy the network OS of their choice. Networks are reconfigurable at run-time, and OVX can automatically recover from physical network failures. OVX like FlowVisor acts as a controller to data plane as well as acts as a data plane to different controllers connected to virtual networks created by OVX. Figure 2.1 gives an overview of OVX. OVX rewrites the OpenFlow messages, creates physical and virtual elements like switches, ports and keeps the mapping between virtual and physical elements for each tenant. In sum, OVX virtualizes OpenFlow or creates virtual Software Defined Networks (vSDN). As OVX is written in Java we used to test OVX's features using *JUnit* test, a unit testing framework for Java. JUnit testing is good to test java classes, java class methods. However to make sure correct end to end delivery of the message JUnit and other unit testing framework is not enough.

2.1.3 Challenges

Developing testing-framework for OVX brings up several challenges that we need to mitigate. OpenVirteX acts as a controller to switches. In particular, because OpenVirteX needs to see a physical data plane, we must create a fake data plane that should give an illusion to OpenVirteX that there is a physical data plane at southbound interface. Creating fake data plane is a challenging issue that we should explore further in the later sections. Furthermore, OpenVirteX acts as a data plane to northbound interface or the controllers, thereby we need to create fake controllers that can connect to the virtual networks created by OpenVirteX. The OpenFlow handshake between the northbound and southbound interface with OpenVirteX needs to be implemented and we need to figure out a way how the fake controllers and fake data plane will complete the OpenFlow handshake with OpenVirteX. The handling of Link Layer Discovery Protocol (LLDP) packet is big challenge. OpenVirteX discovers links between the switches by periodically sending LLDP packets to all the ports of all the discovered switches. So we need to handle periodic LLDPs sent by OpenVirteX where the default time period of sending LLDP packets is twenty seconds. To test OVX server API we need to create a client side of API. The most challenging part in designing OVX Testing Framework is to send messages to OVX and compare OVX response, which includes message to and from specific fake switch(s) or controller(es).

2.1.4 Our Solution

The solution to the problems discussed in the previous subsection is OVX Testing Framework. The OVX Testing Framework is built on top of FlowVisor testing framework which itself is built on of top OpenFlow test. All of them are written in python. The key idea is to run OVX and give OVX illusion that it is connected to data plane and controllers and then we can send OpenFlow messages to OVX and compare the response of OVX with expected message.

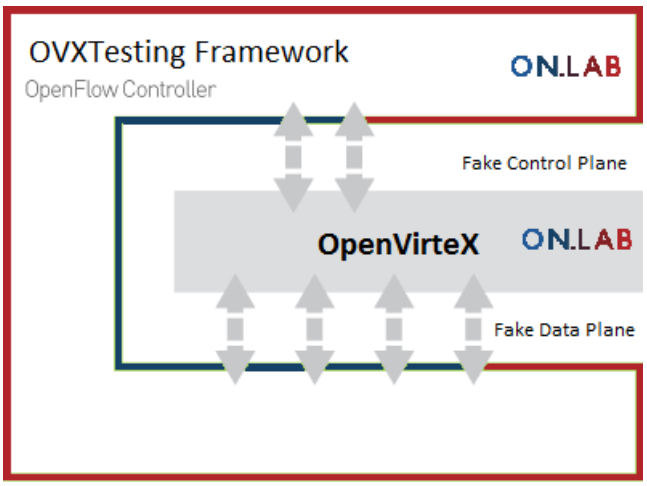

Figure 2.2: Design of OVX Testing Framework

2.1.5 Design and Implementation of OVX Testing Framework

The basic idea of the OVX Testing Framework is to the create communication channel at the north and south bound of OVX to communicate with OVX and give OVX illusion that these channels are connected to either controller or switch. We create fake data and control plane around OVX using sockets and OpenFlow handshake.

2.1.5.1 Fake data plane

OVX discovers switch(es) through OpenFlow handshake and links between switches by sending LLDP as packet-out to one switch for all ports and receiving LLDP packet-in replies from the neighboring switches. If OVX doesn't receive reply for LLDP, OVX assumes that there is no link at that particular port. We use the same procedure to give illusion to OVX that there are switches and links between them by simply creating sockets and sending/receiving OpenFlow messages and handling LLDP packets. When writing test, programmers need to specify topology in JSON format specifying number of switches and links between these switches.

2.1.5.2 Fake controle plane

Once the fake data plane is discovered by OVX, virtual networks can be created using OVX API. Newly created virtual network should point or connect to one of the fake controllers. Default implementation has two fake controllers running on localhost at port number 54321 and 54322 which can be easily modified. The controller of the created virtual network(s) should be one of these fake controllers. When fake controller connects to virtual network, OpenFlow handshake takes place between fake controller and OVX as data plane.

2.1.5.3 ofmsgSndCmp

The most important function of testing framework is *ofmsgSndCmp* which stands for OpenFlow (of) message (msg) send (Snd) and compare (Cmp). This function takes many arguments, and two mandatory and the most important ones are *snd_msg* and *exp_msg*. *snd_msg* contains information about the sender i.e specified

controller or switch and OpenFlow message, on the other hand *exp_msg* contains a list of receivers and expected OpenFlow reply on each receiver by OVX. We just need to create message and expected reply and send it to OVX using ofmsfSndCmp with sender and receiver(s) information. If the message is as expected, test is passed else test is failed. We can see the difference in the expected and received message in the logs. This function also supports no reply, where we just need to send *None* as expected message. We can also see all the messages using wireshark if necessary.

2.1.6 Writing Tests

Before writing a test for specific scenario, we first need to create an environment for that, i.e. determining how many fake controllers and switches we need and also the links between these switches. This is done by using JSON file. We need to create a specific JSON file containing above information and points framework to this JSON file to let framework know what our topology is.

As fake controller(s) and switch (es) are nothing but just sockets, we can send any message to OVX. OVX Testing Framework provides bunch of utility functions and libraries to easily create any OpenFlow message. So in a nutshell, to write a test we need to create request message, expected response, specify request sender, response receiver i.e. controller(s) or switch(es) and use ofmsgSndCmp function to send the message to OVX Testing Framework. As explained earlier ofmsgSndCmp sends the message to OVX from specified controller or switch and compares the expected response at specified controller(s) or switch (es).

2.1.7 Test and Results

Using OVX Testing Framework we wrote more than 30 tests, testing different behaviors of OVX in different scenarios such as how OVX behaves at edge switches, Big-Switch cases, virtual link cases, API tests, flow-mod case, packet-out cases, packet cases and many more. We found many bugs for example, OVX virtual link recovery and route recovery was not working as expected. When we were testing OVX API, we found a bug in OVX that getPhysicalFlow table return physicalFlow in normal circumstances but when OVX receive (explicitly) FlowStatisticRequest from tenant it doesn't return physicalFlow rather it returns virtualFlow. Also OVX rewrites network source and network destination address in the match fields without checking if they are wildcard fields or not when sending flow-mods. These fields do not show in physical flow table like on OVS switches because OVS switches only show non-wildcard filed in flow table.

The most important thing about these tests and results are that we cannot test OVX with such depth using existing techniques, but OVX Testing Framework allows us to do so.

2.1.8 Summary

OVX is a complex software, and in fact it's a network hypervisor. It does complex tasks to implement isolation between the tenants. When a packet comes to OVX, OVX rewrites the packet header and sends it to the appropriate destination. OVX sends messages to both data path and controllers connected to it. Unit test are

not good enough to ensure correctness of OVX behavior and we cannot test end to end behavior of OVX with unit tests. To overcome this problem we designed and developed OVX Testing Framework that gives OVX illusion that it is connected to switches at southbound and controllers at northbound by simply implementing OF handshake between fake controllers and OVX and between fake switches and OVX. OVX Testing Framework provides some utility functions and libraries to create OF messages. Using ofmsgSndCmp we can send message to OVX and compare the response from it with the expected message. This allows us to test the end to end functionality of OVX.

2.2 CLI for ONOS Segment Routing (SR) application

2.2.1 Introduction

ONOS is an open network Operating System (OS) developed at ON.LAB which is a scalable, multi instance distributed network OS. ONOS provides northbound APIs that other network application can use to implement their functionalities and at southbound it provides plugins mechanism to implement various southbound protocols like OpenFlow (OF), NetConf etc. ONOS has many use-cases such as SDN-IP in which ONOS can speak BGP and hence talk to legacy IP network, and Packet-Optical network convergence in which ONOS talks to both packet and optical layers. By controlling both packet and optical layers, ONOS enables us to utilize optical network efficiently in SDN setting. Furthermore ONOS is also used to implement Segment Routing (SR) which we will explain in the next section. In this project we design and develop Command Line Interface (CLI) for ONOS Segment Routing application to allow user to get the switch information by sending OpenFlow v1.3 statistic messages and to create tunnels and policies.

