CONTROL AND DATA PLANES IN SOFTWARE DEFINED DATA CENTER NETWORKS: A SCALABLE AND RESILIENT APPROACH

Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

SAUMYA HEGDE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,
SURATHKAL, MANGALURU - 575025
MAY, 2019

Success is not final, failure is not fatal, it is the courage to continue that counts - Winston Churchill

Dedicated to
my loving family, teachers and friends

DECLARATION

by the Ph.D. Research Scholar

I hereby declare that the Research Thesis entitled **Control and Data Planes in Software Defined Data Center Networks: A Scalable and Resilient Approach** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in **Computer Science and Engineering** is a *bonafide report of the research work carried out by me.* The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

CO09P02, Saumya Hegde,
Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: May, 2019.

CERTIFICATE

This is to *certify* that the Research Thesis entitled **Control and Data Planes in Software Defined Data Center Networks: A Scalable and Resilient Approach** submitted by **Saumya Hegde**, (Register Number: CO09P02) as the record of the research work carried out by her, is *accepted as the Research Thesis submission* in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. Shashidhar G. Koolagudi and Prof. Swapan Bhattacharya
Research Guides

Chairman - DRPC

Acknowledgments

My Ph.D. journey has been long and adventurous, filled with challenges, learning opportunities and finally triumph. I have learned the importance of patience and surrender, the happiness that diligent work brings and the success that a never give up attitude gets. This journey would not have been possible without the support and love of people, who have stood with me through all the ups and downs. I would like to very humbly acknowledge each one of them.

My profound and sincere thanks to my guides **Prof. Swapan Bhattacharya** former director of NITK, Surathkal and **Dr. Shashidhar G. Koolagudi** Associate Professor, Department of CSE, NITK. I cannot thank Prof. Swapan Bhattacharya enough, for without him I could not have dreamed of continuing my research work. His kind words have helped me sail through this journey, especially when the going was tough. His incredibly vast knowledge and meticulous research guidance accelerated my work. I will forever cherish the moments spent with him and always remember the wisdom he shared with me, especially 'To dream Big' and push my boundaries. Thank you sir, for believing in me and for all the support you have given. I would like to thank Dr. Shashidhar G. Koolagudi for being extremely patient, especially during the thesis writing period. I will always remember his kindness and down to earth attitude. His meticulous review of the thesis has helped me immensely. His comments and observations have helped me refine my work. Thank you sir, for all the support and encouragement you have given me.

My sincere thanks to two of my former guides **Dr. T. G. Basavaraju** and **Dr. Prakash S. R.**, who resigned from their post of Assistant Professor at NITK, Surathkal. The discussions with Dr. T. G. Basavaraju have given me a clear understanding of how research work is carried out. His enthusiastic nature has made me look forward to working on my research. I thank Dr. Prakash S. R., with whom I have had many fruitful research discussions. The insights he provided help me even today.

I want to thank **Prof. K. Chandrasekaran** Prof., Dept. of CSE and **Dr. Kumar G. N.**, Associate Prof., Dept. of Mechanical Engg. for being on my Research Progress Assessment Committee. I am grateful to them for having taken the time to attend my seminars and share their knowledge with me. Their valuable feedback has helped me shape and accelerate my work.

My thanks to **Dr. Pushparaj Shetty** Asst. Prof., Dept. of MACS and **Dr. Mohit P Tahiliani** Asst. Prof., Dept. of CSE, for supporting and championing me through this journey. I would like to thank **Prof. Annappa**, Prof. Dept. of CSE, for his kind and motivating words. I also extend my sincere thanks to all the faculty of NITK, for their support.

The time spent in my research lab was exciting and fun because of the many friends around me. I thank **Sumith N**, for being my cheerful friend and companion along this journey. Heartfelt thanks to **Vishnu Yarlagadda** who has helped me keep my research on track and I cannot forget the many times he has readily helped me. I would like to thank the programmer of the research lab, **Mr. Dinesh Kamath** for his support with logistics. I would also like to thank our staff **Ms. Vanitha** for her help with tracking formal letters and communications.

My incredible family has always been my strength and the reason for my smile. My mom **Kausalya Hegde** and my dad **Ananda Hegde** have always been by my side, no matter what and have loved me unconditionally. They have gladly taken on a lot of additional responsibilities at home on account of me pursuing research. Thanks amma and dad for your eternal love. My heartfelt thanks to my late grandmother **Rajeevi Hegde**, who has been an epitome of strength and dignity. She always inspired me to be independent and stand tall. My sincere thanks to my mom-in-law **Lakshmi Hegde** and dad-in-law **Sundara Hegde** for their love and blessings. I would like to thank my maternal uncle **Prof. Mohan Hegde** for his love and blessing throughout my life. Thanks to my awesome girl gang **Dr. Saguna**, **Samanvitha**, **Cdr. Prathibha** and **Puppa** for their love.

Utmost thanks to the love of my life, my husband **Dr. Prakash N. S.** His keen observations, clear thinking and deep rooted values in life, have helped me overcome many a hurdles with ease. He is my strength, my mentor, my confidante and wind under my wings. Thanks honey for your love and the belly laugh inducing wacky jokes when the going was tough.

I am grateful to god for the greatest blessing of my life, my darling children **Ojaskrishna** and **Sujyotsna**. They are my life, my joy and the reason for my everything. They have been incredibly patient with me and tolerated me during my low times. Thank you darlings, Jess and Jo for coming into our life and thanks for your wacky jokes too.

Finally, thank you god, for this blessed life.

Place: Surathkal **Saumya Hegde**

Date: May, 2019

Abstract

The single central controller of Software Defined Network (SDN) eases network management, but leads to scalability problems. It is therefore ideal to have a logically centralized but physically distributed set of controllers. As part of this work we developed a novel placement metric called subgraph-survivability and designed an algorithm for controller placement using this metric, such that the control plane is not only scalable but also resilient to failure of the controller itself. The controller collects the network statistics information and also communicates the forwarding rules to the switches. This lead to the Edge-Core SDN architecture, where the edge and core network have their own edge and core controller. For such networks, we have developed a separate edge and core controller placement algorithms using suitable metrics for each. The scalability problem of the data plane is due to the limited switch memory and increased size of SDN forwarding rule. Using source routing to forward packets, not only alleviates this problem but also complements the Edge-Core SDN model. Here, we have proposed a source routing mechanism that is scalable, is fair to both elephant and mice traffic, and is resilient to link failures, thus making the data plane scalable and resilient.

The algorithm and routing mechanism are validated, through both analytical and empirical methods. The performance metrics of Average Inverse Shortest Path Length (AISPL) and Network Disconnectedness (ND) are used to evaluate our placement algorithms. An improvement of 55.88% for the AISPL metric and 49.22% for ND metric, was observed with our proposed algorithm as compared to the random controller placement. With our source routing mechanism we observe a reduction, in the number of flow table entries and the flow set up time, that is proportional to the number of hops along the path of the packet.

Keywords: SDN, Edge-Core SDN, Controller Placement, Source Routing, Scalability, Reliability, Fairness.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

You can't cross the sea merely by standing and staring at the water.

-Rabindranath Tagore

This chapter introduces the need for a change, in the present networking architecture, that led to the development of Software Defined Network (SDN). This is followed by a description of the SDN architecture and how packet forwarding is carried out therein. This chapter also covers the advantages, challenges and applications of SDN.

1.1 Traditional Networks

In the late 1960s Advanced Projects Research Agency (ARPA) stated a research project to enable communication between its military installations and scientific research centres. Later in mid 1970s they began designing the Internet architecture and communication protocols. Internet refers to the interconnection of thousands of networks around the world, using different technologies. Such an architecture is possible since the Internet is based on an abstraction that is independent of the physical hardware. The Internet, as we know it today, began when Defence Advanced Research Projects Agency (DARPA), originally known as ARPA, started using the Transmission Control Protocol/Internet Protocol (TCP/IP) stack.

Networking technologies underlie all IT activities. The last few years have seen an increase in the number of applications that depend on communication networks, which are fast, scalable, reliable and easy to manage. Consequently, the size of networks has also increased. Current networking architectures only partially meet the requirements of these applications. This is largely because, current networks consist of a set of ad hoc protocols which are developed in isolation; to address one problem at a time. Since, this approach is not built on any fundamental abstraction, the complexity has increased. For example, if an IT administrator wants to add or remove a device from a network, then multiple software and hardware entities must be reconfigured using network level management tools. Also a knowledge of the topology, vendor switch model, switch version etc. are required. New applications have also caused the present day network characteristics, in terms of networking devices used and network traffic generated, to be different from that of the Internet. These developments highlighted the shortcomings of traditional networks and led researchers to redefine at the networking architecture.

Although the shortcomings of traditional networks were known since long, it was difficult to fix them because of the tight integration of the networking devices. Generally, the networking devices operate on two planes. The control decisions that a device has to take, regarding how packets are to be forwarded and managed, are carried out in the control plane. The actual forwarding of packets, based on the forwarding decisions of the control plane, are carried out in the data plane. Traditional networking devices, integrate these two planes in a single networking device. This vertical integration hinders quick development and deployment of new protocols and services, as it would require rebuilding of the entire device. Such an architecture is therefore not suited for the present applications which try to address fast changing user demands related to the network services.

The advent of virtualization, cloud computing and software defined networking, though different technological entities, led to a paradigm shift in the way infrastructure is managed and services are orchestrated. Present day data centers are among the first where this change is happening.

1.2 Data Center Networks

A traditional data center consolidates the storage equipment, networking hardware, and physical servers in one location, in order to provide security, energy efficiency and ease of management. Development of virtualization technologies enabled data centers to host servers and networks that have been virtualized and optimized for cloud applications. Compute virtualization has seen a lot of advancement in the recent years and has proven to be an efficient technology, that the data centers (DC) have benefited from.

Cloud computing allows tenants to use the resources they need, pay only for what they use and scale out on demand i.e. expand with a 'pay-as-you-go' model. Cloud providers have already hugely benefited by leveraging the virtualization technology. Virtualization is the key enabler of cloud computing, since it allows the creation of an abstraction layer which hides the complexity of hardware and software, for example it allows different operating systems to run on the same hardware.

However, the modern day networking has not taken into account the challenges posed by the compute virtualization and cloud based applications (Hamilton, 2009). This is because the networking concepts deployed here were meant for the Internet. Some of these challenges are thus presented.

1.2.1 Challenges and Opportunities

The unique and complex nature of virtualized cloud data center poses certain challenges to data center networking that has not been an issue in traditional networking. Some of these challenges are as follows:

- **Multiple virtual machines on a single host:** The old assumption that each end node is connected to a port and is running a single instance of an OS is no longer a reality. Virtualization has created a new access layer that resides on the physical host machine (Benson et al., 2010). Virtual software switches are developed and are also present inside this physical host machine. These switches connect the Virtual Machines (VM) that are co-located. This ensures that the network traffic between these VMs can pass through the software switch and need not leave the physical host machine. The virtual switch also connects VMs to internet at large.

- **VM migration:** VMs may be moved from one host system to another. This is called VM migration (Zhang et al., 2018). VM migration may be required so as to pack VMs into as few physical machines as possible, to accommodate a different bandwidth requirement of an application running on the VM or to keep the VM running in the event of problems with the host systems. It is necessary to complete this VM migration without disrupting the ongoing communications. This is called live migration. Today, live migration is one of the essential services to be ensured in order to make networks more practical and efficient.

- **Multi-tenancy:** Cloud computing has enabled a single data center to host many tenants. These tenants share the data center network. Each of these tenants may have their own networking requirements and hence each must have its own network management system. These multiple networks must be isolated from each other, such that, one flow does not take up all the resources (bandwidth or forward table space) and the failure of one network should not affect the other (Heng et al., 2012).

- **East West movement of data:** Unlike the traditional client-server data traffic, data centers have movement of data between servers. This is because most applications in data centers are multi-tier in nature i.e. many components of an application may run on back-end servers. For example, a web search application which may access an inverted index spread across 1000 back-end servers.

- **Scale of data center:** A data center is estimated to roughly have up to

1,20,000 VMs with over 20 VMs on each server (Benson et al., 2010). The network must support VMs at such scales in terms of the forwarding table space on the switches, bandwidth management etc.

- **Visibility:** Since, VM to VM communication within a host, does not leave the host, it is not visible to the physical switch which is outside the host, where most of the traffic monitoring applications reside. However, on host network traffic, visibility is essential in order to adhere to the networking policies and the networking technologies must allow this.

- **Dynamic nature of cloud computing:** Tenants in a cloud environment are constantly requesting for new resources or configurations, bringing down existing ones on demand (Wuhib et al., 2012). The network must support the dynamic nature of cloud computing.

- **Dynamic nature of cloud network:** The cloud data traffic is unpredictable and volatile because of its dynamic nature. This may lead to break in service due to overload and the network must be capable of managing it.

Some of the implications of these challenges on the network design are: VM migration is efficient only when location independent MAC address is used for forwarding; multidomain network management is essential for multi-tenant environment; east west movement of data and scale of data center implies that scalability is a serious concern; end point policy enforcement within the host is essential to make on host network visible; dynamic nature of cloud makes traffic predictions difficult in data centers.

On the other hand, Networking in virtual cloud data centers can benefit because of some of its characteristics, such as

- **Single owner** The data center is owned and managed by a single organization unlike the Internet.

- **Central control** There is central control over the host system and the network, again unlike the Internet.

- **Well defined events** Since most events in the data centers are catering to cloud operations they are well defined. Ex. when a VM joins and leaves the network, the network topology and configurations are well defined.

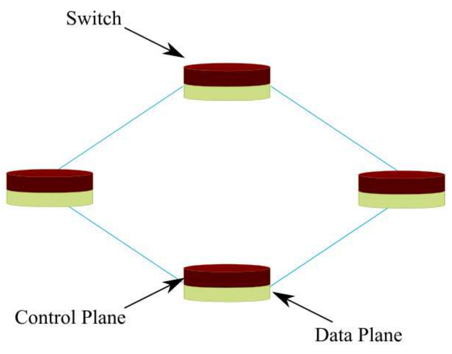
The implications of these opportunities on the network design are: Single ownership and central control creates a possibility of a centralized network management. Well defined events imply optimization of operations like path computation, bandwidth utilization etc. which can now be implemented with greater accuracy.

Widespread use of data centers and the inherent challenges therein, forced researchers to re-look at how networks have been built, leading to some radical solutions.

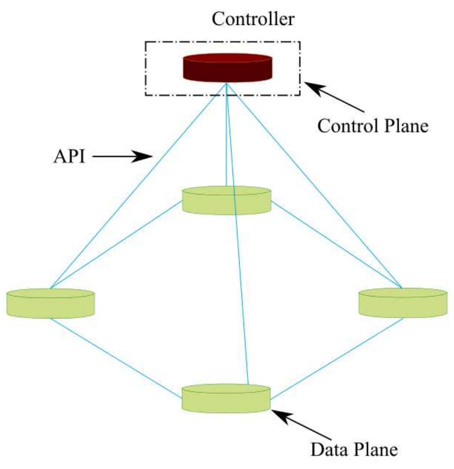
1.3 Software Defined Networking

The networking challenges presented in the previous section show, the need for present day networks to evolve, in order to adapt to the changes brought about by virtualization and cloud computing. However, current networking switches are vertically integrated where the application specific hardware chips, the hardware and the entire software stack are singly sourced. This tight coupling makes it hard to program the switches. It causes vendor lock-in, leading to the monopoly of a switch manufacturer, making it difficult for the switches to evolve fast enough to keep up with the changing demands of the applications. Further, any change to the network switch configurations has to be done manually on a switch by switch basis. This normally leads to an increase in the OPerational EXpenditure (OPEX)

The above reasons made the network inordinately complex with a lot of manual configurations. Due to this, there was a need to rethink the way in which the network has been architected. Traditional networking devices consist of two planes, namely the control plane and the data plane. The control plane is responsible for taking decisions about how to forward packets that the device receives. The data plane is responsible for forwarding these packets according to the decisions that the control plane takes. The control plane requires better programmability and the data plane requires more speed. This difference in requirements has led to the revolutionary idea of removing the control plane from all the switches and placing them on a central server; while the data plane will continue to reside on the networking device. In other words, the network control (software, protocol and state) is decoupled from the network hardware. This is similar to the separation between the applications, operating systems and hardware on a x86 machines. This separation of the control and data planes is the central idea of Software Defined Networking (SDN) (Kim and Feamster, 2013) and is depicted in Figure 1.1



(a) Traditional network architecture



(b) SDN network architecture

Figure 1.1: Traditional vs SDN network architecture

This implies, that the forwarding of packets is still carried out by the switch. However, the routing decisions take place in the centralized controller and these decisions are then communicated to the switches. A protocol is necessary to facilitate the communication between the controller and the switch. The most common protocol used currently is OpenFlow (McKeown et al., 2008). In this regard a SDN switch equipped with OpenFlow or a similar protocol is the need of the hour. The functionality of the controller is thus synonymous to that of an operating system, since it controls and manages the network. Due to the complete access to global network elements and resources, the network behaviour is flexible and can be changed in real time. Using automated SDN programs, it is possible to flexibly optimize / manage / configure and secure the network.

SDN architecture is characterized by:

- Separation of control plane from the data plane; the control plane residing on a central controller.
- The SDN controller can collect the entire networks traffic statistics and topology information by communicating with the switches. This, unlike traditional networks, enables centralized configurations and management.
- The network can be programmed by external applications via the controller eliminating the time consuming and human error prone configuration of the network.

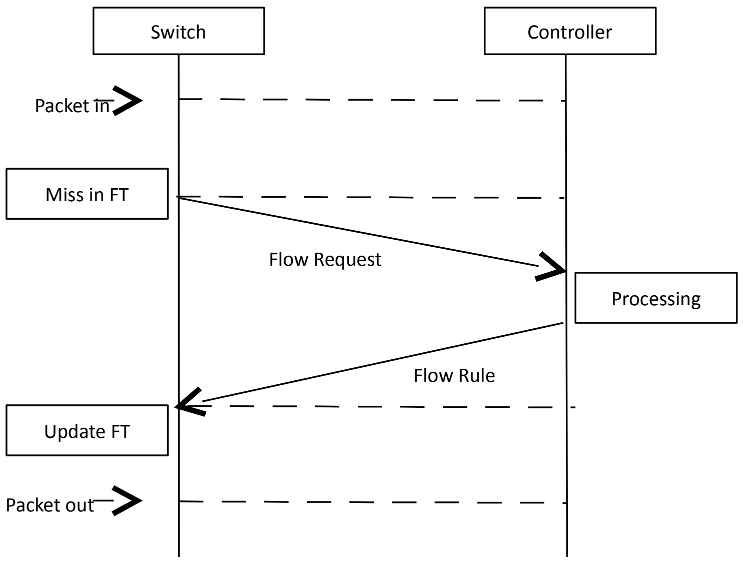


Figure 1.2: Communication between switch and controller to route packets in SDN

In SDN the brain of the switch i.e. the control plane, is moved out and put into the central controller. This makes the switches mere forwarding devices. Hence, they need to communicate with the controller in order to know how and where to forward the packets. The controller with its central view of the networks topology and traffic statistics, can decide the path that the packet has to take in order to reach the destination without fail. When an incoming packet does not have a matching flow entry in the Flow Table (FT), the packet header is encapsulated and sent to the central controller; the controller runs the required routing module in order to identify the path for the flow. It then installs this forwarding information on all the switches along the flow path as shown in Figure 1.2

The forwarding decisions may be done on the basis of all fields like source and destination MAC address, destination protocol etc., and not just based on the source and destination IP address. These additional header information however, increase the size of flow entries to about 356 bits (Kannan and Banerjee, 2013)

If an incoming packet has a matching flow entry, then it is forwarded accordingly, without contacting the controller. This style of packet forwarding must ensure that the first packet of every new flow has to contact the controller. This causes a slight delay in packet forwarding as compared to traditional networks. This shortcoming is negligible when compared to the benefits that SDNs provide. Similar delay is not experienced by subsequent packets of a flow, since, the flow rule is already installed. The delays are mainly introduced due to the communication between the different layers, hence, a standardization of these communication API is essential.

North bound API vs South bound API: The SDN architecture may be considered to have three distinct layers namely, the application plane, the control plane and the data plane. The application layer runs network applications like load balancers, firewalls etc. and use the services of the control plane. The control plane manages the entire network from a central location. The data plane is responsible for actually forwarding the packets. The API used for communication between the application and control plane is referred to as the north bound API. The south bound API is used to communicate between the control plane and the data plane (Klaedtke et al., 2014). Presently not much work has been carried out to standardise the abilities of north bound API. However south bound APIs have seen significant research and development and OpenFlow is an example of a standard south bound API.

In-band vs Out-of-band communication: The control communication between a switch and a controller (i.e. south bound API) may be carried out over the links of the same network, used for data communication between the switches. Such a network is referred to as an in-band communication network.

