LEVERAGING PROACTIVE FLOW TO IMPROVE SCALABILITY IN SOFTWARE DEFINED NETWORKING

OSEGHALE OSEMUDIAMEN VICTOR

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

LEVERAGING PROACTIVE FLOW TO IMPROVE SCALABILITY IN SOFTWARE DEFINED NETWORKING

OSEGHALE OSEMUDIAMEN VICTOR

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: OSEGHALE OSEMUDIAMEN VICTOR

Matric No: WGA120028

Name of Degree: MASTERS OF COMPUTER SCIENCE

Title of Dissertation ("this Work"): LEVERAGING PROACTIVE FLOW TO IMPROVE SCALABILITY IN SOFTWARE DEFINED NETWORKING

Field of Study: COMPUTER SYSTEMS AND NETWORK

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

LEVERAGING PROACTIVE FLOW TO IMPROVE SCALABILITY IN SOFTWARE DEFINED NETWORKING

ABSTRACT

The concept of Software Defined Networking (SDN) is a concept which necessitates decoupling the network control plane from the data plane and makes each plane evolves independently. This controller architecture supporting versatile communication services is a complex paradigm. The need to coordinate the interaction of the control plane components such as QOS modules, routing information base, security policy, resource scheduling, etc. is a complex operation and based on the controller. The coordination of the actions of these components sometimes have conflicting goals and require careful handling. Some of these components exist in the network as distributed protocols and the inconsistency in coordinating these protocols could lead to critical problems in the network. Apart from the coordination of these distributed protocols, there is an issue that can arise due to flow request from data plane devices to the controller, which could eventually lead to a bottleneck as the network flow request scale at large network. The use of a centralized controller to address some of these challenges has been proposed by researchers. By centralizing the controller, the network simplifies the state distribution and hence have a better control over the network consistency. However, it is difficult to scale at larger network size and flow request. It is also important to maintain low latency in request handling and at the same time having high scalability throughput. For this thesis, an implementation of proactive flow programming on the data plane and improved I/O systems is proposed as leverage in the design of the SDN controllers. Through experiments, this work shows that implementing these concepts improve the controller scalability by enhancing the flow handling rate as the network size increases.

Keywords: Quality of Service, Security Policy, SDN Controllers, Throughput

ALIH PROACTIVE LEVERAGING UNTUK MENINGKATKAN KALUNGAN DALAM RANGKAIAN YANG DITERBITKAN PERISIAN

ABSTRAK

Konsep Rangkaian Perisian Terperinci (SDN) adalah konsep yang memerlukan pemisahan pesawat kawalan rangkaian dari satah data dan membuat setiap satah berubah secara sendiri. Seni bina pengawal ini yang menyokong perkhidmatan komunikasi serba boleh adalah paradigma yang kompleks. Keperluan untuk menyelaraskan interaksi komponen pesawat kawalan seperti modul Servis Kualiti (QoS), pangkalan maklumat penghalaan, dasar keselamatan, penjadualan sumber, dan lain-lain adalah operasi yang kompleks dan berdasarkan pengawal. Penyelarasan tindakan komponen ini kadang-kadang mempunyai matlamat yang saling bertentangan dan memerlukan pengendalian yang teliti. Sesetengah komponen ini wujud dalam rangkaian sebagai pengagihan protokol dan keadaan yang tidak konsisten dalam menyelaraskan protokol ini boleh membawa kepada masalah kritikal dalam rangkaian. Selain dari penyelarasan pengagihan protokol ini, terdapat satu isu yang boleh timbul kerana permintaan Aliran dari peranti pesawat data ke pengawal, yang akhirnya boleh membawa kepada kesesakan kerana permintaan aliran rangkaian berskala pada rangkaian yang besar. Penggunaan pengawal terpusat untuk menangani beberapa cabaran ini telah dicadangkan oleh penyelidik. Dengan memusatkan pengawal, kita dapat mempermudahkan pengedaran kawasan rangkaian dan oleh itu mempunyai kawalan yang lebih baik ke atas konsistensi rangkaian. Walau bagaimanapun, sukar untuk berskala pada saiz rangkaian dan permintaan aliran yang lebih besar. Ia juga penting untuk mengekalkan kependaman rendah dalam pengendalian permintaan dan pada masa yang sama mempunyai penskalaan penghantaran yang tinggi. Untuk tesis ini, pelaksanaan pengaturcaraan aliran proaktif pada pesawat data dan sistem I / O yang lebih baik dicadangkan sebagai penekanan dalam reka bentuk pengawal SDN. Melalui eksperimen, kami akan menunjukkan bahawa perlaksanaan konsep ini meningkatkan kemampuan pengawal penskalaan dengan meningkatkan kadar pengendalian aliran apabila saiz rangkaian meningkat.

Kata Kunci: Kualiti Perkhidmatan, Dasar Keselamatan, Pengawal SDN, Daya Pemprosesan

ACKNOWLEDGEMENTS

To my life-coach, Rev. (Dr.) Chris Oyakhilome DSc. DD and my parents Mr and Mrs Oseghale: because I owe it all to you. Many Thanks!

Special mention goes to my enthusiastic supervisor, Madam Fazidah Binti Othman. My Masters has been an amazing experience and I thank my supervisor wholeheartedly, not only for her tremendous academic support during the Thesis writing, but also for giving me so many wonderful supports as my referee for my current employment.

Similar, profound gratitude goes to Bimba Andrew, who has been a truly dedicated academic mentor. I am particularly indebted to Andrew for his constant faith in my lab work, and for his support when so generously responding to all my request. I have very fond memories of my time with him.

Special mention goes to Hayden Gilgan, Gloria Benjamin Dusman, Pastor Rhema for going far beyond the call of duty.

Finally, but by no means least, thanks go to siblings, for almost unbelievable support. They are the most important people in my world and I dedicate this thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

CMIP	:	Common Management Information Protocol
NIB	:	Network Information Base
NMP	:	Network Management Protocol
QOS	:	quality of service
SDN	:	Software Defined Network
VOIP	:	Voice Over IP

CHAPTER 1: INTRODUCTION

According to the Open Network Foundation (Sezer et al., 2013), Software Defined Networking (SDN) is an architecture based on the decoupling of the Network control and forwarding functions which makes the network control directly programmable, centralized and the underlying infrastructure abstracted for applications and the network services. The application layer which comprises of the business logic communicates with the controller with the northbound application programming interface, while the communication of component in the separate planes, controller and forwarding of the data plane uses the Southbound API. The responsibility of control in the data plane devices such as switch and router, by the control plane is enabled by the OpenFlow® protocol, which is a standard communication interface based on the controls and forwarding planes of the SDN architecture (Bakshi, 2013). This forms the basic building blocks for SDN solutions as shown in Figure 1.1 Basic SDN Architecture.

The controller allows to manipulate flow entries of different devices, but the OpenFlow® helps updates these flows.

OpenFlow® enabled switches to have one or several flow tables in a pipeline, and a group table either for packet lookup and also fundamental for packet forwarding; it also consists of a channel (TCP, TLS) to an external controller (Lantz, Heller, & McKeown, 2010). The TCP or TLC connection can be established over the Ethernet or the Internet by the legacy routing protocols.

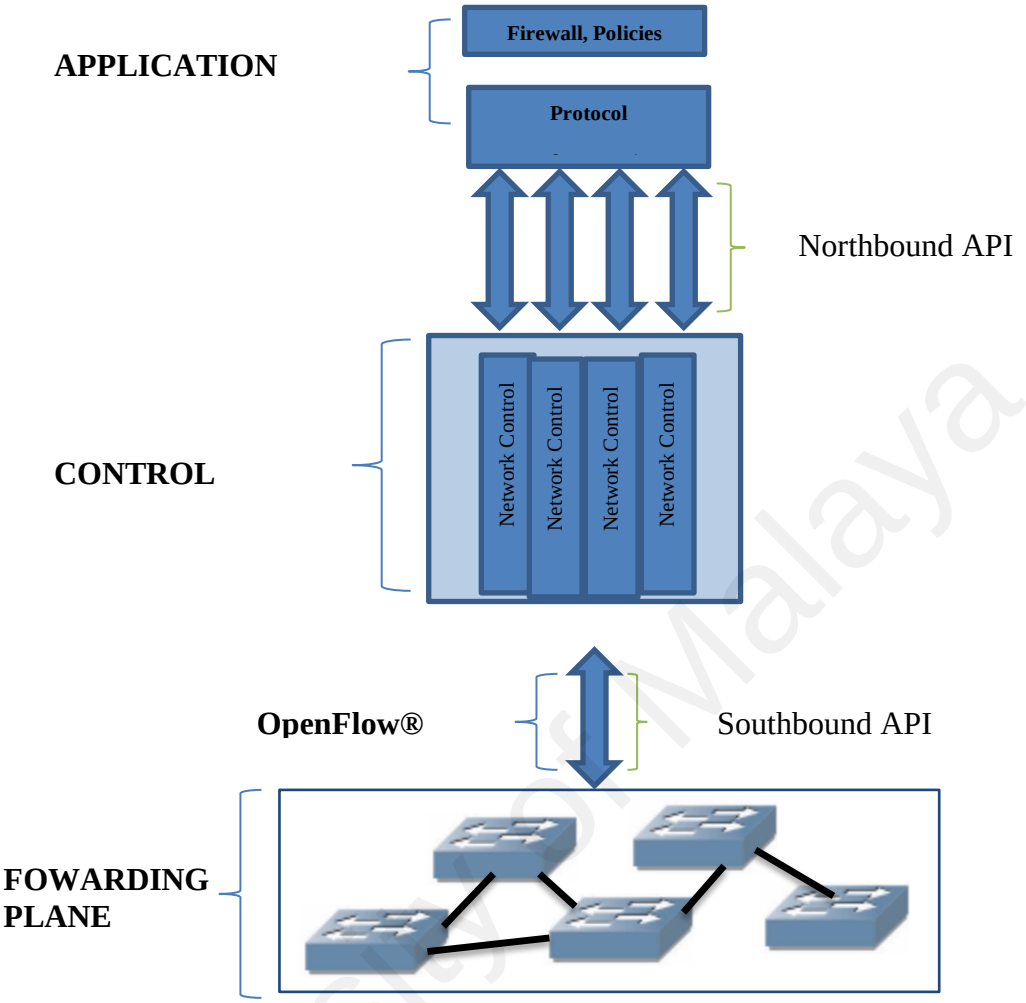

Figure 0.1 Basic SDN Architecture

A switch has to only support a flow table to be compliant with the standard, but multiple tables give scalability when you have to match a large flow or match a large set of data in a group table. The group table group multiple interfaces into a single group, and when traffic is sent to the group, it is forwarded out of all interfaces in that group. So, if a broadcast or multicast application is to be implemented, a group table will be used. The OpenFlow® protocol contains a set of API that the control plane controller leverages to control the elements of the flow entries (DELETE, ADD and UPDATE) in the flow table of the data plane devices in a reactive or proactive state. The flow table's entities or entries consist of MATCH fields, counters and set of instructions that are applied to matching packets. These packet matching is first initiated on the first flow table usually

Table0 and may continue to additional flow tables in the pipeline using the GOTO Tables. Matching is matched in a priority order of higher priority to lower priority. When traffic arrives, it is first matched in against entries in Table 0. Matching flow entry in the flow table executes the instruction associated with the specific flow entry executed i.e. drop, Go to table X, send out of Port X, etc. If for a particular flow, there is no entry found to match such flow in the flow table, the outcome of such flow depends on the controller for processing

The independent evolution of both the control and data plane is enabled by this decoupling. This leads to several significant benefits.