2.2.2 Motivation

Segment Routing (SR) is the new routing technique and can be used to utilize MPLS based network effectively as it leverages the source routing paradigm. The routing operation is done by adding list of instructions at source node. This list of instructions are known as segments. A segment can represent any instruction, which can be topological or service-based. Segment can be local to router or global within the domain. SR allows to enforce a flow through any topological path and service chain while maintaining per-flow state only at the ingress node to the SR domain. It is very easy to apply segment routing in MPLS architecture without any change to forwarding plane. In this case MPLS label can encode as segments. An ordered list of segments is encoded as a stack of labels. Labels are processed from top to bottom and processed label is piped from the label stack. There are many benefits of Segment Routing for example, it uses ECMP shortest path where it is possible. We can implement scalable end to end polices to avoid default routing. It can be efficiently utilized by SDN and IP networks and many more.

One of the use case of ONOS is segment routing. For this use case ONOS needs a Command Line Interface (CLI) so that user can get the state and statistics of the switches. Furthermore in segment routing we can implement routing policies and specific tunnels. To do so segment routing application should support some mechanisms that allow user to define their own routing policies. For these purpose we develop CLI that uses OpenFlow v1.3 statistic messages to get the information from the switches and allows network administrators to configure and create their own policies.

2.2.3 Challenges

Design and implementation of CLI for ONOS brings some challenging issues. We discuss some of these challenges, which are dealt with in the upcoming sections. Currently, ONOS supports OpenFlow v1.0 but we need

OpenFlow v1.3 features like MPLS and groups. Thus we must implement OpenFlow v1.3 functionalities to ONOS. Our CLI must provide common CLI features like *command autocomplete*. Moreover the existing API of ONOS supports only OpenFlow v1.0, and we need to add new API for ONOS segment routing application that supports OpenFlow v1.3 messages. OpenFlow v1.3 messages are not serialized by default Java serializer, and thus we must create new serializers that can serialize OpenFlow v1.3 statistic messages into JSON format so that serialized response can be easily read by the CLI. Furthermore one of the major challenges is to display the response in a easily readable format for example in table format. In this case we use sdncon for formatting.

2.2.4 Design and Implementation of CLI

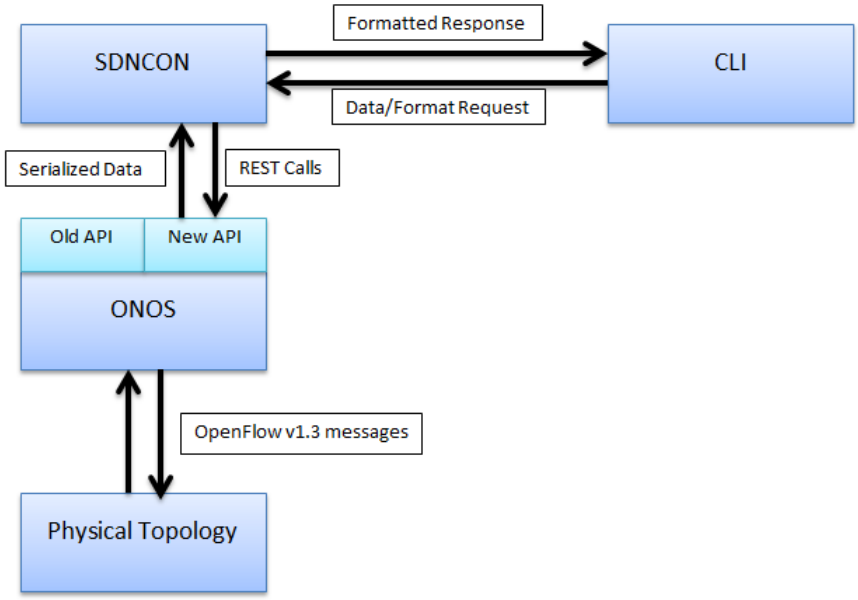

Figure 2.3: Design ONOS-Segment Routing CLI

There are three main parts of CLI including ONOS new API, sdncon and CLI itself. As explained earlier we have to add new API that supports OF13 so that we can get switch statistics from the switches using CLI. In addition CLI output should be easily readable so we need output in the table format to make it user friendly. For this purpose we are using sdncon. Moreover, CLI itself should be easy to use. It must support command autocomplete functionality. We used python2.7 to implement CLI.

We implement both front and back end of the ONOS API by implementing methods that send OpenFlow v1.3 message to ONOS, receive response and handle the request from the CLI in an appropriate way. We also develop new data serializer that converts the raw data into JSON format and sends JSON formatted response as CLI expects JSON formatted response.

SDNCON is the utility developed by Bigswitch Company. SDNCON provides some formats that can be used to display easy readable data. But we need more than that. So we modify SDNCON so that we can specify expected format. When the response comes from ONOS we dipslay it according to expected format.

CLI itself should be user friendly. To do so we implement command and data autocompletion, an old and very useful features of CLI. To implement command autocompletion CLI should have the knowledge or data about that particular command. For example, if a user wants to see the stats of a particular switch, when user presses tab after writing *show switch* we should show all the available switches. For this purpose we need to know how many switches are connected to ONOS. We get this information by sending REST call to ONOS and retrieve the data to display. Furthermore CLI supports all the commands that are needed by Segment Routing application. Figure 2.3 show the architecture on CLI.

2.2.5 CLI Workflow

CLI supports two types of functions. One is to retrieve statistics from the switches and the other is to configure and create tunnels and policies. For retrieving statistics the user types respected command on CLI, where command fires *GET REST* call to sdncon. SDNCON figures out that what is the required format of this command. Then sdncon fires a *GET REST* call to ONOS. ONOS reads the response, retrieves the data according to the request, serializes the data in JSON format and sends to sdncon. SDNCON parses the JSON response into required format and sends it to CLI for display. For creating policies and tunnels CLI sends *POST REST* call, and sdncon forwards the *POST REST* to ONOS REST API, where we have implemented the backend of creating policy and tunnel. If policy is successfully created. ONOS doesn't send any response message to CLI. In case of error, ONOS sends error message according to the error.

2.2.6 Summary

In this section we designed and developed Command Line Interface (CLI) for ONOS Segment Routing application. Segment Routing a is implemented using ONOS which was OpenFlow v1.0 enabled. However, since Segment Routing needs OpenFlow v1.3 features, the existing ONOS API sufficient to implement CLI. Therefore we developed new ONOS API that supports OpenFlow v1.3. In addition CLI should be easily readable, we used sdncon to implement data formatting. In particular, CLI sends request to sdncon, sdncon figures out the format and sends the request to ONOS and ONOS replies with JSON formatted data, which is read by sdncon and displayed by CLI.

2.3 Packet Optical Use Case ONOS

2.3.1 Introduction

Packet Optical use case provides a bandwidth (BW) calendaring application on the top of SDN where it emulates datacenter customers to request network capacity on demand. The bandwidth calendaring application is one of the ONOS applications that allows user to define the required bandwidth using web-portal. The service requests are compiled by ONOS. ONOS then installs flows on both packet and optical switches at the same time according to the service request.

It seems that this Proof Of Concept (POC) for packet optical packet-optical convergence is a big deal for telecom companies all over the globe. We believe that perhaps packet switching can not sustain itself for large data transport since the packet switching is expensive but switching at optical layer is much more granular. With control over both packet and optical layer the large data may not have switch back and forth between packet and optical layer, in fact we can program optical switches in a way that they may not have to send the data back to packet layer. This use case enables high flexibility for operators to perform enhanced traffic management functions for the large enterprise customers. Management and orchestration can even be given to the customers in certain cases.

2.3.2 Motivation

Today Packet and optical networks are managed independently. As a result the optical transport is always over provisioned to meet all possible failure scenarios and busy hour congestion. This has caused Operational Expenses to rise exponentially for carriers where they have to process much of their traffic at IP layer. Note that the beauty of this use case is that the actual network devices such as packet and optical switches are also emulated. For example futures students and researchers can create a large packet-optical topology on their laptop and test existing applications and create new ones. This use case enables them to converge the management of the packet and optical networks such that their customer can add capacity on demand. For example large DataCenters need to perform their own traffic management and reservation on demand. In addition, not all traffic needs to be treated the same way. For example certain traffic such as elastic data can wait in case of sudden failures. Therefore current practices for full redundancy at the transport layer are not necessary for elastic traffic.

2.3.3 Challenges

There is no optical switch emulator which has been developed before. So prototyping and designing this kind of switch emulator is a challenge in itself by providing all the required functionalities and capabilities to handle big data. The other challenge is to create a connection between packet switches and the optical switches. This is achieved by creating tap interfaces on optical switch process and connecting it to packet switch veth-pair. One of the biggest challenges is to discover links between optical switches by the OpenFlow controller (ONOS), as LLDP does not work with optical switches. We used JSON file for sending information about optical links to the controller. This use case consists of multiple components (mininet, OpenVSwitch as packet switch, Linc-oe as optical switch and ONOS as OpenFlow controller) and each component has its integrated functionalities in addition to some individual capabilities. Thus the end-to-end integration of each component is also challenge.