On the other hand, if the control communication is carried out over a separate network links between the two, then it is referred to as an out-of-band communication network. Currently most deployments of SDN use out-of-band communications.

Ternary Content Addressable Memories: The packets arrive at the data plane of a switch at a very high speed. In order to forward these packets effectively without any packet drops, the flow table look ups must be quick. This requires high speed memories such as Ternary Content Addressable Memories (TCAM). A TCAM is a memory chip where each entry can store a packet classification rule that is encoded in ternary format. Given a packet, the TCAM hardware can compare the packet with all the stored rules in parallel and return the decision of the first rule that the packet matches. TCAM allows matching on masked bit value and not just strict binary matching and this special feature facilitates rich policy based forwarding with wild card matches, paving the way for custom forwarding models.

TCAM has become the de facto standard for high-speed routers on the Internet (Lakshminarayanan et al., 2005) and SDN by design uses TCAM. However, TCAMs are expensive and power hungry memory devices. This severely limits the amount of TCAM memory available on the switches. Hence, limited TCAM memory especially in large networks is a serious concern for SDN implementations.

1.3.1 Advantages of SDN

Distributed architecture of the legacy networks is scalable, autonomous, robust and proven. But it has certain disadvantages like difficulty to add new features, forwarding by destination address only, limited external configuration, closed and eventual consistency. SDN on the other hand has the following advantages

- Centralized decisions can be made, in response to the changing state of the network, allowing flows to be managed in real time.
- It is possible to centrally program and manage the network rather than configure it on a device by device basis(Casado et al., 2007).
- The decoupling of the software dependent control plane, from the hardware dependent data plane, facilitates quick innovation at both these layers.

Although the SDN concept may be used in the Internet, it is most appropriate for large data centers like Google (Jain et al., 2013). Large data centers are single owned and it requires direct control and programmability of the network in order to meet the specific requirements fo the application it runs.

1.3.2 Issues in SDN

SDN is an emerging technology and it promises to address many problems that are hard or time consuming to solve with traditional networking tools. But, as with any emerging idea, for SDN to be widely accepted, its basic architecture needs to be reexamined and tweaked.

Centralized controller A central controller has many advantages, however, it does not scale well and may cause single point failure. A controller communicates with its switches to;

- obtaining the network updates regarding the traffic status and topology information
- installing the forwarding rules on the switches to enable forwarding of packets.

As the number of switches in a controller domain increases, the controller becomes less responsive and does not scale well. Additionally, if the central controller fails, its network will be rendered non functional, even though the switches themselves are working fine. Hence, there is a need to tweak the SDN architecture so that it retains the benefits of the centralized controller approach while also making it scalable.

Packet forwarding The forwarding information in a SDN flow table includes header from all layers, unlike the traditional forwarding entries which are restricted to only layer three headers of TCP/IP. Therefore each forwarding rule takes up more space in the flow table TCAM memory. The size of TCAM memory on the switches is small because they are power hungry and expensive. The combined effect of an increased flow entry size and a small flow table size (Kannan and Banerjee, 2013) in SDN enabled switches, severely limits the number of forwarding rules that can be stored on them. If a switch cannot store adequate number of forwarding rules, then, it will have to frequently contact the controller in order to obtain the forwarding rules. Frequent communications impose a burden on both, the switch and the controller, and thus poses a serious scalability concern to packet forwarding as the size of the network grows. Therefore, there is a need for a packet forwarding mechanism on the data plane that is scalable and at the same time, benefits from the centralized controller approach.

1.3.3 Applications of SDN

Programmable networks which are implemented using SDN find several applications because of its central control over all the switches and dynamic control of the traffic. Some of the applications are: dynamic access control as a single point operation; load balancing that is dynamic and includes path balancing; real time energy efficient networking is achievable as centralized view is available; network management becomes easier because of network programmability; rich, dynamic policy based routing is possible as routing is not distributed. SDN is fast becoming the chosen networking solution in more recent domains too. The flexible nature of SDN can handle the heterogeneous and dynamic nature of IoT devices and traffic.

1.3.4 Why SDN?

Generally computer networks consist of a bunch of protocols developed to solve problems as and when they arise. SDN on the other hand moves away from this approach and aims to give a clean slate approach to networking, by rethinking the way in which the network is architected. With the fast growth in compute and storage domains, the networks needs to adapt quickly. This can be achieved only by SDN. Traditional networks obviously find it hard to cater to the dynamic and complex needs of present day networks. All of these reasons imply that SDN will play an important role in the next generation networks.

1.3.5 Outcome of the research

Presented below are the outcomes of this research work. It also details the chapters where the work leading to these outcomes are discussed and the challenges of section 1.2.1 that are addressed.

1. Proposed a new metric called subgraph-survivability for controller placement in Chapter 3, which ensures connectedness of the network even after the failure of one of the controllers. This is especially applicable in *multi-tenancy environments* where failure of one domain should not affect the others. Further, this addressed the challenge of uninterrupted *VM migration.*

2. Developed an edge-core SDN model as presented in Chapter 4, with a separate controller for the edge and core of the network. Here, the host and the first switch it is connected to, form the edge of the network. The remaining switches form the core of the network. The core controller is responsible for continuously collecting information regarding, the network topology and the shortest paths between any two hosts. The edge controller is responsible for embedding this path information in the packets

so that packet forwarding can be carried out without intervention of the core controller. This research outcome increases the *visibility of the edge traffic* by incorporating a separate controller.

3. Proposed a dynamic core controller placement algorithm based on controller load and latency in Chapter 4. Such a placement algorithm ensures scalability as the *size of the data center increases.*

4. Proposed a scalable and fair forwarding mechanism for both elephant and mice traffic by combining source routing with flow splitting in Chapter 5. This will make the data plane scalable, fair and capable of handling a variety of flows, owing to the *dynamic nature of cloud computing.*

5. Developed a path restoration mechanism which takes local corrective actions in the event of link failure without contacting the controller and presented it in Chapter 6. This effectively handles the changes to the topology that arise due to the *dynamic nature of the cloud network.*

1.4 Outline of the Thesis

SDN, due to its clean slate approach, is fast becoming the de facto network architecture in many domains like data centers, IoT, smart cities etc. The work presented here, aims to find solutions to some of the problems that are inherent in SDN, thereby making it an ideal networking solution in the years to come. The issues chosen and addressed as part of this work are, scalability and resiliency in both the control plane and the data plane. The thesis is organized as follows.

Chapter 1 discusses the drawback of traditional networks that has led to the development of SDN. This is followed by a look at SDN architecture and it is compared with the traditional networks. Some of the issues, challenges and applications of SDN are also presented. It covers the motivation and essence of the research work undertaken.

Chapter 2 reviews related literature followed by the identification of research gaps, that motivated this work. Next, the problem statement and research objectives are presented.

Chapter 3 presents a scalable and resilient control plane. This is accomplished using a new metric we propose called sugraph-survivability. The controller placement algorithm is designed using this metric.

Chapter 4 presents a controller placement algorithm for the edge-core SDN model. Since the nature of packet forwarding in the core network differs significantly from that in the edge network, the metrics used here are also different.

Chapter 5 contains a scalable and fair forwarding mechanism, used in the data plane, that combines source routing with flow splitting.

Chapter 6 presents a resilient data plane solution based on graph theoretic approach.

We conclude and present some future work in **Chapter 7**

1.5 Summary

This chapter began with a discussion of some of the problems that traditional networks face, in view of the fast changing demands of applications. This was followed by an introduction to the important concepts of SDN, its architectural features and, its advantages and disadvantages. Some of the probable applications of SDN were presented, followed by an analysis of why SDN is best suited networking solution for the demanding modern day applications. A justification of why this research work on SDN is the need of the hour was presented next. Chapter 2 reviews the literature related to SDN, scalability and resiliency issues with regard to the control and data plane, followed by the outcome of literature review.

Chapter 2

Literature Review

If we can really understand the problem, the answer will come out of it, because the answer is not separate from the problem

-Jiddu Krishnamurti

The last few years have seen considerable work being carried out in the area of Software Defined Networks (SDN). Since it is a relatively new area of research, a major portion of the previous work has focused on formalizing the idea, on developing the controllers and protocols, necessary for the functioning etc. Simultaneously, serious effort is also put to overcome some of the problems inherent in SDN architecture. Important technical hiccups in SDN are mainly because of a single centralized controller and limited memory in the switches on the data plane as is discussed in Chapter 1. The following sections summarize the literature review, pertaining to the scalability and reliability of both the control and the data plane.

2.1 Evolution of SDN

Ethane project (Casado et al., 2007), carried out at Stanford University, defined a new architecture, specifically for enterprise networks. It comprises of two abstraction planes: the control plane, comprising of the the centralized controller, that manages the policy and security issues of the network; and the data plane, consisting of the ethane switches with a secure communication channel to the controller, that is used to enforce the controllers' decisions. This work demonstrates that, such a split architecture eases network management. The promise that the Ethane project has shown, led to the development of OpenFlow (McKeown et al., 2008); a generic protocol that has allowed a controller to communicate with OpenFlow enabled switches in the network. Specifically, OpenFlow allows the controller to insert, modify or delete forwarding entries in the switch flow table, thereby controlling the network behavior. This protocol is currently standardized by Open Networking Foundation (ONF) and is one of the most widely used protocols. Ethane is built for a specific purpose and it therefore lacks general programmatic control of the network. To address this, the Ethane team has developed Nox (Gude et al., 2008), a general purpose network controller, which made use of the OpenFlow protocol. Nox has a user interface that allows the network administrator to program the Nox controller. It is the first of the many general purpose network controllers, such as Ryu, Difane (Yu et al., 2010) etc, paving the way for software defined networks.

2.2 The Control Plane

A centralized controller is a single entity that manages all the switches in the network. It poses the problem of single point of failure and scalability limitations. Addressing these problems has become imperative, if SDN is to be widely deployed.

The emergence of new applications shows that there is a need for separating the control and data planes of the network and its success is demonstrated with the Ethane project. However, as the size of the network grows, it has become evident that centralized controllers are not scalable. Onix (Koponen et al., 2010), is the first controller to propose a distributed hierarchical control plane wherein the lower tier onix controllers take local decisions with regard to the network that they have managed. The localized data and control decisions are aggregated, and this information is made available to the root controller, enabling it to take global decisions. Another approach to solve the scalability problem is to offload some of the work of the controller to the switches, as in (Yu et al., 2010). However, this requires modifications of the switch hardware. This approach is not widely adopted, since changes in the switch hardware take at least a couple of years to design and build. At the same time that Onix was developed, another logically centralized, physically distributed controller called Hyperflow was introduced (Tootoonchian and Ganjali, 2010). Unlike Onix, this has a flat architecture for the controller distribution, with each controller having a global view of the network, but taking local decisions. The control events are communicated to each other using the subscribe publish paradigm. It is built using the Nox controller, developed earlier. They further show that such architectures utilize small control channel bandwidth, keeping a clear room for expansion, making it scalable. Although, this has addressed the single point of failure problem, the latency introduced by the inter controller communication, between the two tiers, need to be addressed. The solutions, that have been proposed until this point, have made the control plane scalable, by distributing the controller functionality across multiple controllers or to the switches. These works have not specifically taken a call on; which switch is to be assigned to which controller. Such questioning is now known as the *controller placement problem.* It is evident from the above literature that multiple controllers are necessary to make the control plane scalable. Heller et al. (2012) wrote the first paper that puts forth the problem of controller placement i.e. how many controllers are required in a network and where these controllers should be placed. They analyse the controller placement problem especially for the WAN topologies. Latency was chosen as the placement metric. After the researchers run their algorithms on various topologies they conclude that a single controller is enough for moderate service level agreements, but do not adequately handle failure in the network. Further, they also conclude that randomly placing controllers in the network is not ideal for large WAN like networks and hence the location of the controllers must be carefully chosen, such that it is optimized for the given metric, such as latency, reliability etc. Zhang et al. (2011) has not only distributed the controllers but has also considered reliability as the criteria for controller placement, using metric such as valid control paths and resiliency protection. These metrics ensure reliable connectivity between the switch and the controller. Also, the network is divided into multiple domains, such that, there are many edges between the switches within a domain and few edges connecting the switches between domains. These controllers reside on a switch belonging to a domain, such that reliability is increased. Another contemporary paper (Levin et al., 2012) has reported the work on eventual and strong consistency of the network views or network state information that is maintained by the controllers. They also discuss how the application level programs need to be tweaked based on the type of consistency. They show that, inconsistent global view can lead to routing loops and other suboptimal performance of the application program. Ensuring consistent global view, entails higher communication costs. The trade off here is between two types of application programs. The first is a complicated but robust application program that knows that the controller data could be stale and takes corrective action accordingly. The second is a simple but less robust application program that does not take into account the fact that the controller data could be stale.

Several placement metrics have since been formally defined, such as, the expected percent of valid paths to the controller, for the WAN environment (HU et al., 2012). The expected percent of valid paths is the percentage of operational paths between the switches and the controller after a network component fails, such as a node or link failure. The authors propose a greedy approach for placing controllers in the network. In order to do this they calculate the probability of failure of a networking component. Hierarchical multi controller architecture has been used to make SDN network scalable (Hassas Yeganeh and Ganjali, 2012). The top tier controllers maintain global network level information and lower tier controllers maintain local domain level information only. Lower tier controllers are enabled to take local decisions (eg. identify elephant or mice flows), without requiring contact to top tier controller. Here, the top tier controllers subscribe to the lower tier controllers, in order to obtain a global network view.

The resilient network problem (Beheshti and Zhang, 2012) is formulated and a metric for protecting resiliency is developed, for the links between the switches and the controller. A switch is said to be protected if it has path to the controller, other than the controllers routing tree. Two algorithms are proposed for controller placement. The first algorithm does a brute force search of all possible controller placements, and then selects a placement, which creates least number of unprotected switches. The second one is heuristic based algorithm, that identifies a densely connected switch as its controller location, so as to increase the switches protection metric.

Work till now focused on static controller placement, but Bari et al. (2013) went a step beyond the controller placement problem by dynamically allocating controllers as the state of the underlying network changes. The monitoring module monitors the controllers and allocates / deallocates switches to the controllers while trying to minimize the flow set up time and communication overhead. Similarly, (Hock et al., 2014) analyse the effect of controller failure on the switches in terms of, how they have to be reassigned to a new controller. They employ a set of secondary controllers that are used when the primary controller fails. Here, the controller placement makes use of a metric which is based on the communication delays between a switch and its primary as well as secondary controller. They prove their placement is pareto-optimal.

The latency and load are both considered as a metric for controller placement by UlHuque et al. (2015). As the load of switches vary, they have dynamically reassigned the switches to controllers so as to equally distribute the load. Kreutz et al. (2013) develop a SDN failure model to describe network failures in SDN and formalize two optimization problems, i.e., *Controller Placement under Comprehensive Network States* (CPCNS) problem and *Controller Placement under Single Link Failure* (CPSLF) problem. They have then propose optimal controller placement algorithms for these failure models. A specialized heuristics, which takes into account a particular set of optimization objectives, is developed by Lange et al. (2015a) and used for controller placement. Lange et al. (2015b) present POCO, a framework for Pareto-based Optimal Controller placement that provides operators for Pareto optimal placements with respect to different performance metrics. For large networks they use a heuristic approach that is less accurate, but yields faster computation. A recent paper by Zheng et al. (2017), is based on a core-edge separated architecture and the control tasks are implemented in a hybrid fashion. The edge switches are clustered into local control groups according to locality of traffic. Frequent coarse grained control tasks are assigned to the network edge. The central controller is only in charge of a very limited number of intergroup or other fine-grained control events. The central controller keeps adapting the grouping of edge switches to maintain its laziness. UlHuque et al. (2017) have partitioned the entire network into sub-regions; based on a maximum latency bound sub-region, it finds out the optimal location of the controller module so that a maximum number of switches can be operated while minimizing the maximum latency. Dynamic controller placement is resource intensive and suitable only in networks that are highly dynamic in nature.

Recent work (Wang et al., 2018) on controller placement in wide area networks, have partitioned the network using a clustering algorithm, such that end to end delay is minimized. Presently, works on controller placement are focusing a wide array of networks such as wireless networks (Dvir et al., 2018), 5G networks (Liu et al., 2018) etc. Also, these works devise complex placement metrics that take into account the specific characteristics of these networks such as higher packet loss or hidden terminal problem of wireless networks of reliability concerns of 5G networks.

2.3 The Data Plane

Destination based routing may lead to scalability issues in SDN as is discussed in Chapter 1. One solution to this problem is to use source routing. Here, the path of the packet from the source to the destination is stored in the packet itself and the switches simply forward the packet based on this information. Source routing is not widely used in traditional network due to absence of a central decision making entity and network wide information. SDN provides both of these approaches and hence source routing is gaining popularity.

Although Secondnet (Guo et al., 2010) is not deployed in SDN it is one of the first work that uses port switching to carry out source routing in data centers. Port switching use the egress port number to store the path information rather than use the link details. They show that source routing is scalable since the servers take the routing decisions and the switches are basically stateless. Soliman et al. (2012) is one of the first papers to discuss source routing in SDN. They have enhanced source routing with reverse path calculation. As the packet enters a switch, the egress port number is replaced by the ingress port number, thereby indicating the reverse path for the packet. Further, they have also showen how link failure can be locally handled, wherein the switch chooses another outgoing link, based on the alternate link information stored in the switch. Source routing frees the controller of the task of installing forwarding information on the switches. Source routing alleviates the scalability problem, but it is not resilient to link failure, since the switches are not actively involved in forwarding packets. In (Stephens et al., 2013), the researchers have focused on making source routing resilient to failure. The main tasks here are; path computation on the controller, source routing, computing the reverse path using the ingress port number of each switch and the Plinko forwarding function. This last function stores alternate routes in the packet, that is to be used when primary path fails. Their algorithm begins by implementing source routing. When a packet encounters link failure, the switch that is local to the failure, finds and stores an alternate path. This information is used to forward the current and all subsequent packets, if the link fails. In (Ramos et al., 2013a)the authors have done extensive work on source controlled routing. A resilient source routing algorithm has been developed as part of this work.

Here, they have computed a primary route and have then found secondary route between source and destination and store them both in the packet. The secondary route is to be used in case of primary route failure. They also ensure none of this lead to loops. Efforts are made to keep the length of secondary route minimal. The experiments were run on Fat tree topology. They have further improved on this work in Ramos et al. (2013b), to support arbitrary topologies. sci propose that latency can be reduced significantly by removing fields in packet headers that are unnecessary in the SDN environment. In (Alizadeh et al., 2013) the authors discuss scalability issues in SDN. They propose the separation of edge and core controllers. Here the edge controller carries out path installation functionality and the core controller carries out the network status collection functionality. Such an approach can improve scalability since the edge controller can quickly install paths as it is not burdened with network status collection functionality. Although sci; Alizadeh et al. (2013) are not about source routing they suggest how source routing is effective by using the unused header bits of packets and separating the status collection from path installation.

Katta et al. (2016) address the scalability problem on the data plane due to limited switch memory, by having a switch that tracks the congestion level for identifying the best path to the destination and not all paths to the destination. They implemented this solution by programming the switch data plane using the P4 programming language (Bosshart et al., 2014). Zhang et al. (2016) develop a switch memory aware routing scheme that reduces TCAM space usage. They aggregate flows between different source and destination pairs and use subnet masks to reduce the number of entries in each table. Another important constraint to the scalability of the data plane is the time it takes to install the long flow table entries onto TCAM memory. Bifulco et al. (2017) use a hybrid approach of hardware and software switches to address this problem, based on the observation that it is faster to install flow rules on software and delete from hardware. Hence, forwarding table updates always happen in software first. Eventually, entries are moved to the TCAM based forwarding tables, to achieve higher overall throughput, and delete from there eventually.

A recent paper (Shirali-Shahreza and Ganjali, 2018) proposed a novel technique of identifying TCP flow that will terminate shortly by looking at FIN packets. This information is then used to expedite the flow rule eviction, thereby reducing the average table occupancy. Judiciously using software tables, that have higher capacity but lower look up times, in conjunction with TCAM memory is another recent solution being proposed (Kentis et al., 2018).

We summarize the controller placement solution in the Table 2.1

Table 2.1: Summary of controller placement solutions

Issues addressed	**Architecture**	**Metric**	**Domain**	**Topology**	**Component of failure**	**Reference**
Onix - Hierarchical controller placement	Hierarchical	Logical groups	Enterprise	Random	None	(Koponen et al., 2010)
Hyperflow - Distributed control plane	Flat	Proximity	Medium enterprise	Star	Component failure	(Tootoonchian and Ganjali, 2010)
Flat control plane	Flat	Reliability	Medium enterprise	Random	Node and link failure	(Zhang et al., 2011)
Effect of different positions of controllers	Flat	Latency	WAN	Random	Node and Link	(Heller et al., 2012)
Kandoo - Offloading controller application	Hierarchical	Proximity	Data center	Tree	None	(Hassas Yeganeh and Ganjali, 2012)
Fast fail over mechanism	Flat	Resiliency protection	Enterprise	Tree	Switch	(Beheshti and Zhang, 2012)
Dynamic controller placement	Hierarchical	Flow set up time	Internet	Random	Switch	(Bari et al., 2013)
Pareto optimal number of controllers	Flat	None	Internet	Random	Node and link	(Hock et al., 2014)

2.4 Outcome of Literature Review

The motivation for the work reported in this thesis on building a scalable and resilient control plane and data plane is derived from the available literature. Control plane scalability has been achieved using multiple controllers. One of the areas, where we felt, there is scope for considerable contribution is the controller placement problem when the controller itself fails. This will make the control plane more resilient. Majority of the previous works, we have reviewed so far, consider only node or link failure not the controller failure.