1.1 SDN

With the evolution of cloud computing, several eco systems and business paradigm are changing and demand for network centric applications, business logics are transcending to a more complex paradigm. To meet these trend, there is a need for a dynamic network logic and open system to enable the dynamic nature of business and network requirement. SDN is a promising architecture where the decoupling of the data plane and the control plane makes the network functions readily programmable, and is a potential advantage for the need of recent network trends. Due to real time performance, cost-effective, dynamic, manageable, adaptable state and high availability requirement, telecom networks are adopting the concept of Software Defined Networking (SDN) which aims to transform the way networks function.

1.2 Problem Statement

According to the work done by (Murat Karakus, 2017), regardless of the definition of scalability or its knowledge by system users, it is an important consideration for SDN architecture. Scalability challenges and mitigation proposed for a network may introduce tradeoffs that may be introduce mitigating factors for other useful properties in the

network. A useful example of this is; in a proactive rule setup in SDN switches reduces the load of the controller, with an outcome of reduced processing time and flow initiation overhead in the controller. However, this constrain the flexibility coming from reactive flow proactive flow setup, reduces decision-making dynamicity of the controller and management of the network.

(Murat Karakus, 2017), also proposed that controller distribution is one way to overcome computational load on controller but it brings consistency and synchronization problems as well.

Despite the benefits of SDN, there is a vital and critical concern about the scalability as the network devices and application interaction grows (increase in the number of data plane devices i.e. router, switches, etc.), flows and bandwidth (Yeganeh, Tootoonchian, & Ganjali, 2013) will lead the controller to become a bottleneck of itself or fail to handle incoming request while providing consistent service level guarantee. These challenges although not peculiar with SDN but they arises from the decoupling and independent evolution of the planes. Because of this decoupling, the flow setup process could create potential overheads which could limit scalability.

When packet arrives at the ingress switch and it does not match any flow table entry, the switch forwards the packet to the controller with flow request on how to treat that packet. The controller forwards the packet to the destination switch and pushed the new flow entries to the switches which then update the entries to the flow table in the network. Subsequent flows are forwarded by the updated flow table rule.

As the flow and network request increases, these flow request and update process can produce overheads which can subsequently lead to limited scalability in the controller as shown in Figure 0.2.

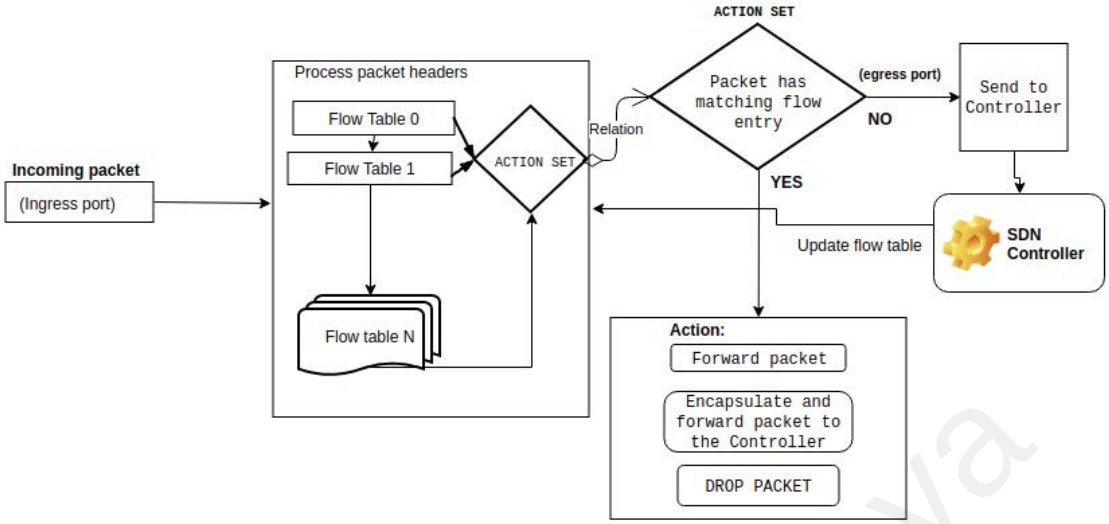

Figure 0.2 Packet forwarding in SDN

Figure 0.2 shows a high-level flow of packet in SDN network, for this illustrative diagram, the details of each component are not included for clarity, as it's intended as a high level overview. In the illustration, when incoming packets arrive at the data plane, it checks the flow table for a match for the packet, it first starts in table 0, if the flow needs to continue to another table, ***GOTO*** statement routes the flow to the table specified in the instructions. If there is not matching flow in the flow table, the packet is pushed to the controller. The controller processes the packets and route the flow to the required egress port and update the data plane flow table with the entry of this packet.

1.3 Objectives

The main aim of this thesis is to critically explore the root causes of limited scalability in SDN and propose a solution. To achieve this, the following objectives are presented.

1. To design an architectural framework for SDN data plane that implements proactive flow.
2. The framework will implement a proactive trigger for packet forwarding to the controller if the packet reaches specified threshold

3. To evaluate the developed module by testing it on a simulated SDN network and compare with a Reactive flow of the same simulation to mitigate scalability issues in SDN.

This thesis explores the root causes of limited scalability in SDN and propose a solution for the scalability challenge.

Representational state transfer (REST) API is used to proactively program the flow table of OpenFlow enabled switches. REST puts together sets of controls applied to designing set of flows that can be used to program the data plane. The evaluation will be carried out on a testbed composing of OpenVSwitch run on Docker container, OpenDayLight controller run on an Ubuntu Server, a virtual network simulator and a modify 'cbench', a bash script for network evaluation to suit the purpose of testing for scalability.

1.4 Methodology

The approach is to first get a detailed clarification of the problem by reviewing relevant literature to identify understand the relevant research aspect of Scalability challenges in SDN and studied methods that has been previously used to solve this problems. With these findings, a design of the framework which uses threshold trigger in a proactive flow SDN; only when a particular flow satisfies the conditions for a trigger will a controller be involved with the flow control of the SDN network herby reducing the flow setup overhead which is identified one of the basic causes of the Scalability challenge in SDN. We tests this framework by implementing it with an Open source controller Open Daylight (ODL) which allows for the implementation of Northbound API programmed with JAVA programming language and REST API.

1.5 Mitigating the Scalability Challenges

By reconnoitering proactively programming flows to the data plane flow setup and request rate that can cause an overhead to the control plane can be addressed and also increasing control plan processing capacity can be a good balance for fulfilling the SDN architectural goal. A comparison of the implementation with ODL (Medved, Varga, Tkacik, & Gray, 2014) and HP VAN SDN (Dixon et al., 2014) controllers in reactive control state. The key significance of our experiment is that with the proactive flow programming implementation leveraging high control plane processing, throughput at high data path devices is still achieved and there is an improved scalability at scale. Flow overhead is limited by the proactive flow. The design and experiments show an extensive insight in throughput, scalability, latency in the load based SDN implementation setup.

1.6 Summary and Layout of this thesis

In this chapter a high level concept of Software Defined Networking (SDN), the eminence of implementing it in the present Information Technology (IT) networking environment is discussed, as well as the challenges in the deployment of SDN. Additionally we highlight the aim of using the proactive flow on data path and improve I/O performance to enable scalability in the deployment of the SDN.

In Chapter 2 a background study is done to understand the relationship between the complexity of the computer network and the elements that cause the overhead in the network because these overheads are a vital cause of the scalability challenge in the SDN. In this chapter, the goal of using proactive flow in the SDN setup to mitigate the scalability challenge is emphasized, and finally a brief discussion on how the tools required to achieve SDN flow programming. In Chapter 3 a review in the Evolution of the legacy network is explained; and the various work done to mitigate the scalability challenge in SDN. In chapter 4 is the explanation or architecture for the proposed system,

and the various critical aspect of the architecture. In chapter 5, an evaluation of the system and the results to determine if the proposed objectives were achievable by this work done. And a conclusion by detailing contribution and future work to be done to improve this work.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

The concept of computing has evolved in the past decade. The continuous replacement of dedicated servers with virtualized cloud data centers with lower cost and high flexibility. This is built on middleware and virtualization technologies. The manual performance of management and usage of the network elements is usually slower and an outage in the network can lead to a significant loss, with the less predictability of network change.

An approach to solve this evolving paradigm is the decoupling of the network from computing volatility into one flat static communication service.