2.3.4 Multi-Layer SDN Controller

Most SDN' controllers today are only managing the packet layers. This project is about managing both packet and optical layer simultaneously. The multilayer SDN is communicating to both packet and optical switches

using OpenFlow interface. One of the main features of ONOS is to abstract the the physical network in a form of graph and represent it to the the applications. The applications will program the network graph instead of talking to individual switches. In this case the SDN as a controller for both packet and optical layer has to discover and manage both layers. In addition the SDN must be able to accept service requests from the applications and program the graph for both layers. Therefore when an application makes a request to connect point A to point Z with specific constraints, the SDN needs to either instruct a path computation engine to allocate resources for both layers. Also the SDN is responsible for processing equipment triggers such as device, link, and port failures to update the graph in real time. The applications will need to trust the network graph to represent the real network. Figure 2.4 shows an architecture diagram for a multilayer SDN that was used in this project.

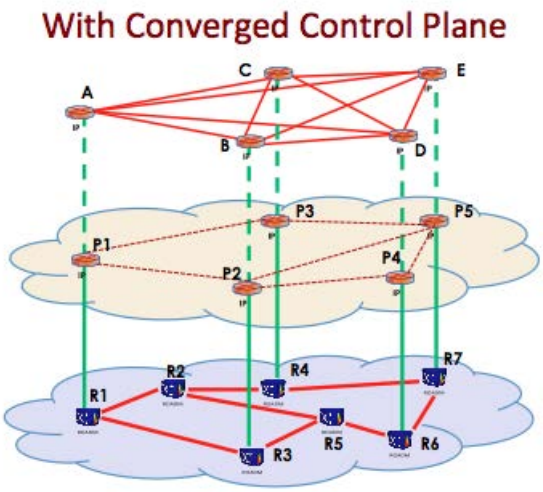

Figure 2.4: Packet-Optical Converge

Figure 2.4 represents a converged packet-optical infrastructure for typical wide area networks. As shown there are three different planes representing the actual optical network transport depicted with a set of Roadms shown as R1, R2, R3, R4, R5, R6 and R7. The optical transport is the physical network with actual optical cables underground connecting the central offices together. At the optical networks there are no packet processing and simply they are highways for all traffic types without consideration to headers or payloads.

On top of the optical transports are riding the packet network which is the 2nd plane shown in the diagram. The packet switches form packet switching capabilities on top of optical circuits. The actual network packet processing and protocols are all performed at the packet layer.

Finally the upper layer represents the logical full or partial logical meshes interconnect that represents connectivity between the end points. Typically network operators depict their MPLS tunnels on top of their packet-optical networks and enable label switching for enhanced traffic management options.

As shown in the figure 2.4 the SDN core represents the network and the north bound interface communicates to the applications such as BW Cal. Service portal, and south bound communicate using various interfaces such as OpenFlow. In this project we only use OpenFlow interface at the south bound. Eventually ONOS allows plug-in mechanism at south bound to support various south bound protocols like NetConf.

2.3.5 Emulations of Packet and Optical Networks

Today, testing the network behavior with real equipment is very difficult and expensive. In this use case we used packet network emulators known as OVS, and optical network emulators known as LINC-OE. The emulated packet and optical network formed a desired topology for testing. The major challenge during this project was to connect end links appropriately to form the proper topology. Once the topology is established, then the SDN controller will be used to manage the network. Figure 2.5 shows the emulation environment that we used for test and integration efforts.

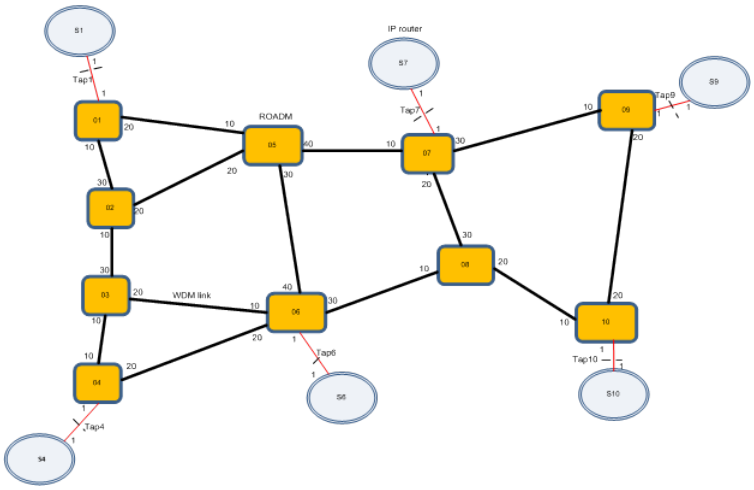

Figure 2.5: Test environment of Packet-Optical

2.3.6 End to End Integration

The project has many complex parts that need to come together for a successful proof-of-concept (POC). Environment on the desktop must first support VM(s) to install each sub-system. Next we had to install appropriate virtual machines using any virtualization product such as virtual box. VM should have Ubuntu (Linux) as Operation System (OS) and the RAM memory of at least 2048MB. We can install everything in one VM but recommended way is to have each component in separate VMs. ONOS can run on XOS (Mac) so it is recommended to install ONOS on host machine if you have XOS on your host machine ONOS. The goal is to install the SDN with appropriate containers to support the specific use case. We need mininet (latest version) to trigger packet switch topology, ONOS as SDN controller, Linc-oe for optical switch or roadm, OpenVSwitch (OVS) for packet switch and Linc-config-generator for generating topology file for Linc-oe.

We start with integration of SDN core with both packet and optical layer topology. The effort involves a good understating of how SDN is attaching itself to every packet and optical nodes. Then we need to find a way to push the appropriate topology to the SDN where the links and interface can be discovered. Today the SDN does not have an autonomous approach to discover the optical topology, but it does uses LLDP protocol to discover the the packet layer. Once the topology can be discovered by the SDN controller, then the ONOS-GUI can be used to represent the the discovered topology.

The ONOS directly communicates to the GUI application for all event changes. For example when or every

time a link or a switch state changes the GUI needs to get a notification from ONOS to reflect the network changes from the ONOS. Integrating Bandwidth (BW) calendaring portal (service layer) with ONOS is the next step. ONOS allows application to show their intent. We managed to show that the ONOS-intents are generated when the time triggers, and then the ONOS takes an action with path computation engine to find the best path possible. We had to make sure that the ONOS correctly assigned resources to the path request. For example, the transport layer (Optical layer) should map the packet layer to the the existing optical layer circuits if BW is already available. It will only create paths at the transport layer only if the circuit layer is not available.

Figure 2.6 is ONOS GUI for converged packet optical emulated switches and ROADMS (optical switches). ONOS in this case was given the optical and packet topology and the interconnects from a topology file. As shown the dark blues represent the optical ROADMS, and light blue represents the packet switches. The ONOS controller was used to establish a connection from two end points between host A, and B representing data centers. The red dashed line represents the actual traffic being passed between the hosts.

The ON LAB ONOS controller abstracts multilayer network topology which provides applications such as Path Computation Engines (PCE) to find capacity in real-time and also recover from failures.

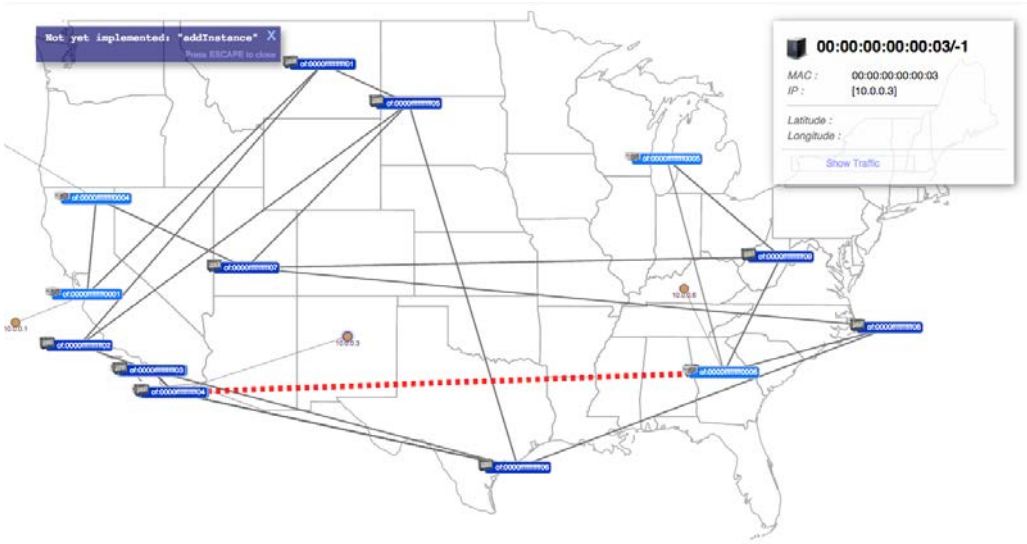

Figure 2.6: ONOS GUI for Packet-Optical

2.3.7 Summary

The Packet-Optical project aims to create a cross layer control infrastructure where networks at both optical and logical layers could be coordinated for best traffic optimization practices. The SDN controller is used not only to abstract the network topology at both layers, but also maintain the performance measurement for traffic Engineering functions such as Path Computation element known as PCE. The SDN (ONOS) graph abstraction is used for easier access and control of traffic-flow forwarding and dynamic BW assignment at both packet and optical layers.

