Exploring Source Routing as an Alternative Routing Approach in Wide Area Software-Defined Networks

by

Mourad Soliman

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University

Ottawa, Ontario

ABSTRACT

Software-Defined Networking (SDN) has been gaining increasing attention in both the research and the industry communities. Separating the control and data planes has brought many advantages such as greater control plane programmability, more vendor independence, possibility of network virtualization, and lower operational expenses. SDN deployments are possible in a variety of contexts: Enterprise networks, Datacenter networks, and Wide Area Networks (WANs). However, SDN raises several concerns for WANs including performance limitations due to the larger propagation delays, the increased controller work load due to a larger number of network elements, and concerns about performance impacts of the controller placement.

This study attempts to address some of these issues by examining the effects of using source routing as an alternative to traditional distributed routing in SDN-based WANs. Our simulations and analysis show that source routing can bring significant gains in SDN performance in WANs, improve network scalability, and reduce the network performance sensitivity to controller placement.

ACKNOWLEDGEMENTS

First, I thank God, the almighty, for giving me the strength, the patience, and the blessing to accomplish the research work presented in this thesis.

I would like to thank and express my great appreciation to my thesis supervisors; Dr. Ioannis Lambadaris and Dr. Biswajit Nandy, for their valuable guidance and supervision. I am very grateful for all the things that I have learned from both of you along the way, and I am very grateful for your support, insights, and dedicating time for me to accomplish this research study.

I would like to thank my parents whose prayers have always been for me with their support and encouragement.

Finally, I would like to specially thank my lovely wife for being there for me in all situations and circumstances with her support, encouragement, appreciation, and patience.

Table of Contents

List of Tables

List of Figures

List of Acronyms

API	Application Program Interface
ARP	Address Resolution Protocol
CapEx	Capital Expenditure
CSMA	Carrier Sensing Multiple Access
FIFO	First-in First-out
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MPLS	Multiprotocol Label Switch
NDDI	Network Development and Deployment Initiative
NIC	Network Interface Card
NS-3	Network Simulator 3
OpEx	Operational Expenditure
OS3E	Open Science, Scholarship and Services Exchange
POCO	Pareto-Based Optimal Controller-Placement
PPBP	Poisson Pareto Burst Process
QoS	Quality of Service
RTT	Round-Trip Time
SDN	Software-Defined Networking
SLA	Service Level Agreement
SLSR	Strict Link Source Routing
TCAM	Ternary Content-addressable Memory
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
WAN	Wide Area Network

1. INTRODUCTION

This chapter will cover the problem statement and the motivation behind this research by mentioning the challenges faced by the SDN architecture in WAN deployments. Then, our research objectives will be discussed. Finally, the research contributions are summarized and followed by the thesis outline.

1.1. Problem Statement and Motivation

Software Defined Networking relies on the separation of control and data planes. This separation allows the control and data planes to evolve independently and brings great advantages regarding easing the development of new control plane applications, reducing network complexity, and increasing interoperability and vendor independence. As a result of these advantages, SDN was adopted as a reliable network architecture for several network deployment use cases. SDN have been leveraged in Enterprise Networks, Datacenter networks, and WANs for Cloud and Service Providers. SDN deployments in WAN in particular have been gaining increasing attention with several successful implementations such as Google inter-Datacenter WAN that leverages SDN principles to achieve above 90% utilization on many WAN links in their network [1].

This increased attention also led to realizing the challenges and concerns that come with the deployment of SDN in WANs.

SDN relies on the OpenFlow communication messaging between the network controller and the OpenFlow switches in the network. WANs are often characterized by long propagation delays between the WAN network nodes that are usually geographically distributed over large areas. In SDN, the controller-switch communications over links with long propagation delays makes achieving acceptable network convergence/re-convergence times more challenging [2], and consequently limits network scalability. Reducing the latency in the communication process between the controller and the switches is usually not an option in a WAN because it is constrained by physics and the physical topology. This raises a need for an approach that reduces the volume of communication between switches and controllers. Since most of the control communication involves state distribution, a method that reduces or eliminates state distribution would have a positive impact on performance.

In addition, since WANs usually cover large areas such as regions, or countries, or even continents, the placements of the network nodes have a direct effect on the latency between them. In particular, from an SDN perspective, the placement of the network controller in WAN becomes a major factor in network design as it affects the network convergence and re-convergence time. A study in [2] has shown that optimizing the network controller in an SDN-based WAN is a complicated and an expensive procedure. In addition when this procedure is completed only a small percentage of the possible network placements are near optimal, once again raising the need for an approach that reduces the sensitivity of such a network design parameter.

Motivated by these SDN-based WANs long propagation delay challenges, as well as network design concerns due to controller placement, this paper explores a promising alternative routing approach in SDN that leverages means of Source Routing. The motivation for this approach comes from the fact that a centralized SDN controller has a global view of the entire network. Therefore, the controller is capable of performing path computations efficiently, making source routing a natural fit for SDN architecture.

In this research study, we illustrate that with some modifications to the traditional OpenFlow protocol, source routing can help address some of the SDN challenges in WANs, bring improvements in network convergence times, and provide increased flexibility of network design and controller placements.

1.2. Research Objectives

In this research, we studied SDN to shed the light on some challenges that arise when considering SDN for WAN deployments. This study led to identifying several challenges imposed by large propagation delays between the dispersed nodes and the controllers when SDN is used in a WAN. Other challenges were found relating to the placement of the controller, network scalability, and switch memory hardware requirements.

The main research objectives in this study are:

1) Research and identify the main challenges of using the SDN architecture in WAN deployments.
2) Study source routing as an alternative routing approach in SDN WAN deployments.
3) Implement a simulation model that leverages source routing functionalities to the standard OpenFlow switch and a corresponding controller simulation model on ns-3.
4) Perform a high level computational analysis as well as a detailed packet level simulation to show the effectiveness of source routing in SDN-based WANs.

1.3. Research Contributions

In this research, we were able to identify some of the challenges faced by SDN-based WAN deployments and, propose a solution that aims to ease and overcome these challenges.

The main contribution of the research is to show the effectiveness of source routing as an alternative routing approach in SDN-based WANs. This was done as a two step process: (1) conducting computational analysis for the possible source routing gains if implemented in a production SDN-based WAN, (2) conducting packet level simulations after implementing source routing functionalities in a simulation model for an OpenFlow switch and controller in ns-3. The simulation models allowed running several experiments over a simulated representation of a real production SDN-based WAN.

The contributions are as follows:

1) Conducting research on SDN-based WAN deployments and identifying the main challenges.
2) Proposing an alternative routing approach based on source routing.
3) Conducting a preliminary computational analysis to study the possible gains of source routing in a production SDN-based WAN.
4) Implementing the proposed routing approach as a simulation model using SDN in ns-3.
5) Showing the effectiveness of the proposed solution on:
    a. Network convergence time
    b. Network state distribution

c. Switch flow table sizes

d. Controller placement considerations

e. Controller processing load and switch control traffic forwarding

Parts of this thesis have appeared in the following publications:

- M. Soliman, B. Nandy, I. Lambadaris, P. Ashwood-Smith. “Source Routed Forwarding with Software Defined Control, Considerations and Implications.” *ACM CoNEXT Student’12*, December 2012, Nice, France.
- M. Soliman, B. Nandy, I. Lambadaris, P. Ashwood-Smith, "Exploring source routed forwarding in SDN-based WANs," *IEEE ICC 2014,* June 2014, Sydney, Australia.

1.4. Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides a background about Software-Defined Network, OpenFlow protocol, and Source Routing. It also provides a review of the related published work. Chapter 3 introduces an analysis of the possible gains of Source Routing if implemented in a production SDN WAN. In Chapter 4, our proposed source routing architecture and simulation model details are discussed. Chapter 5 presents the simulation results and a discussion of the findings. Chapter 6 concludes the thesis with an overview of the proposed source routing model and a summary of the findings of the study, and outlines proposed future work.

2. BACKGROUND AND RELATED WORK

2.1. Software-Defined Networking (SDN)

In a traditional network, a switch is composed of data and control planes, Figure 2.1. The data plane relies on the hardware implementation of the switch. The data plane is responsible for forwarding the packets through the switch fabric from one port to another depending on a forwarding table that matches packets to an output port. The control plane relies on the software implementation of the logic required to build the forwarding table. It contains implementation of network protocols and manages signaling between the switch and other network elements. Typically, a switch's data plane and control plane relies on proprietary hardware and control functions implementations. This implies that the switches hardware functions and software features remain under the control of its vendor with little to no room for external customization.

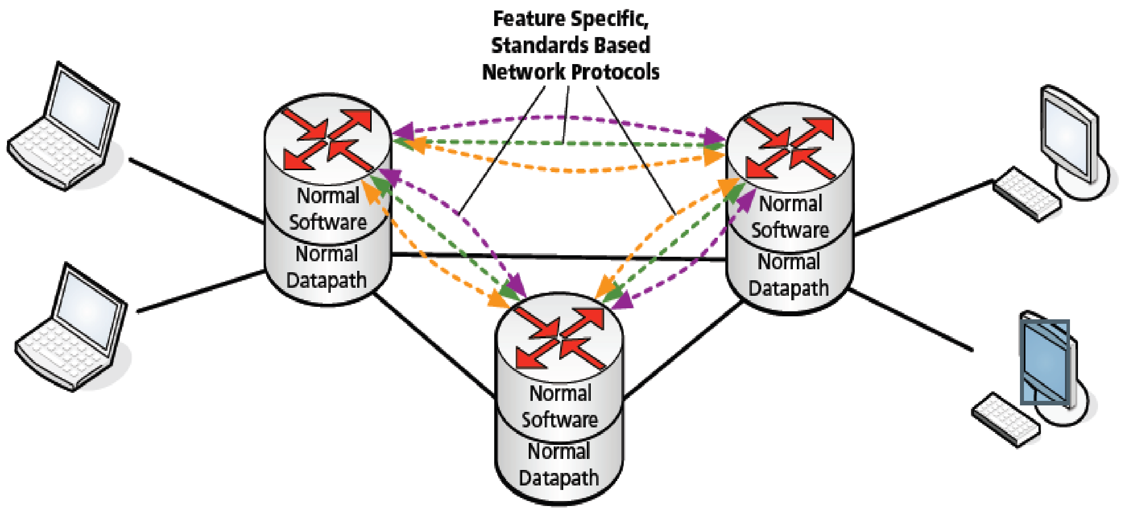


Figure 2.1: Traditional Network Scheme [3]

This traditional architecture of networking was found to impose several limitations in both research and industry fields. Proprietary implementations increase network complexity especially when dealing with products from different vendors, which is the common case for communications service providers. This leads to difficulties in network management, new services rollouts, and scalability. This also makes service providers increasingly tied to vendor product cycles and releases. In the research world, the traditional networking architecture limits the researchers' abilities to experiment with new network protocols since it is difficult to experiment with real equipment. This limits the academic innovation and the evolution of the networking field.

Software-Defined Networking is a new emerging network architecture that is aimed towards overcoming the limitations of traditional networking, and making next generation networks more adaptive, dynamic, manageable, and innovative. Software-Defined Networking architecture relies on the idea of decoupling control and date planes. In this architecture, the data plane remains in the switches possibly depending on commodity hardware implementations. Control functionalities in SDN are moved to a logically centralized controller platform that is open-source, programmable, and vendor agnostic.

Figure 2.2 shows how the SDN architecture is broken into three layers. First, the infrastructure layer contains the data plane forwarding devices, which are simple switches with no control functionality that depend only on an externally-programmable flow table to forward incoming packets. Second, the control layer houses a network-optimized operating system where several controller applications can be developed. Examples of network applications include, routing, policy management, device configuration, security, traffic engineering, etc. These functionalities are housed on one or multiple controllers in the software-define network. The controller has knowledge of the topology of the network of forwarding devices that it manages, and is responsible for configuring the forwarding tables on those devices. The application layer controls the network applications to provide customization of the network infrastructure according to the business and the operational needs. In SDN a uniform vendor-agnostic interface/protocol called OpenFlow is used for handling interactions between the centralized controller and the network devices' data planes [4].

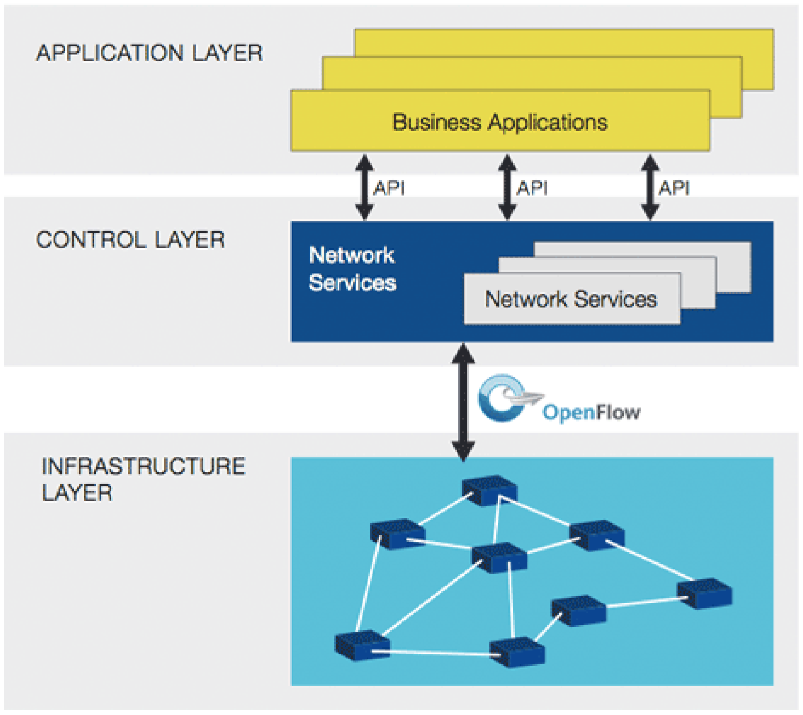


Figure 2.2: Software-Defined Networking Architecture [4]

The SDN architecture brings great advantages to the field of networking. Decoupling control and data planes allows them to evolve independently with different paces. Network control logic programmability enables experimenting with new ideas and protocols in realistic environments. In addition, programmability makes the network more agile and adaptive to the operational needs. Furthermore, the centralized controller that maintains a global view of the network eases network and policy management, and the open source implementations promote vendor independence [4].

2.2. OpenFlow Protocol

In Software-Defined Networks, OpenFlow protocol is considered the mechanism that allows network programmability. The network controller in SDN architecture uses OpenFlow protocol to manage and manipulate the forwarding tables of the network switches. OpenFlow protocol controls and defines the messaging between the controller and the switches and vice versa. The OpenFlow standard defines the flow table configuration control messages and the behaviour of an OpenFlow switch [5].