However from the underlying network, traffic from each tenants need to be segregated for the need of security and performance. This imposed requirement network elements like deep packet inspection, and QOS and Load balancing which are on-demand need to couple to computing functions and this cannot be fully achieved with the flat static communication configuration. The need to map this static network to a dynamic configuration has led to the innovation of SDN, which uses software network switches (e.g. Open vSwitch) to rout packets from the different tunnels over the network established by separate virtual machines. SDN automates network elements and configurations for a unified network-wide communication. The connectivity requirements of the network are communicated to the network controller which used API's such as OpenFlow® to implement the required in the virtual switches (Tourrilhes, Sharma, Banerjee, & Pettit, 2014).

According to a survey made by (Murat Karakus, 2017), there is an inherent scalability issue as a result of the delay in new flow rule setup processing in the control plane, and

several modes of flow setup some of which are *proactive mode* and *reactive mode*. These two are prominent modes to setup a flow in the flow table of SDN.

In proactive mode, latency is not introduced in the flow rule setup from the controller's point of view, however, in the reactive mode, the controller response time is crucial as some latency is introduced.

OpenFlow which is a flow-based programing standard for Ethernet switches was proposed from the success of Ethane and SANE. This standard defines that an OpenFlow enabled Ethernet Switch will forward traffic according to the rules defined in the flow table of such switches; these rules are pushed to the switches by the central controller. The flow tables contain rules which consists of actions with which a packet will be processed based if matched, and communication with the central controller is by a secure OpenFlow Channel.

OpenFlow provides opportunities for researches to test and develop new applications for the control plane and enables researchers to learn more about traffic classifications.

The OpenFlow specification defines that for every flow that arrives at the data OpenFlow enabled switch, the data is first forwarded to the central controller and the controller sends an update to the flow table on the action with which the packet is to be processed. In the scenario of a large network, there is a large flow request and as well a large flow overhead. This is the primary cause of a bottle neck to the control plane. This work which is based on the OpenFlow standard defines that for each flow that arrives at the data path, they are processed by the proactive flow defined in the switch memory. Our concept limits the controller interaction at flow setup. The controller explicitly provisions the network needs. The issue with this centralized management system is that as the underlying network flow requirement increases, there will be need for the controller to scale and in

most cases, this is not so because the controller is not able to meet the flow request of the underlying network device. A solution to this concern is to improve on adequate flow management. VMware and HP provisioned a solution where the controller manages the elephant flows in the underlying network. Another solution to this is to eliminate the network learning and discovery and the core switches functions as the forwarding devices, and tunnel configurations are programmed through OpenFlow®. This division of functions between the control and forwarding path is evolving. On the data-path, complex packets can be treated while the decision policies are handled by the controller functions. This idea relies on the delegation of some of the control functions to the data plane so as to alleviate the load on the controller(s), thereby reducing controller switch communication frequency.

2.2 4D Architecture

The management and control planes management complexity is the root reason for an uncontrollable and fragile network as argued by the authors (Tourrilhes et al., 2014). The packet forwarding functions coupled with the control plane logics distributed among the network elements limits the enforcement of network-wide objectives; however management goals are achieved by ad hoc ways of manual scripting. The lack of network-wide coordination between network elements makes it error prone whenever there are some changes in the network. Control of all functions is realized through inherent tuning i.e. the correct and successful setup of the data path is a factor implicitly important for the management control to function correctly. This brings about dependency of the control plane on the data plane. But from the concept of SDN, they are to evolve independently.

A complete repartitioning of the functionalities of computer networks which is called 4D is a proposed solution in an attempt to address these problems (Greenberg et al., 2005; Yan et al., 2007). It is a four-plane architecture comprising of Decision, Dissemination,

Discovery and Data planes. The distributed protocols handling packet forwarding is separated from (Schehlmann, Abt, & Baier, 2014) the decision logic of the network. Based on the network-wide view of the underlying data plane, the decision plane specifies the network-wide objectives which are translated by some algorithms into control configurations for the data plane devices. The data plane handles packet forwarding, filtering queuing and address translation; these functionalities are controlled by the decision plane to fulfill the networks objectives. Between the decision and data plane is the communication mechanism handled by the dissemination plane. The data planes changes with the change of the plane configuration policies, instead dissemination plane which requires no pre-configuration and needs reliability should not change with changes in the data plane. The discovery of the physical network components and the creation of logical identifiers to present them is the responsibility of the discovery plane. It is also the responsibility of the discovery plane to collect measurement data of the network-wide view for the decision plane to achieve the networks objectives. Exchange of secret keys at the boot strap stage phase is required for the security of the discovery plane. 4D has no real prototype built however it brings researchers attention to very important challenges in the present network management and control plane. And creates a trend aimed at the clean slate design of network architecture.

The work done in this thesis focuses on the data plane from the proposed 4D architecture. 4D only enumerates the data handing functionalities; but in our work, a dive into how to realize these functionalities without the complete control of the decision plane is made. The total control of flows in the data plane by the control plane would likely lead to a bottleneck in the decision plane as the network scales; our work is based on how to allocate certain low level flow to the data plane and let the decision plane handle only elephant and complex decision flows requiring special treatments. We are confident that

our work will contribute significantly in solving the scalability concern in this research area.

2.3 NOX and BEACON

NOX and BEACON is a follow up on the success of SANE and Ethane (Erickson, 2013; Gude et al., 2008). Previous controller architecture has a complex control plane in which all the control functions are hard coded, this lives the users with a difficult task of replacing or writing control components for the desired goal. Modular and flexible control plane architecture will make it easier for users to achieve the network management goals. NOX is designed to provide a modular and flexible design framework for users to achieve complicated control function using OpenFlow® enabled switches (Gude et al., 2008). The OpenFlow® design, where the controller installs every flow in the data path; if the controller does not have enough control plane capacity to handle all these request, it will become a bottle neck of itself, and unfortunately NOX can only utilize a single CPU core. NOX is designed to reduce overhead by not relying on the I/O operations but it is not multi-threaded to leverage multi-core processing. NOX handles each request individually so overhead is introduced by such operation. BEACON is a multi-threaded programmable OpenFlow® controller which allows users to write simple threaded applications and can be run together with NOX in parallel to enable Scale in multi-core processors (Erickson, 2013). However in practice, not all workers tread run at the same rate and this could lead to a bias even under a uniform workload. They use static batching strategy to improve the throughput of individual workers tread and this can lead to large request handling latency under heavy load. A good way to improve the performance of NOX and BACON is to commit the flow handling to the OpenFlow® enabled data plane devices and only control policies are handled by the controllers which would reduce the flow overhead limited workers tread. So we propose that our implementation of proactive flow for flow setup of

data plane and enhancing the scalability of the control plane capacity which is a major contribution to the implementation concept for enhancing scalability in SDN.

2.4 McNettle

The work by Andreas V et.al is a motivation from the argument that the simplicity of the logically centralized controller comes at a cost of control plane scalability and changes in network state is a problem to distributed controllers (Voellmy & Wang, 2012). McNettle is extended by writing event handlers and background programs in high-level programming language with a shared state and memory transaction. The control processing events throughput scales with the number of CPU cores. The event handler includes the packet miss function which updates switch local and network state variables. Messages from each switch are handled in sequence and from different switches are executed concurrently, this provision a synchronized access to the network state. It fully parallelizes waiting and dispatching of threads on I/O devices stabilizing the multi-core load balancing algorithm and prevent excessive cache-trashing due to work migration. McNettle avoids contention on sockets by processing each message on a single CPU core thereby reducing inter-core synchronization except required by user-specified controller logic.

This framework was tested in Haskel and leveraged the multi-core facility of the Glasgow Haskel Compiler and runtime system; it was shown to reduce system calls and optimizes cache usage and runtime overhead (Terei & Chakravarty, 2009). It is also shown that it schedules event handlers, optimizes message parsing, allocates memory and serialization. The implementation also shows that McNettle can achieve a throughput of over 14 million flows per second (Voellmy & Wang, 2012). From the evaluation, McNettle scaled better that NOX a distributed controller for more than 30 CPU cores and the previous stopped scaling at 20 CPU cores.

2.5 DevFlow

The authors argued that the design implementation of OpenFlow® comes at the cost of excessive overheads; and that this cannot meet the need of today's high-performance network. The involvement of the switch implementation where the switch control plane is involved so often and the frequency of the distributed system involving the control plane; both for flow setup and statistic gathering. Devflow is a modification of the OpenFlow® model with the objectives of gently breaking the coupling of the control logic and global visibility (Curtis et al., 2011). In this model, switch-internal communication between the control and data plane are reduced and potentially reduce the need for a controller for most flow setup. Rule cloning, statistics gathering, approximate counters, and the bin packing algorithm are used to achieve these objectives. For the Rule cloning, the action properties of the installed wildcard configuration is augmented with the flag CLONE, and the data plane switches follows the default wildcard behavior when flag is clear otherwise to create a new rule, it locally clones the wildcard rule with all field are replaced with the with values matching the micro flow. This rule are installed in the lookup table and consequently reduces the switch TCAM usage and also the cost of power. DevFlow multipath support is relatively bring into line to the equal-cost multipath (ECMP) routing, however, in ECMP the flows are split evenly across selected paths and it is a none ideal case because some path may consume relatively more bandwidth compared to the others (Chiesa, Kindler, & Schapira, 2017). Devflow allows clonable wildcard rules to randomly select flow path according to some probability distribution. Statistics gathering uses sampling where each packet headers is sent to the controller in 1/1000 probability and Threshold Triggers which sets a threshold per flow to be reached before flow are sent to the controller, thereby living only fine-grained flow at the data path and elephant flows to be processed by the controller. When flows match these wildcard rules, an approximate counter using streaming algorithm is used to track these

flows thereby enabling a fine-grained statistic collection. The evaluation shows that DevFlow balanced data center traffic without as many overheads. In a reactive manner, all flows are installed only if identified as elephant flows. In multiple simulations performed on a large data network simulator show that the pull-based setup with minimum update interval maximizes data throughput in the network (Curtis et al., 2011). In this case, the performance comes at a price, significantly large flow table in comparison with the other schemes.

The Threshold Triggers is most optimal, as the data throughput increases, less traffic is required between the switch and the controller thus flow table size is minimal.