In addition to traffic demands, the monitoring and analysis at both layers can be used to de-fragment the

traffic on various transport links. In other words many of the logical flows may be mapped to transport circuits that may not be suited for restoration. SDN is used to shift the traffic for best optimization practices.

Furthermore, network graphs at the SDN can represent a dash board view of network load and congestion. The administrators of the networks can use these dashboard to make intelligent decisions to best operate the network for users. Another challenge we face is Service orchestration on the north bound of the SDN, where a unified service layer is required to meet many application demands. Open APIs alone will not meet the high demand batch application and or operational subsystem functions expected from SDN. Therefore, the team started working on compilable intent frame works.

The intent framework can eventually compile various models such as YANG [17] to it's APIs. The next challenge is to enable the SDN to expose topology,network maps and states to trusted applications. Although security has not been implemented yet in SDN controller, but the SDN graphs must only be exposed to trusted applications.

In the future Network Function Virtualization will be integrated with SDN for enabling many new applications. For example NFV can be used for load balancing, traffic shaping, and Internet of things. I have listed a few additional applications that SDN needs to support as a control platform:

- Data Center bandwidth slicing on demand where data centers can schedule or buy capacity on demand.
- Real Time Probes and Analytics at the data path level where applications can be aware of the network probes and sensors.
- TE and Optimization Practices for specific traffic types where administrator can enhance end users quality of experience adaptively
- Content Distribution Network (CDN) and Video Service orchestration and QOE measurement where content owners can measure the quality of their premium content
- Mobile Health (MHealth) Metering Observation where health care providers can monitor their patients health and behaviours for preventive measures.

It is mind boggling to see how much pressure new applications such as those mentioned above are putting on the network. It is more exciting to see that with SDN networks are becoming much more programmable, perhaps eventually some day we can look at the networks in the same we look at the computers.

Chapter 3

THROUGHPUT OPTIMUM SCHEDULING ALGORITHM FOR INPUT QUEUED SWITCH UNDER HYBRID DATA TRAFFIC

3.1 Introduction

In the last decade or so there is huge amount of increase in internet data traffic. This increase is due to a variety of emerging internet based applications such as Skype, torrent, youtube and many more. This increase has significant impact on networking devices such as routers and switches. Scheduling policies is one of the victims of these impact. Due to increase in data traffic, network devices now need to schedule queues dynamically in a presence of hybrid traffic.

Maximum weight matching (MWM) algorithms [1] provide 100% throughput for input queued switch in the presence of light tailed (LT) data traffic but perform poorly in the presence of heavy tailed (HT) data traffic [3]. Main reason for this poor performance is, MWM algorithms like Longest Queue First (LQF) [1] schedule queue with the maximum queue length and in the presence of hybrid traffic i.e. some queues experiencing HT and some LT, MWM tends to schedule queues with HT data traffic as their queue length will always be higher than queue length with LT data traffic. This behaviour results in instability of the queues with LT data traffic.

To overcome this problem a new scheduling policy named maximum weight-α (MWA) is introduced for wireless networks [3]. MWA algorithm gives some priority to queue with LT data traffic. This increases the opportunities of scheduling of the queues with LT data traffic.

So far, MWA has been applied in wireless networks such as Cognitive Radio Networks (CRN) [2] to solve scheduling problems in presence of HT data traffic.

The focus of this research is to show that MWA scheduling algorithm provides moment stability to input queued switch under the presence of hybrid data traffic for all admissible arrival processes. Moment stability is a new stability criterion first proposed in [2], [14], aiming to characterize the QoS performance of data networks in the presence of heavy-tailed traffic. Moment stability requires that all the network users with light-tailed traffic arrivals always have bounded queueing delay with finite mean and variance. Compared to strong stability, moment stability not only requires the finiteness of lower order moments, such as mean, but also demands the boundedness of higher order moments, such as variance, provided that such moments exist. What is more important, achieving moment stability can prevent heavy-tailed traffic (e.g., video conferencing and online gaming traffic) from significantly degrading the queueing performance of light-tailed traffic (e.g., email deliveries, audio/voice traffic, and temperature and humidity readings). Based on the definition of moment stability, the network stability region is defined as the closure of the set of all arrival rate vectors for which the queues of all network users can be stabilized by a feasible scheduling policy. Moreover, a scheduling policy is throughput optimal if it stabilizes the system for any arrival rates in the stability region.

We evaluate our results in SDN environment specifically, we are using *mininet* for emulating a network with one $OpenvSwitch$ connected to 8 hosts, $OpenVirteX$ to create two virtual networks from the created network and two controller to implement MWM and MWA scheduler, each connected to one of the two virtual networks.

3.2 Related Work and Motivation

[2] along with some other useful results also concluded that in Cognitive Radio Networks (CRN) MWA stabilized the network for any arrival rate in stability region. [3] shows that MWM perform poorly in the presence of HT traffic and proves that MWA scheduling policy with suitable value of α parameter provides delay stability. [4] and [6] considered system of parallel two parallel queues, one queue experiencing HT traffic and other with LT and shows that MWA is delay stable for such system. [4] extends this result and introduce Log Maximum Weight (LMW) scheduling algorithm, which also guarantee throughput optimization.

Our motivation comes from the result of all the above mentioned work that is MWA is definitely better scheduling algorithm than MWM as it solve the problem related to HT traffic. No or very little amount of work has been done for wired network to solve the scheduling problem that is due to HT nature of data traffic. This paper is the first concrete effort to show that MWA provides through stability in input queued switch for all admissible arrival processes, if suitable value of α parameter is chosen.

3.3 System Model, Definitions and Preliminaries

3.3.1 System Model

Our model is very much similar to [1]. Consider $N \times N$ (N inputs and N outputs) input queued switch in the figure 3.1. The independent and identically distributed (IID) arrival process $A_I(n)$ at input i, $1 \leq i \leq N$, is

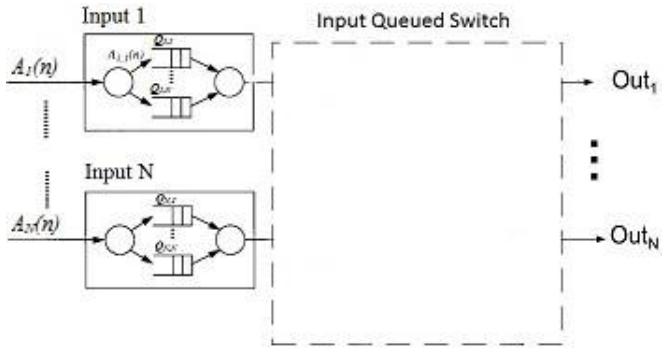

Figure 3.1: System Model and components

a discrete time process at time n. Let j denote the output, where $1 \leq j \leq N$. $Q_{i,j}$ denotes the buffer for input i destined for output j. $L_{i,j}$(n) is the occupancy (i.e. is number of packet in buffer) of $Q_{i,j}$ at time n. Following vector represent the occupancy of all the queues.

$$\underline{L}(n) \equiv (\underline{L}_{1,1}(n), \ldots, \underline{L}_{1,N}(n), \ldots \underline{L}_{N,N}(n))^T \tag{3.1}$$

Arrival rate at input i for output j is defined as $\lambda_{i,j} = E[A_{i,j}(n)]$. The arrival process is said to be admissible if no input or output is fully or oversubscribed i.e.

$$\sum_{i}^{N} \lambda_{i,j} < 1, \sum_{j}^{N} \lambda_{i,j} < 1 \text{ where } \lambda_{i,j} \geq 0$$

3.3.2 Definitions

Definition 1. *Tail Coefficient* C *describe the heaviness of the tail and is defined for non-negative random variable X as*

$$C = inf\{c \in R^{+} | E[X^{c}] = \infty\}$$

Which means that minimum value of coefficient that makes random variable infinite is tail coefficient. One thing that we must focus on is that what would be the value of coefficient that makes HT and LT random variable infinite. As HT random variable will already be with very high value, so very small value of tail coefficient C will make the expected value of random variable to infinite but for LT we need higher values of C.

Definition 2. *An arrival process $A_{i,j}(n)$ is called heavy-tailed if it $A_{i,j}(n)$ has Tail Coefficient less than or equal to 2. Otherwise, the arrival process is called light-tailed [3].*

It can be concluded from the above definition that if variance of arrival process is infinite then arrival process is said to be HT else arrival process is LT i.e. $E[A_{i,j}(n)^2] = \infty$ for HT.