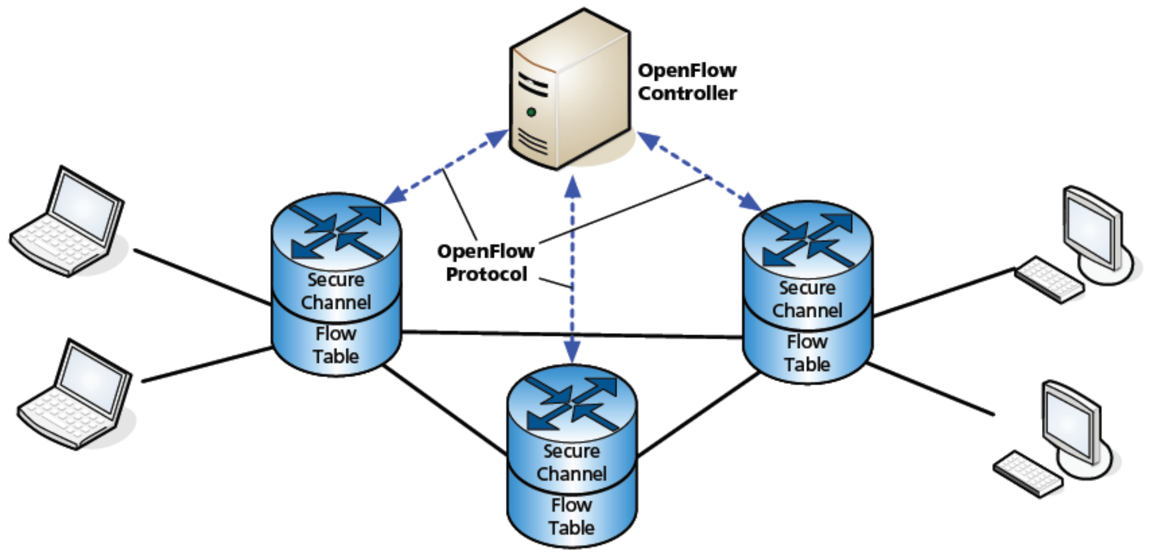


Figure 2.3: OpenFlow Networking Scheme [3]

A typical OpenFlow switch consists of three main parts: a flow table, a secure channel to the controller, and the OpenFlow protocol [5].

The OpenFlow switches flow table contains a list of flow table entries that specify how the switch will deal with the incoming packets. A flow table entry typically contains flow characteristics that are used to match the incoming packet, an action associated with the matching criteria that defines how the matched packet should be processed, and a statistics field that tracks the number of packets matched and the time of the last packet matched [5]. Table 2.1 below shows a set of packet header fields that can be used to match a packet to a table flow entry, according to OpenFlow version 1.0 [6]. Later versions of OpenFlow standard include more header fields for matching such as MPLS labels and traffic classes.

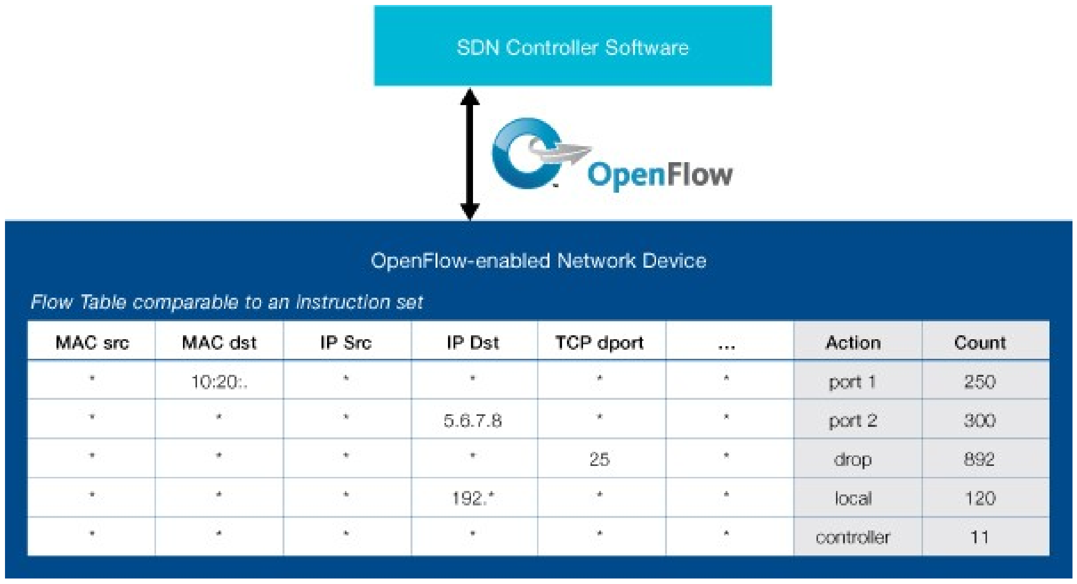
Table 2.1: Packet header match fields

Packet Match Field
Ingress Port
Ethernet source address
Ethernet Destination Address
Ethernet Type
VLAN Id
VLAN Priority
IP Source Address
IP Destination Address
IP Transport Protocol ID
IP ToS bits
TCP/UDP Source Port or ICMP type
TCP/UDP Destination port or ICMP code

Starting from OpenFlow specification 1.1, an OpenFlow switch can contain multiple flow tables that are pipelined, and a packet will move from one table to the other until matched. If a packet matches one of the header rules defined in one of the flow table entries, the associated action is applied.

The applied action can be one of the following [5]:

- Forward the packet to a specific port, or all ports. This enables a packet to be forwarded according to a route through the network.
- Forward the packet to the controller. This allows further processing to be applied at the controller.
- Manipulating packet header fields. This allows applying traffic engineering functionalities, e.g. MPLS.
- Dropping the packet.
- Forward the packet to the switch's normal processing plane if the switch is OpenFlow enabled but also capable of normal networking functionality.


SDN Controller Software
OpenFlow
OpenFlow-enabled Network Device
Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figure 2.4: An example of an OpenFlow Switch Flow Table [7]

If a packet doesn't match any entry in the flow table it is automatically forwarded to the controller through the secure channel.

A secure channel is a logical link between the switches and the network controller. The switch uses this logical link to send events to the switch when a new unknown packet arrives or when a flow table entry expires. The controller also uses this link to send configuration messages to the switch. All the messages exchanged between the switch and the controller must be formatted according to the OpenFlow protocol.

The OpenFlow protocol defines the type and the format of the messages exchanged between the switch and the controller on the secure channel [5]. An example of the type of messages sent from the controller to a switch includes: *Modify-State* messages to add, delete, or modify entries in the switch's flow table [6]. The controller can also send messages to query the switch features and configurations, to collect statistics about switch's flow tables or ports, or to request acknowledgements for completed operations. In addition, the controller can send a message to the switch to send a packet out of a specified port. The last message type is used when the switch sends an unknown packet to the controller, as a response along with the corresponding new flow table entry addition message. An example of messages sent by the switch to the controller includes *Packet-In* messages. The switch sends this message when a packet doesn't match any of the flow entries in the flow table. The switch also sends messages to the controller in the case of events, such as flow removal as a result of flow expiry, port configuration state changes, notifications and general errors [6].

The main responsibility of the OpenFlow controller is to install flow table entries in the flow tables of the switches that it manages. The OpenFlow controller maintains overall knowledge of the network topology, and therefore, can determine the best paths of unknown flows and translate that into flow table entries to be installed to the switches.

Flow table entries can be given timeout values. When the timeout expires, the switch removes the flow table entry from its flow table and sends a notification message to the controller [6]. In OpenFlow standards, two timeout approaches are supported: an idle timeout and hard timeout. The idle timeout specifies how long an entry should be maintained in the flow table since the last time a packet matched that entry. The hard timeout, although far less used, specifies when the entry should be removed regardless of when it was last matched. These two approaches are used to manage the flow table sizes [6].

2.3. SDN-Based WAN Deployments

2.3.1. The WAN Use Case

The great advantages of SDN in terms of network simplicity, programmability, vendor dependence, and lower operational expenses make the architecture a preferred deployment option in variety of use cases. SDN use cases include Enterprise networks, Campus networks, Datacenters, Cloud providers, Carriers and Service providers. The SDN deployments in these use case range from small local networks to large-scale WANs. In Datacenters and Cloud provider networks, SDN allows efficient management of network resources, and enables features like Network Virtualization. SDN can also be utilized to control network policies distribution and fast service rollout. SDN also started playing a role in Service Provider networks allowing automated provisioning of network infrastructure, better network optimization and traffic engineering [7], and significant reductions in CapEx and OpEx. In dynamic WAN deployments, SDN can help ease tasks such as moving large amount of data without the need for expensive inspection devices by programming flow-level information in SDN-enabled routers and switches. In WAN deployments, SDN can also optimize the provisioning of network interconnects reducing the operational expenses, as well as optimize bandwidth allocations across the WAN through the use of automated policy distribution [8], which is further simplified by using the logically centralized network control plane.

The research in this study focuses on the SDN deployments in WANs. There has been an increasing attention and interest in leveraging SDN in WANs for better management of links, QoS, and service rollout. Currently, there have been several successful SDN deployments in WANs, with Google inter-datacenter WAN being one of the most famous examples. Google inter-datacenter WAN has utilized SDN approaches to achieve near 100% utilization in many of the WAN links and a significant average of 70% link utilization. Google leveraged SDN principles, OpenFlow-enabled high performance hardware, centralized traffic engineering, and policy enforcement at the network edges according to dynamic demands, bandwidths, and resource availability in the network to achieve such high utilization results [1].

Another example of SDN adoption for WAN optimization is the research introduced in [9]. The research targets multi-layered WAN optimization by presenting a network management system based on SDN/OpenFlow, OSCARS-TE, to provide traffic management and bandwidth on demand functionalities in WANs. Leveraging SDN in the management of WANs using OSCARS-TE was evaluated in a multivendor multi-layer WAN and was proven to solve the inefficiencies of WAN links traffic congestion [9].

The discussion of SDN adoption in WAN also brings into consideration leveraging SDN in transport optical networks as they provide the backbone for WAN interconnects. There has been several proposals for utilizing SDN in transport network architectures to satisfy WAN datacenter interconnects' demands [10], [11].

Most networking equipment manufacturers are responding to this increasing attention to the SDN use case in WAN by designing and providing solutions for SDN-based WAN optimization, such as Cisco's WAN Automation Engine [12], Alcatel-Lucent's Nuage Virtualized Network Services [13], and Ciena's Multilayer WAN Controller [14], to name a few.

2.3.2. Challenges

Despite the great advantages of leveraging SDN/OpenFlow in WANs, certain challenges related to the nature of the SDN architecture and the nature of WAN topologies still arise.

Network Convergence Time

WANs are usually characterized by geographically dispersed network elements that span broad areas and may cover large regions, countries or continents. This naturally leads to relatively long propagation delays between the different network elements. Since SDN depends on a logically centralized controller, all the nodes must communicate back and forth with the controller to obtain new state information to forward network flows. Furthermore, setting up the path for an unknown network flow often requires the controller to insert flow table entries in several network switches. Consequently, in WANs context, the long propagation delays between the nodes become a major contributor to flow installation time. As an SDN-based WAN expands and increases in complexity, the need to distribute large amounts of state coupled with the long propagation delays between network devices and the SDN controller(s), makes achieving acceptable network convergence/re-convergence times more challenging [2].

Flow Table Size

As mentioned before, the SDN controller is responsible for installing flow table entries in the OpenFlow-enabled devices' flow tables. The flow table size in commodity switches is limited as it relies on the available hardware TCAM [15]. Acquiring larger TCAM to accommodate larger flow tables affects the capital cost of the network switches, which goes against one of SDNs goals of cost reduction. This becomes a challenge in the context of a

WAN that is responsible for providing connectivity and services to a very large number of network elements with several hundred thousands of network flows generated every second.

Efficient flow table management using OpenFlow has been a hot topic in research. The research in [15] studies different optimization techniques' effects on the performance in terms of the flow table miss rate and occupancy "table size". The research introduces a flow table management mechanism that combines the use of controller-defined flow timeouts with explicit controller flow removal messages to the switches [15]. This method achieves a 42% reduction in the table size without affecting the miss rate of the flow table.

Several other proposals have been made in recent literature to overcome the limitation of flow table size in OpenFlow switches. The research in [16] suggests flow table rules aggregation as a possible solution for reducing flow table sizes and consequently TCAM requirements in commodity switches. While aggregation can be impose a limitation as it requires processing that may delay the flow table updates, the research proposes using efficient and fast flow table aggregation schemes to overcome these limitations with promising results [16]. Other proposals included methods such as flow caching algorithms [17], and the use of consistent route update sequences [18].

Controller Placement Problem

Moreover, in a WAN with long propagation delays, SDN places limits on network design flexibility because the controller's placement would directly impact the network performance because all the network nodes have to communicate back and forth with the controller. The controller placement problem was motivated in [2], and was demonstrated to have a direct effect on the performance of the network design. It was demonstrated that in some networks, performance varies widely with different placement choices and that only a small percentage of these choices are near optimal. In real WAN production networks the optimal controller placements might be difficult due to various factors: finances, policies, security issues, and the occasional need to grow or change the network topology without having to move the controller. This may force network operators to accept sub optimal placement initially or later in the life of their network in return of gaining the other promised advantages of SDN. The research in [2] also evaluates the controller placement problem in terms of two performance metrics, average latency between the nodes and the controller, and worst-case latency. It was concluded that a trade-off has to be made between the two metrics when optimizing the controller(s) placement.

The arguments presented in [2] motivated the research for controller placement optimization techniques. This led to several proposals suggesting different frameworks to find the optimal control placement in an SDN network. One of the proposed frameworks in the literature is a resilient Pareto-based Optimal Controller-placement (POCO) [19]. This framework is targeted at finding a range of controller placement that best satisfies specific network needs such as latency limitations, failure tolerance, and node-to-controller allocations. The research evaluated the framework on wide range of topologies to validate the usability, and it was realized that even with controller placement optimizations, more than 20% of the nodes in most topologies need to be controllers to assure reliability in case of node or double link failures [19]. In addition, the research in [20] elaborated more on the controller placement problem by defining a reliability metric and evaluating a number of placement algorithms. The research proposes and validates a greedy algorithm for finding controller placements with close to optimal reliability performance.

Other studies have taken a very different approach to address the controller placement problem. The research in [21] proposes a game theory based scheme that relies on a distributed implementation of an optimization application that can run at each SDN-controller in the network. Using the proposed algorithms a controller can validate its value in the design and decided after comparing with the other controllers in the network whether controllers should be added, moved, or deleted.