2.6 DIFANE

DIFANE proposed a suitable solution for the flow based network. This solution keeps all traffic in the data plane by selectively redirecting packets through intermediate switches or Authority Switches, that store the necessary rules, and relegates the controller to the simple task of partitioning these rules over the switches (Tootoonchian, Gorbunov, Ganjali, Casado, & Sherwood, 2012). DIFANE partitions the rule space and assigns each partition to one or more Authority switches that handle packets and sends the rules to ingress switches. In this model, when a packet arrives at the ingress switches with no matching flow, it is forwarded to the Authority switch which pushes the relevant rule to the ingress switch; the ingress switches caches this rule locally. Subsequent rules can be encapsulated and forwarded to the egress switch by packet redirection. DIFANE controller uses partition algorithm to subdivide the space of all rules. The controller pre-computes the low-level rules based on the high-level policies by substituting high-level names with network addresses. To avoid redirecting all flow traffic to the Authority switch, the authority switches caches all rules in the ingress switch. DIFANE achieves a small delay for first packets of a flow compared and a higher throughput to NOX

(Tootoonchian & Ganjali, 2010). DIFANE scales linearly with the number of authority switches; the Authors propose the use of authority switches to keep packets in the data plane.

In this reviewed literature, the solution proposed is to divide the workload of the network over several instances of a controller. However, difficulties come with optimum scalability solution for SDN networks. From the observation in this literature, flows are reactive as against our proposed framework. These problems can be sufficiently solved by combining the reduction of control overhead, distributing multiple controllers among a network and proactively programming flows.

2.7 HyperFlow

Hyper-Flow extends NOX into a distributed control plane by Synchronizing network-wide state among distributed controllers. HyperFlow localizes the processing of a single flow to an individual controller machine, and minimizes response time forwarding plane requests and subsequently improve throughput. To complement the solutions that are aimed at leveraging the performance of the control plane physical controller machine, several types of research have been done to enable a cluster of the control plane controller working together as a single logical controller for improved scalability. HyperFlow is implemented as an application for NOX by making minimum changes to NOX, and minimum modification to NOX application (Tootoonchian & Ganjali, 2010). By distributed file system, network state is synchronized among these distributed controllers. The processing of flow request is localized to a single controller to minimize the control response time from the data plane request; this scale the system throughput. The first limitation according to the Authors was found in the (WheeIFS) performance. All network events and status information need to be synchronized among the controllers and this requires fast distributed storage systems. This leads to the fact that the global view of

the network not converging in a timely manner. HyperFlow is resilience to network partitioning and components failures because of the distributed instances running in parallel, however, there is no guarantee against network state inconsistency (van Asten, van Adrichem, & Kuipers, 2014). This is a common challenge with the distributed controllers. In Hyper flow, traffic is dropped when it exceeds the controller processing capacity, even if the controller load is reduced by assigning lesser numbers of the switch to it, a single switch can generate a large traffic which will subsequently be dropped by the controller.

2.8 ONIX

The network state inconsistency of the distributed controller is a common challenge with. By default, ONIX is proposed to provides a framework for building distributed coordination in the network control plane (Koponen et al., 2010). The Network Information Base (NIB), is provisioned to provide access to different network states and synchronization framework of different network availability requirements. The NIB uses two data stores SQL-database for slow changing topology information and a Dynamic Hash table for link utilization which has a rapid trip time. This application of these two data storages mitigates the challenges faced by Hyperflow Controllers. ONIX has switch availability and status of link information which so the path computation is not limited to the information provided by the OpenFlow protocol. The computed paths are installed in the switch flow table by the ONIX distributed. Combining the HyperFlow and ONIX, a single controller can be developed to become fully distributed to attain a better scalability and availability in the network. The advantage of the ONIX is the partitioning of the payload over multiple instances thus if one ONIX instance has a limiting traffic throughput due to the high payload, switches can be reassigned to other instances of ONIX.

2.9 SANE

Inspired by the 4D architecture, SANE leverages a clean-slate approach with separation of routing control plane from data plane or forwarding plane (Casado et al., 2006). Just like 4D where the routing and the security is centralized, this approach is also leveraged by SANE. Spanning tree at the Control plane carries traffic between forwarding devices and the central controller similar to the role of the dissemination plane proposed in 4D. However, SANE does not allow the communication between end hosts. SANE argues that its way is better than the other one, because it requires no interaction between the routing and filter/firewall control. We think that although such security policies enforcement is very strong, it on the other hand limits the functionalities that with filters/firewalls a network can realize. For example, with filters/firewalls, not only malicious traffic can be blocked, but legitimate traffic can be shaped or modified, to achieve different goals. Since the central controller is really critical in SANE, whether it can scale up its throughput is very important. Again we argue that, ideas developed in our work in the proceeding chapter would help in scaling the network.

2.10 Ethane

Ethane is a follow-up of the SANE except that Ethane takes a less complex approach than SANE (Casado et al., 2009). Ethane is similar to SANE in Incremental deplorability except that Ethane can be incrementally deployed in an Enterprise network unlike SANE that require a complete replacement of the enterprise network. In Ethane, the central controller is used to compute for the enforcement of security policy for flows in the network (Lara, Kolasani, & Ramamurthy, 2014). The central Controller contains the global network policy for all packets processing. The controller has a global view of the network and the security policies determine the applied filter to each flow presented by each packet. The controller can be replicated to enhance redundancy and scalability. The Switches are simple and Consisting of a simple flow table and a secure channel to the

Controller. When a packet arrives that is not in the flow table, the Switch forwards that packet to the Controller and it is forwarded according to the Controller's directive. Not every switch in an Ethane network needs to be an Ethane Switch. Ethane design allows Switches to be added gradually, and the network becomes more manageable with each added Switch. Ethane showed that there are different ways of replication for the system robustness improvement. Programmed by the policy composition, Language Pol-Eth was proposed to program security policies based on identity binding. Ethane address source address spoof problem by binding entities to their location for security enforcement.

The deployment of the real system at the Stanford's Computer Science department for real experience in designing and evaluation is an important contribution of Ethane. Different ethane switches were built and Ethane is used to achieve the security policies for the campus network. They claim from their evaluation of ethane that one central controller can handle a network with as large as 22,000 hosts. This evaluation was done with the work-load of the campus.

Adding new features to the control plane might not be an easy task. The interaction between the controller's elements are not modularized but hard-wired, this makes the management a non-trivial task. The authors recognized this problem and thus reintroduced the NOX which we will discuss in a later section (Gude et al., 2008). The central controller cannot scale up in Ethane very well as required.

Table 0.1: The various work done to solve scalability challenges SDN, their objectives, findings and challenges

Publication	SDN Implementation	Objectives	Methodology	Findings	Challenges	Flow
(Tourrilhes et al., 2014)	4DArchitecture	Centralization of network wide decision making to ease management complexity	Partitioning network functionalities	Optimal network control for shortest-path routing.	Only handing functionalities in the controllers	O
(Erickson, 2013; Gude et al., 2008)	NOX and BEACON	Modular and flexible design framework using OpenFlow enabled switches.	BEACON runs together with NOX in parallel to enable Scale in multi-core processors. Static batching.	Throughput Scalability is achieved by parallelism.	Bias Large request handling latency under heavy load	O
(Voellmy & Wang, 2012)	McNettle	Writing event handlers. Shared state and Memory Transaction	-Event handler to include the packet miss. -Parallelizes waiting and dispatching of threads on I/O devices. -Avoids contention on sockets.	Maximum throughput. Reduce system calls. McNettle scaled better that NOX.	Large CPU Core requirements	O
(Curtis et al., 2011)	DevFlow	Breaking the coupling of the control logic and global visibility. switch implementation system involving the switch control plane for	Rule cloning, statistics gathering, approximate counters, and the bin packing algorithm Cloneable wildcard rules.	Helps operators target only the flows that matter for their management problem.	Significantly large flow table in comparison with the other schemes.	I

Publication	SDN Implementation	Objectives	Methodology	Findings	Challenges	Flow
		both flow setup and statistic gathering comes	Statistics gathering Threshold	Reducing control-plane for most flow setups. Scalable management architectures.		
(Yu, Rexford, Freedman, & Wang, 2010)	DIFANE	Suitable solution for the flow-based network	Traffic monitor. Authority Switches.	Achieves small delay for first packets of a flow compared and a higher throughput to NOX. Scales linearly with the number of authority switches.	Much Authority Switch needed	O
(Tootoonchian & Ganjali, 2010)	Hiperflow	Distributed event-based control plane for OpenFlow.	Minor modifications to previous control applications. Synchronizing network wide state among distributed controller. Event Propagation.	Improved flow handling rate. Handles fluctuation in network synchronization. Enables network operators deploy any	Network state inconsistency	O

Publication	SDN Implementation	Objectives	Methodology	Findings	Challenges	Flow
				number of controllers to tune the performance. Resilient to network component failures		
(Koponen et al., 2010)	ONIX	Framework for building distributed coordination in the network control plane.	Network Information Base (NIB). Synchronization framework.	Fully distributed to attain. Scalability. Availability.	Scalability and Inconsistency	O
(Casado et al., 2006)	Sane	Actions of both routing and access control in a centralized plane for right security policies.	Separation of plane. 4D approach of centralized routing and the security.	Deployable in current networks. Scale to networks of tens of thousands of nodes.	Central Controller lacks scalability	O

Publication	SDN Implementation	Objectives	Methodology	Findings	Challenges	Flow
(Casado et al., 2009)	Ethane	Define a single fine-grained policy for enterprise network to operate.	Ethernet switch. Ethane Extends SANE Security, flow management; Incremental deplorability; Significant deployment experience.	Compactable design with existing network Manageability. Straightforward to add new features. Ease of innovation and evolution.	Management Complexity	O

O ---- No proactive flow method used.

I ---- Proactive flow method used.

2.11 Summary

In this chapter, a discussed of the evolution of Network system was done and a review of some researches done in the process of attaining a reliable and efficient network that will leverage scalability as an important advantage. This chapter is concluded by summarizing the uniqueness of our proposed techniques as a contribution to this evolution progress. We discovered that much work has not been done to using proactive flow in deploying SDN for enterprises.

The review show that the current controllers will depredate at scale and will not able to meet increasing demands in communication for future network traffic as new flow are introduced. Despite the various optimization efforts, a centralized control logic remains subject to the single-point of failure issue. Needless to mention the harmful consequences that may occur during a controller failure in a dense network. Additionally, as the network expands both in size and space, the centralized model will inevitably encounter several limitations.