- Control plane
  - Functionality of an SDN controller belongs to one of the two distinct categories namely, collection of network topology information and setting up forwarding paths on the switches. It is a scalability concern, to expect a single controller to perform both these functions. Separation of control plane based on these functionalities, which will lead to a scalable architecture, is desirable and has not been fully explored in the literature.
  - Several failure models have been considered, including link failure, switch failure etc. One of the areas, with scope for significant work is, the effect of failure of the controller itself on the rest of the network.
  - Latency is the most common metric that has been used to decide the placement of controllers in a multi controller SDN environment. Connectedness of the network or survivability is an important metric, especially on networks that require high availability.
  - SDN architecture inherently supports the end to end communication principle, but this has not been fully harnessed. Attempting an edge-core separation in SDN architecture is an important challenge.
- Data plane

- Although some of the prior works do implement source routing, it's merits especially in SDN have not been fully demonstrated. Since the entire path that a packet has to take to reach a destination in SDN, is known aprior, source routing is an ideal solution. It requires further investigation in conjunction with the end to end communication principle.
- One of the major scalability problems on the data plane is due to the limited memory on the switches. Scalability solutions, taking into account this aspect of SDN, need to be developed.
- Data center traffic comprises of elephant and mice traffic. Mechanisms, to ensure that SDN technology is fair to both in addressing the related issues, is missing in the literature.
- Responding to link failure in source routed networks requires additional effort as the switches are mere forwarding devices. A local corrective measure without contacting the controller would be ideal and needs to be studied.

Table 2.2 gives the issues and the solution methodology we propose to address the issues considered in this work.

Table 2.2: Issues in management of control and data planes, and Proposed Solution

Plane	Issues	Proposed Solution
Control Plane	1. **Scalability**: status collection and forwarding rule installation	**Multicontroller**: separate for status collection (core controller) and forwarding rule installation (edge controller)
	2. **Resilience**: network disconnect on controller failure	**Biconnectedness**: between controller domains
Data Plane	1. **Scalability**: limited switch memory and rule installation on all switches along a path	**Source Routing**: path in packet and only one communication by controller for a path
	2. **Resilience**: path unknown when link fails	**Path Protection**: store backup paths locally so that there is no need to contact controller

2.5 Motivation for Further Investigation

A review of the literature in the area of SDN and related fields has led us to the observation that, if capability of SDN is to be widely and completely harnesed, its architecture must be modified so that it becomes more scalable and resilient to failures in the context of both control and data planes. This is especially true when we consider SDN in large data center networks.

Given below are the specific problems we tried to address in this thesis.

- On the control plane:
  1. Employing a single controller, to manage the entire network, does not adequately address the issues of diverse nature and latency requirements of the applications. This also does not cater to the varying traffic characteristics within different domains.
  2. When a controller fails, the switches under its control may not forward packets reliably. This may also affect the transmission of packets between switches that are fully functional in other controller domains.
- On the data plane:
  1. The data plane has scalability issues due to (i) limited TCAM memory on SDN switches and (ii) larger size of forwarding flow rules. This limits the number of flow rules that can be stored on these switches.
  2. In the event of a failure occurring in the data plane, the packet forwarding imposes additional computation and communications overhead on the control plane. Ideally, the control plane must enable the data plane to handle failures locally.

2.6 Problem Statement

On the control plane, an edge core separation, ensures scalability. In terms of network survivability, the network must be resilient to controller failure. On the data, plane the forwarding mechanisms must enable the end to end principle, while being fair to all traffic and also being resilient to link failure. Here, the most important design issue, is to ensure, that the control and data plane solutions, are coherent and work well together.

With this understanding the following problem is addressed in this thesis *Design, implement and evaluate a scalable and resilient control plane and data plane, in a software defined data center network.*

Our research objectives are

- **Scalable and resilient control plane** Given a network consisting of switches, and links connecting these switches, assign a switch to one of the controllers $\{C_1, C_2 \dots C_n\}$ such that

    1. C_i can manage the communication, between the switches under its control, without contacting any other controller.
    2. If controller C_i fails then
        (a) Switches managed by C_i are rendered non functional.
        (b) All other switches are up and can communicate.

- **Scalable control plane for the edge-core SDN model** Given an edge-core SDN network, to develop separate edge and core placement metrics and design algorithms, for controller placement using these metrics.

- **Scalable and resilient data plane** Given a network consisting of switches, and links connecting these switches, route packets from the source to the destination such that

    1. Scalability:
        (a) The number of forwarding entries in the intermediate switches are reduced .
        (b) The controller to switch communication is reduced.
    2. Fairness: The routing must be fair to both the elephant and the mice traffic.

3. Resiliency: In the event of node-link failure re-route the packets, with as little communication with the controller as possible.

2.7 Scope of Work

Networks under a single administrative entity, forms the basis of the network model used in this work. Large data center models are normally of this kind. The controller placement problem can be analyzed with respect to various failure models, however in this work, only survivability of the network when the controller fails is, considered. Inter controller communication models are not studied. Source routing is best suited for networks which are centrally monitored and controlled. Therefore, the scope of this work is restricted to data center networks that use source routing, with multiple controller domains. We focus on the scalability and resiliency issues, with respect to both source routing and controller placement problems, when we consider controller failures. The network topologies in data centers are usually densely connected, in order to ensure, that there are no oversubscribed links. Hence it is safe to assume that, in a data center network, there exist switch disjoint paths between any pair of switches.

In the network model model considered for this work, the controller is placed on a dedicated server. It is not collocated on any of the switches. An out of band connection is assumed between the switches and controllers.

In this model, existing network connectivity among switches is considered and it does not add or remove any switch or link to the network, in order to optimize the solution of the problem considered.

2.8 Framework of the Proposed Work

As mentioned in the previous discussion the research contributions are limited to the domains of the control and data planes. The research framework adopted has been presented below.

2.8.1 Control Plane

Scalability through edge and core separation: SDN uses a central controller to handle multiple switches. This poses scalability issues since the controller has two distinct functions namely.

- i To continuously collect the network topology information and the traffic statistics.

- ii To install the forwarding rules in the switches along the path a packet has to take, every time the first packet of a new flow is encountered.

Here, the controllers are divided, as edge controllers and core controllers. The edge controller handles computations that need to be completed fast at the edge itself. The core controller handles the core switches in the cloud or data centers. The issue involved and addressed this regard is, optimizing the number of the edge controllers and core controllers

Scalability through controller placement: The second issue addressed in this thesis is to assign an edge switch to an edge controller and a core switch to a core controller. In order to do this efficiently a placement metric has to be chosen. The chosen metric may be different for edge and core controllers, as their functionalities differ. Once a placement metric is chosen an algorithm has to be developed to assign the switches as per the metric. This involves:

1. Formulating an edge controller placement metric, based on the functionality of the edge controller. The metric must ensure data plane resiliency without deteriorating the latency.

2. Formulating a core controller placement metric, based on the functionality of the core controller. The metric must ensure data plane resiliency without deteriorating the latency.

3. Designing and implementing an algorithm for edge switch assignment, to edge controller using the developed edge metric.

4. Designing and implementing an algorithm for core switch assignment, to core controller using the core metric developed.

Resiliency of network during failure of a controller means that the switches under its control, are no longer reliable, since it has lost contact with the controller, even though all switches are functional. However, we would like to ensure that this surely does not affect connectedness of the rest of the network. This involves

1. Evaluating the effect of edge and core controller failure on the network in terms of connectivity, packet forwarding capability etc.
2. Ensuring that the network is resilient to controller failures.

2.8.2 Data plane

Scalability of data plane: The limited TCAM memory available on the SDN switches and the increased size of the SDN flow rules, restrict incorporation of more flow rules. Based on earlier motivating results, it can be said that source routing provides an elegant solution to the scalability problem. This is because, with source routing the path information is stored in the packet itself and not on the switches. The switches therefore behave as simple forwarding elements that look at the packet header and forward the packet accordingly. Definitely this increases the packet overhead; however the overhead is within control for short path lengths. This issue involves

1. Devising encoding schemes to ensure that the overhead imposed due to insertion of path information into the packet, is minimized.
2. Developing a mechanism for multi controller forwarding, that takes care of how the packet is routed between two controller domains, such as from the edge to the core domain.

Resiliency of data plane: Data plane resiliency ensures that packets that are in transit would not be dropped when a link on the path fails. Additionally, corrective measures can be taken locally, at the switch level, without contacting the controller. This will reduce the latency, however, it is also a challenge since the switches have limited intelligence in SDN. This issue involves:

1. Designing of a mechanism using which the switch can handle link failure; by taking local corrective measures, using minimal resources of the switch and with minimal controller communication overhead.

2.9 Summary

This chapter reviewed the literature that was published in the area of SDN; focusing especially on the scalability and resiliency issue therein. The findings of the review were summarized and some important lacunae which needed to be addressed were listed. Based on these lacunae the research problem statement and the objectives were developed and presented. This was followed by a section on the scope of work and assumptions made. The detailed framework of the research work was also discussed. Chapter 3 discusses the approach taken and solution designed, to address the scalability and resiliency problems of the control plane.

Chapter 3

Building Scalable and Resilient Control Plane

If you're not prepared to be wrong, you'll never come up with anything original.

-Sir Ken Robinson

3.1 Overview

This chapter addresses the controller placement problem i.e. the problem of assigning a switch to a controller based on carefully chosen metrics, in a multi-controller environment. A controller placement metric known as subgraph-survivability and a controller placement algorithm, that uses this metric, is presented in this chapter. We thoroughly analyses the survivability of the software defined network in the event of a few a switch being non-functional as a result of controller failure. The controller placement algorithm ensures that rest of the network remains connected even when some switches become non functional. The correctness of the algorithm is demonstrated using formal proofs. Further the algorithm is evaluated against the random controller placement algorithm, based on the Average Inverse Shortest Path Length and Network Disconnectedness. The proposed algorithm is run on real topologies and some synthetic topologies. The results obtained show that the proposed subgraph-survivable controller placement algorithm performs better, especially when large networks are considered.

3.2 Logically Centralized, Physically Distributed Controllers.

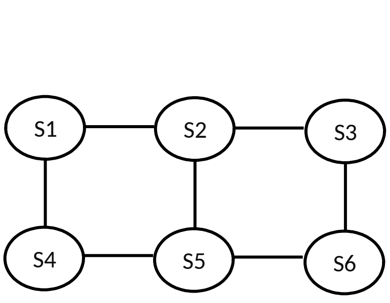
A centrally controlled network has many advantages as discussed in Chapter 1. However, such an architecture is not scalable, has a single point of failure and a weak response times as the number of switches grows. Hence, physically distributing the controllers, while still maintaining a logically centralized control gives the best of both worlds. The introduction of these multiple controllers gives rise to the question of how the switches are assigned to a controller or in other words the controller placement problem (Heller et al., 2012). The placement metrics chosen depends on the network topology and the application domain.

There are two techniques by which the control plane can be made resilient. One is through replication and the other is using partitioning. In the first technique, a cluster of controllers is used to manage the network. Here, the failure of one controller, makes the other redundant controller take over. Although, this approach gives better resilience, it is still not scalable. The second technique, partitions the network into multiple domains, each one managed by its own controller. This method is scalable. Further, replicating the controller in each of the domains makes it resilient too. This work focuses on partitioning the network, rather than replicating the controllers.

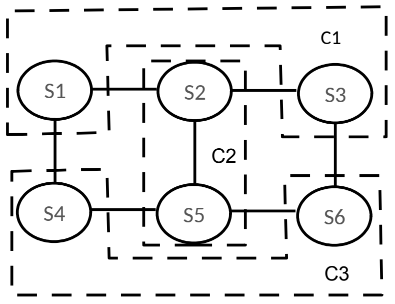
3.3 Motivating Scenario

Randomly assigning switches to the controllers, especially in a data center, where the network is expected to be up and available at all times, may lead to the following problems

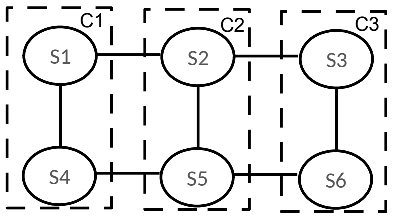
1. Although two switches that want to communicate, may belong to a single domain, the intermediate switches on the path connecting them might belong to different domains. This necessitates inter controller communication, to ensure two switches under a single domain can communicate.



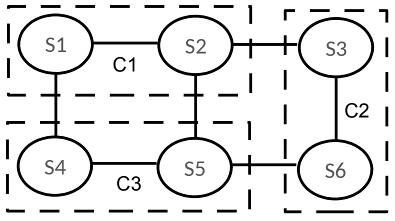
(a) Topology



(b) Placement 1



(c) Placement 2



(d) Placement 3

Figure 3.1: Comparison of Different Controller Placement Scenarios

2. When a controller fails, the switches under its control cannot forward packets reliably, since they do not obtain the forwarding rules. These non functional switches may disconnect rest of the network.

This problem is illustrated in Figure 3.1. Figure 3.1(a) shows network topology, with six switches s_1 to s_6. C_1, C_2 and C_3 are three controllers and each can handle up to two switches. The problem is to assign a switch to any one of the three controllers, such that;

1. If the source and destination switch belong to a single domain, then all switches along the path between them, must also belong to the same domain

2. If any controller fails, then the switches under other controllers can still communicate or in other words the network is not disconnected.

In Figures 3.1(b), 3.1(c) and 3.1(d), the dotted blocks indicate controller assignment. The switches within a dottted block are allocated to the same controller. Considering that the switches are assigned to the controllers as shown in Figure 3.1(b) where $C_1 = \{s_1, s_3\}$, $C_2 = \{s_2, s_5\}$, $C_3 = \{s_4, s_6\}$. The removal of any controller due to failure will not break the connectivity between the domains. However, it fails to satisfy the first condition given above. In order for switches s_1 and s_3 under the controller C_1 to communicate, s_2, a switch under C_3, has to be contacted.

Considering the next scenario where the switches are assigned to the controllers as shown in Figure 3.1(c) where $C_1 = \{s_1, s_4\}$, $C_2 = \{s_2, s_5\}$, $C_3 = \{s_3, s_6\}$. Although this satisfies the first condition, it fails to satisfy the second condition. This is because, if controller C_2 is removed, then s_2 and s_5 switches are unable to forward packets and this causes the network to disconnect. Switches under the control of C_1 and C_3 cannot communicate with each other.

Considering the third assignment of switches, as shown in Figure 3.1(d) where, $C_1 = \{s_1, s_2\}$, $C_2 = \{s_3, s_6\}$, $C_3 = \{s_4, s_5\}$. Here, if C_2 controller fails, then switches s_3 and s_6 are non functional. However, this does not cause the network to disconnect. Switches under the control of C_1 can still communicate with switches under the control of C_3. This holds true when any of the controllers fails.

Therefore, the switches must be assigned to controllers such that, removal of any one controller does not cause the network to disconnect. This holds true, only if the clusters of switches under different controllers are biconnected. The proposed work discussed in this thesis is restricted to the case where only one controller fails at any given time. One approach, to mitigate failure, is to build a redundant cluster of controllers. This approach although expensive allows for failover mechanism. However, in this work, the network is made more resilient in the event of targeted attack on the cluster of controllers. Such a solution complements the redundant controllers approach and makes the system more robust.

3.4 Scalable and Resilient Control Plane

Placement Metric: The network under consideration here is a large densely connected network under a single administrative control, such as data center networks. Although, the controller placement algorithms can be designed for different failure models, this work is confined to the network survivability on controller failures.

Although latency is a commonly used placement metric, it is not relevant in the data center models. As the average number of hops between a switch and the controller is about 8 (Benson et al., 2010). Latency is therefore not a significant factor for controller placement.

Data centers run applications that require high availability and fault tolerance. Such networks are survivable if they continue to operate in the presence of targeted attacks. In the proposed work, survivability is considered as one of the performance metric. Survivability is ensured by providing diversity, so that fate sharing does not occur. In networks this can be accomplished by providing path diversity.

Data center networks are densely connected to ensure high availability and also to make sure that the links are not oversubscribed. It is therefore safe to assume, that switch disjoint paths between any source and destination switch exists. Two paths are switch disjoint if they do not share any switches. The SDN model that is proposed in this work, places the controller on a server and not on one of the switches. The communication between the switch and the controller may be carried out in two ways; using the existing network infrastructure or having a separate set of links between the switches and the controller. The former is referred to as in band connection while the later is referred to as out of band connection. In our work we consider only out of band connection from the switch to the controller. Further, the controller placement problem is solved, without changing the existing network topology i.e. without adding or removing any new link or switch, to or from the network.

Controller Placement Model: Given a network consisting of switches, and links connecting these switches, assign a switch to one of the controllers $\{C_1, C_2 \ldots C_n\}$ such that

1. C_i can manage the communication, between the switches under its control, without contacting any other controller.

2. If controller C_i fails then

    (a) Switches managed by C_i are rendered non functional.

    (b) All other switches are up and can communicate.

The Network Model: The network is represented as an undirected graph $G = (V, E)$, wherein node v_i belonging to the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents a switch and $n = |V|$ gives the total number of switches and the edge set E represents the links connecting the switches. Vertices which are incident with a common edge are said to be adjacent. Two distinct adjacent vertices are neighbours. The set of neighbours of a vertex v is given by $N(v)$. Let $C = \{c_1, c_2, \ldots, c_k\}$ denote the set of controllers.

Subgraph: Given a graph $G = (V, E)$ and set of vertices S s.t. $S \subseteq V$, then the subgraph of G induced by S is denoted as $G[S] = (S, E_S)$ where $E_S = \{uw, s.t.E : \{u, w\} \subseteq S\}$.

Connected Subgraph: Given a graph $G = (V, E)$ and a set of vertices S where $S \subseteq V$, then the subgraph $G[S]$ induced by S is connected if there exists a path $P = \{u, v_i, v_j, \ldots, w\}$ between every pair of vertices $\{u, w\}$ of S such that all vertices of $P \subseteq S$.

Reduced Graph: Given a graph G and a set of subgraphs $SS = \{S_1, S_2, \ldots, S_k\}$ such that $\bigcup_{1 \leq i \leq k} V[S_i] = V[G]$, G is reduced to R by contracting the edges within each subgraph G_i, such that all vertices and edges in a subgraph collapses to a single super vertex. The number of vertices in the reduced graph is equal to the number of subgraphs in SS which is k .

Subgraph-Survivability: Given graph G is to be reduced to R, let $P_{i,j}$ denote the path between any two super vertices i and j of R. Let $P^k_{i,j}$ denote the k^{th} path between i and j. Then $C(P^k_{i,j}) = c_x$ such that c_x is the controller, to which nodes on path P_{ij} are connected . Subgraph-survivability is ensured if $\bigcap(C(P^k_{i,j}))$ for any two $k = \emptyset$

Problem Definition Given a connected, undirected graph $G(V, E)$ where $n = |V|$ is the total number of nodes in the graph, m is the maximum number of nodes in a subgraph, the problem is to find a set of subgraphs $SS = \{S_1, S_2, \ldots, S_k\}$ such that,

i. $|S_i| \leq m$

ii. k is minimal and $k > 0$

iii. $S_1 \cup S_2 \cup \ldots \cup S_k = G$

iv. $\forall i, j S_i \cap S_j = \emptyset$

v. for any node $w \subset P_{uv}$ if $(u, v) \subset S_i$ then $w \subset S_i$

vi. G is *subgraph-survivable*

3.4.1 Subgraph-Survivable Controller Placement Algorithm

Subgraph-Survivable Controller Placement *(SSCP)* approach is given in Algorithm 1. It groups the switches of a given data center network and assigns switches within each group to a controller such that it is subgraph-survivable. The algorithm proceeds as follows

- Step 1: Depth First Search (DFS) traversal of the graph is carried out and the tree edges and back edges are labelled. Let T be the DFS tree thus obtained.

- Step 2: In this DFS tree T, m or less number of adjacent nodes are grouped together into subgraph S_i such that the following are true

  - S_0, the root group of the DFS tree has a single child
  - S_i, the non root group of the DFS tree, which has a child group S_j such that some group belonging to the subtree rooted at S_j, has a back edge to an ancestor group of S_i. In other words if the graph G is reduced on the basis of the subgraphs obtained above, the resulting graph R will have no articulation point.