Definition 3. *[14] A network is moment stable, if there exists a scheduling algorithm, under which the steady-state queue length of any user with light-tailed arrival has bounded mean and variance.*

3.3.3 Queuing Dynamics

$L_{i,j}(n)$ represent number of packet in queue $Q_{i,j}$ and $A_{i,j}(n)$ is arrival process at time n. Let $S_{i,j}(n)$ be scheduled packets from input i to output j where $S_{i,j}(n) \in \{0,1\} \ \forall \ (i,j)$, then the queueing dynamic can be represented by

$$L_{i,j}(n+1) = L_{i,j}(n) + A_{i,j}(n) - S_{i,j}(n) \tag{3.2}$$

where $L_{i,j}(n+1)$ represent the next state occupancy of the queue $Q_{i,j}$

3.3.4 Preliminaries

In this paper we are using following symbolic representations:
$L^{\alpha}(n)$ represent queue length raised to the power α matrix and the associated vector

$$\underline{L}^{\alpha}(n) \equiv (\underline{L}_{1,1}^{\alpha_{1,1}}(n), \ldots, \underline{L}_{1,N}^{\alpha_{1,N}}(n), \ldots \underline{L}_{N,N}^{\alpha_{N,N}}(n))^T \tag{3.3}$$

Let

$$\mathfrak{L}_{i,j}(n) = \frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1}$$

associated matrix

$$\mathfrak{L}_{(}n) = \left[\frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1} \right]$$

and associated vector

$$\underline{\mathfrak{L}}(n) \equiv (\frac{L_{1,1}^{\alpha_{1,1}+1}(n)}{\alpha_{1,1}+1}, \ldots, \frac{L_{1,N}^{\alpha_{1,N}+1}(n)}{\alpha_{1,N}+1}, \ldots, \frac{L_{N,N}^{\alpha_{N,N}+1}(n)}{\alpha_{N,N}+1})^T \tag{3.4}$$

O_{ne} represent $N \times N$ matrix with all coefficient equal to 1

$$O_{ne} \equiv [1_{i,j}]$$

and the associated vector

$$\underline{O_{ne}} \equiv (1_{1,1}, 1_{1,2}, \ldots 1_{1,N}, \ldots 1_{N,N})^T$$

Note that

$$\underline{\mathfrak{L}}(n)^T \underline{O_{ne}} = \sum_{i,j=1}^{N} \frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1} \tag{3.5}$$

3.4 Overview Of Main Result

Theorem. *Maximum Weight-α scheduling algorithm is stable for Input queued switch under the presence of hybrid data traffic for all admissible arrival processes if $1 < \alpha_{i,j} + 1 < C_{i,j}$ for HT and $2 \leq \alpha_{i,j} + 1 < C_{i,j}$ where $C_{i,j}$ is the tail coefficient of arrival process $A_{i,j}(n)$ defined in definition 1 and 2.*

Proof. The detailed proof is given in Appendix A. Basically, we will show that under MWA scheduling algorithm, the input queued switch will experience negative Lyapunov drift as the queue length increases, i.e.,

$$E\left[\underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}} - \underline{\mathfrak{L}}(n)^T \underline{O_{ne}} | L(n)\right] \leq -\varepsilon \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}}(n) + k$$

□

where $\underline{\mathfrak{L}}(n)^T \underline{O_{ne}} = \sum_{i,j=1}^{N} \frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1}$ from 3.5 and $\varepsilon > 0, k > 0$.

V(n) = $\underline{\mathfrak{L}}(n)^T \underline{O_{ne}}$ is Lyapunov function and using the result of [7] and [1] we show that the system is stable. Lets evaluate term $-\varepsilon \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}}(n)$, -ε indicates that the drift is negative and $\sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}}(n)$ is more or less equals to $\sum_{i,j=1}^{N} L_{i,j}(n)$ which shows that if $\sum_{i,j=1}^{N} L_{i,j}(n)$ is large the Lyapunov drift also tends to be negative.

Note that Kumar and Meyn [7] and Nick et al. [1] uses quadratic Lyapunov function but Lyapunov function used in this paper depends on the value of $\alpha_{i,j}$, if $\alpha_{i,j} = 1 \; \forall (i,j)$ i.e. LT, then $V(n) = \sum_{i,j=1}^{N} \frac{L_{i,j}^{1+1}(n)}{1+1}$ quite same as [7] and [1].