Proactive flow improves scalability, which perfectly fits the demands of fine-grained functional of controllers. In proactive flow, we install flow rules on the data plane, so some responsibility of flow control is pushed to the data plane and only for flow that reach certain threshold is pushed to the controller for processing. With this, the shared responsibility of the control plan and data plane will allow for scalability in SDN as the network increase. This is a motivation for this thesis.

CHAPTER 3: BACKGROUND OF CONCEPT OF NETWORK COMPLEXITY AND PROACTIVE FLOW

3.1 Introduction

There are six major components of an SDN network.

The first plane is the management plane. The management plane consists of network applications which are responsible for the management of the control logic in the SDN. In place of a command line interface, SDN enabled networks use programmable interfaces for flexibility and ease to the task of implementing new applications and services, such as routing, load balancing, policy enforcement, or a custom application from a service provider. It also allows orchestration and automation of the network via existing APIs.

Second is the control plane that is the most intelligent and important layer of an SDN architecture. It contains one or various controllers that forward the different types of rules and policies to the infrastructure layer through the southbound interface.

Third, the data plane, also known as the infrastructure layer, represents the forwarding devices on the network (routers, switches, load balancers, etc.). It uses the southbound APIs to interact with the control plane by receiving the forwarding rules and policies to apply them to the corresponding devices.

Fourth, the northbound interfaces that permit communication between the control layer and the management layer are mainly a set of open source application programming interfaces (APIs).

Fifth, the east-west interfaces, which are not yet standardized, allow communication between the multiple controllers. They use a system of notification and messaging or a distributed routing protocol like BGP and OSPF.

Sixth, the southbound interfaces allow interaction between the control plane and the data plane, which can be defined summarily as protocols that permit the controller to push policies to the forwarding plane.

The OpenFlow protocol is the most widely accepted and implemented southbound API for SDN-enabled networks.

OpenFlow is normalized by the Open Networking Foundation (ONF) (M. Smith et. al, 2014), backed by the leaders of IT industry like Facebook, Cisco, Google, HP, and others. For this reason, understanding the OpenFlow architecture is important to grasp the notion of SDN, which we are going to present in the next subsection. Before that, we should realize that OpenFlow is just an instantiation of SDN, as there are many existing and under development southbound APIs, for instance, CISCO OpFlex (United States of America Patent No. US10033622B2, 2015), which distributes some of the complexity of managing the network to the infrastructure layer to improve the scalability.

As previously explained, the intent of this thesis is to implement a scalable mechanism to reduce controller overhead in a software defined network. In this perspective, a possible improvement in the current OpenFlow (OF) communication model can be achieved by more precisely proactively programming flow to meet predefined rule. This mechanism enables switching nodes to instantly redirect traffic in advent of node or link overload without having to re-compute a backup path. Implementing proactive flow in OF-based networks requires the controller to proactively forward backup rules on each switching

node. Thus, this mechanism can potentially introduce an additional load on the controller and as such, requires a careful deployment.

3.2 Overload Caused by Complexity of Computer Network

Computer networks mainly deployed simple data communication channels among connected computers. The question of data flow was the main decision of the computer network. However, this has changed with the evolution of the architecture and requirements of computer networks. As a result of this trend, computer network requires complex operations like traffic engineering, security policies and well grained control to address each block of control decision with fair isolation from other component; this brings about the concepts of virtualization which is an underlying technology for software defined networks.

In an enterprise network, the routing decision responsibility is handled using the OSPF protocol component, while global routing decision is the responsibility of BGP protocol component. In enterprise network packet filter placement and configuration component are responsibility of blocking, dropping are related traffic engineering policies, and the packet redirection serves for load balance across multiple servers, suspicious traffic forwarding to the Intrusion Detection System; the QOS(quality of service) routing enables capability of voice over IP (VOIP) traffic with low delay and loss rate; the virtual private intra-networks is handled by the tunneling services, and so on. With this complexity of managing, a network which requires scale will introduce more and more complexity for handling critical functions. This trend leads to an accurate prediction that network will continue to grow and become more complex.

Although SDN architectural approach and control components helps to decompose this complexity in operation and management of networks into more manageable pieces, it is also critical to note a fundamental behavior of network control components that at

the same time modifies the behavior of the underlying shared physical network; i.e. modular network control components are in reality not isolated from or independent of one another. The decision of one component may depend on the decision of another component (e.g. best-effort routing may determine the residual bandwidth available for voice over IP traffic). Thus, components need to communicate their decisions with each other, and their execution schedule must be managed. The network behavior (e.g. network load distribution) caused by one component may inadvertently change the input conditions for another control component. Thus, *unintended feedback* and *implicit dependency* is possible and must be managed. These continuous interaction and flow of information in the network can be a major challenge to the network and concurrent actions of interdependent network control components may lead to an inconsistent network state. Thus, concurrency must be managed as well as finding a modular way to manage the components interactions. The control decision a component makes may fail to be implemented due to network hardware outages, and transient effects may be observed during a network state transition. Thus, the implementation of control decisions must ensure the correct transition of network state and component interaction. In summary, the network state dependency is critical in creating network interaction problems that must be solved to ensure SDN scalability.

There is little support for solving this network problem. The widely used protocols *Network Management Protocol* (SNMP) and *Common Management Information Protocol* (CMIP) are analogous to low level device drivers; they provide the means for network control components to interact with the network, but they are not meant to solve the higher-level problems. These tools serve to assist a human operator to monitor the network and to carry out simple network configuration changes. For example, they help a human operator recognize and analyze changes in the network load, and they enable the human operator to analyze the effects of changing the network topology based on past or

present network conditions. However, these network management tools do not manage the interactions among modular network control components at run time. The problems identified in this thesis are not caused by flaws in individual network control components but rather by their dynamic interactions. It should be quite clear that it will take a system that orchestrates the network control components to solve these problems. Such a system is analogous to a network "operating system". But unlike a traditional operating system (e.g. Linux, FreeBSD) that manages applications running on an individual device, a network "operating system" will orchestrate the network control components that govern the behavior of a network of devices. However, because of the distributed nature of these individual network control components, such a network "operating system" is much harder to design than a traditional operating system. Determined by the speed of light, there is an in-eliminable delay in the network no matter how fast the network can be built, and this fundamentally makes it a complex task to collect and synchronize the state and information distributed among individual components across the entire network. This delay could cumulate over operational time lead to increased network payload or flow data.

3.3 The Goal of Using Proactive Flow

The goal of proactively programming flow is to enable a reduction in the interaction between the network components and thereby reducing the flow request rate; this reduction of flow and resource consumption can help the network scale with increased network devises and user interaction. In this section I assume that the network is controlled by centralized control components, the controller in this is a central system that provides a layer of indirection between all the centralized control components and the underlying network of devices. For the proactive flow design, the REST API is used to push flow to the controller to register in the data plane. REST offers additional decoupling, so as to allow for extensive scalability of the network. There is no

conversational state, this allows for a wide range of scalability adding additional switch nodes behind central controller. REST API is used to program a proactive flow at runtime which enabled the network to scale with increased load or resource request, as new flows can be introduced at any point of network runtime.

The support of dynamic loading of routes and flow tables without restarting the whole system more easily make REST API very flexible to extend and it is very easy to migrate to different platforms and data path devices.

3.4 Flow

For this work, HTTP protocol is used to make HTTP requests for the various flow processing actions and calling the respective payload from the controller and how the data will be formatted as shown in Figure 0.1below.

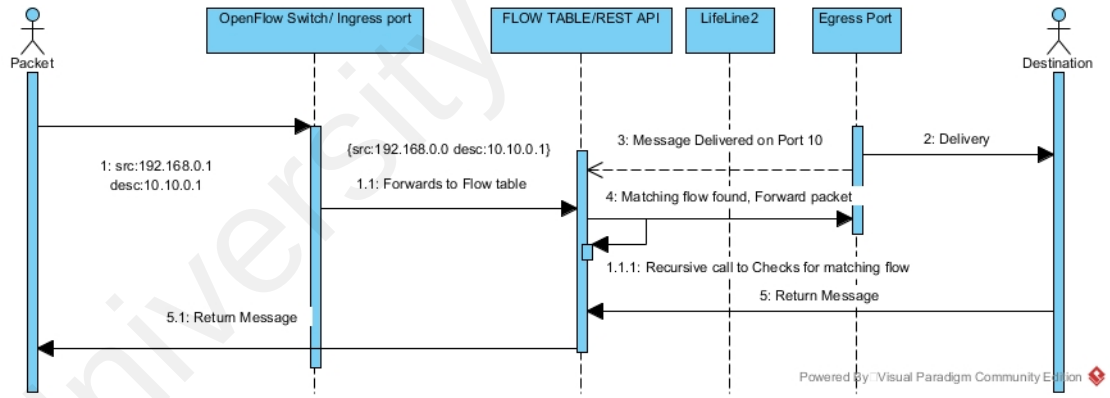

Figure 0.1: Case diagram for the Proactive Flow with preset rules

Listeners are used to interact with the controller in order to create a stream on which these messages or interaction will be sent. Apart from reading the network inventory of switches and hosts, we will use **Models** to parse the output of a request to get the topology. After our flow know how to communicate with the controller and network inventory and other information from the network topology, OpenFlow standard flow processing

standards will be used to process the flow with network requirements and also communicate certain instructions to the devices in the network.

To realize flexible and finer granularity routing in network in which users have the ability to control the routing decision for each individual or flow in the network, proactive flow can be leveraged and it comes with the advantage of increased scalability. For example, different security policies can be realized by controlling whether a flow should be allowed or not in the network; dynamic traffic engineering can be achieved because the network operators now have the ability to flexibly route the flow traffic in any arbitrary way that they consider optimal; network operators can also dynamically route flows through any arbitrary middle boxes in the network, for monitoring purposes.

This research focuses on how proactive flow can be used so that data path can handle most flow request at flow setup and from network devices thereby optimizing the performance of a controller machine from the workload characteristics of OpenFlow at flow setup and flow processing.