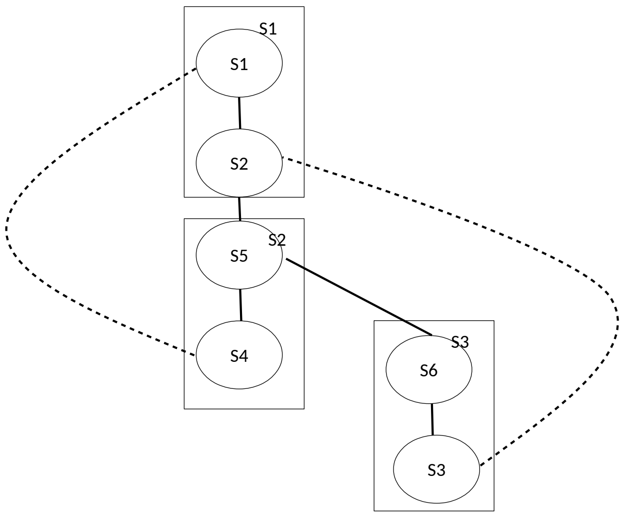


Figure 3.2: SSCP algorithm output for the example scenario

Thus, if any subgraph, i.e. controller domain, is rendered non-functional as a result of its controller failing, the remaining subgraphs are still connected.

fbnGlobal in the algorithm is a pointer that points to the farthest child node which has a back edge to an ancestor node of the current node. The DFS tree, obtained previously is traversed and the nodes are grouped together, as long as *fbnGlobal* points to a descendant node. This ensures that the group formed is not an articulation group. When the node pointed to by *fbnGlobal* is reached, a new group is started and *fbnGlobal* is updated as in line 14 of Algorithm 1. Further, a new group is started when a node with two or more children are reached, as this may disconnect the graph G. Also, the unexplored branches in the tree are next traversed, as shown in line 18 to 25. The algorithm stops once the entire tree is explored.

For the example topology given in Figure 3.1(a), the DFS tree and the subsequent subgraphs S_1, S_2 and S_3 are obtained by Algorithm 1 as shown in Figure 3.2.

The algorithm takes linear time since it involves:

1. obtaining the DFS tree from the input graph G, whose complexity is $\mathcal{O}(|V|+|E|)$

2. a traversal of the DFS tree itself, step 2 through 28

Algorithm 1 Subgraph-Survivable Controller Placement algorithm

Input: The graph $G(V, E)$; m the maximum number of nodes allowed in a subgraph

Output: S the subgraphs

```
1: $u \leftarrow DFS(G)$
2: $fbnGlobal \leftarrow \infty$ {$fbnGlobal$ is a pointer the farthest back edge node}
3: $fbnLocal \leftarrow 0$ {$fbnLocal$ is a pointer to the farthest back edge node
   encountered in the current subgraph grouping}
4: repeat
5:   while $(!fbnGlobal)$and$(u.child \neq null)$and$(count(u.child) < 2)$or$(n < m)$or$(u \neq null)$ do
6:     $S_i \leftarrow S_i \bigcup \{u\}$
7:     $fbn \leftarrow max(fbn, max(u.fbn))$ {$u.fbn$ is an array with the depth of the
       node which has a back edge to $u$}
8:     $path \leftarrow u.path$
9:     $n++$
10:    $u \leftarrow u.child$
11:  end while
12:  $n \leftarrow 0$
13:  $fbn_i \leftarrow fb$
14:  $fbnGlobal \leftarrow maximum(fbnGlobal, fbn)$
15:  $fbn \leftarrow 0$
16:  $i++$ {subgraph index}
17:  if $u == null$ then
18:    $u \leftarrow unexplored\_branch$
19:    $fbnGlobal \leftarrow maximum(unexplored\_fbn, unexplored\_fbn\_i)$
20:  end if
21:  if $(u.child) > 1$ then
22:    if $path = left$ then
23:      $unexplored_branch = right\_child(u.child)$
24:    end if
25:  else
26:    $unexplored_branch = left\_child(u.child)$
27:  end if
28:  $unexplored_fbn = maximum(u.fbn)$
29: until $unexplored = null$
```

The time complexity of this algorithm is therefore $\mathcal{O}(|V| + |E|)$

3.4.2 Correctness Proof

Lemma 1. *The subgraph S_i of G obtained using the SSCP algorithm is connected.*

Proof. The subgraph S_i is obtained by picking adjacent nodes, which are connected by tree edges in the DFS tree T of graph G, in lines 5 to 11. Since the tree edges of T represent adjacent nodes in graph G, S_i is also connected. □

Lemma 2. *G obtained by the SSCP algorithm is subgraph-survivable*

Proof. The following loop invariant is used:
In the DFS tree T, at the start of the *while* loop, the nodes of the subgraph S_i under consideration, have a descendent node k. This node has a back edge to a node which is an ancestor of nodes in subgraph of S_i. This is true in all cases except for the root and leaf subgraph.
Initialization
Prior to the first iteration of the loop, subgraph $i = 1$ is empty

Maintenance
$Safenode$: This refers to a node which has a child node with a back edge to one of the subgraph that is obtained thus far, i.e. a parent subgraph.
In the DFS tree T, a Farthest Back Edge (fbn) node is the farthest node, from which, there is a back edge to any of the subgraphs Sa that is obtained thus far. The DFT is traversed down, until the fbn node. The traversed nodes, called $safenodes$, are added to the current subgraph S_i, except for fbn, provided the size of subgraph is not greater than maximum allowed. This ensures that an ancestor of current subgraph S_i has a back edge from a descendent node of S_i as shown in Figure 3.3.
The loop in line 4 to 28, of Algorithm 1, ensures loop invariance by adding only $safenodes$ to the subgraph.
Termination
Except for the root and leaf subgraph node, all subgraphs have a child node pointing to a parent subgraph.

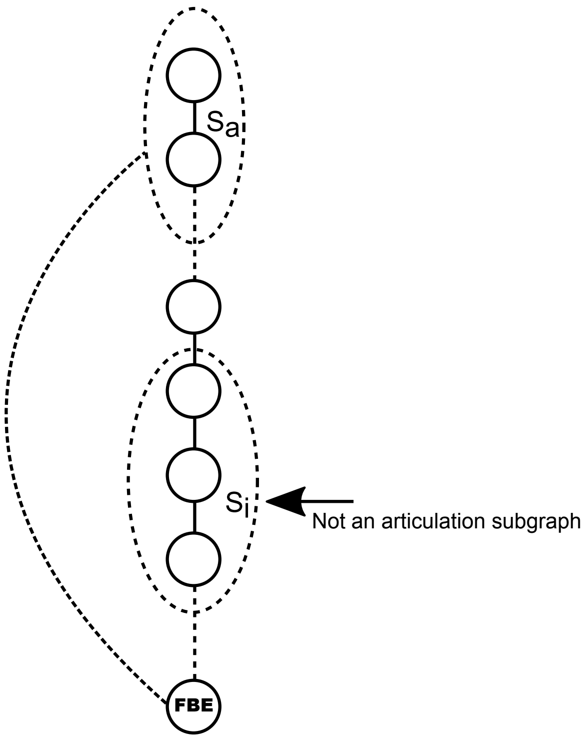


Figure 3.3: Creation of subgraphs by grouping nodes in *SSCP*

□

Theorm 3. *Given a graph G the SSCP algorithm gives a set of connected subgraphs* $S = \{G_1, G_2, G_k\}$ *s.t.* $\bigcup_{(1<=i<=k)} V[Gi] = V[G]$ *and G is subgraph-survivable.*

Proof. The algorithm is run until all nodes of the graph G are included in some graph S_i as indicated in the while loop in line 4 to 28. Hence based on Lemma 1 and Lemma 2, the Theorm 3 is proved. □

3.4.2.1 Connected-Random Controller Placement Algorithm

In random controller placement algorithm, a node is randomly assigned to a controller, such that, nodes belonging to a controller are all connected to each other.

The algorithm begins with a subgraph consisting of a randomly chosen node. Next, one of the neighbors of the subgraph is chosen and included in the subgraph. This ensures the nodes of the subgraph are connected. This process is repeated until a maximum size of subgraph is reached or there are no more nodes in the neighborhood

The process is repeated starting from any unassigned node in the graph, until all nodes are assigned.

Algorithm 2 Connected-Random Controller Placement algorithm

```
Input: The graph $G(V, E)$; $m$ the maximum number of nodes allowed in a
    subgraph; random node $u$.
Output: $S$ the subgraphs
 1: $i = 0$
 2: $n = 0$
 3: repeat
 4:   while $(n < m)$and$(u \neq null)$ do
 5:     $i = S_i \bigcup \{u\}$
 6:     $n++$
 7:     $assigned(u) = true$
 8:     $updateN_i$ {include unassigned neighbours of $u$ into $N_i$}
 9:     $u = random(N_i)$
10:   end while
11:   $i++$
12: until $assigned(V) = true$
```

The Connected-Random Controller Placement (*CRCP*) algorithm, given in Algorithm 2 begins by assigning a vertex u to the subgraph S_i. A neighbour set N_i maintains all the neighbours of the subgraph S_i. The next node to be added to the subgraph S_i is chosen randomly from the neighbour set N_i. The subgraph formation stops when it has the required number of nodes or there are no more nodes to be added. The next subgraph formation starts with any unassigned node and continues till there are no more unassigned nodes in the graph G.

3.5 Evaluation

The algorithm is implemented and analyzed in Matlab. The performance of the *SSCP* algorithm is evaluated by comparing it with the *CRCP* based on proposed evaluation metric *Network Disconnectedness.*

We define *Network Disconnectedness* as

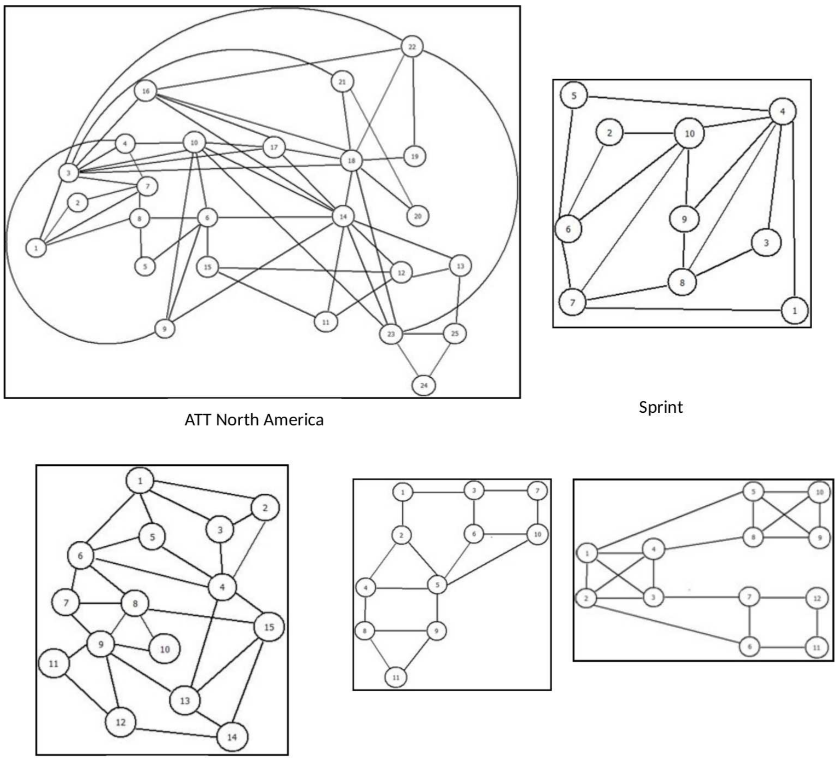


Figure 3.4: Network Topologies used for Evaluation

$$ND = \frac{DP}{CP} \tag{3.1}$$

Here, DP is the number of disconnected pairs of nodes after a subgraph is removed and CP is the total number of connected pairs of nodes in the graph G before the subgraph is removed. This metric is a direct measure of how many paths in the graph are disconnected, as a result of a controller failing or in other words a subgraph removal. If ND has a value 1, then it indicates total disconnection and a value of 0 indicates no disconnection.

The algorithm is also studied using the widely used measure such as, *Average Inverse Shortest Path Length (AISPL)* (Latora and Marchiori, 2001), which is obtained using the formula

$$AISPL = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{L_{ij}} \tag{3.2}$$

Table 3.1: Comparison of CRCP and SSCP Algorithms
Legend: CRCP: Connected-Random Controller Placement; SSCP: Subgraph-Survivable Controller Placement; AISPL: Average Inverse Shortest Path Length; ND: Network Disconnectedness

Topology	**No. of Nodes**	**No. of Edges**	**CRCP**		**SSCP**		**% Improvement**	
			AISPL	**ND**	**AISPL**	**ND**	**AISPL**	**ND**
ATT North America	25	56	0.2958	0.3669	0.4069	0.2300	37.5	37.2
Sprint	10	17	0.2685	0.5333	0.5593	0.2000	73.5	62.4
Telecordia	15	28	0.2933	0.7429	0.4849	0.2571	65.3	65.39
Topology 1	11	16	0.2564	0.7091	0.3818	0.3455	48.9	51.2
Topology 2	12	20	0.2816	0.4545	0.4343	0.3182	54.2	29.9

where N is the number of nodes and L_{ij} is the shortest path length between nodes i and j

It must be noted that, non-existent path between disconnected nodes contributes a zero to the final value and hence, *AISPL* gives information about degree of disconnectedness along with average shortest path. A high value of *AISPL* indicates connectedness and a low value indicates higher disconnectedness.

The proposed algorithm is run on network topologies like Sprint, TelCordia, ATT North America (Knight et al., 2011) and two other synthetic topologies which are shown in Figure 3.4a-e. However, the proposed work can be applied to any network that needs to be partitioned and therefore some Internet topologies are considered along with data center topologies.

Table 3.1 compares the values of *AISPL* and *ND*, obtained when using the *CRCP* and *SSCP* algorithms, for various topologies. The table shows that the value of *AISPL* is more for *CRCP* when compared to that for *SSCP*. This indicates that with *CRCP* we have more disconnectedness and longer average path lengths It may also be observed that the value of *ND* is less for *CRCP* when compared to that for *SSCP*. Both these changes in the values show that the network is better connected with the *SSCP*. Hence, it may be concluded that with *SSCP* the nodes that get disconnected from each other are the ones that belong to the removed subgraph only and the rest of the network remains better connected. Hence, the proposed algorithm helps make the network scalable by assigning switches to multiple controllers. It also makes the network resilient, since the removal of a set of switches whose controller has failed, will not disconnect the rest of the network.

3.6 Summary

In order to ensure scalability, multi-controller SDNs are steadily gaining importance. Ensuring the survivability of such networks, in the event of a controller failing, is also critical to modern applications. Motivated by this need for scalability and resiliency in the control plane, a placement metric called *subgraph-survivability* and an algorithm for controller placement was proposed in this chapter. The algorithm ensures scalability and resiliency in the SDN multi-controller scenario. Apart from this intended application, the algorithm is significant in other domains too, such as, road transport system and social networks. Theoretically, this research proposed an algorithm to obtain k subgraphs of maximum size m such that k is minimal and there exists biconnectivity between the subgraphs. This algorithm can be used in any networked domains such as social networks etc. to ensure that the removal of a subgraph does not fragment the rest of the network. Chapter 4 extends the solution presented here to the Edge-core SDN model.

Chapter 4

Scalable Controller Placement in Edge-Core SDN

Being out on the edge, with everything at risk, is where you learn and grow the most.

-Jim Whittaker

4.1 Overview

SDN when deployed in networks that includes, a large number of networking elements and heavy traffic flow volume, does not scale well. A single centralized controller can not efficiently manage the networks requirements. Distributed controller architectures are recommended in such cases. The edge-core SDN is an extension to this classical SDN architecture. Here, the data plane is divided into edge and core networking components which are being managed by edge and core controllers on the control plane. Classical SDN architectures generally use static mapping of switches to controllers. When traffic conditions change, such static allotment may lead to unbalanced assignment of load among the controllers. This chapter presents a dynamic controller placement algorithm for the edge-core SDN architecture, using a different set of metrics for the edge and core of the network. The switch migration procedure for transfer of switch from one controller to another, necessary for dynamic load balancing is discussed next. Further, mechanisms by which source routing helps to better establish the

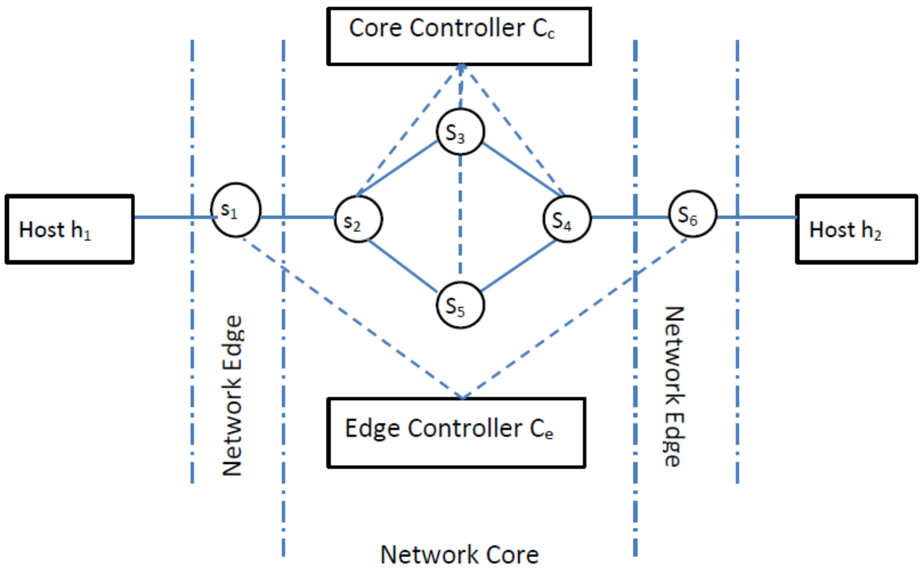


Figure 4.1: Edge-Core SDN architecture

edge-core SDN architecture, is also presented.

4.2 Edge-Core Model

The network can be categorised into three components, namely: the end hosts, edge switches and core switches. The end hosts send and receive packets. The edge switches are connected to the end hosts and they form the ingress and egress switches. The remaining switches are the core switches and they form the network fabric. Complex network services and policies are usually implemented at the edge switches, while basic packet forwarding is carried out at the core switches. Since, the functionalities carried out by these two sets of switches are different, ideally they should be managed by different controllers. This is the idea behind edge-core SDN architecture, i.e. it separates the edge of the network from the core, while also decoupling the control and data plane, as in regular SDN. The edge-core SDN architecture is shown in Figure 4.1

In the classic SDN architecture, a single controller that has a complete view of the network topology, manages the data plane on all the switches. Such an arrangement might lead to reduced controller performance and scalability concerns, especially in large-scale networks such as data centres or wide area networks. For example, a controller might not be able to handle the heavy traffic flows initiated within the network, due to its limited processing capacity. As a consequence some packets might be discarded without being processed. At the same time, the controller is also burdened by the responsibility of collecting the network and traffic status information, in order to maintain an up to date network view. Additionally, a single centralized controller has a single point of failure problem. All these factors necessitate a logically centralized but physically distributed control plane. Although placing multiple controllers can overcome the above limitations, new challenges arise, such as how many controllers are required and how the switches are assigned to these controllers.

Most of the available literature discusses the static controller placement. Load balancing schemes in multi controller scenarios are largely unexplored. Dynamically adjusting the controller placement in accordance with the changing traffic load on each switch is desirable. This is a motivation to develop a dynamic controller placement algorithm that balances the load between the controllers, by choosing the optimal switch for migration, while also avoiding packet loss. Further, different and appropriate parameters, for the edge and core of SDN, are chosen, depending on their functionalities.

4.3 Controller Placement in Edge-Core SDN Architecture

The network topology of an edge-core SDN that uses source routing, is modelled as an undirected graph $G(V, E)$ where vertex set V represents the set of switches and edge set E represents the set of links between these switches. Here $V = S_c \cup S_e$ where S_c is the set of core switches and S_e is the set of edge switches. Let C represents the set of controllers, where $C = C_c \cup C_e$ where C_c is the set of core controllers and C_e is the set of edge controllers. Switches in S_c have to be assigned to one of the controllers in C_c and the switches in S_e have to be connected to a controller in C_e. Once this initial switch allocation is done, the next step is to balance the load on the edge controllers according to the varying traffic conditions.

Core Subgraph: Given an undirected graph $G = (V, E)$ and a set of vertices S_c s.t. $S_c \subseteq V$, then the subgraph of G induced by S_c is denoted as $G[S_c] = (S_c, E_{S_c})$ where $E_{S_c} = \{uw, s.t.E : \{u, w\} \subseteq S_c\}$.