2.4. Source Routing

Motivated by the increased interest in the adoption of SDN in WANs, and the challenges of SDN-based WANs, the research in this study explores source routing as a promising alternative routing approach in SDN. The fact that having a centralized control entity that has a global view of the network, performing path computations makes source routing a natural fit for SDN architecture.

Source routing is a general technique where the sender of a packet specifies the route that the packet should take through the network of switching devices. Two flavors exist, Strict and Loose [22]. Strict Source Routing requires that every hop be included in the header, while Loose Source Routing allows for some hop-by-hop behavior mid path. The research in this study focuses on Strict Source Routing. Source Routing on its own is not a novel technology, however, it hasn’t been a preferable option in IP networks due to extra per-packet overhead of the source routing headers [23], and security concerns [24].

Although source routing hasn't been a favorable choice in networking in general, there have been attempts to leverage the advantages of source routing in networks to address scalability concerns. AXON [25], SecondNet [26] are examples of source routing architectures in large datacenter networks. AXON [25] proposes a novel source-routed Ethernet protocol to transfer data through a network of source-routing-enabled Ethernet-compatible devices called Axons, and illustrate that such architecture provides greater performance and scalability than conventional switched Ethernet networks architecture.

SecondNet [26] doesn't propose a novel protocol, however, they introduce the notion that a stack of MPLS labels, consumed one by one at each hop, and where each hop identifies the next link to take, can be used to create Strict Source Routed behavior. Although depending on MPLS labels might seem a good approach to implement source routing, it has some drawbacks. MPLS labels are large in size, 20 bits, and the stack size upper bounds would limit the diameter of the network. Another main problem with MPLS is the fact that popping labels hides the path the packet has taken so far. This is considered a very serious problem in a service provider network as it makes network troubleshooting even more difficult and reduces security.

In addition, there is proposal from the IETF for Segment Routing [27], an alternative source routing based solution for simplicity, better traffic engineering and fast reroute capabilities. The IETF draft mentions its applicability to SDN and illustrates its ability to provide several orders of scaling gains over conventional hop-by-hop routing due to its elimination of the RSVP-TE signaling protocol. This method is also an MPLS label stack implementation of a source routing. Segment routing has been attracting attention in the network research communities for several use cases such as leveraging segment routing with SDN capabilities in a new carrier Ethernet architecture for service provider wide area transport networking [28].

Source routing has also gained attention in the SDN domain as an alternative to conventional hop-by-hop routing in datacenter networks, with proposals such as Rain Man [29] and SlickFlow [30]. The research in [29] proposes a scalable SDN datacenter solution that also leverages means of source routing and is compatible with architectures like SecondNet. The same study also demonstrates that centrally controlled source routed networks are more scalable than centrally controlled hop-by-hop routed networks.

The research study in [30] also focuses on the datacenter domain, as the authors propose a source routing approach as a more resilient mechanism to achieve fast failure recovery. SlickFlow [30] relies on encoding primary and alternative path information into the packet header fields. Packets can then be rerouted in case of failures by the switches without the need of communicating to the controller. The evaluation results for SlickFlow [30] show that it can achieve efficient failure recovery, with recovery times that are about 60% lower than traditional routing.

3. ANALYSIS: SOURCE ROUTING GAINS IN SDN-BASED WAN

As mentioned before, several new WAN deployments are implementing the SDN approach to benefit from its advantages. However, there are many issues and limitations that come into consideration as a result of SDN adoption in WAN, as mentioned in *Chapter 2*. In this chapter, we present an analysis of an alternative routing approach in SDN-Based WANs that is directed towards elevating some of the present challenges and allowing better performance and overall design flexibility for the network. We present a computational analysis of the aspects of flow convergence dynamics in various scenarios of an SDN-Based WAN to examine the following fundamentals:

- Network state distribution
- New flow convergence time
- Network scalability
- Performance sensitivity with respect to controller placements

In this chapter, background information is introduced *Section 3.1*. Then, the analysis methodology is explained in *Section 3.2*. *Section 3.3* gives in depth details of the analysis implementation. The results of the analysis are discussed in *Section 3.4*, and finally the findings of the analysis are summarized in *Section 3.5*.

3.1. Source Routing in SDN-Based WAN

In a conventional Software-Defined network, a new flow setup process using a reliable controller implementation takes the following steps [31]:

a) A packet of an unknown flow arrives at the ingress switch, and doesn't match any of the switch's flow table entries.
b) The switch generates a new flow request and sends it to the controller.
c) The controller determines the path for the new flow.
d) The controller responds with new state to be distributed to all the intermediate switches along the path and waits for acknowledgements[1].
e) The switches update their flow tables with new forwarding actions, and send acknowledgements to the controller.
f) Upon receiving all acknowledgements, the controller respond to the ingress switch, and the packet is forwarded to the first hop.
g) The packet flows to destination through the newly installed forwarding actions at each switch along the path.

The steps (a), (c), and (e) depend on the resources and the capabilities of the switch and the controller. In a WAN, the steps (b), (d) and (e) have the greatest contribution to the flow setup time, i.e. convergence/re-convergence time, because they are constrained by the propagation latencies. State distribution and receiving acknowledgements, steps (d) and (e), take at least time equal to the maximum Round Trip Time (RTT) among the RTTs

[1] The OpenFlow protocol does not automatically provide acknowledgements; however, it allows the controller to request on demand acknowledgements of switch operations using a type of messages called Barrier Messages [6].

between the controller and each of the intermediate switches along the flow's path. This is because the controller would distribute state to intermediate switches simultaneously.

The use of source routing in a WAN completely eliminates steps (d) and (e) from the flow setup time. This is because in source routing, the path information would be put in the form of a source route and pushed down to the ingress switch only. The source route would then be carried by every packet in the flow and inspected at every hop.

3.2. Analysis Methodology

In this analysis, a production SDN-based WAN is studied. The focus of the study is to examine the convergence time of a new flow from any source to any destination in the network. In this context, the convergence time is defined as the time the first packet of an unknown packet takes from source to destination. This includes the time taken to request new flow information from the controller, distribute new state by the controller, and propagate along the determined path from source to destination. The convergence time is also influenced by sources of network delays which include queuing delays, transmission delays, and propagation delays which are the dominant factor in WANs. State installation delays were not considered in this analysis as they are highly dependent on the switch implementation. In addition, the convergence time is also affected by the routing approach that is used in the network: conventional hop-by-hop or source routing.

Flows were analyzed between every possible source-destination pair in the network to calculate average convergence time in the network for various scenarios to compare the performance and the scalability of the routing approach used in the SDN-based WAN. The performance results, in terms of convergence time, were compared for two routing approaches: tradition hop-by-hop routing, and suggested source routing.

3.3. Analysis Implementation

The implementation of the analysis depended on three fundamental parts: modeling the network nodes and interconnections, modeling the network convergence dynamics, and estimating network delays.

3.3.1. OS3E Network Topology

As a first step, the OS3E network was selected as a realistic production SDN-based WAN for this analysis. OS3E is a network service developed by the Network Development and Deployment Initiative (NDDI) which is a partnership between Internet2, Indiana University and the Clean State Program at Stanford University. OS3E – which stands for Open Science, Scholarship and Services Exchange - is intended to support global scientific research. OS3E is designed based on the software defined network architecture and built using the first production deployment of OpenFlow technology. The network consists of 34 Nodes in 34 different locations across the united states, interconnected though the Internet2 high speed core network [32], Figure 3.1.

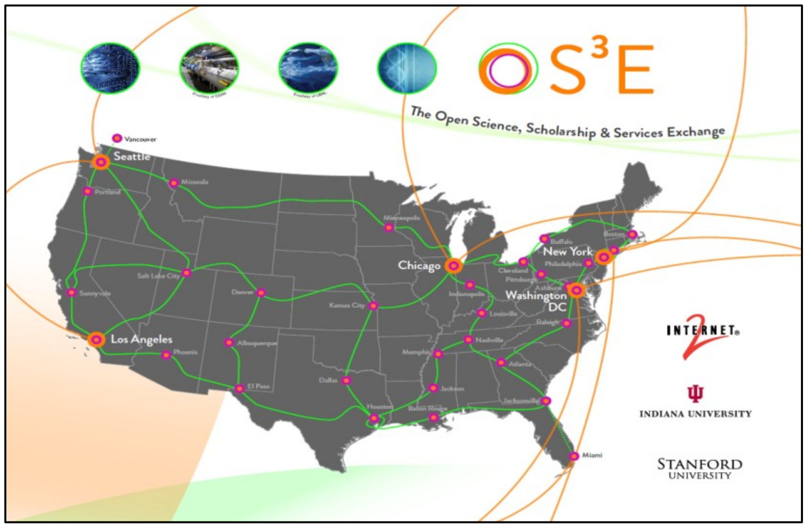
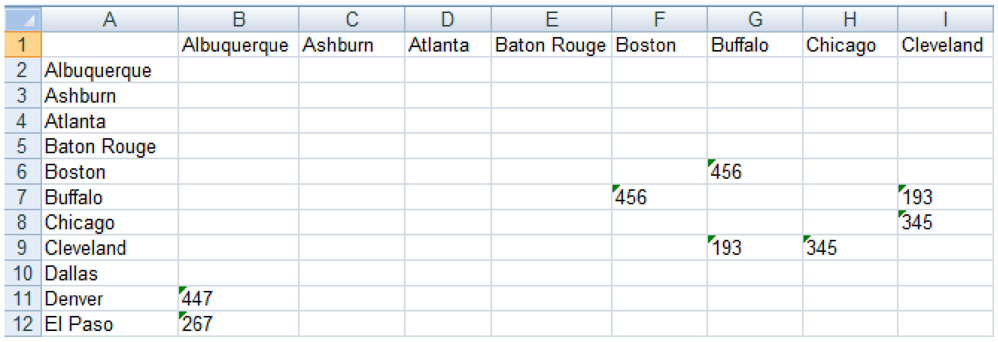


Figure 3.1: OS3E Network [32]

3.3.2. Network Modeling

To represent the network entities and their relationships, the network information needs to be organized in a meaningful and automatable way. For the OS3E case, this meant collecting information about the relationship between the 34 nodes that comprise the network and the length of the links that connect every pair of nodes in the network.

The nodes that make up the network were noted from the map, and organized in a spreadsheet alphabetically in the first row and the first column. The values at the nodes cross cells were arranged to represent the length of the link between the nodes. If there is no value recorded, this means that there is no link between the two corresponding nodes. For example, a value of 456 in the cell crossed by the row "Buffalo" and column "Boston" means that the length of the fiber link between Buffalo and Boston is 456 miles, and vice versa. The distances between the nodes were recorded from Google Maps manually. It was assumed that the fiber cables will go along the suggested main routes suggested by Google Maps to travel between any of the two cites. This provides a fair estimation as the effect of an error on the propagation delay in a fiber link would be minimal considering the speed of light in Fiber (2*10^8 m/s). A difference of ±100 miles leads to a ± 0.8 ms in the propagation delay. This process was done manually and iteratively for all the connections using Google Maps. A sample of the spreadsheet is show in Figure 3.2.



	A	B	C	D	E	F	G	H	I
1		Albuquerque	Ashburn	Atlanta	Baton Rouge	Boston	Buffalo	Chicago	Cleveland
2	Albuquerque								
3	Ashburn								
4	Atlanta								
5	Baton Rouge								
6	Boston						456		
7	Buffalo					456			193
8	Chicago								345
9	Cleveland						193	345	
10	Dallas								
11	Denver	447							
12	El Paso	267							

Figure 3.2: OS3E Network Adjacency Matrix

After organizing all the information in a spreadsheet, a network modeling application was developed in Java to properly model the network and represent the network nodes and relationships in a way that allows further and more complex analysis to be performed.

A graph database was used as a basis for modeling the OS3E network. Graph databases use structures with nodes, edges, and properties to represent the data. Graph databases provide index-free indexing that allows every element to contain direct pointer to its adjacent element(s) [33]. These characteristics for graph databases made them a natural fit to comprehensively model a network such as the OS3E. There are several graph database implementation projects available for download. The implementation that is selected for this study is Neo4j graph database [34]. Neo4j is a widely used open-source graph database implementation. It was chosen because of the abundant support and documentation that is available for the online community, and because of implementation familiarity as it is developed in JAVA. The application turns the information in the excel file to a meaningful graph database using the Neo4j graph database libraries. Neo4j provides basic graph database functions such as creating nodes and relationships, adding properties, and deleting nodes and relationships. Neo4j, in addition to its basic functions, provides many powerful features. It allows the implementation of traversals, graph algorithms like Dijkstra's algorithm, indexing and much more [34]. It is also important to note the Neo4j provides the tools to ensure that all database actions are performed in the proper and safe way.

The topology spreadsheet served as an input to the application. The network application translated the cities into graph database nodes with the city name as a property. The application also modeled the links as graph database edges with properties containing the length of each link. Figure 3.4 shows a visualization of the graph database after it is built. The visualization is obtained from NeoClipse [34]. NeoClipse is an open source 2D visualization tool for Neo4j graph databases. While the graph in Figure 3.4 doesn't show the right node location with respect to the map, it accurately shows the node interconnection links.

Figure 3.3 shows an overview of the components used in network modeling.