3.5 Summary

In this chapter, the background discussion is on the complexity of computer network and the interaction between components which can lead to overhead in the control system and loss of processing or hardware capacity. We will further discuss the need for the development of proactive flows with the use of *representational state transfer* REST API which is neither propriety nor a contribution of this thesis. We will finally conclude this chapter with the design and implementation of our proposed solution.

CHAPTER 4: IMPLEMENTATION

4.1 Introduction

A basic feature of OpenFlow is the controller is responsible for establishing every flow in the network. Whenever a switch receives the first packet of a flow, because there is no flow entry configured on the switch to match this flow, the first packet will be forwarded to the controller. The controller runs user defined applications to process a flow request. As the network grows in requests, it will become a network bottleneck. We investigate how to flexibly use threshold triggers in a proactive flow to optimize the scalability of a controller machine under the workload characteristics of OpenFlow. This interaction is reduced by proactively programming the flow in SDN to limit the controller interaction and when a packet matches a predefined match, the PacketOut message is sent to the controller, to process such packet. To realize this flexibility and more granularity in SDN control, a flow-based threshold-based trigger approach has the advantage, by giving network operators the flexibility of controlling the flow processing in the network. This can be explained in a scenario where different flow policies are realized by controlling whether a DROP or ACCEPT Acton should be applied for a particular flow in the network.

This concept of dynamicity in traffic engineering where the network operators have the ability to flexibly route flow traffic in any arbitrary way that they consider optimal and dynamically route flows through any subjective middleboxes in the network can be applied in the decision-making process of the network operator in setting up the threshold. Every network operator has a plan for the network according to specific network requirement such as expected capacity and required software and client base to use the network, this understanding will help the operator to adequately select the best practice for network planning (Di Francesco, Kibilda, Malandrino, Kaminski, & DaSilva, 2017).

4.2 Design Consideration

The objective using proactive flow at the data plane is to distribute workload among the SDN planes in is order to maximize the system's throughput. We observed that how such distribution is done will directly affect the scalability, and at the same time the fairness in allocating the capacity of the system. Optimizing the performance of a controller means more than just hitting the highest aggregate flow request handling throughput. A controller that does so but unintentionally starves some subset of requests is useless. More generally, a controller that has arbitrary performance bias against certain requests is undesirable. A controller that achieves high throughput but has uncontrollable latency is also undesirable. Optimizing performance requires a balance between fairness, latency, and throughput. So installing flow rules in the Data plane will enable the fair distribution of workload.

4.3 Fair Capacity Allocation

The capacity of the controllers must be "fairly" allocated among source switches that generate requests according to a well-defined fairness policy. Especially when the offered workload is larger than the capacity of the controller. Fair distribution of workload in the SDN network. To achieve fairness, the controller performs periodic monitoring of each ingress port from the data plane switch. Controller does this by polling each switch every t=10 seconds for port statistics using Open Flow *message* ***ofp_port_stats***. Controller sends OpenFlow port status request message **ofp_port_stats_request** (with **port_no = OFPP_ANY**, for all ports of a switch) to all the switches connected to the controller. Each switch responds by sending **ofp_port_stats_reply** message. Controller then computes port utilization for each port using transmitted bytes count **tx_bytes** using Link Load Algorithm (Ian F. Akyildiz, 2016);

Algorithm 1: Identifying Link Load

Input: Threshold T, time-interval t, port Speed, 100-Mbos Ethernet

Output: List of conjected ports

```
Begin:
  LinkLoadIdentification()
   Initialize LinkLoad  as Empty
   for <every switch> do
      for <each port> do  #Compute Port utilization
        U = (data * 8 bits * 100)* bandwidth * interval 5.
         If U ≥ T then
          Add link to LinkLoadList LL.
       end if
     end for
  end for
```

Figure 0.1 Algorithm to Identify Link Load

If a port is utilized up to a given threshold, such ports link is added to a list of links to be ignored for packet forwarding, and this links are subsequently updated in the flow table so subsequent flow will not be routed through these links. With this, only controllers that are ready to process packets will receive packets.

4.4 Interaction

To achieve the desired goal of this design, our proactive flow setup needs to be setup based on the network requirements, so this is the sole responsibility of the network operator to make the required decision.

The starting point is an analysis of the intended network traffic. The system network flow first takes into consideration the intended traffic for the network, and uses the TCP layer

SWITCH PORT	source mac address	destination mac Address	Ethernet type	VLAN ID	IP source	IP destination	IP port	TCP sport	TCP dport	action
*	*	*	*	*	10.0.0.1/24	172.0.0.1/30	*	*	*	Port 6, port 7, port 8 ...

SWITCH PORT	source mac address	destination mac Address	Ethernet type	VLAN ID	IP source	IP destination	IP port	TCP sport	TCP dport	action
*	*	*	*	*	10.0.0.1/24	173.0.0.1/30	*	*	*	push to controller

Figure 0.2 Layer 3 IP header for the flow

3 header IP address Figure 0.2 to determine the source and destination of the packet. So a fine grain flow is created to be installed in the data plane.

With this, every packet with source IP that matches specific flow table entry, is processed with the respective action.

By the RFC 791 standard, the length of an IP datagram including internet header and data allows up to 65,535 octets datagram length. For this choice of network design as show in Figure 0.2, the Maximum Transmission Unit (MTU) that can be transmitted by a protocol at an instance taking the default Ethernet interface excluding the Ethernet frame header and trailer is 1500 byte. This means that one frame contains 20 byte IP header, 20 byte TCP header, leaving a 1460 byte of the payload that can be transmitted in one frame MSS (Maximum Segment Size).

In this case, there is no encapsulation header i.e. IPsec, MPLS headers etc. The flow table proactively contains rules where by IP headers matching this MTU is forwarded based on Source and destination IP addresses.

In a likely case where additional encapsulation with MPLS label swapping, IPsec etc. an additional header will be introduced in the packet as shown in Figure 0.3.

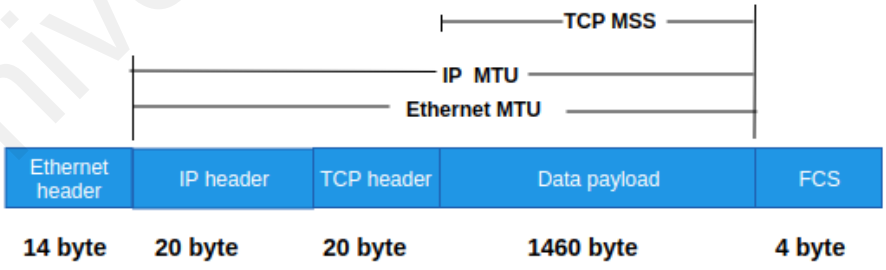

Figure 0.3 TCP Ethernet frame

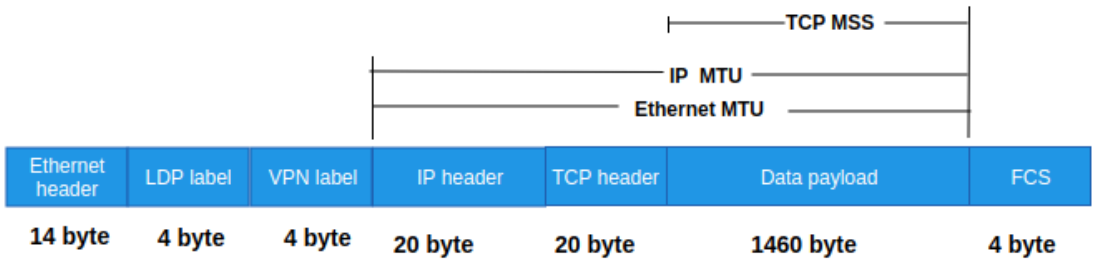

Figure 0.4 Additional TCP Header

When this traffic arrives in the data plane, it is pushed to the controller for processing and routes the packets through the network data plane.

Figure 0.5 shows the gives a pictorial insight from traffic definition to the action taken for packets based on the match in the flow table.

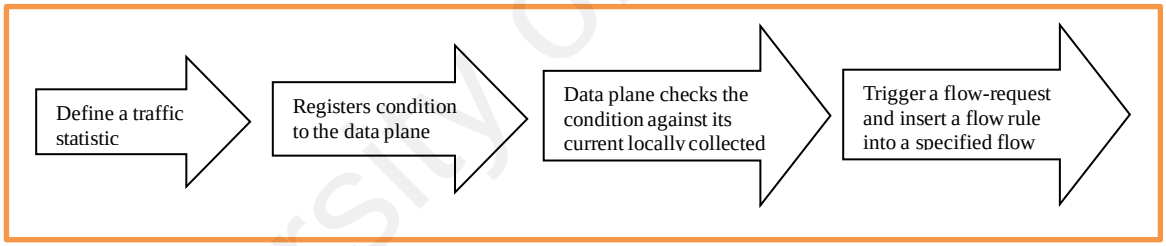

Figure 0.5 Proactive Flow Setup

We will leverage a proactive design of the flow setup based on Ethernet address. Packet delivery to the control plane is controlled by a flag bit i.e. 1. A set flag bit implies a threshold that will trigger control plane for rule configuration.

4.5 Packet Statistic

Our architecture depends on the OpenFlow packet processing rule. The OpenFlow policy on a switch consists of a set of flow control rules. Each rule has a pattern, a priority, actions, and counters. A new flow in the switch is matched to a set of rules which checks the flow priority and takes the actions on the flow table of such a flow. The OpenFlow per flow meter enables the implementation of rate limiting. Meters are used to measure

the rate of packet assigned to it and enables the flow control of such packets. To match the statistic of a packet introduced to the network against our proactive flow, we use OpenFlow **ovs-ofctl** interface which accept an argument that describes a flow or flows, ***output(port=port,max_len=xbytes) (o***utput the packet to the OpenFlow port number, with maximum packet size set to ***xbytes).*** When a packet larger than ***xbytes*** is received in the port, it is processed using the Openflow ***PacketIn*** message. The ***PacketIn*** message is a way for the switch to send a captured packet to the controller. The ***PacketIn*** is a message which consists of header, and a buffer ***id*** which is used to assign unique value for buffered packet, and the length of the captured packet ***total_len***. The port ***in_port*** is a reference to the port in which the packet is received. A field which is a representation of the reason a packet has been captured and forwarded; this could ba an action because of match, or a miss in the match flow tables, or an error. A packet with a 56 bits payload is treated and a normal packet and the specified action in the flow table is applied according to the flow table mapping configurations and when packets with additional flow instructions or otherwise arrives at the flow table mapping, the threshold condition is met and so the trigger involves the controller in such a flow. By this selective approach of involving the controller in flow processing, the flow overhead that is incurred by a reactive flow which is a primary cause of the scalability limitation is reduced.