Edge Subgraph: Given an undirected graph $G = (V, E)$ and a set of vertices S_e s.t. $S_e \subseteq V$, then the subgraph of G induced by S_e is denoted as $G[S_e] = (S_e, E_{S_e})$ where $E_{S_e} = \{uw, s.t.E : \{u, w\} \subseteq S_e\}$. The controller placement problem in an edge-core SDN architecture can be described using the following steps

- In the core network, given a set of core controllers $C_c = C_{c1}, C_{c2}, \ldots, C_{cn}$, a switch in S_c must be assigned to any one of the controllers C_{ci} using a metric that is relevant to the core network.
- In the edge network given a set of edge controllers $C_e = C_{e1}, C_{e2}, \ldots, C_{en}$, a switch in S_e must be assigned to any one of the controllers C_{ei} using a metric that is relevant to the edge network
- Dynamically balance the load among the edge controllers in accordance with the changing traffic pattern by
  - Choosing the best candidate switch S_{ei} to be migrated
  - Formulating a switch migration procedure to migrate a switch s_{ei} from c_{ei} to c_{ej}

4.4 The Edge-Core SDN architecture

The edge-core SDN architecture is an attempt to overcome the shortcomings of SDN, by adopting the insight underlying Multiprotocol Label Switching (MPLS). The first issue is that the SDN hardware switches in the data plane, must support look up over multiple fields in order to look for matching entries. This makes the switch hardware design more complex which is against the promise of a simple hardware in an ideal network. The second issue is that the end host characteristics and network core characteristics are not treated differently. For example, if the external network protocol has changed from IPV4 to IPV6, there will be a change in the corresponding behaviour of the core network too. Also implementation of network services and policies must ideally be carried out in a location which is as close to the origin of data as possible i.e. the edge of the network. The edge-core SDN architecture aims to address these issues by separating the edge from the core network and at the same time it retains the flexibility of centralized control plane provided by regular SDN.

In the edge-core SDN architecture the edge and core are controlled by different controllers.The responsibility of edge switch is to provide complex services such as network security and policy implementation.The core on the other hand has only one duty, that of carrying out basic packet forwarding. As shown in the 4.1 when a host h_1 sends a packet it first reaches the edge switch s_1, which then contacts the edge controller C_e.The controller installs the flow rule on the switch, after taking into consideration the policies defined for that flow. Now the edge switch sends the packet to the core.The core switches contact the core controller, which install the forwarding rules on all the switches along the path the packet has to take. However, if forwarding is carried out in this manner, then the core switch has to carry out the matching action on its forwarding table for all packets. To avoid this we use the source routing which allows the path information to be embedded in the packet header itself. For example in Figure 4.1 if h_1 has to send a packet to h_2, then the egress edge switch s_1 will contact the edge controller C_e which will obtain the entire path information from the core controller C_c. This path information is passed to the edge switch s_1, which will then embed it in the packet header and forward the packet. When this packet reaches the switch s_2 and all the subsequent switches along its route to h_2, the switch need not contact the controller nor perform a forwarding table match action. It can simply forward the packet using the path information embedded in the packet header. Source routing is ideal in such networks, where a central entity has the view of the entire network, and can take decisions regarding the path along which the packet has to be forwarded. Delegating this decision to the switch is counter intuitive to the design philosophy of SDN, but provides an elegant solution to the edge-core design.

4.5 Controller Placement

The controller placement problem addresses the questions of how many controllers are required for a given network and how the switches are managed between these controllers. Here, we extend this to the edge-core SDN with respect to edge and core controllers. The choice of a good metric for controller placement is critical for the efficient transfer of packet in the network. Some of the characteristics of SDN architecture that might influence the choice of a metric are switch to controller latency and PacketIn messages. An increase in the latency between the switch and controller degrades the performance, which is undesirable, since this is a communication cost that is peculiar to SDN and is absent in traditional networks. PacketIn are sent from the switch to the controller, when the switch encounters a packet for which it has no forwarding information available. This happens when the first packet of a flow hits the ingress switch. PacketIn messages constitute a large portion of the load on the controller.

4.5.1 Controller placement in Core

As discussed earlier we choose source routing as the forwarding mechanism for the edge-core architecture. Consequently, the core switches need not contact the core controllers to forward these packets, since the path information is embedded in the packet header. The switch to controller latency therefore is not relevant in core network. Also the PacketIn messages are not generated by the core switches for the same reason. The core controllers function is to collect and maintain information about the core network, such as topology and switch status information. Hence, the appropriate metric to use in the core network is a load on the controller, in terms of the number of switches assigned to it.

Algorithm 3 Core Controller Placement

```
1: $T_h = n/k$
2: $Availableset = C_c$
3: $i = 1$
4: $s_{ci}.color = GRAY$
5: $s_{ci}.\pi = C_{ci}$
6: $C_i.load = 1$
7: **for** $s_{cj} \in S_c - s_{ci}$ **do**
8:     $s_{cj} = WHITE$
9:     $s_{cj}.\pi = Nil$
10:    $ENQUEQUE(Q, s_{cj})$
11: **end for**
12: **while** $Q$ not empty **do**
13:    $s_x = DEQUEQUE(Q)$
14:    **for** $s_x. \in ADJ\|s_{ci}\|$ **do**
15:        **if** $s_x.Color == WHITE$ **then**
16:            $s_x.Color = GRAY$
17:            $s_x.\pi = C_{ci}$
18:            $ENQUEQUE(Q, s_x)$
19:            $C_{ci} = C_{ci} + 1$
20:            **if** $C_{c_i}.load \geq T_h$ **then**
21:                $Availableset = Availableset - C_{ci}$
22:                $i{+}{+}$
23:            **end if**
24:        **end if**
25:        $s_x.Color = BLACK$
26:    **end for**
27: **end while**
```

Algorithm 3 presents the Core Controller Placement algorithm. Here given a core network represented as a graph $G[S_c]$, with k core controllers in the set C_c and n core switches, the algorithm does a the breadth first search of the graph, assigning switches to a controller until it reaches the controllers threshold T_h, after which the switches are assigned to the next controller until all switches are assigned to a controller as shown in Algorithm 3.

4.5.2 Controller placement in Edge Network

The edge network comprises of those switches that are connected directly to the end host and therefore are the first hop physical switches that a packet will hit. On receiving the first packet of a flow, for which the edge switch has no forwarding information, it contacts the edge controller in order to get the flow rules. The latency between the switch and controller is therefore an important metric. Also the PacketIn messages generated as a result of this switch to controller communication constitutes a significant load on the controller. This depends on the traffic pattern and changes over time. Hence, the load on the controller is another metric for controller placement. Given an edge network represented as a graph $G[S_e]$, with k edge controllers and n edge switches we assign switches to controller such that the metric of switch to controller latency and controller load are optimised as shown in Algorithm 4

4.5.2.1 Dynamic Controller Load balancing in Edge

Algorithm 4 assigns switches to the controllers initially without considering the varying load on controllers due to PacketIn messages. However, once the traffic flow starts, over a period of time, one of the controller may be heavily loaded while the others remains under utilized, due to the variations in network traffic volume in different parts of the network. Algorithm 5 presents a method to dynamically balance the load among controllers depending on the traffic load on the controller. The algorithm finds an optimal switch that has to be migrated from the heavily loaded controller. A proper switch migration algorithm which ensures that no packets are dropped during the migration is also suggested in Algorithm 6.

Algorithm 4 Controller Placement in Edge

```
1: $T_h = n/k$
2: $Availableset = C_e$
3: $i = 1$
4: for $s_{ei} \in S_e$ do
5:     for $c_{ei} \in Availableset$ do
6:         if $Latency(s_{ei}, c_{ei}) == MIN$ then
7:             $s_{ei}.\pi = c_{ei}$
8:             $c_{ei}.load = c_{ei}.load + 1$
9:             if $c_{ei}.load \geq T_h$ then
10:                $Availalbeset = Availabeset - c_{ei}$
11:                $i + +$
12:            end if
13:        end if
14:    end for
15: end for
```

A SDN enabled switch may have multiple controller connections at the same time. The controller can play three different roles with respect to switches, namely Master, Slave and Equal. A master controller will have full control over the switch. It receives all the messages from the switches and it can install flow rules on the switch flow tables. A switch can have only one master controller at any given time. Equal controller is similar to a master except that a switch can have multiple equal controllers at the same time. A slave controller on the other hand can monitor the network topology status, but it cannot edit the flow tables. Controllers operating in these three roles enable a switch to seamlessly migrate from one controller to another. The switch migration is carried out using the following steps

1. The target switch sends equal role message to current controller to initiate the migration

2. Upon receiving confirmation from the current controller, the switch sends

master role request to target controller

3. Finally the switch sends slave role request to current controller

Algorithm 5 Dynamic Load Balancing

```
1: for each $PacketIn$ message from $s_{ei}$ do
2:   $Tot_i = Tot_i + 1$
3:   if 100th $PacketIn$ message at controller then
4:     for edge switch $s_{ei}$ do
5:       $R_i = (1 - w) * R_i + w * Tot_i$
6:     end for
7:     if $c.load \geq T_c$ then
8:       for $c_{ej} \in neighbour[c]$ do
9:         $Getload(c_{ej})$
10:        if $c_{ej}.load == MIN$ then
11:          $Ctarget = c_{ej}$
12:        end if
13:      end for
14:      for edge switch $s_{ei}$ do
15:        $s_{ei}.weight = B * (s_{ei}.degree/Maxdegree) * (R_i) * 10 + a * s_{ei}.latency$
16:        if $s_{ei}.weight == MAX$ then
17:          $Starget = s_{ei}$
18:        end if
19:      end for
20:      $SwitchMigration(c, Starget, Ctarget)$
21:    end if
22:  end if
23: end for
```

Algorithm 5 begins with a load statistics collection module, that runs on the controller, which collects the information about switch load. The load on the edge controller is mainly due to the PacketIn messages from the switches. A list data structure is maintained in the controller module where the switch-id along with the running average of the number of PacketIn message received is stored. Running average is used since instantaneous average will not reflect the arrival rate of PacketIn messages in the past and focuses only on the present load and this may lead to unnecessary switch migrations. The degree of switch in terms of connections, is an important parameter since more packets will flow through a switch with higher degree. The algorithm calculates the load on the controller after every 100th packet it receives. If the controller load is above the threshold T_c, then the switch migration procedure will be initiated. For this the controller selects a target switch. The controller then collects the load information from its neighbouring controllers. The controller with the least load will be taken as the target controller. Finally the target switch is assigned to the target controller without affecting the ongoing packet transfers in the network as shown in Algorithm 6

Algorithm 6 SwitchMigration(C, S, C_t)

```
1: $S$ sends equal role message to current controller $C$
2: $S$ sends master role request to target controller $C_t$
3: $S$ sends slave role request to $C$
```

4.6 Results and Analysis

The algorithms presented above are evaluated in a simulation environment built using Mininet (min, 2014) for the data plane and FloodLight (flo, 2014) for the control plane. The network topology considered is a jelly fish topology (Singla et al., 2012) with 250 switches and 300 end hosts. Since Mininet does not have a built in traffic generation tool, iperf and ping commands were used to generate traffic. Additionally, a FTP server is run on one host and clients on the other host.

Load on Controller: The load on the controller is measured as the number of PacketIn messages reaching the controller per second. The threshold load is set at 600 PacketIn messages. Figure 4.2, shows that at 70 second, the PacketIn

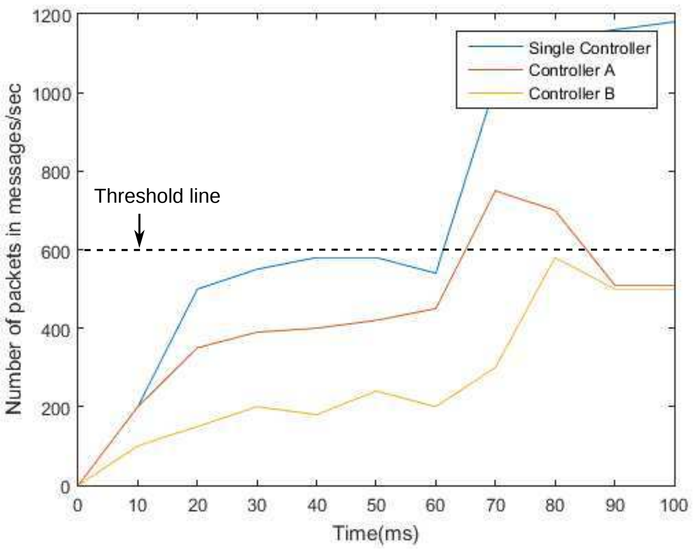


Figure 4.2: Effect of excess load on controllers with and without dynamic load balancing

arrival rate is made to exceed the threshold. In the case of a single controller represented by the blue line, the packet arrival rate remains in this state and this may degrade the controller performance. A multiple controller scenario, consisting of Controller A and controller B, are represented by red and yellow lines respectively. At 70 second the load on controller A exceeds the threshold at which point the migration procedure starts. This procedure transfers some switch from heavily loaded controller A to lightly loaded controller B. This results in the flow request rate at controller A gradually decreasing and the load getting more or less balanced between the two controllers at 90 sec. The switch migration procedure reduces the overloading of the controller gives a stable controller performance.

Response Time: In order to evaluate the algorithms effect on the response time of the controller, it is evaluated against a growing number of network switches. From the experiments it is observed that the response time increases when the number of flows and network elements increase, as is shown in Figure 4.3. It can be seen that the response time of a single controller represented by blue line increased, as the number of switches and hence the traffic in the network increased. This is because an increase in the number of switches and packet flows means, that

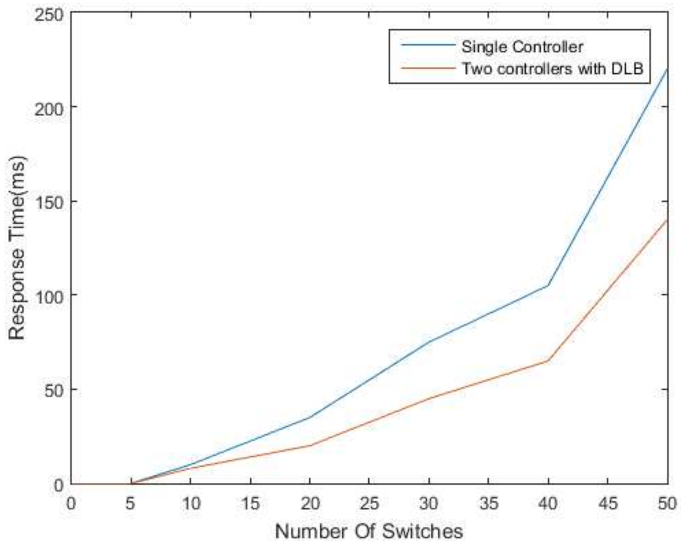


Figure 4.3: Response time of controller with and without dynamic load balancing as network size grows

the controller is busy collecting the network state information and updating the flow tables, leading to degraded response time. The growth rate of response time of controller A in a multi controller scenario, where dynamic load balancing is carried out, is observed to be slower. This is because the load on the controller A is shared with controller B. However, it is not exact half of the value obtained with a single controller, because of the overhead caused by the switch migration procedure and the fact that the load is approximately balanced.

Effect of switch migration: The switch migration requires multiple round of communication between the target switch, and the current and target controllers. Response times of 50 PacketIn messages for controller A, are plotted before and after migration. Figure 4.4 shows the increase in response time just before migration at 60 seconds, after which it stabilizes. This is due to the additional packets exchanged between the controllers and the switches. But no loss of message or duplication was observed. The migration procedure had minimal impact on response time.

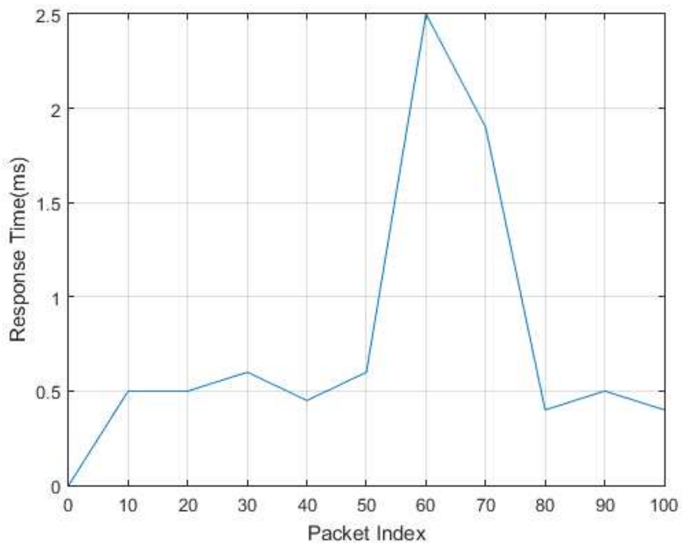


Figure 4.4: Effect of switch migration on response time of controllers

4.7 Summary

This chapter presented the controller placement algorithms for the edge-core SDN architecture. Having two sets of controllers, one each for the edge of the network and the core of the network, was found to be a better design since the the functionality of these two controllers are different. Accordingly, having a different set of metrics for these two domains was also essential. Source routing, wherein the path computation is carried out at the edge, became a good fit for the edge-core SDN. Since source routing was used, the load on the core controller was fairly stable. However, it varied with time at the edge controller. A dynamic load balancing controller placement algorithm with switch migration, so that there was no disruption to the ongoing packet transfers, was also presented. The algorithms were evaluated using the jelly fish topology in Mininet using the FloodLight controller. The results show that the algorithms performed good load balancing between controllers with minimal interruption due to migration. Although source routing proved to be beneficial in this scenario, it can be further harnessed to provide a scalable and fair forwarding of packets in the data plane, as is presented in Chapter 5

Chapter 5

Scalable and Fair Forwarding of Elephant and Mice Traffic

If one ox could not do the job they did not try to grow a bigger ox, but used two oxen. When we need greater computer power, the answer is not to get a bigger computer but to build systems of computers and operate them in parallel.

-Rear Admiral Grace Hopper

This chapter presents a source routing based forwarding mechanism which ensures minimum memory utilization on the switches, making it scalable. It provides equal priority to both elephant and mice traffic. The chapter begins with an analysis of the reasons for choosing source routing as the forwarding mechanism for data packets. Subsequently, the challenges and opportunities in making the flows scalable and fair are presented. This chapter also discusses the use of labels for the purpose of conveying policy information. Use of labels allows routing decisions to be made at the edge of the network, making the network core simple. Based on these observations an elegant solution, which combines source routing with flow splitting is presented. This algorithm splits large elephant flow into smaller mice flows and distributes them on multiple paths. Further, it ensures that latency sensitive mice flows are not slowed down by elephant flows. The proposed solution is analyzed, with regard to flow table memory requirement, processing time on switches and the number of communications between controller and switch. Later in the chapter the experimental set up, results and analysis are presented.

5.1 Overview

SDN controllers contain information about the topology and traffic of the entire network. The controller can hence take centralized decisions about packet routing. It can also react dynamically, to changes in the network traffic and redirect flows accordingly. SDN routing is significantly different from the distributed routing algorithms used in traditional networks, wherein forwarding decisions are taken locally by the switches. Routing algorithms, which are essentially distributed across switches in traditional networks, are now executed in the central controller. Path computations are done, even before the actual flow begins, using the information available at the controller namely; the network topology and traffic status information. In traditional networks, policy enforcement is carried on a switch by switch basis, mostly in the form of configuration changes. However, SDN allows policies enforcement from a central location. Absence of switch by switch configuration reduces the Operational Expenses (OPEX) in networks. Replacing manual configuration by programs reduces the error rate.

Although, SDN has a centralized view of the network and supports network programmability, it faces scalability issues. Figure 5.1 shows an example network, wherein a packet is routed from source host s to destination host d. Since the switches are decoupled from their control planes, when the first packet of a flow arrives at a switch, the switch does not have the necessary information to handle the packet. Hence, the switch has to contact the controller to obtain the forwarding information. Once this forwarding rule is obtained, it can be stored on the switches flow table, to be used by subsequent packets of the same flow.

In order to forward the packets of a flow, five communications, depicted by dotted lines in the Figure 5.1, between the controller and switch, are necessary. Switch s_1 first contacts the controller in order to obtain the flow rules. Since the controller is aware of the entire path the packet has to take, in order to reach the destination d, it establishes contact with the switches S_2, S_3 and S_4, that are along the path of the packet, and proactively installs the forwarding rule. This avoids the switches from having to contact the controller. In spite of these intelligent measures by SDN, the following two issues persist and cause scalability concerns:

(i) Excessive communication between the controller and the switch.

(ii) Space requirement on TCAM based flow table, to store the forwarding rules.