3.5 Simulation

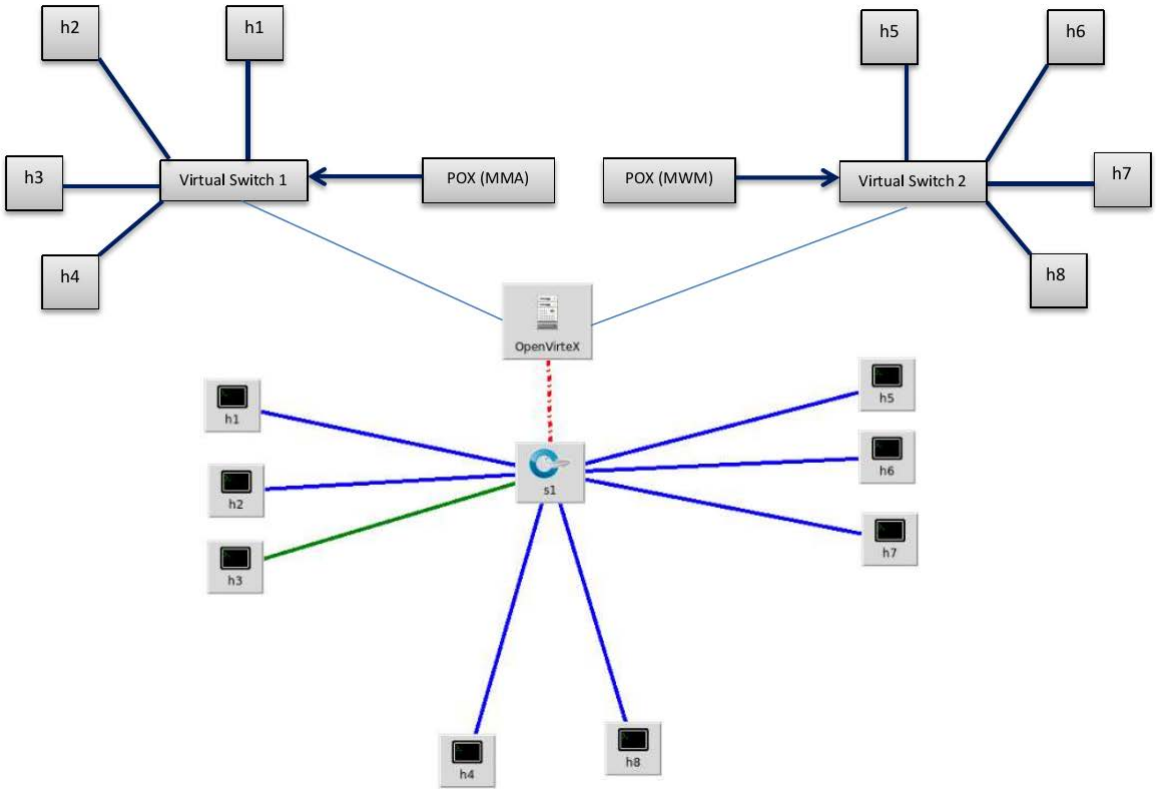

Figure 3.2: Test environment in SDN setting

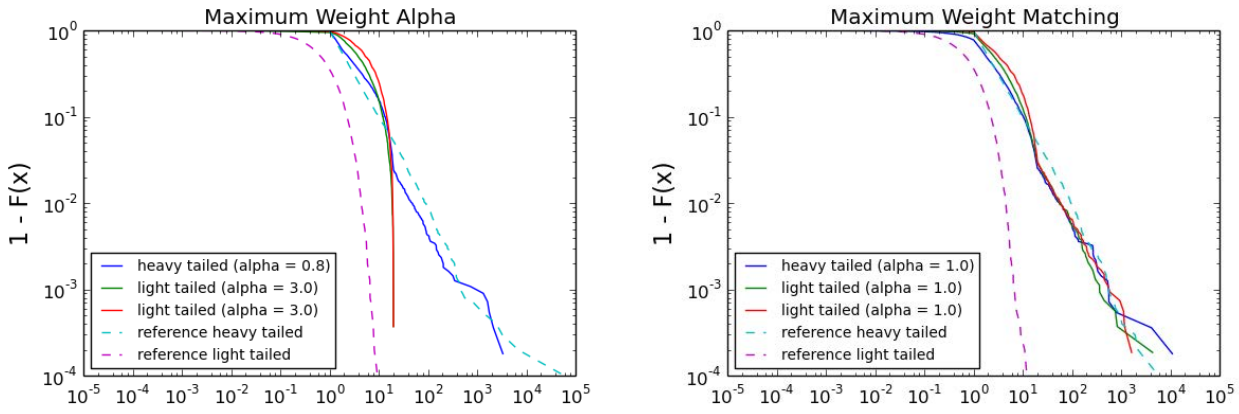

Figure 3.3: Queing Delay comparison

In this section, we use simulations to illustrate our theoretical result. We are using Software Defined Networking (SDN) settings for the simulation as shown in figure 3.2. Specifically we are using *mininet* [8] to create network with one OpenVSwitch (OVS) connected to 8 hosts and *OpenVirteX* [9] to create two virtual networks from the mininet network with 4 hosts each. We implement two packet scheduler using *POX* [15] SDN controller. Note that our scheduler does not install any flow in the switch which makes sure that each packet goes to the scheduler. Both scheduler schedules 1 packet per second. One scheduler implements MWM and other MWA, and each scheduler is connected to one of the two virtual networks. The reason why we are using *OpenVirteX* to create two virtual switches instead of creating two physical switches using *mininet* is to nullify the doubt that performance of two switches can be different. Host $h1$ to $h4$ are in one virtual network and $h5$ to $h8$ are in other virtual virtual network. In first virtual network $h1$ sends HT traffic to $h4$ and $h2, h3$ sends LT traffic to $h4$. In second virtual network $h5$ sends HT traffic to $h8$ and $h6, h7$ sends LT traffic to $h8$. HT traffic follows pareto distribution with $\alpha = 1$ and LT traffic follows exponential distribution with $\lambda = 0.3$. We are using ping to send the traffic. To send the HT traffic we increase the size of the packet using $-s$ ping's option which in results fragments the packet if the size is greater than MTU and hence perfectly simulates HT arrivals. To remove ARP delay each host is preinstalled with required ARP entries. As we are using ping, hosts receive ping reply which is forwarded to respective host without queuing. For simulation we have taken $E[A_{h1}] = 20$ packets, $E[A_{h2}] = 3$ packets, and $E[A_{h3}] = 3$ packets with each packet 1500 bytes. Note we can send ping packets size larger than 65535-byte so to handle this we are sending multiple packets according the packet size if packet size is greater than 65535-bytes.

Lets take a look at figure 3.3 which shows the comparison between the queuing delay of all three hosts in both scenarios. Graph of MWM in figure 3.3 shows that LT queues experience unbounded delays as they follow pareto distribution with tail index equal to 1. On the other hand MWA graph show that both LT queues are stable as they decay exponentially and follows exponential distribution if we choose $\alpha_{h1} = 0.8, \alpha_{h2} = 3.0$ and $\alpha_{h3} = 3.0$. Hence proving out theorem.

3.6 Summary

MWM provides 100% throughput for input queued switch under the presence of LT traffic [1] but performs poorly under the presence of HT traffic [3]. This is mainly because MWM algorithms like Longest Queue First (LQF) schedules the queue with the longest occupancy and in case of hybrid traffic, the queues with HT traffic are always scheduled which makes LT queues unstable.

In this paper we have shown that input queued switch with hybrid data traffic is moment stable with MWA scheduling algorithm for all admissible arrival processes if suitable value of α parameter is used. Value of α is chosen according to the tail coefficient $C_{i,j}$. For HT arrival process $1 < \alpha + 1 < C_{i,j}$ and $2 \leq \alpha < C_{i,j}$ for LT. This gives LT queues some priority and hence increase their throughput and makes switch stable.

Chapter 4

CONCLUSION

In this thesis we worked on three different SDN based tools. We build OVX test framework to test OVX so that OVX can perform appropriate virtualization on the designated partitioned network. In this regard, we addressed the problem where the existing testing techniques were not enough to capture the detailed virtualization. As the unit testing could only test the individual components, it could not ensure the end to end-packet delivery, because unit testing has very limited view of OVX. With OVX Testing Framework we can send OpenFlow packets to the OVX as we give illusion to OVX that there are controllers at Northbound and switches at Southbound. We can compare the OVX response by creating expected message(s). By doing this we can ensure end-to-end delivery.

Segment routing is a new approach to dynamically control the tunnels for various QoS and traffic management practice. With respect to segment routing, we developed command line interface that uses OpenFlow v1.3 (OF) messages to get the information from the switches and also let the administrator create tunnels and policies. The CLI needed to conform to the OF 1.3 events. Since ONOS does not support OF 1.3 and segment routing application requires OF 1.3 features like groups MPLS etc. OF 1.3 support is added to ONOS. Furthermore existing ONOS API was not OFv1.3 enabled. Therefore, we developed new APIs that could enable us to support the appropriate CLI for segment routing application.

The next problem we addressed in this thesis was to establish optical emulator that models the transport network. We used linux *tap* itnerfaces to connect optical and packet layer. In this project, we employed the cross-layer design approach where the application on the northbound of SDN can program both the packet and optical layer. In this regard we had to test with small topology at first to make sure ONOS can discover and program the paths. The next big challenge was to validate the functionality of NB applications such as PCE and calendaring BW on demand can program the network. We showed that with multi-layer topology the traffic optimization and restoration are much more effective and flexible.

In the last we showed that MWA is throughput optimum scheduling algorithm for input queued switch under the presence of hybrid data traffic. We evaluated the performance of the MWA scheduling algorithm in SDN settings using mininet for emulating physical forwarding infrastructure, OpenVirteX for creating virtual

networks and two pox controllers to implement two different scheduling algorithms. By comparing the results we can verify that MWA is moment stable by ensuring the queues with LT traffic have bounded mean and variance surely.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Nick McKeown, Adisak Mekkittikul, Venkat Anantharam, Jean Walrand. "Achieving 100% Throughput in an Input-Queued Switch" *IEEE Transcation on Communication, Vol. 47, NO. 8, August 1999*

[2] Pu Wang and Ian F. Akyildiz. "Network Stability of Cognitive Radio Networks in the Presence of Heavy Tailed Traffic" *appear in IEEE SECON 2012, Seoul, Korea, June 2012;*

[3] Mihalis G. Markakis. "Scheduling in Switched Queueing Networks with Heavy-Tailed Traffic" *PHD dissertation of Mihalis G. Markakis, june 2013*

[4] Jagannathan, K., Markakis, M., Modiano, E. and Tsitsiklis, J.N. "Throughput optimal scheduling in the presence of heavy-tailed traffic" *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton,* 953 - 960, DOI:10.1109/ALLERTON.2010.5707012

[5] Mihalis G. Markakis, Modiano, E.H. and Tsitsiklis, J.N. "Max-weight scheduling in networks with heavy-tailed traffic" *INFOCOM, 2012 Proceedings IEEE* 2318 - 2326, DOI:10.1109/INFCOM.2012.6195619

[6] Markakis, M.G., Modiano, E.H., and Tsitsiklis, J.N. "Scheduling policies for single-hop networks with heavy-tailed traffic" *IEEE conference on Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton*112 - 120, DOI:10.1109/ALLERTON.2009.5394854

[7] P. R. Kumar and S. P. Meyn. "Stability of queuing networks and scheduling policie.s" *IEEE Trans. Automat. Contr.* vol. 40, Feb. 1995.

[8] Bob Lantz, Brandon Heller, and Nick McKeown. "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks." *9th ACM Workshop on Hot Topics in Networks*. October 20-21, 2010, Monterey, CA.

[9] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Elio Salvadori, Guru Parulkar and Bill Snow. "OpenVirteX: Make Your Virtual SDNs Programmable." *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*. Co-located with ACM SIGCOMM'14, August 22, 2014 Chicago, IL.

[10] Bob Lantz, Brian O'Connor, Jonathan Hart, Pankaj Berde, Pavlin Radoslavov, Masayoshi Kobayashi, Toshio Koide, Yuta Higuchi, Matteo Gerola, William Snow and Guru Parulkar. 'ONOS: Towards an Open, Distributed SDN OS" *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*. Co-located with ACM SIGCOMM'14, August 22, 2014 Chicago, IL.

[11] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. "Extending Networking into the Virtualization Layer" *Proc. of workshop on Hot Topics in Networks HotNets-VII*, (2009).

[12] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner. "OpenFlow: enabling innovation in campus networks." *ACM SIGCOMM Computer Communication Review.* Volume 38 Issue 2, Pages 69-74, April 2008. New York, NY, USA.

[13] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown and Guru Parulka. "Can the Production Network Be the Testbed?" *Proceeding OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*, Article No. 1-6, USENIX Association Berkeley, CA, USA 2010.

[14] P. Wang and I. F. Akyildiz "On the Stability of Dynamic Spectrum Access Networks in the Presence of Heavy Tails." *to appear in IEEE Transactions on Wireless Communications*, Sept. 2014.