According to the open flow specification, the flow meter is enabled and used to take statistics. For our architecture, we register the flow, using the OpenFlow interface ***ovs-ofctl*** to pass in the flow;

ovs-ofctl add-flow<bridge><match-field>actions=controller(key=value...) (4.1)

Taking this action will send packets to OpenFlow controller as "packet-in" message. The key-value pairs **max_len=xbytes**: which limit the maximum length of packets sent to the controller when a packet match proactive flow rule.

The wild card rule can route packets to all the required destination so this will require that the network administrator has a specific knowledge of the type of traffic to be handled by the network. Since we need to update the rules as the dynamics of the network changes, we use rule priority so because there might be situations where the packets might match several rules in the flow table and this can cause Rule overlaps.

4.6 Registers Condition to the Data Plane

This flow table is registered or installed in the Data plane memory at the start of the network, checks the condition against its current locally collected. To maintain consistency in the network, when a new switch is added to the network, the flow setup modules deletes all previous flows and install the table miss flows. A broadcast tree is maintained so that every packet forwarded trough the network is dropped due to unintended loops in the network

4.7 Summary

This chapter discusses the implantation of proactive flow, and the various elements of a network traffic that is vital to design a fine-grained proactive flow in SDN. First, we examined the architecture of OpenFlow and the challenge with control plane interaction in packet control which is a scalability bottleneck. We discussed that even with a fine grain proactive flow setup, a mitigating factor of scalability could still exist, which is link failure. This can occur when a controller link is overwhelmed with traffic so much that traffic to such links are dropped. To mitigate this, we extend the controller plane functionality for a fair distribution of workload. This is done by pooling port utilization, with this, traffic will only be received from underutilized ports, thereby avoiding packet drop by over flooded ports.

CHAPTER 5: EVALUATION

The hardware used for the evaluation 3 Ubuntu Server machines. The Helium OpenDayLight controller hosted on a machine with Intel Core i7-4790 CPU @ 3.60GHz, 32G of system memory, and 1GB/s Ethernet channel.

The Open vSwitch run on Docker container and hosted on a machine with Intel Core i7-4790 CPU @ 3.60GHz, 16G of system memory, and 1GB/s Ethernet channel.

The the packet generator is hosted on a machine with Intel Core i7-4790 CPU @ 3.60GHz, 16G of system memory, and 1GB/s Ethernet channel.

The traffic will flow from a physical port on the traffic generator (port A) to a physical port on the Open vSwitch and then back to the physical port (port B) on the traffic generator. This script uses the TRex Realistic Traffic generator (TRex, 2015). TRex is an open source, low cost, stateful and stateless traffic generator fueled by DPDK. It generates traffic based on pre-processing and smart replay of real traffic templates.

The performance of the system is analysed based on the average packet loss, the average throughput, and end to end delay based on work done by (Larry L. Peterson, 2007) and (James F Kurose, 2014). Packet loss is the number of packets that fails to arrive at the destination. It is represented as loss percentage

$$Packet\ Loss = \frac{Number\ of\ packets\ dropped * 100.}{Total\ number\ of\ packets\ sent} \quad (5.1)$$

The *Average Throughput* is the average amount of data delivered in unit time represented in Mbps. Average throughput is calculated using the formula:

$$Average\ Throughput\ = \sum_{i=1}^{N} \frac{\text{Data bits recieved in flow i}}{N} \quad (5.2)$$

(where N is total simulation time * to total number for flow).

5.1 CPU utilization

We implemented with a linear topology with an Open vSwitch and a Helium OpenDayLight controller. We generate traffic using with the setup for both reactive and proactive flow. The traffic generator keeps generating packet of random *bytes* and a varied flow rate of between 250 *flows per seconds* to 20000 flows per second to test the CPU utilization of the controller when implementing the proactive and reactive flow. The CPU utilization of each proactive and reactive flow is taken on separate run time to get the load average of each. To calculate Linux CPU usage time, subtract the idle CPU time from the total CPU time as follows:

Total CPU time since boot = user+nice+system+idle+iowait+irq+softirq+steal

Total CPU Idle time since boot = idle + iowait

Total CPU usage time since boot = Total CPU time since boot - Total CPU Idle time since boot

Total CPU percentage = Total CPU usage time since boot/Total CPU time since boot X 100

Each packet processing instruction involves the CPU, the more packets are pushed to the controller for processing there is an expected spike in CPU usage Figure 0.1 shows that there is a higher CPU utilization with the reactive flow compared to the proactive flow setup. The proactive setup has matching flow for the packets based on source and

destination IP address on the flow table, only flows without entry in the flow table were pushed to the controller which accounts for the increment of

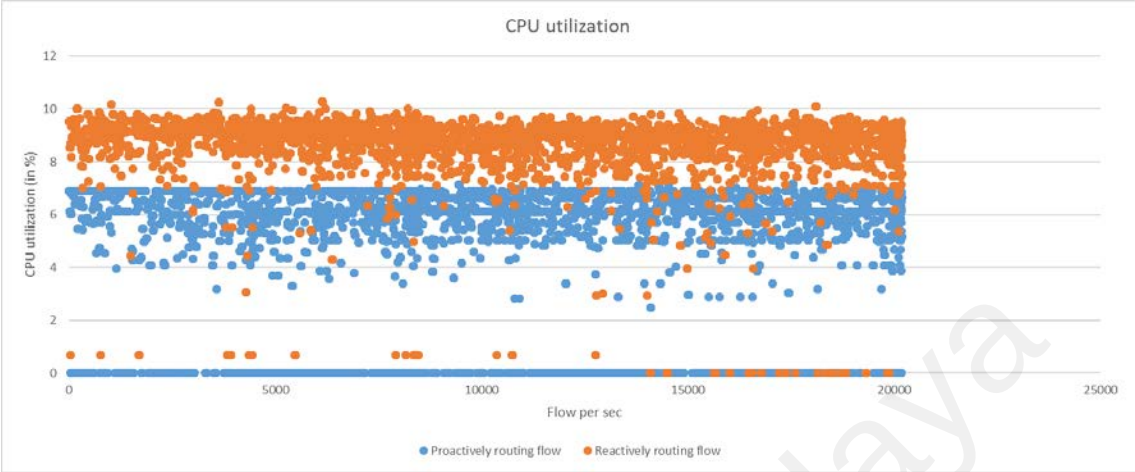

Figure 0.1 CPU Utilization of both reactive and proactive flow

5.2 Packet Loss

Packet loss for each flow shown in Table. 5-1

Table 0.1 Table showing the total number of runs, reactive and proactive flow setup

Number of packets sent (pps)	Reactive routing flow (packet loss in %)	Proactive routing flow (packet loss in %)
1000	0	0
2000	1.9	0
3000	2.3	0
4000	3.2	2.39
5000	3.91	2.63
6000	4.7	3.43
7000	9.55	4.49
8000	13.69	6.38

Number of packets sent (pps)	Reactive routing flow (packet loss in %)	Proactive routing flow (packet loss in %)
9000	14.16	6.78
10000	15.44	7.87
11000	17.26	6.63
12000	18.19	5.32
13000	23.56	6.11
14000	25.23	10.23
15000	16.34	12.22
16000	25.34	12.45
17000	29.39	18.75
18000	33.22	16.98
19000	45.56	23.93

It is observed that the setup system with proactive flow reported much lesser packet loss than the system with reactive flow. Initially, packet loss remained approximately equal for both the system as the network did not have a high volume of flow to process. When the number of flows increased, proactive flow results in much lesser packet loss than reactive flow setup. Fig. 5-2 shows the variation of packet loss with varying number of packet rate of flows.

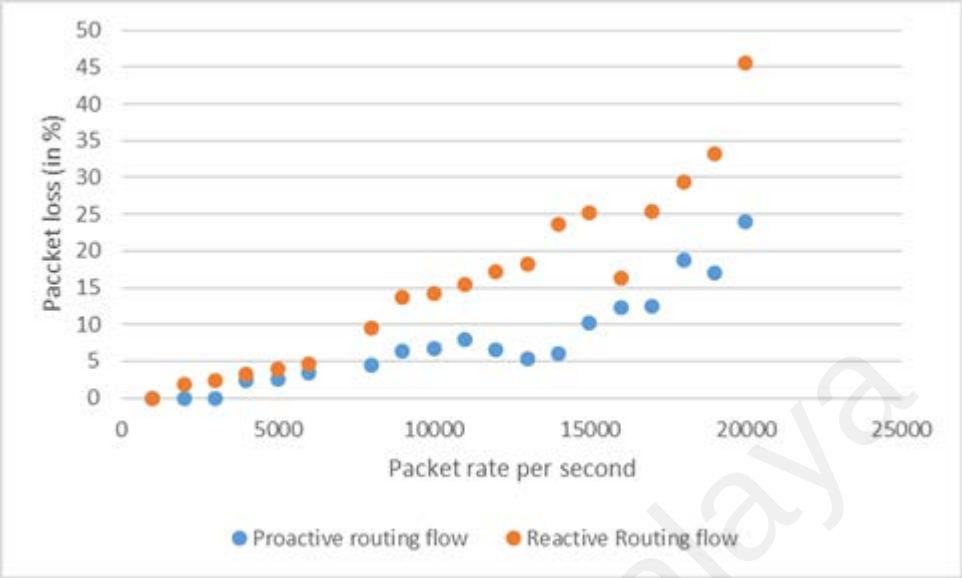

Figure 0.2 Plot showing the average packet loss

5.3 Average Throughput

We measure the average throughput for both the proactive flow SDN setup and the reactive flow setup.