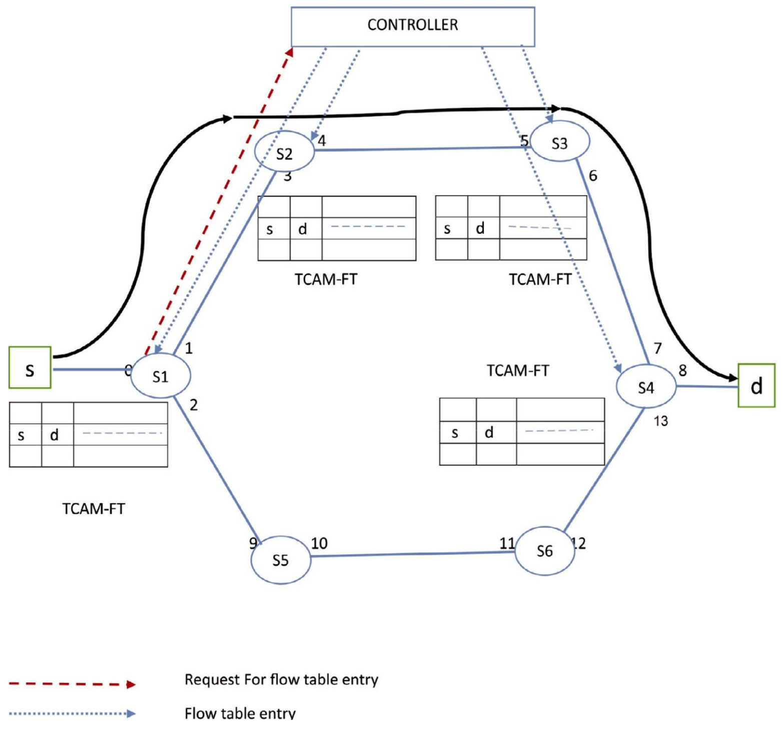


Figure 5.1: Scalability issues in the data plane of SDN

The traffic characteristics of the data center network is significantly different from that of the Internet. In data centers, less than 10% of all flows are elephant flows and the rest are mice flows. However, elephant flows account for more than 80% of the entire traffic volume i.e. majority of flows in the data center, are short, but a large number of the packets belong to a few long lived flows (Benson et al., 2010). Care must be taken to ensure that, the routing algorithm designed for data center, must cater to both latency sensitive mice flow and throughput-sensitive elephant flows. Hence, any routing algorithm must take into account the unique characteristics of data center network topology and traffic patterns. In this work we propose to make routing in SDN scalable and at the same time making it fair for both mice and elephant traffic.

5.2 Scalability in SDN: Destination Based Routing vs. Source Routing

5.2.1 Challenges

Switches, in networks that separate the control plane from the data plane and use destination based forwarding, need to update their flow tables when a new packet arrives. This situation is encountered when the switch receives the first packet of every flow. A new flow table entry is then done by the controller, for every switch along the path of the packet. Such a scenario results in the following scalability issues.

- **R1**: Forwarding table size: Limited size of TCAM based flow tables on the switches.
- **R2**: Forwarding entry size: The size of forwarding entries are individually large, measuring around 15 field tuples, thus requiring approximately 356 bits. (sci).
- **R3**: Scalability of the controller: In order to maintain a centralized view, the controller has to periodically collect information about network statistics. It is further burdened with having to install the flow rules on the switches.

5.2.2 Opportunities

A routing algorithm which stores very little information in the switches and requires minimal communication between the switch and controller is highly desirable. This requirement can be handled by using labels to route packets. Since the entire path, from source to destination, is computed in the controller before the flow begins, the path information can be stored in the packet header and need not be stored as forwarding rules on the switches along the path. This implies that routing decisions are carried out at the edge, while keeping the core network simple. The path information is conveyed only to the ingress switch and not to all the switches along the path. This essentially addresses the above scalability issues **R1** and **R2**, as no information is stored in the switches. In addition it fulfils requirement **R3**, since the controller does not install flow rules in intermediate switches.

Traditionally, source routing has not been widely accepted, due to the following reasons.

- Overhead incurred as a result of embedding path information in the packet.
- The impracticality of having a single server, which has the entire network topology information and can calculate the route accordingly.

These challenges are however, overcome in an SDN enabled data center because,

- The packet overhead incurred is limited, since the average number of hops in a data center is 68 (Benson et al., 2010).
- The central controller is a convenient location to store network information and compute the routes.

An additional advantage is that, source routing speeds up packet forwarding at the switches, since flow table lookups are not required.

5.3 Fairness: Elephant and Mice Flows

5.3.1 Challenges

Elephant flows are large flows, that are generated by operations like virtual machine migration, virtual machine cloning, data backup, large file transfer etc. These flows are throughput sensitive. Mice flows, are generated by applications that use distribute/aggregate functions in their computations. These flows are bursty and latency sensitive. Elephant flows, which are essentially long lived, tend to fill the network buffers, introducing queuing delays to the latency sensitive mice flows, which share the same buffers. Such delays occur inspite of the presence of alternate paths.

Adaptive routing algorithms are ineffective in the presence of mice traffic, as they are bursty and transient, thus denying the algorithms time to act on them. Therefore, routing in data centers often use stateless, hash based multipathing such as Equal Cost MultiPathing (ECMP), which randomly distributes flows across the available equal cost paths. However, using the same approach for elephant flows can cause some paths to be overloaded, as several elephant flows may get hashed onto the same path, while other paths may be lightly loaded.

5.3.2 Opportunities

The above observations led us to design a routing algorithm that caters to both types of flows, by splitting up an elephant flow into several mice flows. This facilitates packet level load balancing, which ensures fairness to both elephant and mice traffic.

5.4 Policy Enforcement

The data center traffic has to follow strict service level agreement with its tenants. This may involve latency constraints, bandwidth constraints, security constraints such as avoiding certain paths, or application level constraints such as permitting certain high priority applications taking high bandwidth paths etc. Network policies can be categorised as: *end point policy* and *routing policy.* This categorization is helpful, as the endpoint policies can be easily converted into flow entries, which can be stored at the edge switches. However, the routing policies have to be encoded in the packet header, since there is no control information exchange between the intermediate switches and the controller. The intermediate switches are to be made intelligent enough to understand this encoded information to achieve network policy implementation.

An example of routing policy, is priority levels of the packet. This information

Table 5.1: Source Routing with Flow Splitting

	Without flow splitting	**With flow splitting**
Without source routing	Not scalable Not fair	Not scalable Fair
With source routing	Scalable Not fair	Scalable Fair

is used by the intermediate switches, to decide which packets are to be dropped when a switch buffer is full. The packets marked as low priority may be dropped. Hence, we find that most of the routing policies can be handled at the edge, but some policies have to be implemented at the switch and this policy information must therefore be embedded in the packet itself.

5.5 Solution Methodology

Source routing makes SDN scalable, while splitting elephant flows ensures fair forwarding of both elephant and mice traffic. Together, they give an elegant solution for scalable and fair routing.

When flow splitting is used with destination based routing, packets of a flow are forwarded along multiple paths. This requires forwarding rules to be stored on all switches, along all the multiple paths, for each flow. This increases the switch TCAM utilization, causing a serious scalability concern. On the other hand, if flow splitting is used with source routing, the forwarding rules need not be stored on the switches and the overhead, in terms of flow table entries, remains a constant, irrespective of the path chosen. Flow splitting may lead to packet reordering at end nodes. This is not a major concern if symmetric topologies, like the ones widely deployed in data centers, are used. Table 5.1 shows how flow splitting and source routing work together, facilitating scalable and fair routing.

The framework that is presented here, addresses the data plane scalability and fairness challenges of SDN.

5.5.1 Policy Database and Proactive Path Computation

As described in previous section, network policies can be divided into endpoint policy and routing policy. Endpoint policies can be handled by the edge switches. Routing policies on the other hand concern intermediate switches. Hence the routing policy information has to be encoded in the packet headers.

Overhead of header The Overhead O of the header, is defined as a measure of how much additional information needs to be stored in the header per hop, when source routing is used.

$$O(h) = \frac{B(h)}{H(h)} \tag{5.1}$$

$B(h)$ is the number of additional bits used in the header to encode the information about the policy compliant path P that the packet has to take.
$H(h)$ is the number of hops the packet with header h has to take from source s to destination d i.e. the length of the path

Traditional $L1$ and $L2$ headers are not used in source routed SDN core network. However, present day SDN still maintain these headers. Hence, these already existing headers, like the Mac header, can be rewritten with the source route information, instead of introducing an additional header for the same. This implies that in such cases, the $B(h)$ in Equation 5.1 is 0 and we get a zero overhead.

All source to destination paths are proactively computed and these paths are compliant with the user defined end point policies that are stored in the policy database. The path information may be stored as a sequence of switches-ids of the switches along a packets path. It can also be stored as a sequence of output ports, through which the packet leaves the switches in order to reach the destination. In the later case, the number of ports on a switch is constant whereas the number of switches in a network can vary over time. Also, the number of ports is much lesser than the number of switches in the network. The output port number needs to be unique only to a switch, whereas a switch identification must be unique across the entire network. Therefore, output port numbers are a better choice than the switch identification, for specifying the path information as is shown in Equation 5.3.

The network model The data center network is modeled as an undirected graph $G = (S, E)$, where each node s_i in the vertex set $S = \{s_1, s_2, \ldots, s_n\}$ represents a core physical switch and $n = |S|$ is the total number of nodes; the edge set E represents the links between these switches.
Each switch s_i of S has a set of ports

$$PO_i = \{po_{i,0}, po_{i,1}, \ldots po_{i,n}\} \tag{5.2}$$

RC is the route controller of the SDN network G under consideration.

The policy compliant path

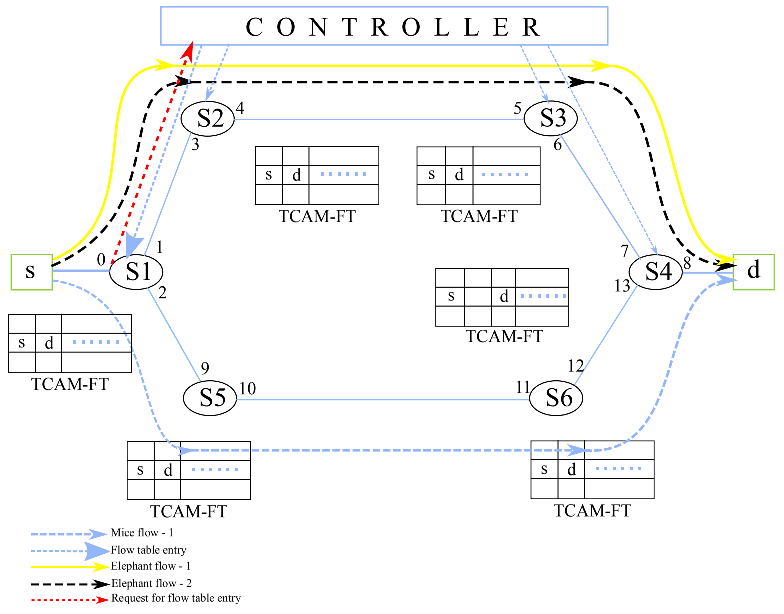
Given the network G, the policy R; the path P between the source and destination pair (s, d), we formally define the encoded path in the packet h as follows.

$$P(s, d) = \{p_{s,a}, \ldots, p_{d,b}\} \tag{5.3}$$

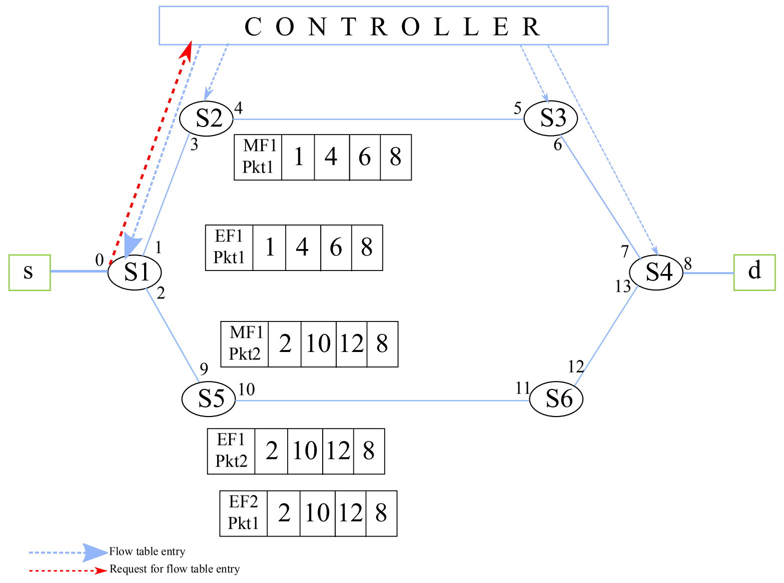
such that $O(h)$ is minimized.
Here $p_{x,y}$ gives the output port y of switch x that the packet has to traverse along path P

5.5.2 Obtaining the Path from Controller

When an edge switch encounters the first packet of a new flow, it checks its local cache, to see if path information is stored for this source destination pair, with the given policy requirement. If such an entry is not cached, then the packet is sent to the controller. The controller responds with k equal cost, policy compliant paths to the edge switch. The edge switch caches these routes. Cache hit is likely to be high because, in a data center, most of the flows are destined to a few destinations.



(a) Using flow tables



(b) Using source routing and flow splitting

Figure 5.2: Comparison of forwarding techniques

5.5.3 Flow Splitting

The edge switch embeds the packet with one of the k paths, chosen in a round robin manner. This approach is essentially turning the elephant flows into mice flows and performing per-packet forwarding. By splitting the flow across available k paths, it ensures that long flows do not fill the network buffers along a single path and adversely affect the latency sensitive short flows, while also ensuring a high throughput.

Figure 5.2 compares flow table based forwarding in SDN, with the proposed flow splitting based source routing. Figure 5.2a shows the initial architecture where flow table based forwarding is used. Here, flows are hashed onto different paths. This incurs an additional overhead as a result of storing forwarding rules for the multiple paths. This approach is neither scalable nor fair, as two elephant flows are hashed onto a single path. Figure 5.2b shows the proposed architecture, which uses source routing with flow splitting. It can be observed that source routing does not require an update in the flow tables of intermediate switches. Flow splitting ensures fair bandwidth utilization of multiple paths with no additional overhead on memory.

5.5.4 Forwarding by Intermediate Switches

The intermediate switches forward the packet by looking at only the forwarding port number embedded in the packet. This approach solves the scalability problem that has been encountered with destination based forwarding because, no information is stored in the switches and the controller does not have to push forwarding information to the intermediate switches. This keeps the core switches simple and they can operate fast. The reverse path is not computed at each switch, while the packet is being forwarded, as is the case in some previous work (Soliman et al., 2012), for two reasons. The first reason being that, the all-to-all paths have already been proactively computed and second reason being that this slows down the forwarding process. The aim here is to keep the core switches as simple as possible. Once the packet reaches the egress switch, it is overwritten with the original Mac header and sent to the destination host.

5.6 Analysis

This section presents an analyses of source routing with flow splitting in SDN environments, with respect to the flow table memory requirements, the processing time on intermediate switches and the number of communications between controller and switches. We compare it with traditional routing and source routing.

Let n be the number of switches in the network, p be the path length described as hop count between a source s and destination d, let f be the size of forwarding rules, k the number of equal cost multiple paths, t the time required to lookup the flow table, r the time required to forward a packet at a switch and pl the packet header look up time.

5.6.1 Flow Table Memory Utilization

Traditional routing requires that the flow rules be stored on all the switches along the path of the flow. Therefore, the memory required on the flow tables for a flow between a source s and destination d is given as

$$M_FT(s,d) = p \cdot f \tag{5.4}$$

For multipath traditional routing the memory requirement is

$$M_FT(s,d) = p \cdot f \cdot k \tag{5.5}$$

In contrast source routing does not utilize the flow table memory, since it does not require flow rules to be stored.

5.6.2 Total Processing Time on Intermediate Switches

In traditional routing, at each switch, the packet header has to be looked up, followed by flow table look up, in order to find matching flow rule. This is followed by the actual packet forwarding. This is repeated on all the intermediate switches. Hence, the processing time on intermediate switches for a flow from source s to destination d is given as

$$PT(s, d) = pl \cdot t \cdot r \cdot p \tag{5.6}$$

On the other hand, source routing with flow splitting do not require a flow table lookup, since the path information is stored in the packet itself. Hence, processing time on intermediate switches is given as

$$PT(s, d) = pl \cdot r \cdot p \tag{5.7}$$

5.6.3 Number of Communications between Controller and Switch

In traditional routing, the controller must insert the forwarding rule on all the switches along the path of the packet. Hence, the number of controller to switch communications is

$$C(s, d) = p \tag{5.8}$$

In source routing, only two communications take place i.e., between the controller and the ingress and egress switches. In source routing, with flow splitting across multiple paths, the number of updates is two for every path, hence

$$C(s, d) = k \cdot 2 \tag{5.9}$$

5.7 Implementation and results

This section presents the performance evaluation of the proposed routing with respect to traditional SDN routing. The data plane of the SDN network has been emulated, using the Mininet (min, 2014) emulator. Floodlight (flo, 2014) is used as the SDN controller. A synthetic topology with 256 nodes and 30 hosts is built and iperf is used to generate the traffic.

In order to evaluate the number of communications between the controller and the switches, the reduction in the number of forwarding rules or flow mods, distributed by the controller is measured. This is done for various flow path lengths. In order to do this, several flows are generated, that each have a different path length between their source and destination nodes. In traditional SDN, when a new flow is encountered by the controller, all the switches along the flow path have to be notified with the appropriate flow entries by the controller. However, with the proposed implementation of source routing in SDN as described in Section 5.5.1, it can be seen that only the edge switches are required to be notified. The required number of flow mods remains at two for only these end nodes, irrespective of the path length. Therefore, we see a reduction in the number of flow modes for the intermediate switches. Figure 5.3 gives the percentage decrease in flow mods. It can be observed that when path length is two, there is no reduction since both, destination based and source routing, require two flow mods. But, as the path length increases, the destination based forwarding requires flow mods proportional to the number of intermediate switches.

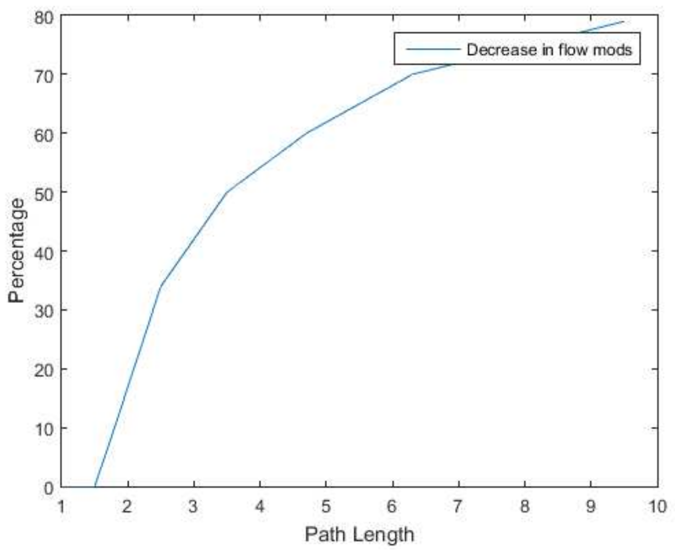


Figure 5.3: Percentage reduction in number of flow entries in the switches

Forwarding of packets by the switches is faster with source routing, since flow table look ups are not required. Instead forwarding action on the packet can be taken by referring the header entries itself. HTTP files of varying size are transferred between two hosts, with a path length of eleven between them. Figure 5.4 shows the impact of source routing on the delay experienced by the flow. Initially due to TCP bursts, the small files are transferred very fast and not much difference can be observed in those cases. However, as file size increases beyond 500MB, i.e. the number of packet look ups increase, change in delay can be clearly seen and is maintained throughout. This is because for large files, the number of time consuming table look ups increases in traditional SDN. However, in the proposed approach table lookups are not required at all.

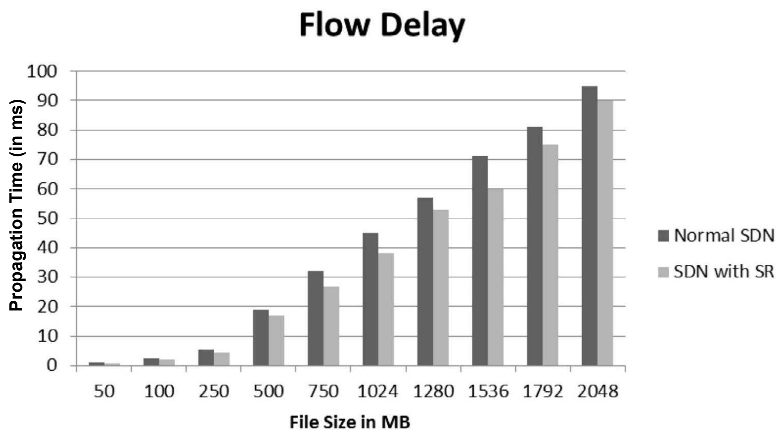


Figure 5.4: Delay experienced by the packet flow

Although decrease in delay is observed by the flows in the presence of source routing, the edge switches see a decrease in the performance due to increase in the number of actions to be carried out per flow as discussed in Section 5.5.2, at these switches. Figure 5.5 shows the decrease in throughput at the edge switches, when source routing is used.