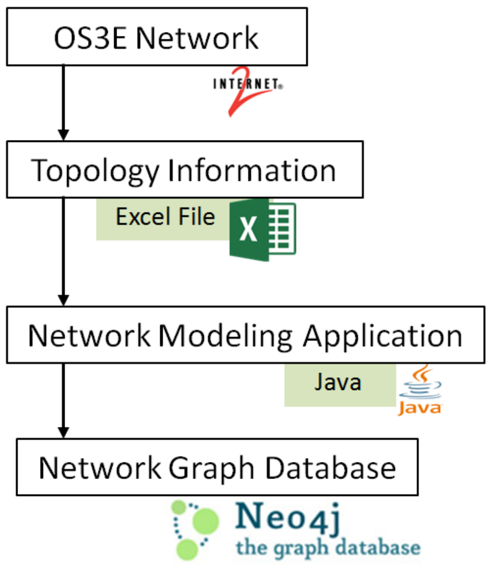


Figure 3.3: Applications used for network modeling

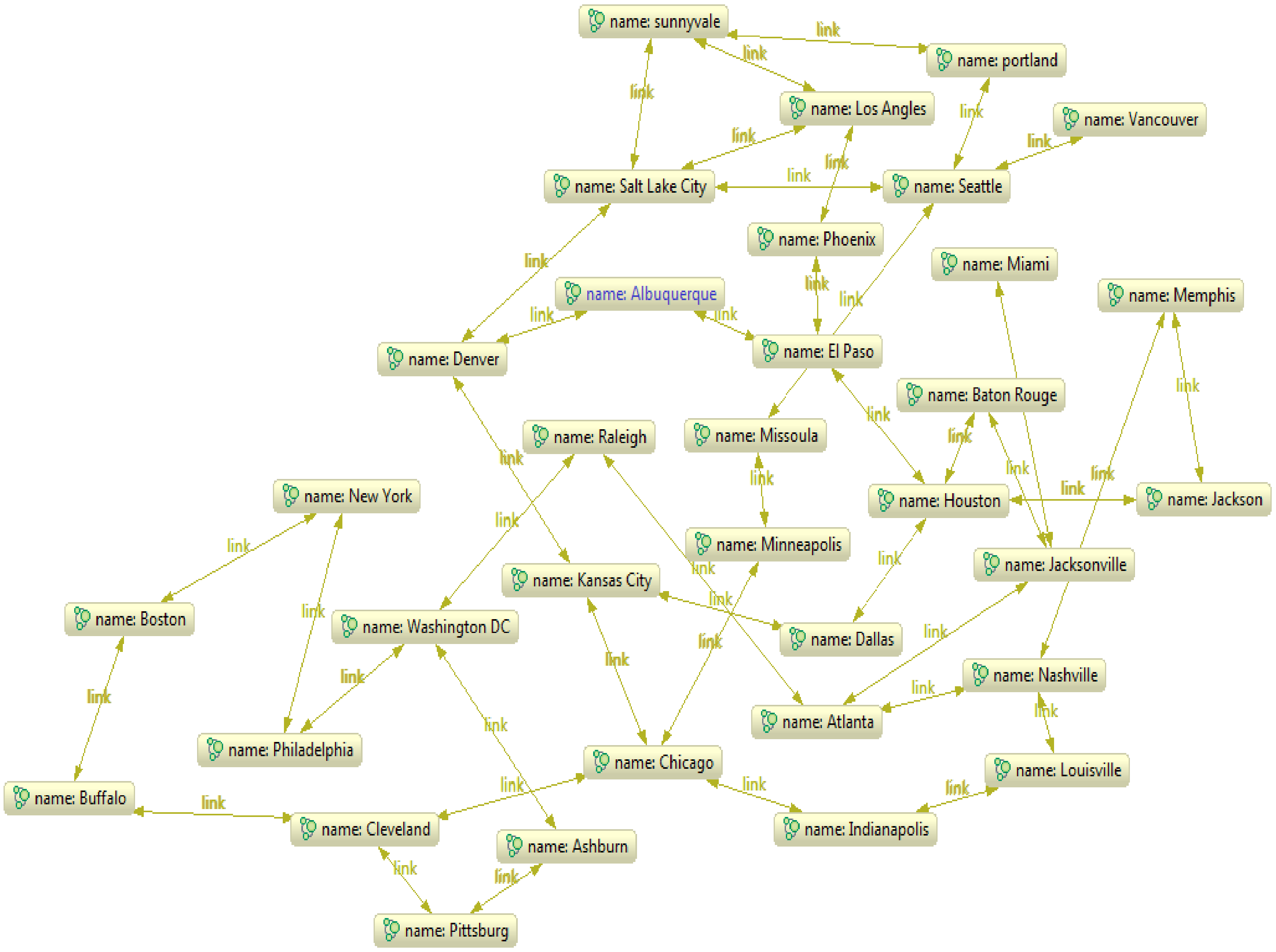


Figure 3.4: Visualization of the graph database representation of OS3E Network

Topology Route Generation

As mentioned before, the flows between all possible source and destination pairs will be analyzed. This requires determining the paths that the data packets will take from source to destination, as well as the paths that the control packets will take from the controller to the other nodes to set up the required paths and install the new network state. The path in the network depends on the nodes relative location as well as the links between them. The paths for control packets also depend on the controller location. The position of the network controller was initially chosen in Chicago based on the analysis results from [2] so as to minimize average latency from the network nodes to the controller.

One of the powerful features of Neo4j and graph databases is traversals. A traversal is a method of finding information in the graph database by visiting its nodes and examining their relationships [33]. In addition, there are graph algorithms defined in Neo4j. These graph algorithms require the following arguments: relationship type and the relationship direction. Some of the algorithms require a third argument, such as the Dijkstra's algorithm which also requires a property type of the relationships to weigh when determining the shortest path. These algorithms return a *PathFinder* object, which is a tool that can utilize the used algorithm to find paths in the graph.

In our case, the algorithm that was chosen to find the path between any pair of nodes was the Shortest Path which is based on finding the path with the least number of path links (minimum number of nodes). Other algorithms, such as A* search algorithm and Dijkstra's algorithm, were available and can be based on the physical length of the path links. However, the Shortest Path algorithm was used for simplicity.

Figure 3.5 shows an example of the paths that are found between two nodes: Chicago and Los Angeles. The solid arrow represents the optimal path according to the shortest path algorithm, and the dotted arrows represent some of the other possible non-optimal paths that were eliminated by the algorithm. The graph on the top left corner shows the corresponding physical paths in the OS3E network map.

Once the path is determined between two nodes, an Iterator can be attached to the path to iterate over all the path nodes. At each node sources of possible network delays can be calculated to simulate a realistic packet flow through the path.

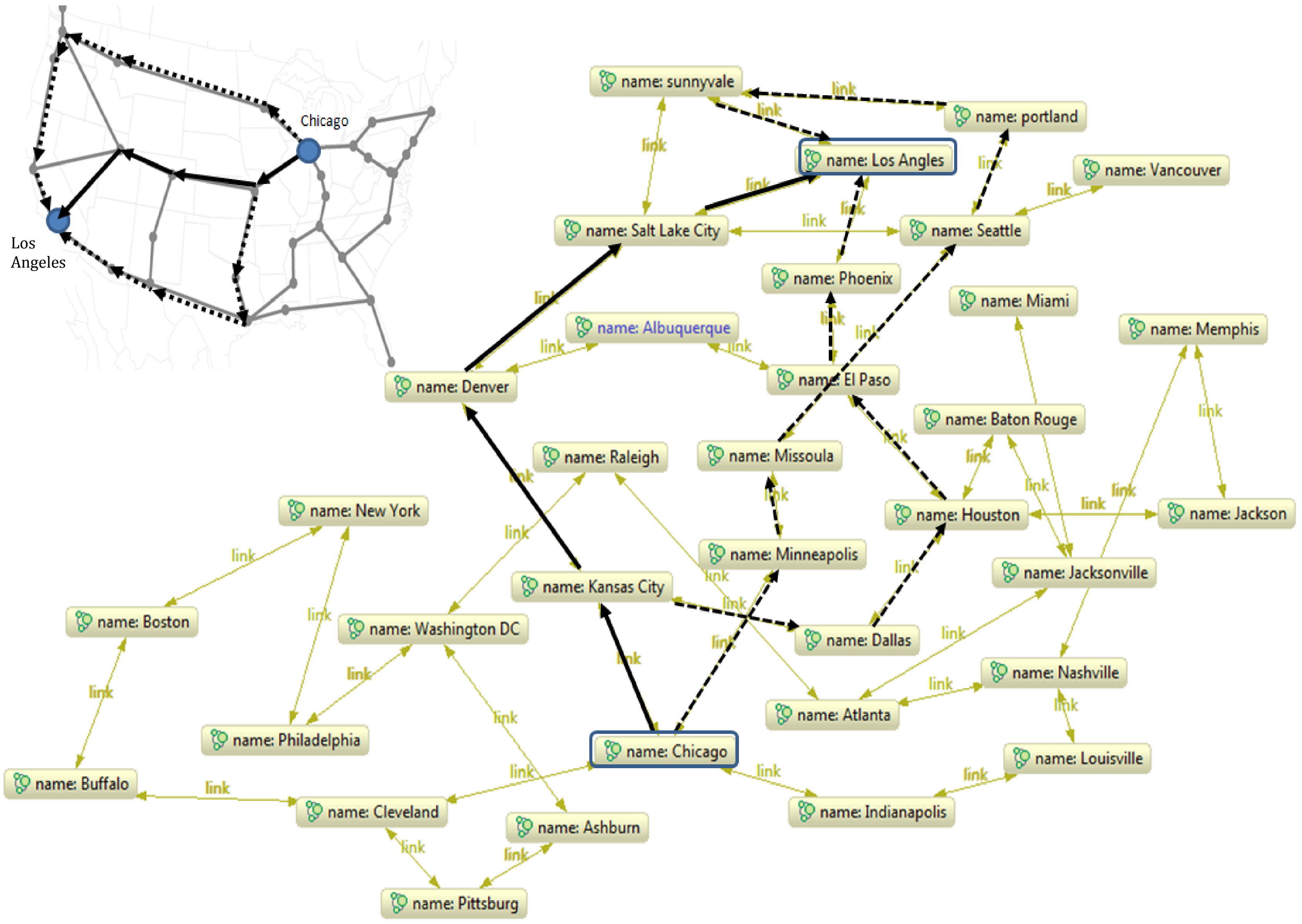


Figure 3.5: Determining the shortest path between two nodes

3.3.3. Network Delays

Sources of network delays were taken into account: propagation delays and transmission delays. Queuing delays were not modeled in the analysis as it was found that a packet level simulation would provide much better representation of the effects of network congestion delays. Switch state installation delays were also excluded as they are highly dependent on switch implementation.

Propagation delays depend on the physical distances in the topology of OS3E network between each pair of the 34 nodes, which are located in 34 different cities across the US. OS3E is a fiber optic network, so distances were divided by (2x10^8 m/s) to obtain propagation delays.

$$propagation\ delay = \frac{Link\ length\ (mile) \times 1.609\ \left(\frac{km}{mile}\right) \times 1000\ (\frac{m}{km})}{2 \times 10^8\ (\frac{m}{s})}$$

Transmission delays reflect the link rates used in the network, which are assumed to be 1Gbps, and depend on packet sizes. Random normal distributions with averages of 1KB and 100 Bytes are used for data and control packets respectively.

$$transmission\ delay = \frac{packet\ size\ (Byte) \times 8\ (\frac{bits}{Byte})}{1Gbps}$$

Table 3.1 gives an idea about the ranges of delays used to obtain the presented analysis results. However, varying the analysis inputs had little material impact on the relative results.

Table 3.1: Ranges of network delays used in the analysis

Delays		Min	Average	Max
Propagation (one way)		0.17 ms	2.04 ms	5.8 ms
Transmission	Data	-	8 µs	-
	Control	-	0.8 µs	-

3.3.4. Analysis Comparison Scenarios

In this analysis two routing approaches are compared: conventional hop-by-hop routing, and source routing. These scenarios are explained below using an example of a new unknown flow that originates from Seattle and is destined to Los Angeles. The controller is based in Chicago, and the path that is defined for the flow is (Seattle, Portland, Sunnyvale, Los Angeles), refer to Figure 3.5.

Traditional Hop-by-hop

In this approach, when a new flow arrives at the ingress point “Seattle”, the ingress switch contacts the controller at “Chicago”. The controller decides the path and before replying back to the ingress switch, it distributes new state information to the intermediate switches so that they can handle the flow when it arrives without the need to contact the controller. The controller waits for acknowledgements that the new actions were successfully installed in the intermediate switches’ flow tables before replying back to the ingress switch. Because the controller distributes the new state to the intermediate switches simultaneously only the longest roundtrip time between the controller and a intermediate switch is taken into consideration for convergence.

In this approach, the time that the first packet of the flow takes to get to destination includes:

- Roundtrip propagation between Seattle and Chicago
- Highest Roundtrip propagation of the following:
    - between Portland and Chicago
    - between Sunnyvale and Chicago
    - between Los Angeles and Chicago
- Propagation from Seattle -> Portland -> Sunnyvale -> Los Angeles

Source Routing

In the source routing approach, when a new flow arrives at the ingress point "Seattle", the ingress switch contacts the controller at "Chicago". The controller determines the path and then sends the path information back to the ingress switch. The ingress switch will add the path headers to the corresponding packets and forwards them accordingly. The intermediate switches need no communications with the controller as they will just rely on the packet path headers for forwarding.

In this approach the time that the first packet of the flow takes to get to destination includes:

- Roundtrip propagation between Seattle and Chicago
- Propagation from Seattle -> Portland -> Sunnyvale -> Los Angeles

Figure 3.6 below is a flow chart that shows the details of the analysis as a simulated packet flows from source to destination.

Figure 3.6 outlines a flow chart describing all the steps used to carry out the computational analysis.