[15] POX Controller:
`http://www.noxrepo.org/pox/about-pox/` 1 November 2014

[16] sdncon:
`https://github.com/mandeepdhami/netvirt-ctrl/tree/master/sdncon` 11 October 2104

[17] yang:
`https://tools.ietf.org/html/rfc6020` 5 November 2014

Appendix

Appendix

Proof preliminaries

In this appendix we are using system model, definitions and preliminaries described in section 3.3 and [1] with few additions. Consider figure 3.1, $A_{i,j}(n)$ is the arrival process at input i for output j. $A(n)$ represents the matrix of arrival process at each queue

$$A(n) \equiv [A_{i,j}(n)]$$

and $\underline{A}(n)$ be the associated vector

$$\underline{A}(n) \equiv (A_{1,1}(n), \ldots, A_{1,N}(n), \ldots, A_{N,N}(n))$$

$\lambda_{i,j} = E[A_{i,j}(n)]$ is arrival rate of arrival process $A_{i,j}(n)$, rate matrix of arrival process is

$$\Lambda \equiv [\lambda_{i,j}]$$

and associated vector is

$$\underline{\lambda} \equiv (\lambda_{1,1}, \ldots, \lambda_{1,N}, \ldots, \lambda_{N,N})$$

The service matrix indicates scheduled queue at time slot n

$$S(n) \equiv [S_{i,j}(n)]$$

where

$$S_{i,j}(n) = \begin{cases} 1, & \text{if } Q_{i,j} \text{ is served at n} \\ 0, & \text{else} \end{cases}$$

$S(n) \in S$ is serves matrix. It is important to note that $\sum_{i=1}^{N} S_{i,j}(n) = \sum_{j=1}^{N} S_{i,j}(n) = 1$ and hence $S(n) \in S$ is permutation matrix, which also include the case where an empty queue is served but it does not affect the result [1]. The associated vector is

$$\underline{S}(n) \equiv (S_{1,1}(n), \ldots, S_{1,N}(n), \ldots, S_{N,N}(n))$$

Note that:

$$\sum_{i,j=1}^{N} S_{i,j}(n) = N$$

Using 3.2 the next state vector becomes

$$\underline{L}(n+1) = \underline{L}(n) - \underline{S}(n) + \underline{A}(n)$$

Note for simplicity we assume that queues are always non-empty and there is not case where empty queue is served.

The scheduling decision by Maximum Weight α is given by

$$\underline{S}^*(n) = arg\ max_{\underline{S}(n)\in S}\ \underline{L}^T(n)^{\alpha}\underline{S}(n)$$

which in other words is

$$arg\ max_{\sum_{i,j}^N S_{i,j}(n)} \sum_{i,j}^{N} L_{i,j}^{\alpha_{i,j}}(n) S_{i,j}(n)$$

Proof of the theorem

Lemma.

$$\sum_{i,j=1}^{N} \left[(\lambda_{i,j} - S_{i,j}(n)) L_{i,j}^{\alpha_{i,j}}(n) \right] = -\varepsilon \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}}(n)$$

where $\varepsilon > 0$.

Proof. Note that:

$\sum_{i,j=1}^{N} S_{i,j}(n) L_{i,j}^{\alpha_{i,j}}(n) = \underline{L}^T(n)^{\alpha}\underline{S}(n)(MWA)$,

$\sum_{i,j=1}^{N} \lambda_{i,j}(n) L_{i,j}^{\alpha_{i,j}}(n) = \underline{L}^T(n)^{\alpha}\underline{\lambda}$

For the admissible arrival rate there exit $\varepsilon > 0$ such that

$$\sum_{i,j=1}^{N} \lambda_{i,j} = N - \varepsilon$$

and as $S(n) \in S$ is permutation matrix we know that

$$\sum_{i,j}^{N} S_{i,j}(n) = N$$

so,

$$\sum_{i,j=1}^{N} \lambda_{i,j} - \sum_{i,j}^{N} S_{i,j}(n) = -\varepsilon$$

hence proving the lemma. □

Now we are ready to prove the theorem. Consider Lyapunov function

$$V(\underline{\mathfrak{L}}(n)) = \underline{\mathfrak{L}}(n)^T \underline{O_{ne}}$$

then

$$V(\underline{\mathfrak{L}}(n+1)) = \underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}}$$

where using 3.4

$$\underline{\mathfrak{L}}(n+1) \equiv (\frac{L_{1,1}^{\alpha_{1,1}+1}(n+1)}{\alpha_{1,1}+1}, \dots, \frac{L_{1,N}^{\alpha_{1,N}+1}(n+1)}{\alpha_{1,N}+1}, \dots, \frac{L_{N,N}^{\alpha_{N,N}+1}(n+1)}{\alpha_{N,N}+1})^T$$

where

$$\frac{\underline{L}_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1} = \frac{1}{\alpha_{i,j}+1}\left(L_{i,j}(n) + A_{i,j}(n) - S_{i,j}(n)\right)^{\alpha_{i,j}+1}$$

let $\Delta_{i,j}(n) = A_{i,j}(n) - S_{i,j}(n)$ so

$$\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1} = \frac{1}{\alpha_{i,j}+1}\left(L_{i,j}(n) + \Delta_{i,j}(n)\right)^{\alpha_{i,j}+1}$$

Now there are two cases 1) case: when $\alpha_{i,j}+1 < 2$ i.e HT, 2) case: when $\alpha_{i,j}+1 \geq 2$ i.e. LT.

Consider Case 1: $\alpha_{i,j}+1 < 2$:

Taking 0^{th} order Taylor expansion with Lagrange form of the remainder [2] around $L_{i,j}(n)$ we have

$$\frac{1}{\alpha_{i,j}+1}\left(L_{i,j}(n) + \Delta_{i,j}(n)\right)^{\alpha_{i,j}+1}$$

$$= \frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1} + \Delta_{i,j}(n)\nu^{\alpha_{i,j}}$$

where $\nu \in [(L_{i,j}(n)-1), (L_{i,j}(n)+A_{i,j}(n))]$ which makes

$$\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1} = \frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1} + \Delta_{i,j}(n)\nu^{\alpha_{i,j}}$$

Taking summation $\sum_{i,j=1}^{N}$ on both sides

$$\sum_{i,j=1}^{N}\left(\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1}\right) = \sum_{i,j=1}^{N}\left(\frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1} + \Delta_{i,j}(n)\nu^{\alpha_{i,j}}\right)$$

from 3.5

$$\underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}} = \underline{\mathfrak{L}}(n)^T \underline{O_{ne}} + \sum_{i,j=1}^{N}\left(\Delta_{i,j}(n)\nu^{\alpha_{i,j}}\right)$$

taking expectation given $\underline{L}(n)$

$$E\left[\underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}} - \underline{\mathfrak{L}}(n)^T \underline{O_{ne}} | \underline{L}(n)\right] = E\left[\sum_{i,j}^{N} (\Delta_{i,j}(n)\nu^{\alpha_{i,j}} | \underline{L}(n))\right] \tag{1}$$

Now consider an event $\Gamma_{i,j}(n) : \{\Delta_{i,j}(n) < 0\}$, right hand side of 3.5 becomes,

$$E\left[\sum_{i,j}^{N} (\Delta_{i,j}(n)\nu^{\alpha_{i,j}} | \underline{L}(n))\right]$$

$$= E\left[\sum_{i,j=1}^{N} (\Delta_{i,j}(n)(L_{i,j}(n)-1)^{\alpha_{i,j}}) | \underline{L}(n), \Gamma_{i,j}(n)\right]$$

$$.P(\Gamma_{i,j}(n) | \underline{L}(n))$$

$$+E\left[\sum_{i,j=1}^{N} (\Delta_{i,j}(n)(L_{i,j}(n)+A_{i,j}(n))^{\alpha_{i,j}}) | \underline{L}(n), \Gamma^c_{i,j}(n)\right]$$

$$.P(\Gamma^c_{i,j}(n) | \underline{L}(n))$$

as $0 \leq \alpha_{i,j} < 1$ and $L_{i,j}(n), A_{i,j}(n)$ both are non-negative integers, so it not difficulty to verify that.

$$(L_{i,j}^{\alpha_{i,j}}(n) - 1) \geq L_{i,j}(n)^{\alpha_{i,j}} - 1,$$

$$(L_{i,j}^{\alpha_{i,j}}(n) + A_{i,j}(n)) \leq L_{i,j}^{\alpha_{i,j}}(n) + A_{i,j}^{\alpha_{i,j}}(n)$$