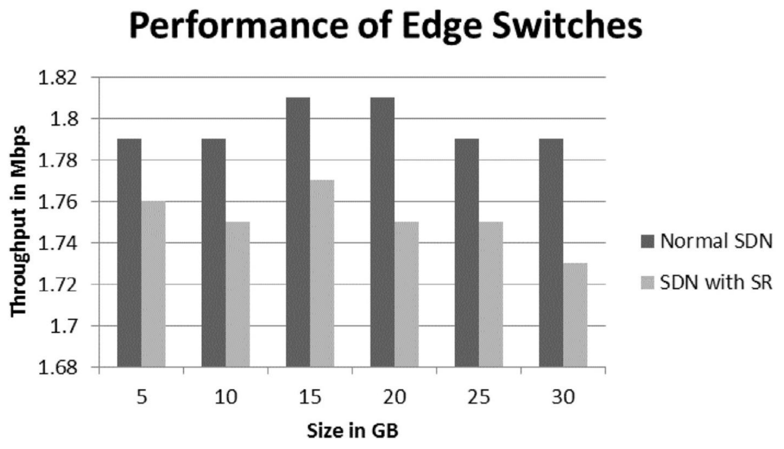


Figure 5.5: Performance of edge switch

In order to evaluate the efficiency of the algorithm, the time taken by the controller to set up a flow under constant minimum workload is measured. The time taken in the destination based SDN to push all the flow mods and set up the flow increases, as the number of flow mods to be pushed increases with path length. On the other hand, in source routed SDN, the length of the path to be encoded in the packet increases with the increase in flow path length. Since the encoding work is pure CPU based and the input output work remains the same, the increase of flow setup time with path length is comparatively lower. This can be seen in Figure 5.6. Only in the case of the path length being two, the time taken is more than the previous case, since more number of actions is to be inserted in the edge switch flow mods.

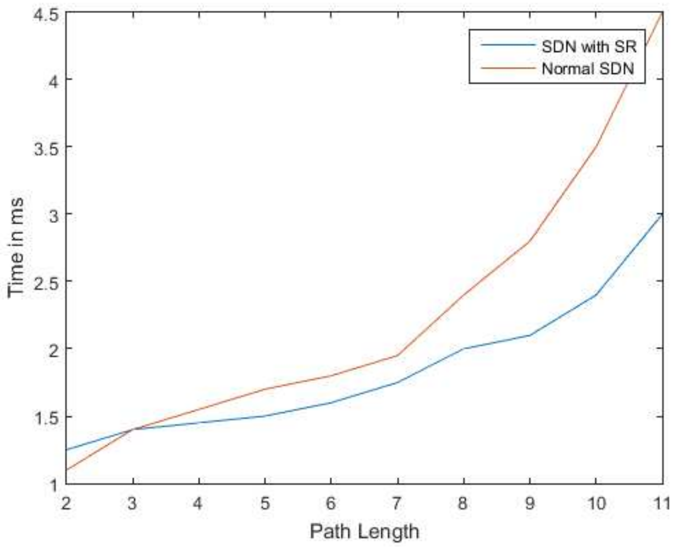


Figure 5.6: Time taken to set up flow

Next the controller throughput is compared as the path lengths of the flows are varied. From Figure 5.6 it is seen that the traditional controller represented by the red line takes more time to set up flows as the path length increases. This is because the number of intermediate switches increases, meaning an increase in the number of flow table entries. SDN controllers also take more time to compute the path, but not as much as traditional SDN controllers. A similar effect can be seen on the controller throughput as well. Figure 5.7 shows the decrease in controller throughput as the path length decreases. This is because, under traditional SDN routing, the intermediate switches are to be programmed by the controller for each and every control decision to be implemented as discussed in Section 5.5.1. Additional controller resources are used in updating the switches regarding route and policy information in traditional controllers. In order to evaluate the overhead incurred, the number of flow entries required and also the number of bytes required to encode the path and policy information, is measured. From Figure 5.8, it can be seen that, to implement various routing policies in traditional routing, equal number of flow entries are to be pushed to each intermediate switch. However, with source routing, no flow entries are put on the switches. The path and policy information are carried in the packet header itself.

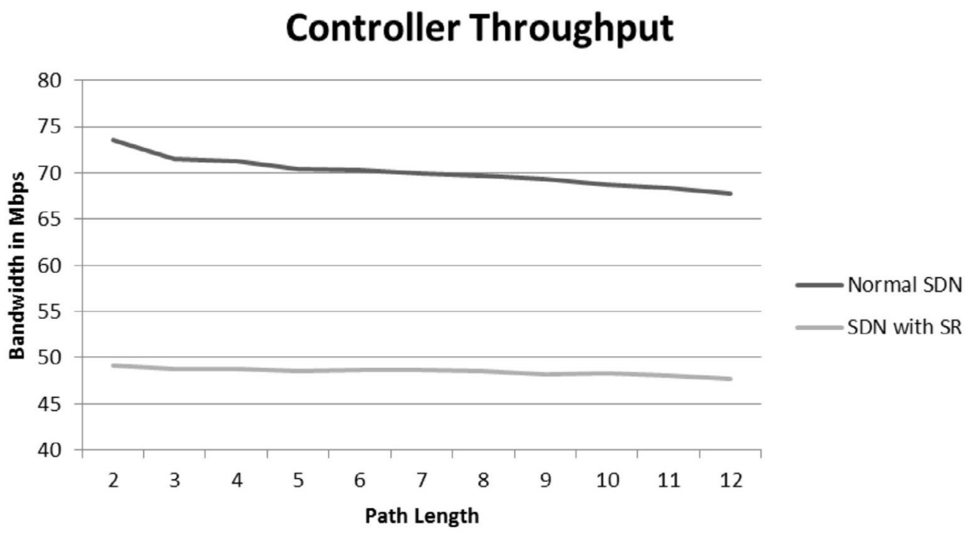


Figure 5.7: Performance of the controller

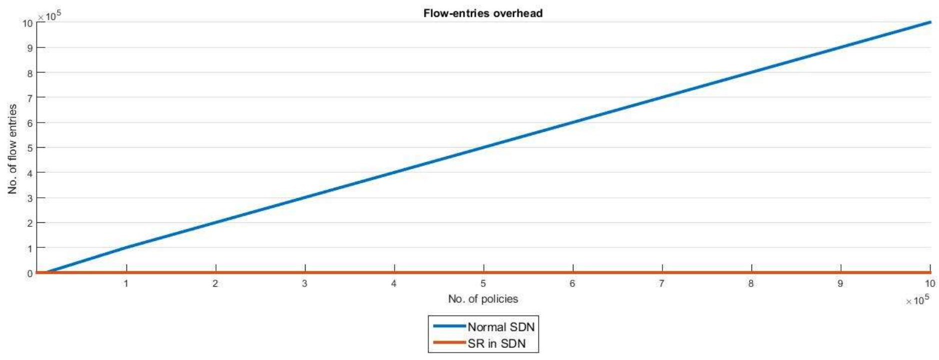


Figure 5.8: Number of flow entries required on the switches

Although, source routing shows a decrease in the flow entries required to implement the policies, it consumes a few bytes in the packet header to achieve the same. From Figure 5.9 it can be seen that, in the worst case, the number of bytes required to implement a single policy increases linearly with path length.

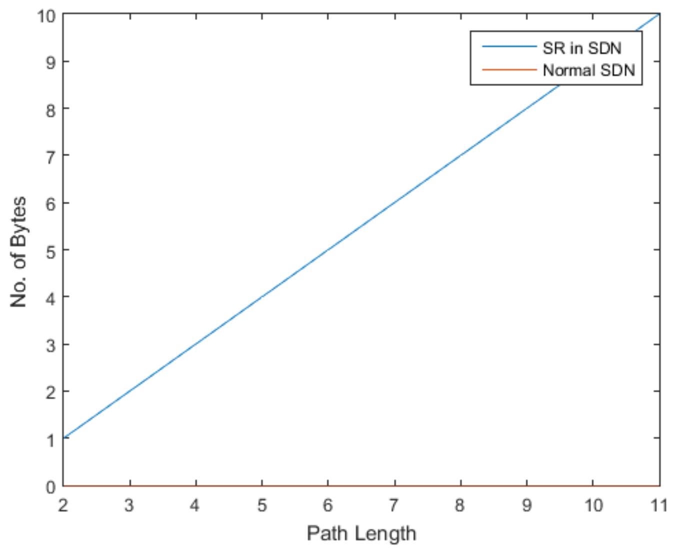


Figure 5.9: Overhead due to inclusion of policy

5.8 Summary

SDN that used destination based forwarding, faced the problems of limited flow table size and the additional communication between switch and controller that was required to disseminate forwarding information to switches. It can also have major design issues if intermediate switches had to identify and handle short and long flows differently. When the routing algorithm used labels, to encode path and policy information and perform routing decisions at the edge, it was essentially separating the edge and core functionality and making the network data plane scalable. It also followed the end to end principle in general. In this regard this chapter presented a mechanism to ensure fairness to both elephant and mice flow. Chapter 6 proposes various techniques of path restoration for source routing in the event of a failure.

Chapter 6

Path Restoration in Source Routed SDN

The work of restoration cannot begin until a problem is fully faced.

-Dan B Allender

6.1 Overview

Software defined networks have a central controller and central view of the network, allowing for source routing to be used as a scalable routing technique instead of the traditional destination based forwarding. However, with source routing the switches are reduced to simple forwarding devices, incapable of finding alternate paths in the event of link failures. In this chapter we look at techniques to provide resiliency when packets are in transit and a network link failure occurs. Path restoration is one such mechanism, where a bypass path for the failed link is used. Such bypass paths are stored locally on each of the switches, for all of its outgoing links. This mechanism ensures that the recovery mechanism is scalable, since it avoids contacting the controller and takes local corrective measures. Two approaches are proposed here, for storing the bypass paths. In the first method, the bypass path is stored between all pairs of nodes. In the second method the bypass path is stored between few selected nodes. These node are the chosen two hop neighbours, using either the two colourable graph approach or the vertex cover approach. The analysis shows that the second method; using the vertex cover approach reduces the total number of bypass paths stored, without compromising the resiliency. Also the solutions are topology dependent and not path dependent, allowing for most of the computations to be done proactively.

6.2 Link Failure and Source Routing

A basic requirement of SDN, deployed in data center networks, is high availability and resilience to failures. In order to cater to this the network must remain connected in the event of networking element failure and packet forwarding must be carried out with as little disruption as possible. As presented in the previous section, source routing, in software defined data center networks, is scalable in terms of both, the packet overhead and switch memory requirement. One of the issues with source routing is the fate of packets in transit when a link failure occurs. This is because unlike destination based forwarding where the switches learn the alternate paths on reconvergence, in source routing the switches are dumb forwarding devices which simply forward packets as per the next hop information in the packet. Hence, for source routing to be a viable solution, the issue of packet forwarding, in the event of link failure, must be addressed.

There are two main techniques by which networks can handle packet forwarding when a link failure occurs, as follows:

I Path protection: Resend the packet from the source on an alternate, link disjoint path. The packet which has been in transit is dropped.

II Path restoration: Continue to forward the packet in transit, without dropping it, via a path which bypasses the failed link, to a subsequent node along the path.

In this work the second technique is explored. The information about the path which bypasses the broken link can be obtained using either of the following:

i The switch which detects one of its adjacent link has failed, contacts the controller and gets the bypass path ii The switch stores the bypass paths for all of its neighbours in its local memory

The problem in method (i) is that, it takes time to contact the controller and also places extra load on the controller and therefore is not a scalable option. On the other hand method (ii) requires storage space on the memory constrained switches. However, the second option of path restoration is faster and a solution based on it is proposed and evaluated in the following sections. Also simultaneous multiple link failures are not considered.

The following requirement for resilient source routing is identified

R1 In the event of link failure, packets in transit are re-routed correctly without having to retransmit them i.e. ensure path restoration.

R2 Take local corrective measures so as to minimize the communications between the switch and the controller.

R3 Minimize the average memory requirement of the switches in the network.

R4 The solution that we propose must be topology dependent and not path dependent

6.2.1 The Network Model

A data center network is modeled as an undirected graph $G = (V, E)$ where each node v_i in the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents a physical switch, $n = |V|$ is the total number of switches and the edge set E represents the links between the switches. Two vertices which are incident with a common edge represent adjacent switches.

6.2.2 Notations

Shortest Path: Given a graph $G(V, E)$ let $P(s, d)$ the shortest path from the source s to the destination d be a vertex set

$$P(s, d) = \{v_1, v_2, \ldots, v_m\} \tag{6.1}$$

where v_i is the i^{th} node in the path and m is the total number of nodes in the path.

Second Shortest Path: Given a graph $G(V, E)$ let $SP(s, d)$ the second shortest path from the source s to the destination d be a vertex set

$$SP(s, d) = \{v_1, v_2, \ldots, v_l\} \tag{6.2}$$

where v_i is the i^{th} node in the path and l is the total number of nodes in the path.

One Hop Neighbour Set: The one hop neighbour set of a node v_i be defined as

$$ON(i) = \{v_1, v_2, \ldots, v_d\} \tag{6.3}$$

where $v_k \in ON(i)$ is an adjacent node of v_i and $d = |ON(i)|$ is the degree of the node v_i.

Two Hop Neighbour Set: The two hop neighbour set of a node v_i be defined as where $v_k \in TN(i)$ is a node at a distance of two hop from v_i and $t = |TN(i)|$ is the total number of nodes at a distance of two hop from v_i.

$$TN(i) = \{v_1, v_2, \ldots, v_t\} \tag{6.4}$$

Vertex Cover Neighbour Set: The vertex cover neighbour set of a node v_i be defined as

$$VN(i) = \{v_1, v_2, \ldots, v_c\} \tag{6.5}$$

where $v_k \in VN(i)$ is a vertex cover node reachable from v_i without crossing another vertex cover node and $c = |VN(i)|$ is the total number of such neighbouring vertex cover nodes of v_i.

One Hop Bypass Path: Given a node v_i and its one hop neighbour $v_k \in ON\left(i\right)$ then the one hop bypass path $OB\left(i,k\right)$ is given by

$$OB\left(i,k\right) = SP\left(i,k\right) \tag{6.6}$$

where SP is the second shortest path from i to k i.e. the shortest path from i to k without considering the direct link.

Two Hop Bypass Path: Given a node v_i and its two hop neighbour $v_k \in TN\left(i\right)$ then the two hop bypass path $TB\left(i,k\right)$ is given by

$$TB\left(i,k\right) = SP\left(i,k\right) \tag{6.7}$$

where SP is the second shortest path from i to k i.e. the shortest path from i to k without considering the direct two hop link.

Vertex Cover Bypass Path: Given a node v_i and its one hop neighbour $v_k \in VN\left(i\right)$ then the vertex cover bypass path $VB\left(i,k\right)$ is given by

$$VB\left(i,k\right) = SP\left(i,k\right) \tag{6.8}$$

where SP is the second shortest path from i to k i.e. the shortest path from i to k without considering the shortest link to the vertex cover neighbour

6.2.3 Problem Definition

Given an edge l between two nodes v_i and v_j as $v_i <-> v_j$ along the path $P(s,d)$ has failed, the problem is defined as finding an alternate path between (s,d) which bypasses the link l without contacting the controller. This alternate path is used to forward the packets that have already left the egress switch before the controller can update the new path information at the egress switch. The solution must also minimize the amount of information stored in the switches.

6.3 Path Restoration

In order to meet the requirements 1 and 2, set in the previous section, the bypass path has to be stored in the intermediate switches, in order to enable local corrective measures that can be taken. The challenge is to use as little of the switches memory as possible and to ensure that these bypass paths are topology dependent and not path dependent as per requirement 3 and 4.

We propose three approaches to solve this problem. In first approach the bypass path information is stored on all switches. The information stored is the bypass path to all its neighbouring switches. In the second approach the bypass information is stored on only the two hop distance switches. The information stored is the bypass path to the two hop neighbour switches only. In third approach the bypass path information is stored only on the vertex cover nodes. The information stored is the bypass path to only the neighbouring vertex cover nodes.

6.3.1 Bypass Path to One Hop Neighbour

Consider the example topology shown in Figure 6.1. Here the bypass routes stored in switch a are:

1. $<a, c, d, b>$ to the adjacent node b if link ab fails
2. $<a, b, d, c>$ for the adjacent node c if link ac fails

Node a threfore stores two entries which are the same as its degree.

Every node $v_i \in V[G]$ stores the *one hop bypass path* $OB(i, k)$ for all $k \in ON(i)$.

Packet arriving at switch Consider a packet is at node v_i on its path from source node s to destination node d. Consider the edge v_i to v_k is down as a result

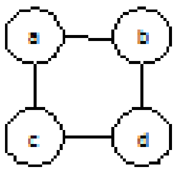
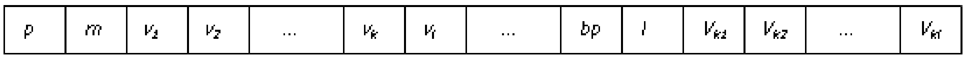


Figure 6.1: Bypass route to one hop neighbour example

of link failure. In such a case the bypass path $OB(i, k)$ stored at v_i is copied to the packet header, and may be used for source routing.

Figure 6.2 shows the header format. $v_1, \ldots v_k, v_l, \ldots$ represents the hop information along the primary path and m its length. $v_{k1}, v_{k2}, \ldots$ represents the hop information along the bypass path and l its length. p is the pointer to the next hop information along the primary path. sp is the pointer to the next hop information along the bypass path. m is decremented by, every time the packet makes a hop along the primary path and a value of 0 indicates the packet has reached the destination. l is decremented by one, every time the packet makes a hop along the bypass path and a value of 0 means the packet has taken the bypass and is now back on the primary path.



p	m	v_1	v_2	...	v_k	v_l	...	bp	l	V_{k1}	V_{k2}	...	V_{kl}

Figure 6.2: Header format with bypass route

This ensures the packet reaches node k via the bypass path. Thereafter, it follows the original path embedded in the packet from node k to destination node d. The controller eventually notes that the link is down and identify an alternate path between s and d. For packets yet to be dispatched, this alternate path is used.

6.3.2 Bypass Path to Two Hop Neighbour

In order to reduce the total number of bypass path entries stored on the switches, we explore the idea of storing bypass path between only a few select nodes. This would imply that the packet may have to crank back along its path before it takes a bypass path. However, if the number of hops the packet would have to crank back is bounded, then efficiency is maintained while we reducing the number of bypass paths stored.

Consider the example topology shown in Figure 6.3. Here we choose only the two hop neighbours to store the bypass path information between them. We represent them as black nodes a, c, e.The second shortest path between a and c is represented as $bp1$ and the second shortest path between c and e is represented as $bp2$. The bypass path information stored on node a:

1. $<a, bp1, c>$ to the two hop neighbour c if link ab or ac fails

The bypass path information stored on node b:

1. $<c, bp1, a>$ to the two hop neighbour a if link cb or ba fails
2. $<c, bp2, e>$ to the two hop neighbour e if link cd or de fails

The bypass path information stored on node e:

1. $<e, bp2, c>$ to the two hop neighbour c if link ed or dc fails

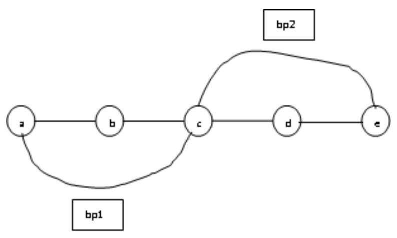


Figure 6.3: Bypass route to two hop neighbour example

If the link coming after the black node along the path fails then the packets can directly take the bypass path to the two hop neighbour along the path. However, if the link is not preceded by the black node along the path fails then the packet has to crank back to the black node and then take the bypass path. For example in Figure 6.3, for a packet at node a along the path a to e if link ab fails, then the packet can take the bypass path $<a, bp, c>$ to node c. However, if the packet is at node b, and link bc fails, the packet may have to crank back to node a and then take the bypass path $<a, bp, c>$ to node c, since node b does not store any bypass paths.

Identifying the two hop neighbours, in a graph, is essentially identifying if a graph is two colourable. If it is, then we choose all the nodes with one colour as the two hop neighbours Thus identifying these two hop neighbours is topology dependent and not path dependent, and this can be done proactively by the SDN controller.

The above approach involves carrying out the following steps proactively by the SDN controller

1. Identify if a graph is two colourable.
2. If step 1 is true, label the nodes of the graph with colours black and white.
3. Store the bypass path information only between the black nodes which are two hop neighbours

6.4 Analysis

Source routing does not burden the switch by storing forwarding rules in it. However, as discussed in previous sections, in order to introduce resiliency the bypass path information is stored locally in the switches. In this section the overhead, due to information stored, is analyzed.