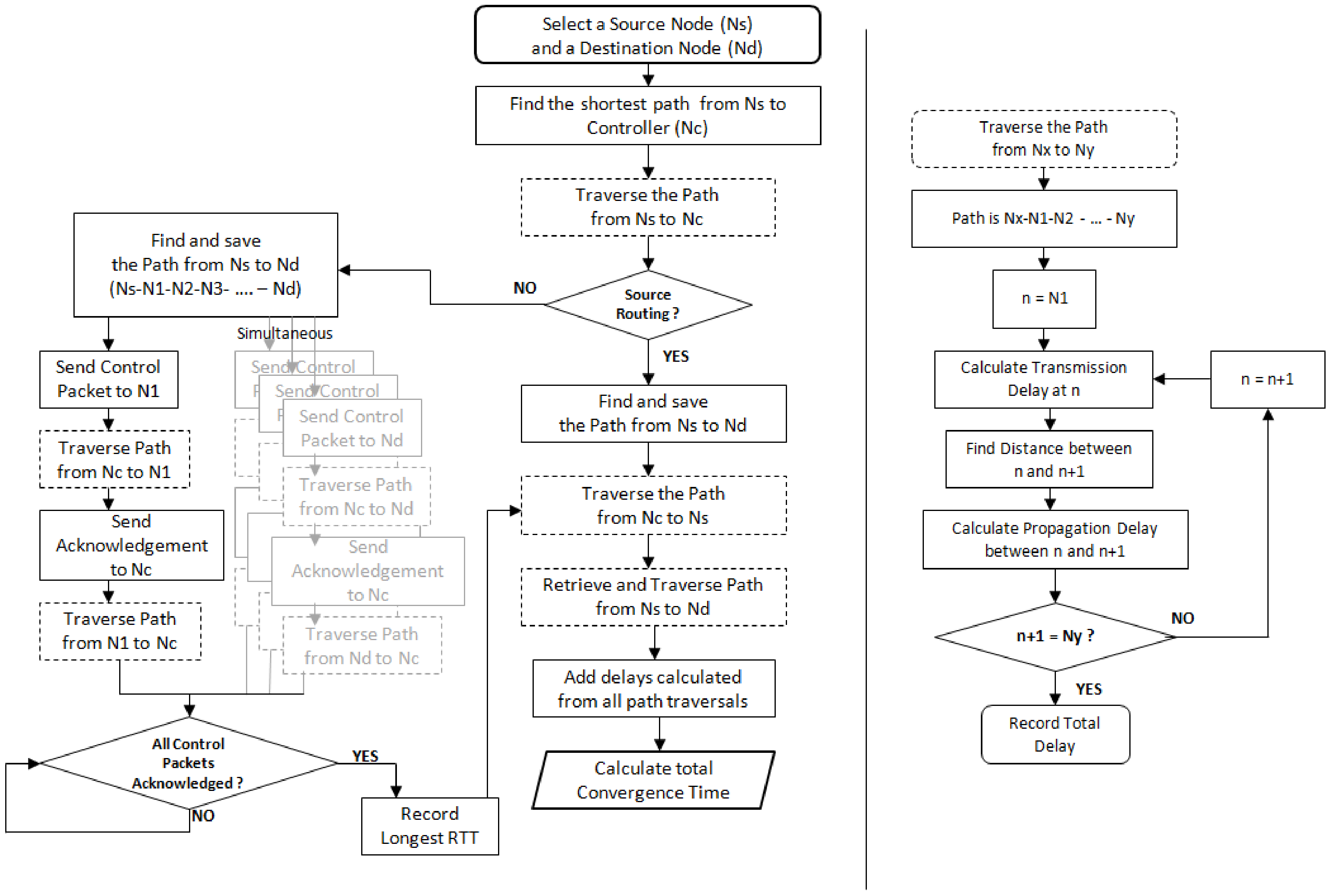

Select a Source Node (Ns) and a Destination Node (Nd)
Find the shortest path from Ns to Controller (Nc)
Traverse the Path from Ns to Nc
Source Routing ?
NO
YES
Find and save the Path from Ns to Nd (Ns-N1-N2-N3- …. – Nd)
Simultaneous
Send Control Packet to N1
Traverse Path from Nc to N1
Send Acknowledgement to Nc
Traverse Path from N1 to Nc
Send Control Packet to Nd
Traverse Path from Nc to Nd
Send Acknowledgement to Nc
Traverse Path from Nd to Nc
All Control Packets Acknowledged ?
YES
NO
Record Longest RTT
Find and save the Path from Ns to Nd
Traverse the Path from Nc to Ns
Retrieve and Traverse Path from Ns to Nd
Add delays calculated from all path traversals
Calculate total Convergence Time
Traverse the Path from Nx to Ny
Path is Nx-N1-N2 - … - Ny
n = N1
Calculate Transmission Delay at n
n = n+1
Find Distance between n and n+1
Calculate Propagation Delay between n and n+1
n+1 = Ny ?
NO
YES
Record Total Delay

Figure 3.6: Analysis flow chart

3.4. Analysis Results and Discussion

As mentioned before, this analysis was conducted based on network convergence time as a primary metric. The analysis intends to shed the light on four main areas that can be affected by the difference in routing dynamics between traditional hop-by-hop routing and source routing in an SDN-based WAN. The four main areas are state reduction, network convergence time, network performance with respect to scalability consideration, and network sensitivity to controller placement options.

3.4.1. State Reduction

One of the major differences between source routing and hop-by-hop routing is the dynamics of state distribution. In source routing, new state information will be pushed to only one node, the ingress switch, instead of having to distribute state to all the nodes along the determined path for the flow. The intermediate nodes won't need to receive state information as they will inspect the newly added source routing header for forwarding decisions. This leads to a significant reduction in the state distribution proportional to the length of the path from source to destination according to the equation below, which is plotted in Figure 3.7.

State Reduction = 100-(100/n) ; n is the number of links in the path.

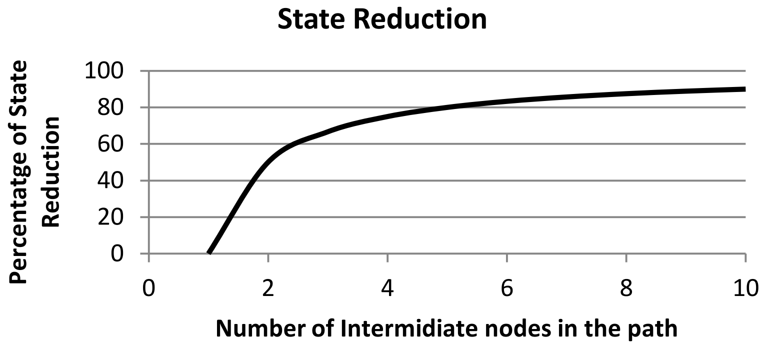


Figure 3.7: State reduction using source routing

As shown in Figure 3.7, using source routing can lead to significant results in state reduction as the path length increases.

An elementary set of results were collected using the OS3E network for the percentage of state reduction that can be achieved in this network using source routing. These results depended on calculating the number of path links between each source and destination pair, and using it to find the average path length from one source to all destinations, and consequently, the average state reduction corresponding to flows originating from each source. The results are shown in Table 3.2 below.

The state reduction achieved in the whole network averaged over all the source-destination pairs is found to be about 77.6%, which is considered a significant improvement in performance.

Table 3.2: Results for State Reduction in OS3E network using Source Routing

City	Average Path Length	Average State Reduction (%)	City	Average Path Length	Average State Reduction (%)
Vancouver	**5.5**	**81.8**	**Pittsburg**	4.27	76.6
Philadelphia	**5.5**	**81.7**	**Jackson**	4.24	76.4
New York	5.4	81.4	**Salt Lake City**	4.21	76.2
Miami	5.3	81.0	**Atlanta**	4.21	76.2
Portland	5.2	80.6	**Baton Rouge**	4.09	75.6
Boston	5.1	80.2	**El Paso**	4.06	75.4
Sunnyvale	4.8	79.1	**Nashville**	4.06	75.4
Raleigh	4.7	78.7	**Albuquerque**	4.03	75.2
Washington DC	4.7	78.5	**Louisville**	4.03	75.2
Ashburn	4.6	78.3	**Minneapolis**	3.84	74.0
Phoenix	4.6	78.1	**Indianapolis**	3.75	73.3
Seattle	4.5	77.8	**Cleveland**	3.72	73.1
Buffalo	4.5	77.8	**Denver**	3.69	72.9
Los Angeles	4.5	77.7	**Houston**	3.66	72.7
Memphis	4.4	77.2	**Dallas**	3.63	72.5
Missoula	4.3	76.7	**Kansas City**	**3.3**	**69.7**
Jacksonville	4.3	76.7	**Chicago**	**3.24**	**69.1**

3.4.2. Network Convergence Time

Convergence time was calculated for all the flows between every source and destination pair. Convergence times were averaged for all the flows originating from the same source to all the possible destinations, Figure 3.8. The results clearly show that the average convergence time for all the source nodes in the case of source routing is substantially lower than the case of hop-by-hop routing, while keeping the same trend of variation.

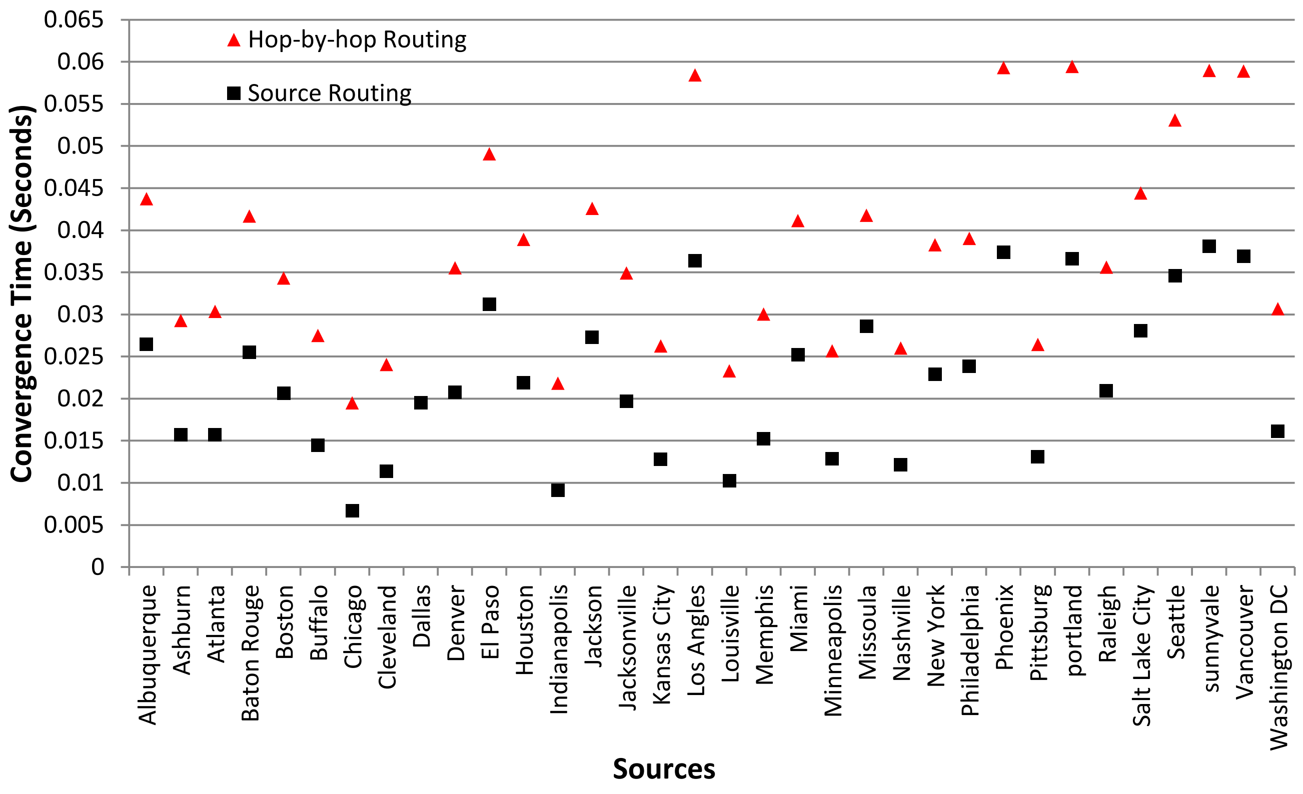


Figure 3.8: Average network convergence time

In this network, using source routing achieves a 41.7% decrease in the average convergence time compared to hop-by-hop routing. Results also show that in source routing, 92% of the flows have convergence time less than 40ms, against only 41% for hop-by-hop routing, Figure 3.9. This is considered a significant improvement in

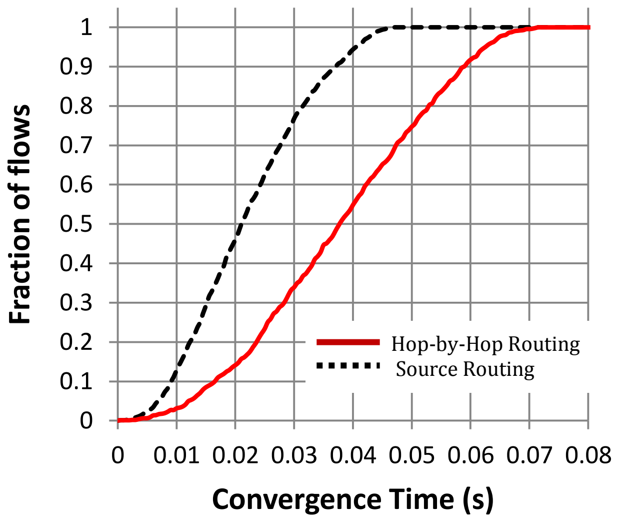


Figure 3.9: Cumulative distribution curve for network convergence times

overall network convergence performance that directly affects flow initialization as well as network fault recovery.

Another way of analyzing the results is from the perspective of one source to all its destinations. For example, Figure 3.10 below shows the convergence time from Seattle to all the other destinations.

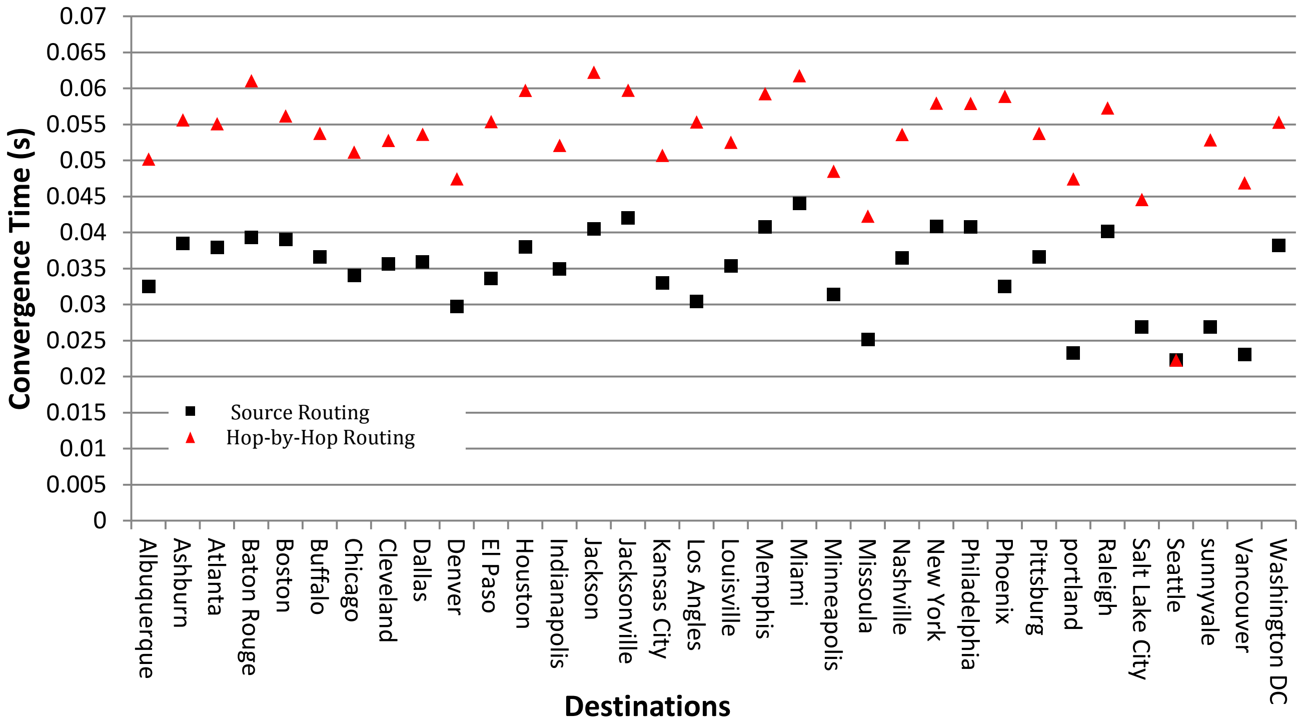


Figure 3.10: Network convergence time from one source to all destinations

The results in Figure 3.10 show that the convergence times in the case of source routing show less variations than in the case of hop-by-hop routing. This is simply because in traditional routing the convergence time includes the roundtrip time from each intermediate switch to the controller. This parameter of stability can be quantified using standard deviation measurements from the average convergence time.