Applying these inequalities we have,

$$E\left[\sum_{i,j}^{N} (\Delta_{i,j}(n)\nu^{\alpha_{i,j}} | \underline{L}(n))\right]$$

$$\leq E\left[\sum_{i,j=1}^{N} (\Delta_{i,j}(n).(L_{i,j}^{\alpha_{i,j}}(n) - 1)) | \underline{L}(n), \Gamma_{i,j}(n)\right]$$

$$.P(\Gamma_{i,j}(n) | \underline{L}(n))$$

$$+E\left[\sum_{i,j=1}^{N} (\Delta_{i,j}(n).(L_{i,j}^{\alpha_{i,j}}(n) + A_{i,j}^{\alpha_{i,j}}(n))) | \underline{L}(n), \Gamma^c_{i,j}(n)\right]$$

$$.P(\Gamma^c_{i,j}(n) | \underline{L}(n))$$

by solving

$$E\left[\sum_{i,j}^{N} (\Delta_{i,j}(n)\nu^{\alpha_{i,j}} | \underline{L}(n))\right] \leq E\left[\sum_{i,j=1}^{N} \Delta_{i,j}(n).L_{i,j}^{\alpha_{i,j}}(n) | \underline{L}(n)\right]$$

$$-E\left[\sum_{i,j=1}^{N} \Delta_{i,j}(n) | \underline{L}(n), \Gamma_{i,j}(n)\right].P(\Gamma_{i,j}(n) | \underline{L}(n))$$

$$+E\left[\sum_{i,j=1}^{N}(\Delta_{i,j}(n)\cdot A_{i,j}^{\alpha_{i,j}}(n))|\underline{L}(n),\Gamma_{i,j}^{c}(n)\right].P(\Gamma_{i,j}^{c}(n)|\underline{L}(n))$$

Now consider the follow facts:

when $\Delta_{i,j}(n)<0$ that is $\Gamma_{i,j}(n)$ then $\Delta_{i,j}(n)=-1$, when $\Delta_{i,j}(n)\geq 0$ that is $\Gamma_{i,j}^{c}(n)$ then $\Delta_{i,j}(n)\leq A_{i,j}(n)$ which implies $\Delta_{i,j}(n)A_{i,j}^{\alpha_{i,j}}(n)\leq A_{i,j}^{\alpha_{i,j}+1}(n)$. Since $\alpha_{i,j}+1\leq C_{i,j}$ we know that $E[A_{i,j}^{\alpha_{i,j}+1}(n)]$
$<\infty$ [3], [6], hence $+E\left[\sum_{i,j=1}^{N}\Delta_{i,j}(n).A_{i,j}^{\alpha_{i,j}}(n))|\underline{L}(n),\Gamma_{i,j}^{c}(n)\right].P(\Gamma_{i,j}^{c}(n)|\underline{L}(n))$ is bounded summation. Hence there exist $b_{i,j}$ such that

$$E\left[\sum_{i,j}^{N}(\Delta_{i,j}(n)\nu^{\alpha_{i,j}}|\underline{L}(n))\right]$$

$$\leq E\left[\sum_{i,j=1}^{N}(A_{i,j}(n)-S_{i,j}(n)).L_{i,j}^{\alpha_{i,j}}(n)|\underline{L}(n)\right]+\sum_{i,j=1}^{N}b_{i,j}$$

applying the expectation and putting it in 1 we have

$$E\left[\underline{\mathfrak{L}}(n+1)^{T}\underline{O_{ne}}-\underline{\mathfrak{L}}(n)^{T}\underline{O_{ne}}|\underline{L}(n)\right]$$

$$\leq\sum_{i,j=1}^{N}\left((\lambda_{i,j}(n)-S_{i,j}(n).L_{i,j}^{\alpha_{i,j}}(n))\right)+\sum_{i,j=1}^{N}b_{i,j}$$

let $k=\sum_{i,j=1}^{N}b_{i,j}$ and using lemma, it is clear that there is negative drift in Lyapunov function and hence proving the theorem.

$$\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1}=\frac{1}{\alpha_{i,j}+1}\left(L_{i,j}(n)+\Delta_{i,j}(n)\right)^{\alpha_{i,j}+1}$$

Now consider case 2: $\alpha_{i,j}+1\geq 2$:

$$\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1}=\frac{1}{\alpha_{i,j}+1}\left(L_{i,j}(n)+\Delta_{i,j}(n)\right)^{\alpha_{i,j}+1}$$

Taking 1^{th} order Taylor expansion with Lagrange form of the remainder around $Li,j(n)$ we have

$$\frac{L_{i,j}^{\alpha_{i,j}+1}(n+1)}{\alpha_{i,j}+1}$$

$$=\frac{L_{i,j}^{\alpha_{i,j}+1}(n)}{\alpha_{i,j}+1}+\Delta_{i,j}(n)L_{i,j}^{\alpha_{i,j}}+\frac{\Delta_{i,j}^{2}(n)}{2}\nu^{\alpha_{i,j}-1}\alpha_{i,j}$$

By taking summation, expectation given $\underline{L}_{(}n)$ and solving, we have

$$E\left[\underline{\mathfrak{L}}(n+1)^{T}\underline{O_{ne}}-\underline{\mathfrak{L}}(n)^{T}\underline{O_{ne}}|\underline{L}(n)\right]$$

$$=E\left[\sum_{i,j=1}^{N}(\Delta_{i,j}(n)L_{i,j}^{\alpha_{i,j}}+\frac{\Delta_{i,j}^{2}(n)}{2}\nu^{\alpha_{i,j}-1}\alpha_{i,j})|\underline{L}(n)\right]$$

it can be rewritten as

$$E\left[\underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}} - \underline{\mathfrak{L}}(n)^T \underline{O_{ne}} | \underline{L}(n)\right] \tag{2}$$

$$= E\left[\sum_{i,j=1}^{N} \Delta_{i,j}(n) | \underline{L}(n)\right] \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}}$$

$$+E\left[\sum_{i,j=1}^{N} (\frac{\Delta_{i,j}^2(n)}{2} \cdot \nu^{\alpha_{i,j}-1} \cdot \alpha_{i,j}) | \underline{L}(n)\right]$$

Now we just have to prove that the last term upper bound is finite. As $\alpha_{i,j} - 1 > 0$ it can be easily verified that:

$$(L_{i,j}(n) + A_{i,j}(n))^{\alpha_{i,j}-1} \leq 2^{\alpha_{i,j}-1} \cdot (L_{i,j}(n)^{\alpha_{i,j}-1} + A_{i,j}(n)^{\alpha_{i,j}-1})$$

and also

$$\Delta_{i,j}^2(n) \leq A_{i,j}^2(n) + 1$$

Using the inequalities the last term upper down becomes:

$$E\left[\sum_{i,j=1}^{N} (\frac{\Delta_{i,j}^2(n)}{2} \cdot \nu^{\alpha_{i,j}-1} \cdot \alpha_{i,j}) | \underline{L}(n)\right]$$

$$= 2^{\alpha_{i,j}-2} \cdot \alpha_{i,j} \cdot E\left[\sum_{i,j}^{N} (A_{i,j}^2(n) + 1) | \underline{L}(n)\right] \sum_{i,j=1}^{N} L_{i,j}(n)^{\alpha_{i,j}}$$

$$+2^{\alpha_{i,j}-2} \cdot \alpha_{i,j} \cdot E\left[\sum_{i,j}^{N} A_{i,j}^{\alpha_{i,j}+1}(n) | \underline{L}(n)\right]$$

$$+2^{\alpha_{i,j}-2} \cdot \alpha_{i,j} \cdot E\left[\sum_{i,j}^{N} A_{i,j}^{\alpha_{i,j}-1}(n) | \underline{L}(n)\right]$$

as $2 \leq \alpha_{i,j} \leq C_{i,j}$ terms $E\left[\sum_{i,j}^{N} (A_{i,j}^2(n) + 1) | \underline{L}(n)\right]$, $E\left[\sum_{i,j}^{N} A_{i,j}^{\alpha_{i,j}+1}(n) | \underline{L}(n)\right]$, $E\left[\sum_{i,j}^{N} A_{i,j}^{\alpha_{i,j}-1}(n) | \underline{L}(n)\right]$ are bounded summation [3], [6] and hence there exist $b_{i,j}$ such that:

$$E\left[\sum_{i,j=1}^{N} (\frac{\Delta_{i,j}^2(n)}{2} \cdot \nu^{\alpha_{i,j}-1} \cdot \alpha_{i,j}) | \underline{L}(n)\right]$$

$$= \sum_{i,j=1}^{N} b_{i,j}$$

which make 2:

$$E\left[\underline{\mathfrak{L}}(n+1)^T \underline{O_{ne}} - \underline{\mathfrak{L}}(n)^T \underline{O_{ne}} | \underline{L}(n)\right]$$

$$\leq E\left[\sum_{i,j=1}^{N} \Delta_{i,j}(n) | \underline{L}(n)\right] \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}} + \sum_{i,j=1}^{N} b_{i,j}$$

$$\leq \left(\sum_{i,j=1}^{N} (A_{i,j}(n) - S_{i,j}(n)) \right) \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}} + \sum_{i,j=1}^{N} b_{i,j}$$

Let $k = \sum_{i,j=1}^{N} b_{i,j}$ and from lemma:

$$E\left[V(n+1) - V(n) | \underline{L}(n)\right] \leq -\varepsilon \sum_{i,j=1}^{N} L_{i,j}^{\alpha_{i,j}} + k$$

hence proving theorem.