Table 0.2 Average Throughput

Number of flows	Reactive routing flow (Mbps)	Proactive routing flow (Mbps)
1	6.55	6.54
2	6.58	6.58
3	6.81	6.82
4	6.83	6.83
5	6.24	6.53
6	6.36	6.56
7	6.43	6.43
8	5.64	6.46

Number of flows	Reactive routing flow (Mbps)	Proactive routing flow (Mbps)
9	5.02	5.74
10	4.51	5.18
12	4.32	5.19
13	5.21	6.76
14	5.45	6.23
15	5.45	6.29

We generate traffic for 120 seconds and measure average throughput with constant proactive flow over the experiment.

It is observed that proactive flow setup reported much better throughput than reactive flow setup. In the proactive flow, packet drop is reduced which resulted in higher throughput. Fig. 5-3 shows average throughput with varying number of flows.

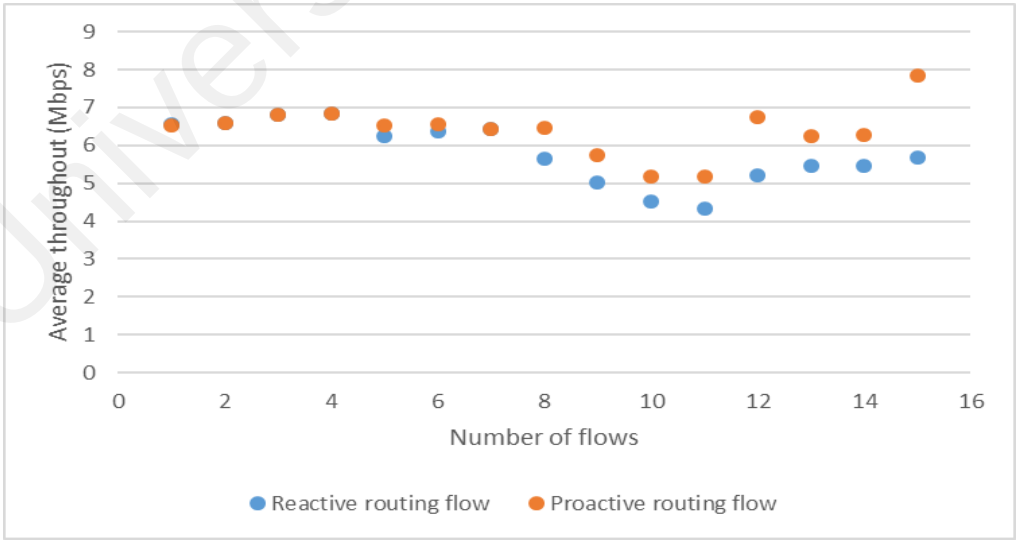

Figure 0.3 Average Throughput

5.4 Summary

From Fig.5.1, It is observed that at run time, the proactive setup CPU utilization was lower than for the case of the reactive flow. This is because the controller did not interact in most of the flow setup, so CPU resources are conserved. From the experiment, the lesser the CPU resource a controller process will consume the better the system overall performance. With proactive flow, the SDN will scale at high network packets. In the case of reactive flow, the flows do not match with any flow table entry and so according to the open flow specification; the flow is pushed to the controller for processing. We observed that as the flow increased, the CPU resource utilization increased significantly, and the packet loss slightly increased, showing that the controller was not processing the flow at scale, this is an indication that at a high flow rate, there is a possibility of a bottle neck due to the amount of CPU required to process flow in the controller.

We have systematically evaluated and compared different design choices. The results have shown that the proactive implementation design can achieve optimal scalability, while at the same time having optimum throughput. Scalability control make OpenFlow implementation of proactive flow a popular choice for different networking scenarios, but the performances of the OpenFlow controller must be optimized for high throughput.

CHAPTER 6: CONCLUSION

In this thesis, we argue that as the interaction of flow request between the control and data plane increases an unavoidable growth in the network flow increases. The fundamental complexity of reactive flow setup of the control plane lies in the fact that, different network control components are interacting with each other in and this increases with the increment of operational time. It is achievable to implement proactively programmed flows in data plane to manage the flow request rate to the controller and all together using high and affordable processing capacity for the control plane deployment, thus scalability is improved; this also gives the system the benefit of ideal and good responsiveness. Using the proactive flow, we eliminated the flow setup time and subsequent flow request from the controller. This scaled the controller significantly.

6.1 Future Work

Our work has a limitation in the case of dynamic network changes which may affect the objective of the desired goal, in order to mitigate this, further studies needs to be done.

Up to now, we have only studied how to solve the scalability problem of OpenFlow controllers by using relatively simple applications, such as “learning switch” or “routing”. We have also only considered the network in steady state where there are no changes or failures. However, in reality there could be much more complicated application scenarios, or the network state is changing dynamically. As a result, we plan to investigate more complicated scenarios in the future.

First of all, one interesting problem to address is how to design data structures for applications with scalability as the primary objective, especially under dynamic condition where the network is undergoing changes. For example, how to design better routing tables and security policy data structures, to support scalable and efficient accesses by

concurrent worker threads. If there are concurrent modifications, accesses should be efficient to minimize synchronization overhead, while at the same the correctness must be enforced. We think a systematic evaluation of the system's performance under different failure or changing conditions is necessary.

REFERENCES

Byungjoon Lee, S. H. (2014). IRIS: The Openflow-based. *ICACT*, pp. 1237-1240.

Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., & Shenker, S. (2006). SANE: A Protection Architecture for Enterprise Networks. Paper presented at the *USENIX Security Symposium*.

Byungjoon Lee, S. H. (2014). IRIS: The Openflow-based. ICACT, 1237-1240.

Chiesa, M., Kindler, G., & Schapira, M. (2017). Traffic engineering with equal-cost-multipath: An algorithmic perspective. *IEEE/ACM Transactions on Networking (TON), 25*(2), 779-792.

CISCO. (2015). United States of America Patent No. US10033622B2.

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., & Banerjee, S. (2011). DevoFlow: scaling flow management for high-performance networks. Paper presented at the *ACM SIGCOMM Computer Communication Review*.

Di Francesco, P., Kibilda, J., Malandrino, F., Kaminski, N. J., & DaSilva, L. A. (2017). Sensitivity Analysis on Service-Driven Network Planning. *IEEE/ACM Transactions on Networking (TON), 25*(3), 1417-1430.

Dixon, C., Olshefski, D., Jain, V., DeCusatis, C., Felter, W., Carter, J., . . . Recio, R. (2014). Software defined networking to support the software defined environment. *IBM Journal of Research and Development, 58*(2/3), 3: 1-3: 14.

Erickson, D. (2013). The beacon openflow controller. Paper presented at the Proceedings of the second *ACM SIGCOMM* workshop on Hot topics in software defined networking.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., . . . Zhang, H. (2005). A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review, 35*(5), 41-54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review, 38*(3), 105-110.

Ian F. Akyildiz, A. L. (2016). Research Challenges for Traffic. *IEEE Network*, 30 - 58.

James F Kurose, K. R. (2014). *Computer Networking A Top-down approach.* Fifth Edition, Pearson.

Kanagavelu Renuga, K. M. (2015). SDN Controlled Local Re-routing to Reduce Congestion in Cloud Data Center. *International Conference on Cloud Computing Research and Innovation (ICCCRI)*, 26-27.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., . . . Hama, T. (2010). Onix: A distributed control platform for large-scale production networks. Paper presented at the *OSDI*.

Lantz, B., Heller, B., & McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. Paper presented at the Proceedings of the 9th *ACM SIGCOMM* Workshop on Hot Topics in Networks.

Lara, A., Kolasani, A., & Ramamurthy, B. (2014). Network innovation using openflow: A survey. *IEEE communications surveys & tutorials, 16*(1), 493-512.

Larry L. Peterson, B. S. (2007). *Computer Networks a Systems Approach*. Elsevier, 4th Edition.

M. Smith et. al. (2014). OpFlex Control Protocol. Internet Draft, Internet Engineering Task Force.

Medved, J., Varga, R., Tkacik, A., & Gray, K. (2014, June). Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (pp. 1-6). IEEE.

Murat Karakus, A. D. (2017). A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN). *Computer Networks 112*, 279-293.

Schehlmann, L., Abt, S., & Baier, H. (2014, November). Blessing or curse? Revisiting security aspects of Software-Defined Networking. In *10th International Conference on Network and Service Management (CNSM) and Workshop* (pp. 382-387). IEEE.

Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J., . . . Rao, N. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine, 51*(7), 36-43.

Terei, D. A., & Chakravarty, M. M. (2009). *Low level virtual machine for Glasgow Haskell Compiler* (Doctoral dissertation, Bachelor's Thesis, Computer Science and Engineering Dept., The University of New South Wales, Sydney, Australia).

Tootoonchian, A., & Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. In *Proc. NSDI Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN).*

Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012). On Controller Performance in Software-Defined Networks. *Hot-ICE, 12*, 1-6.

Tourrilhes, J., Sharma, P., Banerjee, S., & Pettit, J. (2014). SDN and OpenFlow evolution: A standards perspective. *Computer, 47*(11), 22-29.

TRex. (2015). TRex Retrieved from Realistic traffic generator: https://trex-tgn.cisco.com/

van Asten, B. J., van Adrichem, N. L., & Kuipers, F. A. (2014). Scalability and resilience of software-defined networking: An overview. *arXiv preprint arXiv*:1408.6760.

Voellmy, A., & Wang, J. (2012). Scalable software defined network controllers. A*CM SIGCOMM Computer Communication Review, 42*(4), 289-290.

Yan, H., Maltz, D. A., Ng, T. E., Gogineni, H., Zhang, H., & Cai, Z. (2007, April). Tesseract: A 4D Network Control Plane. In *NSDI* (Vol. 7, pp. 27-27).

Yeganeh, S. H., Tootoonchian, A., & Ganjali, Y. (2013). On scalability of software-defined networking. *IEEE Communications Magazine, 51*(2), 136-141.

Yu, M., Rexford, J., Freedman, M. J., & Wang, J. (2010). Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Communication Review, 40*(4), 351-362.