According to Table 3.3, using source routing brings a 44.5% decrease in standard deviation compared to traditional routing in this case. Having a stable convergence time over all destinations provides better performance quality and allows better control for Service Level Agreements (SLAs).

Table 3.3: Standard Deviation of network convergence time

Routing Approach	Standard Deviation
Source Routing	4.6 ms
Hop-by-hop Routing	8.3 ms

3.4.3. Scalability

As propagation latency is a major factor in network convergence time in SDN WANs, the performance of source routing was compared to traditional routing as the network scales in diameter. Scalability analysis covered two steps. First, a simple analysis was conducted to study the effects of increasing the network diameter while keeping the same number of nodes, i.e., using an incremental multiplication factor against the original interconnection distances. Second, a more thorough analysis was conducted to study four cases of different network radii along with different numbers of nodes in the areas covered by these radii, Table 3.3. Radii were measured from the geographical center of the network (Kansas City).

Table 3.4: Scalability analysis for different network radii scenarios

Radius (miles)	Number of Nodes Included
700	10
1000	18
1300	26
2100	34

Scalability analysis results show that although the overall network convergence time naturally increases in both approaches with increasing network radius multiplication factors, i.e. using an increasing multiplication factor with link lengths when calculating propagation delays, the rate of increase for source routing is 42% lower than the rate of increase for hop-by hop routing, Figure 3.11(a).

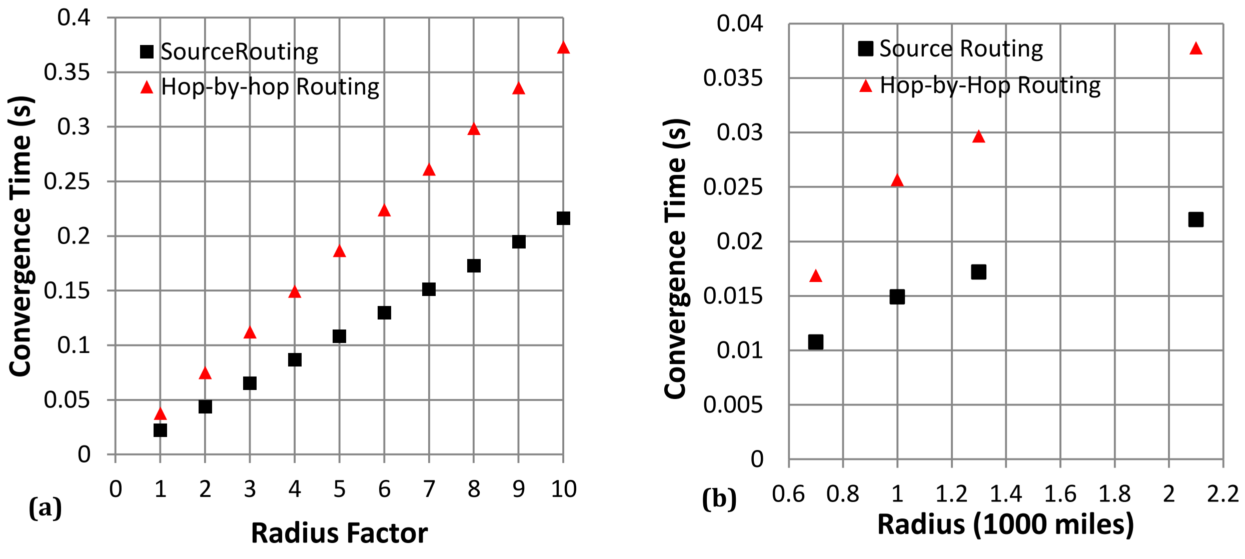
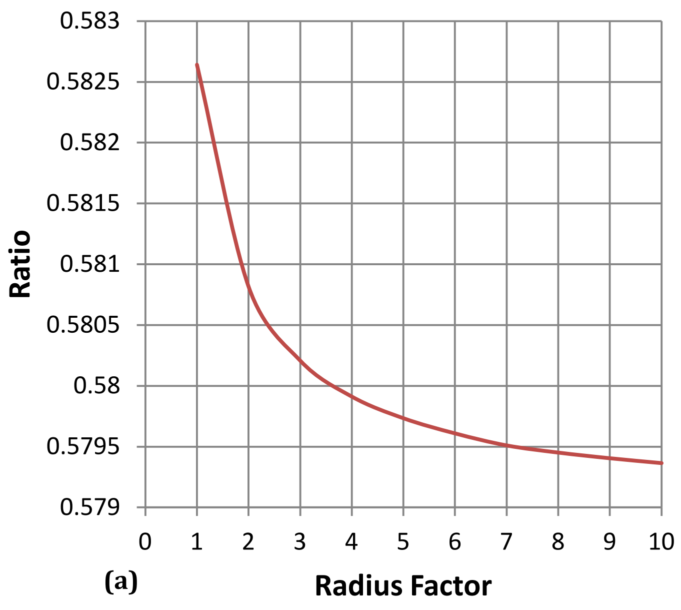


Figure 3.11: Convergence time as the network expands (a) by using a radius multiplication factor with constant nodes, (b) by considering different network radii

The first part of this scalability analysis focused solely on the effects of increasing distances, and consequently increasing the propagation latencies, however, the second part of this analysis incorporates an increase in the number of nodes along with the increase in radius. Results, plotted in Figure 3.11(b), show that source routing achieves a 46% lower average rate of increase than hop-by-hop routing.

Figure 3.12(a, b) provide a different way of looking at the results considering the ratio between the source routing convergence time and hop-by-hop convergence time. The graphs show that as the radius increases, solely by delay or with increasing the number of nodes, source routing scales better in a decaying exponential relationship proportional to the network radius.



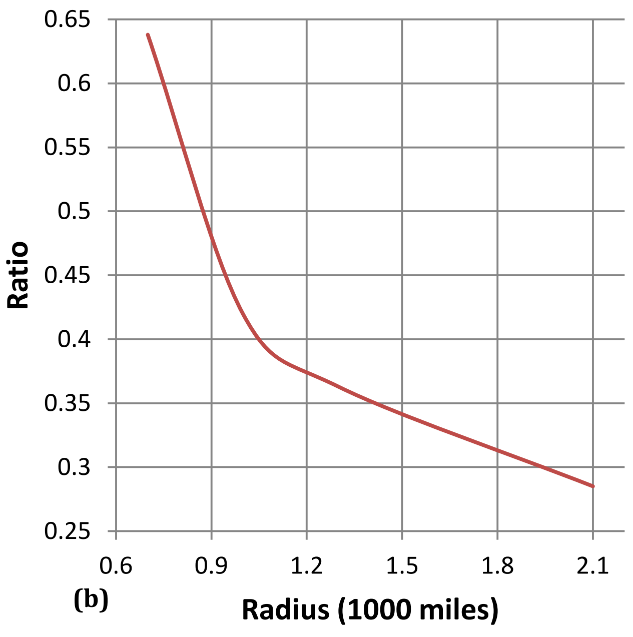


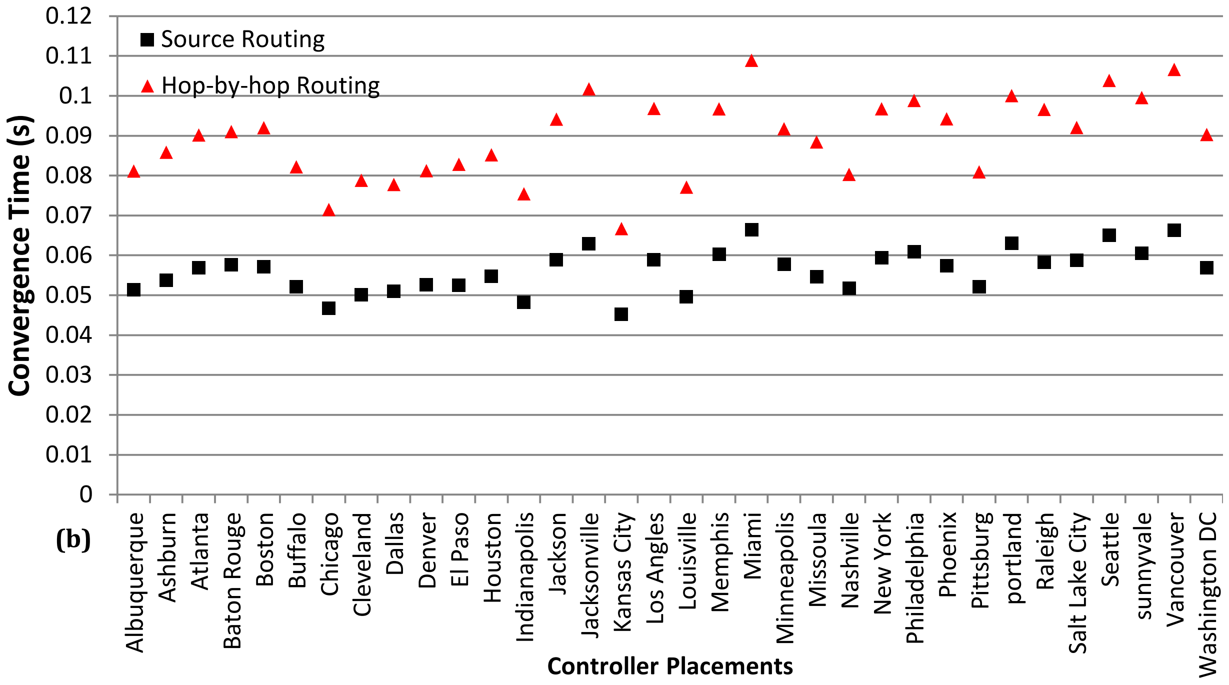
Figure 3.12: Ratio of source routing convergence time to hop-by-hop routing as the network expands (a) by using a radius multiplication factor with constant nodes, (b) by considering different radii.

3.4.4. Controller Placement

Network convergence time was analyzed for all source-destination pairs using all the possible controller placements over the 34 different nodes of the OS3E topology iteratively. The controller placement analysis in [2] relied on two performance metrics: average and worst-case node-to-controller latency. Similarly, our analysis relies on two performance metrics: average convergence time, and worst-case convergence time using both source routing and traditional hop-by-hop routing.

First, we note that our results agreed with the results obtained in [2] regarding optimal controller placements for both metrics with hop-by-hop forwarding. The results show that Chicago is the optimal placement for minimum average convergence time, and Kansas City is optimal placement for minimum worst-case convergence time.

Second, comparing source routing and hop-by-hop routing for each possible controller placement reveals that source routing performs better in all placements for both metrics, Figure 3.13(a, b). Source routing achieves a 41% and 32% reduction in average convergence time and worst-case convergence time, respectively, for the optimal controller placement. Moreover, source routing achieves an average of 45% and 37% reduction over all the other placements in average convergence time and worst-case convergence time respectively.



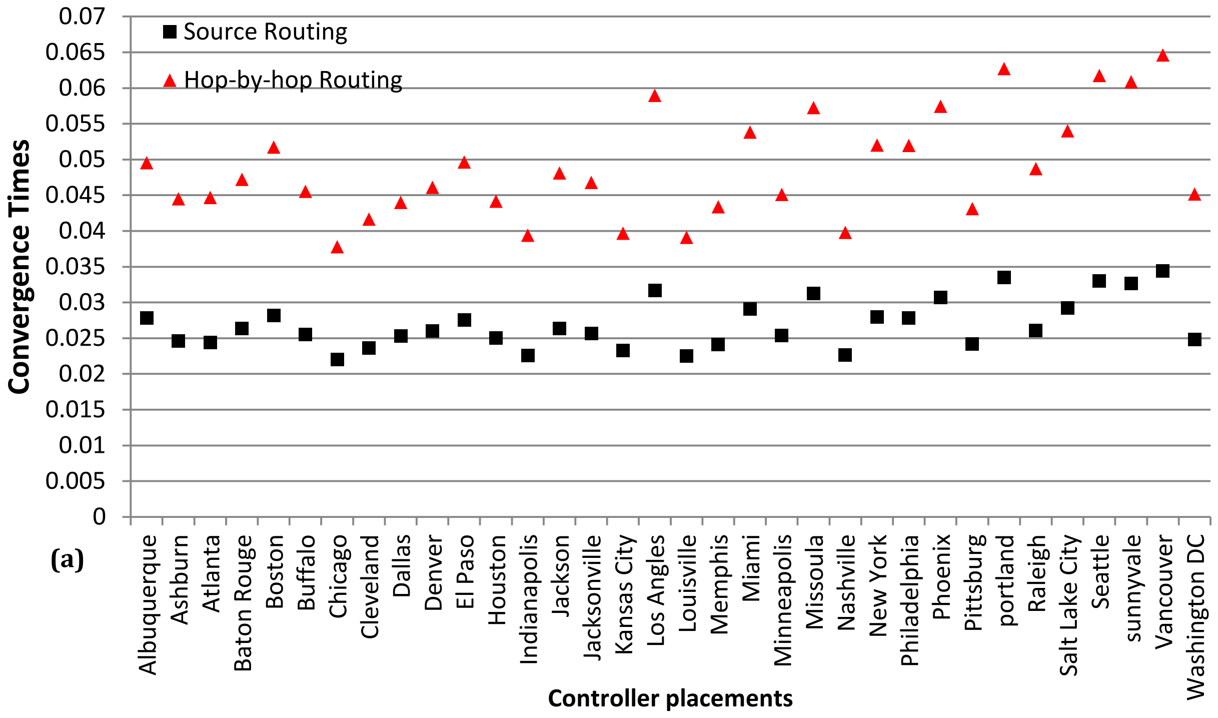


Figure 3.13: (a) Average convergence times for all placements, (b) Worst-case convergence time for all placements.

Third, Figure 3.13(a) shows that the average convergence times for the different placements are more variable in the case of hop-by-hop routing than in source routing. This is reflected by a 53% reduction in the standard deviation of the average convergence times over all the placements when using source routing. Figure 3.13(b) shows a similar behavior for the worst-case convergence time, as source routing brings a 47% reduction in the standard deviation over all the placements. This means that when source routing is used, in addition to reduced convergence times, less variation in convergence performance will be experienced with different controller placements, giving network design engineers more flexibility in placing network controller(s), or expanding networks without changing controller placement while still keeping acceptable levels of network convergence times.