6.4.1 Bypass Path to One Hop Neighbour

Switch Memory

The *one hop bypass paths* on all the nodes for each of its neighbours is stored. This memory requirement across the entire network $MOB\left(G\right)$ is given by

$$MOB\left(G\right) = \sum_{i=1}^{n} \sum_{k=1}^{d} \left|OB\left(i,k\right)\right| + 1 \tag{6.9}$$

where $|OB\left(i,k\right)| + 1$ gives the bypass path length in terms of number of hops. Since we consider a software defined data center network, the bypass path has a minimum length of 2 hops and a maximum length of 8 hops. Hence

$$2 * n * d \leq MOB\,(G) \leq 8 * n * d \tag{6.10}$$

Packet Header Space

Source routing introduces an overhead on each packet since it has the path embedded in it. This overhead is increased when fault tolerance is provided, as discussed in the proposed technique. However, this technique shows that this overhead is minimal.

This header requirement on the packets $POB\,(s, d)$:

$$4 \leq POB\,(s, d) \leq 16 \tag{6.11}$$

since now the packet has to carry both the shortest path from s to d and the bypass path from i to k

Latency

The overall latency of source routing is reduced because of two reasons.

1. The controller does not have to install the forwarding rule on all the switches along the path, rather it passes the path information to the ingress switch so that it can replace the MAC header of packet. It later contacts the egress switch in order to restore the MAC header.

2. The forwarding itself is faster, as the switch can forward packet just based on header content and it does not have to do a flow table lookup.

The latency for packet forwarding can be analyzed as follows: Given that m is the number of nodes in the path between a source s and destination d, p is the probability of one link failing along the path, l is the length of the bypass path across the broken link, t is the time it takes to extract and act upon the next hop information from the packet and u is the time it takes to copy the bypass route to the packet header from the bypass table in the switch, then the latency $L\,(s, d)$

is given as follows

$$\begin{aligned} L\left(s,d\right) &= p*\left(\left(m-1+l\right)*t+u\right)+\left(1-p\right)*m*t \\ &= p*\left(\left(l-1\right)*t+u\right)+m*t \end{aligned} \tag{6.12}$$

6.4.2 Bypass Path to Two Hop Neighbour

Switch Memory

The *two hop bypass paths* are stored on the black nodes for each of its neighbours This memory requirement across the entire network can be analyzed as follows

If E_w and E_b give the number of two hop bypass path entries which are stored in the white and black nodes respectively, then the total number of entries E in the network is given by

$$\begin{aligned} E &= E_w + E_b \\ &= 0 + E_b \end{aligned} \tag{6.13}$$

as no entries are stored on the white nodes. The number of black nodes in the graph G with n nodes is

$$N_b = v_1, v_2, \ldots, v_{\frac{n}{2}} \tag{6.14}$$

Similarly the number of white nodes is

$$N_w = v_1, v_2, \ldots, v_{\frac{n}{2}} \tag{6.15}$$

The number of two hop neighbours of a black node v_i, $TN\left(i\right)$ is dependent on the number of one hop neighbours or degree of v_i's white node neighbours.

$$TN\left(i\right) = ON\left(j\right) exceptv_i \forall v_j \in ON\left(i\right) \tag{6.16}$$

$$t = \sum_{j=1}^{d} \left|ON\left(j\right)\right| - 1 \forall j \in ON\left(i\right) \tag{6.17}$$

where t is the number of entries in black node.

Equation 6.17 implies that the number of entries in a black node is directly proportional to the degree of its white neighbour nodes in $|ON(j)|$ or degree of j. This in turn implies that the number of bypass entries on a black node are also directly proportional to the degree of its white neighbour nodes.

The total number of entries in the network is given by

$$E = E_b = \sum_{k=1}^{\frac{n}{2}} t \forall k \in N_b \tag{6.18}$$

Hence, the number of bypass entries reduces if the nodes with lower degree are chosen as white nodes. However, this cannot be ensured with two color algorithms.

The packet header space and latency experienced by the packet is similar to that of one hop bypass path method which has been discussed in previous section.

From the above observation, choosing high degree nodes as black nodes, reduces the number of entries in the switches. Therefore, the chosen minimum vertex cover nodes are identified as the black nodes, because minimum vertex cover nodes are generally, nodes of high degree.

So, instead of using $TN(i)$ in Equation 6.16, $VN(i)$ is used as given in Equation 6.5.

6.5 Summary

In this chapter a solution to ensure resilient source routing in software defined networks was proposed. The solution aimed to provide a backup path for packets in transit when a link failure occurred The aim was been to enable the switches to take local corrective measures without having to consult the controller. Also, it was necessary to ensure that the solution was scalable with respect to the switch memory space and the packet overhead. The proposed approach catered to these requirements by storing a bypass path for the failed link at the switches. The number of such bypass paths was limited by the number of neighboring switches and the bypass path length itself was limited to 6 - 8 hops. Chapter 7 presents a detailed conclusion and possible future work stemming out of this research work.

Chapter 7

Conclusion and Future Work

The end of a melody is not its goal: but nonetheless, had the melody not reached its end it would not have reached its goal either.

-Friedrich Nietzsche

7.1 Conclusion

In this work, a scalable and resilient edge-core SDN framework was presented. As the number of networking elements and traffic flows increased, a single centralized controller could neither manage the large network, nor handle the diverse functional requirements of the core and edge switches. Therefore, there was a need for multiple controllers in large SDN deployments. Distributed controller architecture was recommended to resolve scalability issues. The model considered in this work had an edge controller that was responsible for routing packets and a core controller responsible for collecting network topology information. The control plane, at both the core and edge, had been further divided into multiple domains, each managed by a controller. This research proposed controller placement algorithms for both the edge and the core networks, in the edge-core SDN model.

The core controllers are placed in such a way that, the network was not partitioned as a result of controller failure, by using a novel placement metric called subgraph-survivability. The controller placement algorithm used this metric to ensure that the controller domains are biconnected. This guaranteed that the failure of a controller did not disconnect the rest of the network, although its own domain was rendered non functional. The correctness of this algorithm was shown using formal proofs. The algorithm was evaluated against the random placement algorithm, using the performance metrics Average Inverse Shortest Path Length and Network Disconnectedness. These algorithms are run on real topologies and also some random topologies. The performance metrics of Average Inverse Shortest Path Length (AISPL) and Network Disconnectedness (ND) were used to evaluate our placement algorithms. An improvement of 55.88% for the AISPL metric and 49.22% for ND metric, was observed with our proposed algorithm as compared to the random controller placement. The results obtained show that the subgraph-survivable controller placement algorithm performs better, especially when considering large networks. The work that is presented here, is significant both in theory and practice. Theoretically an algorithm has been proposed to obtain k subgraphs of maximum size m such that k is minimal and there exists biconnectivity between the subgraphs. In practice this solution ensures scalability in the multi controller SDN.

The edge controllers were placed such that they could efficiently handle the changing network traffic loads. A dynamic load balancing controller placement algorithm was developed, that identifies the switch which is to be migrated and also the controller to which it is to be migrated. A switch migration procedure followed such that there was no disruption to the ongoing packet transfers. Although an increase of 2ms in the response time, of the Packet In messages at the time of migration was observed, no packet losses were incurred. This work showed that it was ideal to have two sets of controllers with unique controller placement metrics, one for the edge network and the other for the core network, because of the difference in the controller functionality.

On the data plane, it was observed that, destination based forwarding does not scale well because of the increased number of switch to controller communications, limited size of flow tables and increased size of flow table entries in the switches. Therefore, in this work labels were used within packets to convey control information regarding path and policy. This made the core of the network simple; while all routing and policy decisions could be taken at the edge. Further, the routing algorithms split the elephant traffic into mice traffics and distributed them across multiple paths. The entire mechanism used here was topology independent. With our source routing mechanism we observe a reduction in the number of flow table entries and the flow set up time that is proportional to the number of hops along the path of the packet.

Source routing made the data plane scalable, however, the switches were reduced to simple forwarding devices, incapable of finding alternate paths in the event of link failures. The solution presented in this work aimed to provide a backup path for packets in transit, when a link failure occurred The aim was to enable the switches to take local corrective measures without having to consult the controller. Also, there was a need to ensure that the solution was scalable with respect to the switch memory space and the packet overhead. The proposed approach catered to these requirements by storing a bypass path, for the failed link, at the switches. Two methods were presented for storing the bypass paths. In the first method the bypass path was stored for all pairs of nodes. In the second method the bypass paths were stored for only a few selected nodes. These node are the two hop neighbours, chosen using either the two colorable graph algorithm or the vertex cover algorithm. The analysis showed that the second method, using the vertex cover approach reduced the total number of bypass paths to be stored, without compromising the resiliency. Also, the solutions presented are topology dependent and not path dependent, allowing for most of the computations to be carried out proactively.

Through this work, it was observe that, separating the edge and core of the SDN network makes the control plane scalable and also allowed for routing decisions to be carried out at the edge, thereby enforcing the end to end networking principle. Intelligent placement of multiple controllers made the control plane scalable and also resilient to controller failure. The placement algorithm performed better when it was compared to random placement algorithm. The novel proposed approach of packet forwarding was fair to both elephant and mice flow. This also made the data plane scalable by reducing the number of flow table entries. Further, the load on the controller ws reduced, when the switches took local corrective measures without consulting the controller. The dynamic load balancing of controllers using a switch migration algorithm, ensured that there was no disruption to the ongoing packet transfers.

7.2 Future work

In this section, suggestions are made for some future research work that can be carried out with regard to scalability and resiliency of the control plane and data planes in SDN.

The framework that is presented in this work ensures scalability of the control plane, by placing multiple controllers and clearly separating the functionality of the edge and core controllers. This concept of edge and core separation may be extended to other networks such as IoT networks. Such networks extensively use computation and aggregation at the edge, in order to reduce the traffic in the core network. Building an edge-core SDN network, that understands and supports such edge computation can make the network scalable. Another possible future work can be, to dynamically adjust the number of controllers in the network in response to the load. As the load increases new controllers may be included and load balancing algorithms can be run to redistribute the load. Similarly, when the load decreases some controllers may be shut down and the load may be optimally distributed among the remaining controllers.

By implementing sub-graph survivability this work has ensured that, the failure of one controller shall not lead to failure of entire network. However, there are several other failure models that can be examined, and failure protection or management can be provided. One such failure scenario, is that of the link to the controller. These links are critical because their failure causes the failure of entire network, even though the switch and controllers are working correctly. As future work, in such a scenario, probability of failure of the link can be computed and back up paths can be introduced. The placement metrics can be developed accordingly, along with the placement algorithms. Another option for future work is considering the failure models of other SDN domains, such as SDN for wide area networks and SDN for IoT networks. These networks have failure models which are significantly different from that of regular SDN. For example, software failures that may be introduced by SDN applications. SDN controller code is highly reused and optimized to minimize failure, however SDN application codes, like the ones that have been used for traffic shaping etc., are highly prone to failures.

In this work, the data plane has been made scalable by reducing the number of flow table entries in the switches. This is accomplished by adopting source routing. Future work on this can explore reducing the size of the flow table entries. SDN based forwarding may not require all the header fields used in traditional networks, such as time to live etc. and such information may be removed. Such a study can not only make the data plane scalable but also make it faster and more streamlined, by removing redundant unused information. Further, flow table utilization can be improved by implementing intelligent eviction rules for the table entries. These rules may be based on the traffic patterns that have been identified using machine learning techniques. Our work on the data plane has shown that flow splitting allows better utilization of the links and fairness to both the elephant and mice traffic. However, if large networks are considered, the split packets may have to be reordered. A study of the effect of such reordering and mechanisms, in order to prevent it, may be the future research scope. Source routing and flow splitting together give better scalability and network utilization. In order to make the source routing resilient to link failure, bypass paths are stored on the switches. This work can be extended so that the space required for storing such back up paths can be reduced. One idea could be to encode the back up paths or design new data structures. Normally, encoding forwarding information on the switches makes forwarding packets slow. However, in this case only rarely used back up paths are encoded, thereby reducing memory utilization without adversely effecting the speed of packet forwarding.

References

(2014). Floodlight controller. http://www.projectfloodlight.org/floodlight/. Accessed: 2014-07-27.

(2014). Mininet. http://mininet.org/. Accessed: 2014-07-27.

Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., and Shenker, S. (2013). pfabric: minimal near-optimal datacenter transport. In *SIGCOMM*, pages 435–446.

Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., and Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. In *IEEE/ACM/IFIP International Conference on Network and Service Management (CNSM)*, Zurich, Switzerland.

Beheshti, N. and Zhang, Y. (2012). Fast failover for control traffic in software-defined networks. In *GLOBECOM*, pages 2665–2670.

Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC 2010, pages 267–280, New York, NY, USA. ACM.

Bifulco, R., Matsiuk, A., and Silvestro, A. (2017). CATENAE: A scalable service function chaining system for legacy mobile networks. *Int. Journal of Network Management*, 27(2).

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM 2017, pages 1–12, New York, NY, USA. ACM.

Dvir, A., Haddad, Y., and Zilberman, A. (2018). Wireless controller placement problem. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4. IEEE.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110.

Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. (2010). Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 15:1–15:12, New York, NY, USA. ACM.

Hamilton, J. (2009). Data Center Networks Are in My Way. *Stanford Clean Slate CTO Summit*.

Hassas Yeganeh, S. and Ganjali, Y. (2012). Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN 2012, pages 19–24, New York, NY, USA. ACM.

Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. *Computer Communication Review*, 42(4):473–478.

Heng, L., Dan, Y., and Xiaohong, Z. (2012). Survey on multi-tenant data architecture for saas. *International Journal of Computer Science Issues (IJCSI)*, 9(6):198.

Hock, D., Gebert, S., Hartmann, M., Zinner, T., and Tran-Gia, P. (2014). Poco-framework for pareto-optimal resilient controller placement in sdn based core networks. In *NOMS*, pages 1–2.

HU, Y. N., WANG, W. D., GONG, X. Y., QUE, X., and CHENG, S. (2012). On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, 19, Supplement 2(0):92 – 171.

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. (2013). B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM 2013, pages 3–14, New York, NY, USA. ACM.

Kannan, K. and Banerjee, S. (2013). Compact tcam: Flow entry compaction in tcam for power aware sdn. In Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R. K., and Sinha, P., editors, *ICDCN*, volume 7730 of *Lecture Notes in Computer Science*, pages 439–444. Springer.

Katta, N., Hira, M., Kim, C., Sivaraman, A., and Rexford, J. (2016). Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 10:1–10:12, New York, NY, USA. ACM.

Kentis, A. M., Pilimon, A., Soler, J., Berger, M. S., and Ruepp, S. R. (2018). A novel algorithm for flow-rule placement in sdn switches. In *4th IEEE International Conference on Network Softwarization*. IEEE.

Kim, H. and Feamster, N. (2013). Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119.

Klaedtke, F., Karame, G. O., Bifulco, R., and Cui, H. (2014). Access control for sdn controllers. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 219–220. Citeseer.

Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M. (2011). The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 –1775.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. (2010). Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA. USENIX Association.

Kreutz, D., Ramos, F. M., and Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN 2013, pages 55–60, New York, NY, USA. ACM.

Lakshminarayanan, K., Rangarajan, A., and Venkatachary, S. (2005). Algorithms for advanced packet classification with ternary cams. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 193–204, New York, NY, USA. ACM.

Lange, S., Gebert, S., Spoerhase, J., Rygielski, P., Zinner, T., Kounev, S., and Tran-Gia, P. (2015a). Specialized heuristics for the controller placement problem in large scale sdn networks. In *Proceedings of the 2015 27th International Teletraffic Congress*, ITC 2015, pages 210–218, Washington, DC, USA. IEEE Computer Society.

Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Hock, D., Jarschel, M., and Hoffmann, M. (2015b). Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17.

Latora, V. and Marchiori, M. (2001). Efficient behavior of small-world networks. *Phys. Rev. Lett.*, 87(19):198701.

Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. (2012). Logically centralized?: State distribution trade-offs in software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN 2012, pages 1–6, New York, NY, USA. ACM.

Liu, J., Shi, Y., Zhao, L., Cao, Y., Sun, W., and Kato, N. (2018). Joint placement of controllers and gateways in sdn-enabled 5g-satellite integrated network. *IEEE Journal on Selected Areas in Communications*, 36(2):221–232.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G. M., Peterson, L. L., Rexford, J., Shenker, S., and Turner, J. S. (2008). Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74.

Ramos, R. M., Martinello, M., and Rothenberg, C. E. (2013a). Data center fault-tolerant routing and forwarding: An approach based on encoded paths. *Dependable Computing, Latin-American Symposium on*, 0:104–113.

Ramos, R. M., Martinello, M., and Rothenberg, C. E. (2013b). Slickflow: Resilient source routing in data center networks unlocked by openflow. In *LCN*, pages 606–613.

Shirali-Shahreza, S. and Ganjali, Y. (2018). Delayed installation and expedited eviction: An alternative approach to reduce flow table occupancy in sdn switches. *IEEE/ACM Transactions on Networking*, 26(4):1547–1561.

Singla, A., Hong, C. Y., Popa, L., and Godfrey, P. B. (2012). Jellyfish: Networking data centers randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 17–17, Berkeley, CA, USA. USENIX Association.

Soliman, M., Nandy, B., Lambadaris, I., and Ashwood-Smith, P. (2012). Source routed forwarding with software defined control, considerations and implications. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student 2012, New York, NY, USA. ACM.

Stephens, B., Cox, A. L., and Rixner, S. (2013). Plinko: Building provably resilient forwarding tables. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pages 26:1–26:7, New York, NY, USA. ACM.

Tootoonchian, A. and Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA. USENIX Association.

UlHuque, T., Jourjon, G., and Gramoli, V. (2015). Revisiting the controller placement problem. In *IEEE Conference on Local Computer Networks*, pages 1–4, Clearwater Beach, USA.

UlHuque, T., Si, W., Jourjon, G., and Gramoli, V. (2017). Large-scale dynamic controller placement. *IEEE Trans. Network and Service Management (TNSM)*, 14(1):63–76.

Wang, G., Zhao, Y., Huang, J., and Wu, Y. (2018). An effective approach to controller placement in software defined wide area networks. *IEEE Transactions on Network and Service Management*, 15(1):344–355.

Wuhib, F., Stadler, R., and Lindgren, H. (2012). Dynamic resource allocation with management objectivesimplementation for an openstack cloud. In *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pages 309–315. IEEE.

Yu, M., Rexford, J., Freedman, M. J., and Wang, J. (2010). Scalable flow-based networking with difane. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 351–362, New York, NY, USA. ACM.

Zhang, F., Liu, G., Fu, X., and Yahyapour, R. (2018). A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys & Tutorials*, 20(2):1206–1243.

Zhang, S. Q., Zhang, Q., Tizghadam, A., Park, B., Bannazadeh, H., Leon-Garcia, A., and Boutaba, R. (2016). Sector: TCAM space aware routing on SDN. In *28th International Teletraffic Congress (ITC 28)*, Würzburg, Germany.

Zhang, Y., Beheshti, N., and Tatipamula, M. (2011). On resilience of split-architecture networks. In *GLOBECOM*, pages 1–6.

Zheng, K., Wang, L., Yang, B., Sun, Y., Uhlig, S., undefined, undefined, undefined, and undefined (2017). Lazyctrl: A scalable hybrid network control plane design for cloud data centers. *IEEE Transactions on Parallel Distributed Systems*, 28(1):115–127.

Publications

Journal

1. Saumya Hegde, Shashidhar Koolagudi, Swapan Bhattacharya, Scalable and Fair Forwarding of Elephant and Mice Traffic in Software Defined Networks, in Elsevier Computer Networks, Volume 92P2, 2015, Pages 330-340

Conference

1. Saumya Hegde, Shashidhar Koolagudi, Swapan Bhattacharya, Path Restoration in Source Routed Software Defined Networks, 9th International Conference on Ubiquitous and Future Networks, Italy, July 4-7, 2017 DOI: 10.1109/ICUFN.2017.7993885
2. Saumya Hegde, Roshni Ajaygosh, Shashidhar Koolagudi, Swapan Bhattacharya, Dynamic Controller Placement in Edge-Core Software Defined Networks, IEEE TENCON, 2017, Malaysia, Nov 5-8, 2017

Resume

Saumya Hegde

Personal Details

Name: Saumya Hegde

Date of Birth: 29 February 1976

Qualification

M.Tech. Computer Engineering, National Institute of Technology, Surathkal, 2007

B. E, Computer Engineering, Nitte Mahalinga Adyanthaya Memorial Institute of Technology, Nitte, 1997.

Current Employment

National Institute of Technology Karnataka, India, Nov 1998 to date

Assistant Professor

Permanent Address	**Work Address**
Saumya Hegde	Saumya Hegde
4-25/1A	Dept. of Computer Engineering
Opposite Vidyadayinee School	NITK Surathkal
Surathka 575014	India
India	
hegdesaumya@gmail.com	hegdesaumya@gmail.com