Finally, to show the difference between the two controller placement metrics, the analysis in [2] finds that when optimizing for average latency, 90% of the time, the worst-case latency is within 50% of optimal. The same results were obtained for hop-by-hop routing in our analysis. However, our analysis shows that when optimizing for average convergence time, in the case of source routing, 90% of the placements keep the worst-case convergence time within only 40% of optimal. Moreover, 50% of the placements in the case of source routing keep the worst-case convergence time within only 20% of optimal. The analysis in [2] concludes that a trade off has to be made between the two placement metrics. While this conclusion still holds true with our analysis, to an extent, the effects of the trade-off using source routing are proven less severe, giving network design engineers more flexibility on optimizing controller placements while keeping acceptable levels of both average and worst-case convergence times.

3.5. Summary

In this chapter, we illustrated that source routing can stand as an alternative approach to conventional hop-by-hop routing in SDN-based WANs. The analysis discussed in this chapter focuses on the applicability of source routing to address major problems in SDN related to state distribution, network convergence, scalability, and controller placement.

Source routing has been demonstrated to provide significant gains in network convergence time. Analysis also shows that source routing provides better scalability than hop-by-hop routing in WAN as the network diameter increases. In source routing, the controller only communicates to ingress nodes which limits the growth of the controller burden as the network expands, providing a more scalable solution compared to hop-by-hop routing.

Source routing brings a new dimension to the controller placement problem as it limits the convergence performance variation experienced across different controller placements in the network. This allows for greater flexibility in controller placements when optimal placements cannot be achieved for various reasons without sacrificing acceptable performance guarantees. Controller placement flexibility is also enhanced using source routing as it eases the trade-off between the two analyzed controller placement metrics: average and worst-case convergence time.

4. SOURCE ROUTING MODEL IMPLEMENTATION

In this section, a detailed description of the proposed source routing solution is provided. The proposed alternative SDN routing approach is implemented in our study as a simulation model in ns-3. Therefore an overview about ns-3 and the current relevant models is given, followed by a discussion of the enhancements and the implementation details that led to the development of the simulation models that will be utilized to drive our experiments and findings.

4.1. Source Routing Model Architecture

As mentioned before, SDN deployment in WAN environments impose limitations and issues that need to be addressed. A general argument can be made that in the case of proposing a centralized control of a significant number of network nodes (as in SDN), the scale, speed, cost and quality of the solution can be significantly improved by reducing the state to be distributed by the controller to the network nodes [35].

In SDN, the centralized control maintains knowledge about the entire network of nodes, and in some cases the traffic demands and characteristics. The controller is responsible for configuring the forwarding tables of the forwarding nodes with new state information. This procedure of state distribution can be done in a pro-active approach during network initialization or over specified intervals. It can also be done in a reactive approach as a response to requests for new state from the forwarding devices as a result of receiving an unknown flow.

As mentioned before, OpenFlow is the protocol widely used for communication between the controller and the data plane network devices. Using this protocol the controller installs n-tuple matching rules on every hop along any path from source to destination, along with a matching forwarding action for packets that match these rules.

This chapter will introduce an SDN routing alternative that aims to reduce the state distributed to the network devices by the controller by leveraging means of source routing. Since the controller has information about the exact end to end paths between any source and destination in the network, it can insert this path information in a header that can be added to packets at the ingress node, inspected at every intermediate node, and stripped off at the egress node.

4.1.1. Strict Link Source Routing (SLSR)

An example is presented here to show how a path can be expressed in a source route using the link interfaces at each node along the path. This method represents Strict Link Source Routing (SLSR) [35], since the controller strictly forms the path from the links along the route from source to destination.

Figure 4.1 shows a simple network composed of five switches and a controller. The switches are named (SwitchA, SwitchB, SwitchC, SwitchD, SwitchE), and the interfaces on each switch are numbered locally (1, 2, 3, 4, etc).

If a flow is to go from SwitchA to SwitchE in the network, a possible path is: *SwitchA -> SwitchB -> SwitchD -> SwitchE***.** This path can be expressed as a SLSR Route as a sequence of *Switch(Forwarding Interface)* entries as follows: *SwitchA(2), SwitchB(3), SwitchD(3).* Since each forwarding interface leads to a known node in the network, the path can be even more simplified by eliminating the switch identifiers as follows: {2, 3, 3}.

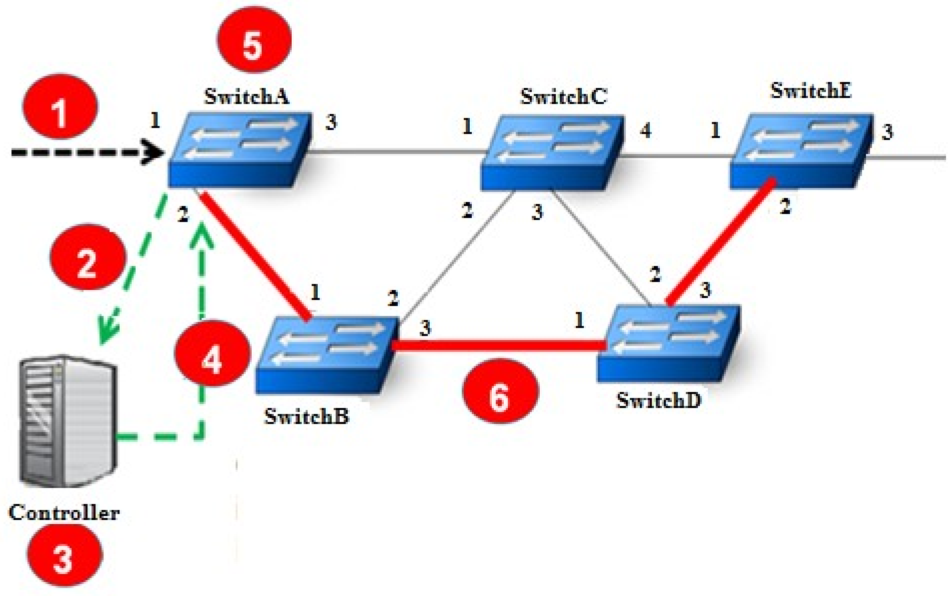


Figure 4.1: Source Routing in SDN Overview

The following steps, identified on Figure 4.1, show an example of the end to end actions that take place as a packet that belongs to an unknown flow traverses from source to destination.

[1] SwitchA receives a new packet.

[2] SwitchA performs a flow lookup in its flow table to match the new packet. When the packet is not matched, SwitchA sends a new flow request to the controller.

[3] Controller computes the path from source to destination using specific network algorithms and protocols, and utilizing the network-wide knowledge.

[4] Controller replies back to the ingress switch, SwitchA, with a flow table entry to be installed in its flow table to apply an action on the new packet.

[5] The ingress switch, SwitchA, receives the control packet and inserts the flow entry in its table, Figure 4.2. The flow applies the action on the packet by forming a header representing the path and appending the header to the packet. The same action will be performed on all following packets belonging to the same flow.

Source IP	Destination IP	Source Port	Destination Port	Action
10.10.0.5	10.10.0.6	1105	1205	Add SLSR {2,3,3}

Figure 4.2: Source routing flow table entry

The n-tuple matching rule in Figure 4.2 shows a simple example. Matching rules can be more complex and detailed than shown above.

[6] The path header can be embedded using various efficient encodings. A possible simple header format can be composed of 8 bits for each hop along with an 8-bit field to identify the location along the path, and a hop count, shown in Figure 4.3. This results in a 40 bit header, which is considered a very light weight header to represent a 3-hop path.

Path Header					Packet HEADERS	Packet PAYLOAD
2	3	3	Location	Hop Count		
			1	3		

Figure 4.3: Source routing path header

[7] The ingress switch uses the first hop forwarding link in the path header and forwards the packet through interface number 2 after incrementing the location identifier to 2.

[8] Each intermediate node along the path starting with SwitchB does the following:

- Verify if the *Location* identifier is greater than the *Hop Count*. If this is true then the switch is the egress switch.
- If not, the forwarding link entry at the corresponding *Location* is used.
- Increment the *Location* identifier.

In the case of SwitchB, the forwarding interface identifier at *Location (2) is* 3*.* The *Location* identifier is incremented from 2 to 3, and the packet is forwarded through interface number 3.

The same procedure is followed until the packet reaches SwitchE. At SwitchE, the *Location* identifier will exceed the *Hop Count* which means that packet reached the egress switch. The switch will strip the path header and match the packet to its forwarding table.

Comparing SLSR to traditional hop-by-hop routing in SDN reveals that for the same path (SwitchA->SwitchE) traditional hop-by-hop SDN routing approach will require 3 times the amount of state distribution. New flow table entries will have to be pushed down to SwitchA, SwitchB, and SwitchD, instead of just to the ingress switch, SwitchA. This also means that in the case of source routing the burden on the controller is reduced by 66%. This fact can be generalized as state distribution and consequently controller I/O burden in this case is linked to the average number of links along the path. Table 4.1 below shows an example of the reduction gains along multiple path lengths.

It is important to note that the number of paths in a network grows exponentially with the network size (number of nodes and the diameter). This makes the state distribution burden on the controller even greater. An argument can be made that the bigger the SDN network is in terms of number of nodes (reflected in the number of possible paths) and the diameter of the network (reflected in the average path length), the better source routing performs compared to hop-by-hop routing in terms of reducing state distribution and consequently controller's I/O burden [35].

Table 4.1: Reduction in state distribution achieved by Source Routing

Average Path Length	Percentage of State Distribution and Controller I/O burden **Reduction**
1	0%
2	50%
3	66%
4	75%
N	*(100-100/n) %*

The use of Strict Link Source Routing enables implementing additional features for reverse path calculation and local link failure recovery.

Reverse Path Calculation

As the packets traverse the defined path, the reverse path can be calculated by changing the output interface number in the path header with the input interface number at each intermediate node. Referring to Figure 4.4, when SwitchB receives the packet, the *Location* identifier value is 2, which points to the 2nd hop forwarding interface "3". Before *SwitchB* forwards the packet through Interface 3 and

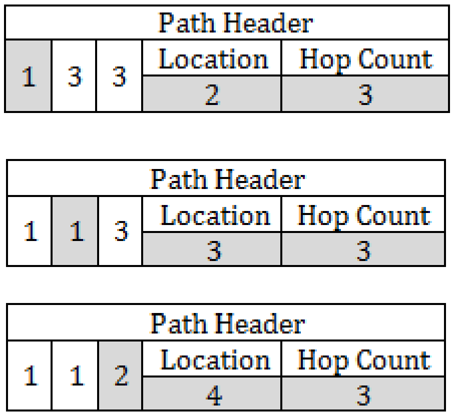


Figure 4.4: An example of reverse path calculation

increment the *Location* identifier, the entry at the previous location, 1st location, "3" in the path header can be changed to the input interface on which the packet arrived, interface number “1”. Performing the same operation at each intermediate switch leads to having a sequence of link identifiers that represent the exact symmetrical reverse path. The resultant path would be in the reverse order. For example, Figure 4.4 shows that the path will be {1, 1, 2}, but it is reversed. The path {2, 1, 1} is the path that can be used to traverse from SwitchE to SwitchA through the same route.

This path can be stored in a flow table entry at *SwitchE*’s flow table to handle packets destined to SwitchA. This eliminates the need for reverse flows to trigger communication to the Controller. Consequently, an extra 50% reduction in controller state distribution burden can be achieved [35].

It is important to note that the reverse path calculation as described above becomes irrelevant if path symmetry between two network nodes is not intended as part of the network design and traffic optimization.

Local Link Failure Recovery

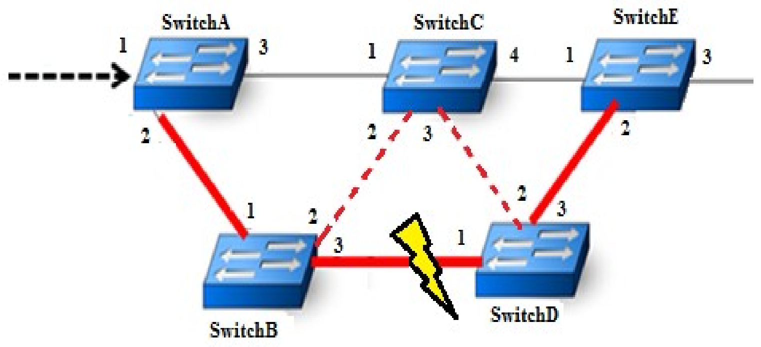
Typically, in the case of link failures, the switch where the link failure occurs informs the controller. The controller then re-computes alternative paths for the flows affected by this link failure and pushes the new path to these flows’ ingress switches. Flows will be affected during the period taken to inform the controller, re-computing new paths, and redistributed the new state.

Using SLSR allows implementing interesting features for fast local link failure recovery. This can be done by substituting a failed link identifier with an alternative link identifier or a sequence of link identifiers that merge with the flow’s path after the failure location. This will be done based on prior configurations at each intermediate switch that matches a link failure to a sequence of alternative identifiers to follow.

For example, in Figure 4.5, if packet reaches *SwitchB* and Interface 3 is found down, the switch can change the entry "3" in the path sequence with {2,3} to reach *SwitchD* but through *SwitchC*, based on pre-stored information.

This technique can be used locally and temporarily to prevent packet loss by detouring around the failure until alternative path is re-computed and new state is distributed to the flow ingress points by the controller.

(a)



(b)

Path Header				
2	3	3	Location	Hop Count
			2	3

Modified Path Header					
2	2	3	3	Location	Hop Count
				2	3

Figure 4.5: (a) Network link failure (b) Local link failure recovery

4.2. Network Simulator 3 (ns-3)

Ns-3 is an open source discrete-event network simulator. Ns-3 introduces an open simulation environment that addresses the needs of modern networking research and is enhanced by well documented community contributions [36]. Ns-3 infrastructure encourages the development of simulation models which are sufficiently realistic to simulate real network devices, topologies, and interconnections.

In ns-3 simulation environment, a generic term - that originated in graph theory - is used to describe the basic computing device abstraction, the node. This abstraction is represented by a C++ class "*Node*", that represents computing devices functionalities in the simulation [36]. In addition, the *Net Device* abstraction covers both hardware and software of the network interface cards (NICs), and is represented by a C++ class "*NetDevice*". Using the concepts of object-oriented programming, several customized versions of NetDevice class are created, such as *CsmaNetDevice*, *PointToPointNetDevice*, and *WifiNetDevice*. The *NetDevices* are installed on the *Node* entities. To represent the media over which the data flows in that network, ns-3 defines the *Channel* abstraction that is represented by *Channel* class, from which customized versions are created, such as *CsmaChannel* and *WifiChannel*. These classes provide methods for managing communications objects and connecting nodes together through *Net Devices*.

To simulate large networks in ns-3 one would need to arrange many connections between *Nodes*, *NetDevices*, *Channels*, as well as install protocol stacks, assign IP addresses, etc. Ns-3 provides *Topology Helpers* to perform these common tasks that increase in complexity with large networks, and combines them into models that are easy to use [36].

4.3. OpenFlow Switch Model

One of the ns-3 models that are of great interest to this research is the OpenFlow switch model. This model is part of the standard ns-3 libraries and provides a representation of the basic functionalities on an OpenFlow Switch. This model provided a baseline that allowed for enhancements and expansions for implementing the source routing functionalities and conducting SDN-based WAN simulations.

The OpenFlow Switch Model relies on OpenFlow standard specifications [37]. However, there are two major differences between this model implementation and real OpenFlow switches. First, there is no secure channel between the switch and the controller. The controller in this case is internal, i.e. the switch doesn't send control packets to the controller but rather calls controller functions directly whenever needed. Second, the model uses a virtual flow table (TCAM), as opposed to hardware TCAM typically used by OpenFlow switches [36]. Despite of these limitations, the model provided a baseline that allowed several modifications, as will be described in the following subsections, to enable more realistic simulations of SDN networks in ns-3.

Figure 4.6 shows the overall architecture of the existing OpenFlow switch model in ns-3 compared to a traditional Node in ns-3.

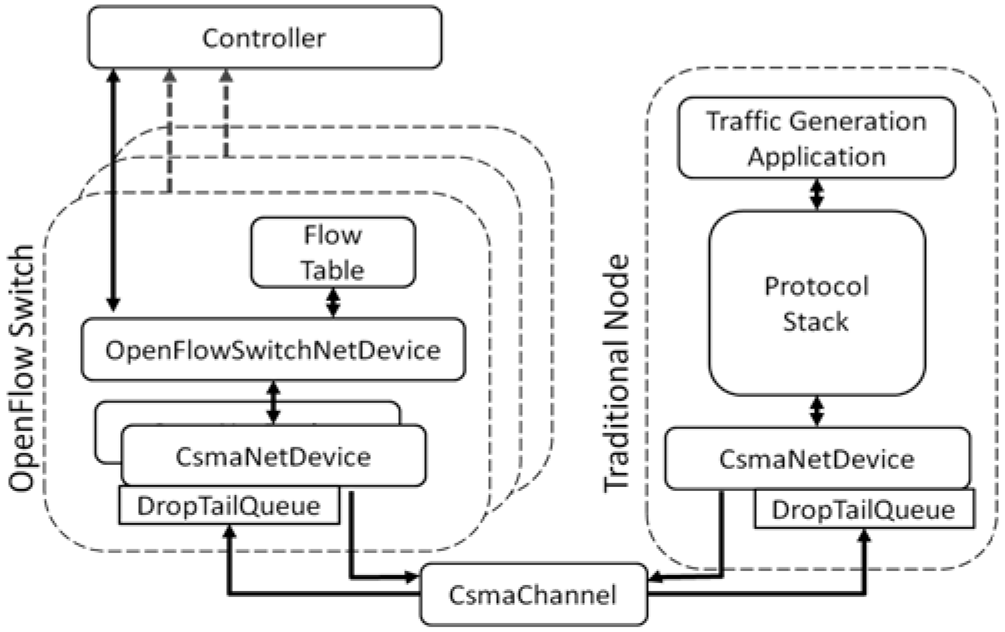


Figure 4.6: Architecture of the OpenFlow switch model in ns-3, and a Traditional Node

The following is a description of the modules that compose the architecture in Figure 4.6:

<u>CsmaNetDevice</u>

This module is used to represent Ethernet ports in ns-3. The module corresponds to layer 1 and 2 in the protocol stack. The OpenFlow switch model restricts the choice of what type of *NetDevices* to use to *NetDevices* with specific APIs compatible with the provided implementation of the OpenFlow model. This restriction was a reason for using the *CsmaNetDevice*. This module is used in our simulations with CsmaChannel, as will be described later, for simulating Ethernet point to point connections in the simulated SDN network.

DropTailQueue

This module represents one of the optional attributes that can be attached to the *CsmaNetDevice*. The *DropTailQueue* is a simple FIFO queue that drops incoming packets when the queue is full. The queue has two limitation modes: maximum packets, or maximum bytes.

CsmaChannel

This module represents the link that will be used between any two nodes in our simulations. As mentioned before, the current OpenFlow switch model imposes a limitation on the compatible *NetDevices* to use. This led to choosing the CsmaNetDevice module as a representation of the network port. Consequently, the *CsmaChannel* was chosen as the representation for the link between the ports of different nodes. Although not originally intended for WAN links, the CSMA channel model provides parameters, and room for customization that was utilized to make this model suitable for the purpose of our simulations. The *CsmaChannel* provides two important attributes: *DataRate* which sets the transmission rate used by the *CsmaNetDevices* at the ends of the *CsmaChannel*, and *Delay* which sets the propagation delay for the channel. These parameters allow the channel to be customized to simulate the delays experienced by the network links that will be simulated.

The *CsmaChannel* is intended for Multiple Access to simulate carrier sensing and collisions; however, in our simulations collisions don't occur. The *CsmaChannel* in our context is used as a point-to-point link, and two channels are used concurrently between any two nodes to represent full-duplex operation. Carrier sensing in this case is disabled, so that the channel will not prevent transmission and schedule back-off/retransmission if CsmaNetDevices requires a transmission while packets are already propagating on the channel. In the absence of an approved ns-3 model for a full-duplex fiber channels, this is a recommended implementation by several implementation discussions on the ns-3 group design forums.

Protocol Stack

In a traditional node, the protocol stack provides ARP operations, IPv4 addressing and interfaces (Layer 3) and UDP protocol operations (Layer 4). The protocol stack can be utilized to run different traffic generation applications. ns-3 provides simple traffic generation applications and allows for developing customized applications.

OpenFlowSwitchNetDevice

This is the main module of the OpenFlow switch implementation in ns-3. This module relies on a collection of *NetDevices* that are set as ports. The *OpenFlowNetDevice* then controls these *NetDevices* and forwards packets from one to the other. The module also includes a software configurable flow table. The flow table contains entries that relate matching keys to actions to be performed on the packets that match those keys. The flow table entries are installed by the OpenFlow controller using OpenFlow messaging between the *OpenFlowNetDevice* and the controller. The OpenFlow configuration messages used in this model are kept in the same format as the real OpenFlow specifications. The module provides a configurable attribute *FlowTableLookupDelay* which models the delay taken by performing a lookup in the flow table.

Below is a list of some of the actions that are supported by the model:

– OFPAT_OUTPUT	: Output flow to a specified port
– OFPAT_SET_VLAN_VID, OFPAT_SET_VLAN_PCP	: Change VLAN Tags
– OFPAT_SET_DL_SRC, OFPAT_SET_DL_DST	: Modify Data Link Layer header
– OFPAT_SET_NW_SRC, OFPAT_SET_NW_DST	: Modify Network Layer header
– OFPAT_SET_TP_SRC, OFPAT_SET_TP_DST	: Modify Transport Layer header
– OFPAT_SET_MPLS_LABEL, OFPAT_SET_MPLS_EXP	: Modify MPLS Labels

<u>The Controller</u>

The controller is the entity responsible for installing entries in the flow table attached to the *OpenFlowSwitchNetDevice*. The controller builds the flow table entries from: an action structure that describes what to be done with the flow packet when it is matched, and the information "keys" that will be used to match the flow. In ns-3, Controllers can be added by the users under a specific name space extending the parent class (*ofi::Controller*) [36].

As mentioned before, the controller in this implementation is internal. This means that the controller doesn't reside on a separate node. The controller can be accessed by *OpenFlowSwitchNetDevice* modules locally through specific APIs. This means that the communication between the controller and the switches happens instantly with no delays. This is considered a major limitation for the OpenFlow switch model in ns-3, because it prevents the ability to appropriately simulate flow insertion delays, and network convergence times.

Figure 4.7 below is a flow chart that shows a high level description of the Open Flow Switch model functionality.

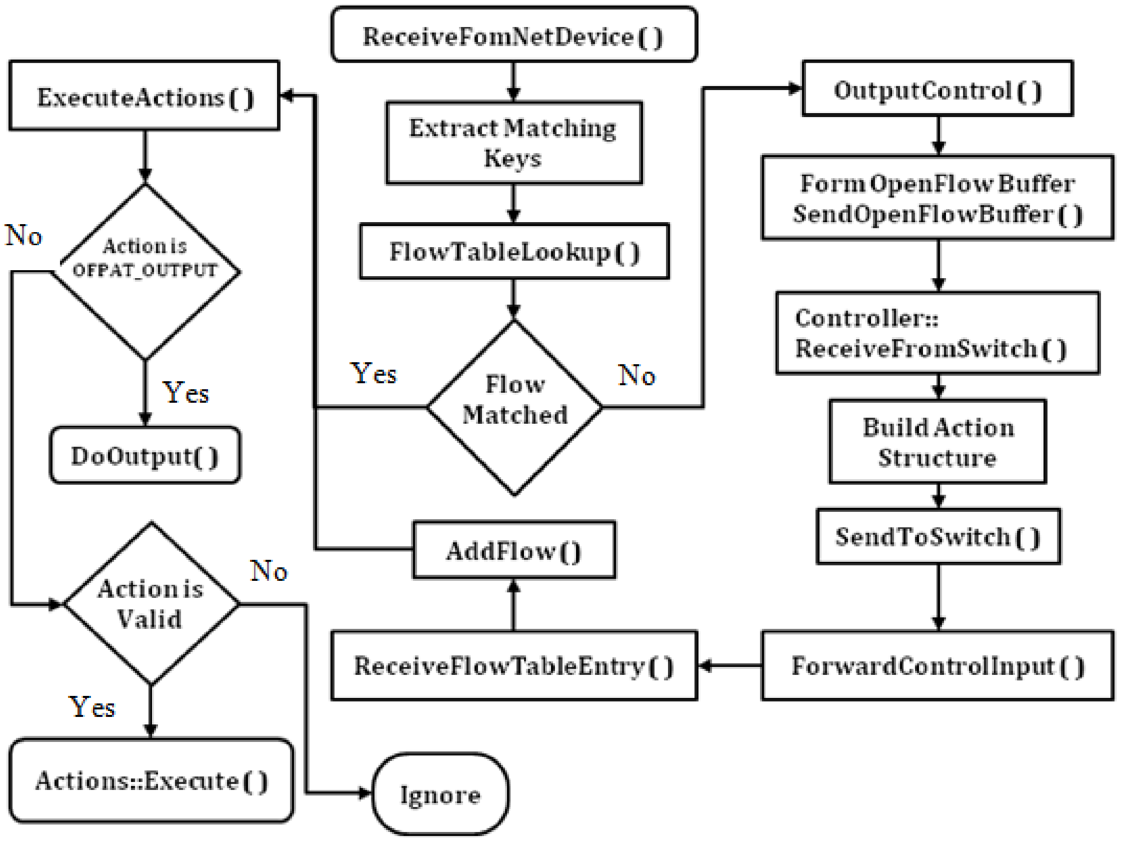


Figure 4.7: Flow Chart for the OpenFlow switch existing functionality

4.4. Source-Routed OpenFlow Switch and Controller

Since the objective of the research is to perform simulations to show the effectiveness of source routing in SDN and compare it to the traditional hop-by-hop routing approach, several modifications and enhancements needed to be applied to the OpenFlow Switch model discussed in *Section 4.3*. The enhancements can be summarized as:

1) Implementing source routing functionalities in the OpenFlow switch model.
2) Implementing realistic switch-to-controller communication model and a secure channel between the controller and the switches for control traffic.

3) Developing realistic OpenFlow controller models for both source routing and hop-by-hop routing for comparison.

The rest of the chapter will discuss these implementations in more details. Figure 4.8 below shows the overall architecture of the intended simulation model.

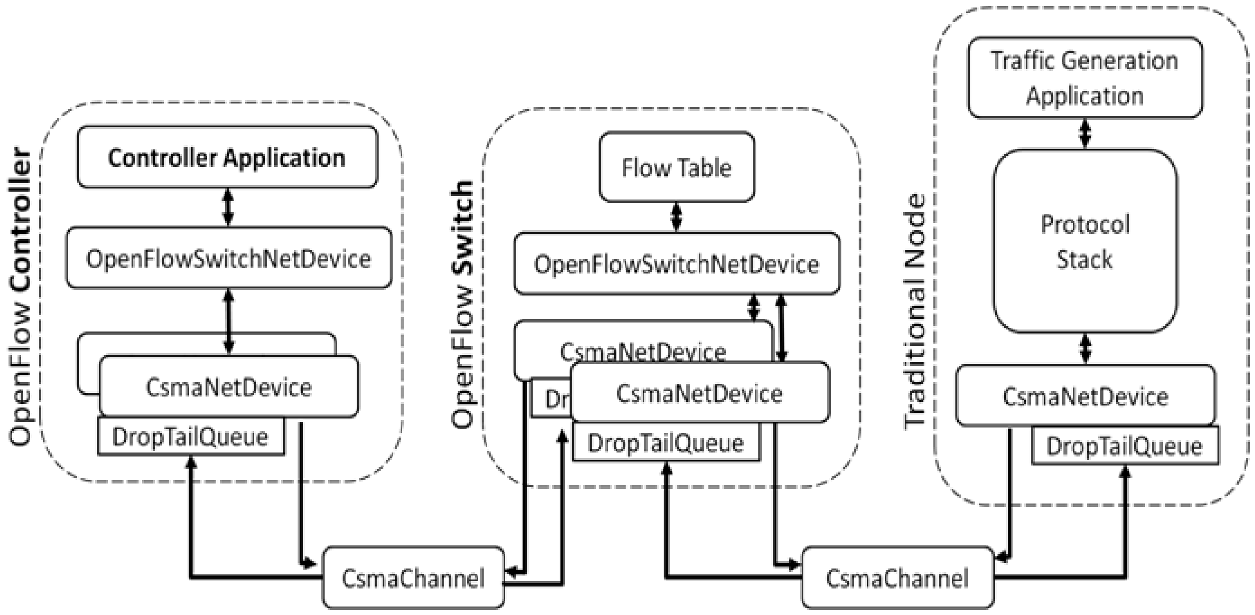


Figure 4.8: Architecture of the intended simulation models after the enhancements are applied

4.4.1. Source Routing Functionality

To be able to support source routing depending on the architecture presented in *Section 4.1*, the *OpenFlowSwitchNetDevice* module was enhanced. The two main functionalities that were added to the module are:

1) The ability to form a source routing path header from information received from the controller and appending them to a matched flow packet.

(Ingress Switch)

2) The ability to parse a source routing path header and use the embedded information to forward the corresponding packet accordingly.
   (Intermediate Switch)

New OpenFlow Action Type

OpenFlow protocol supports certain types of actions that the controller can install in an OpenFlow switch's flow table [6]. To implement the first functionality in the list above, *OpenFlowSwitchNetDevice* has to support a new action type that can be installed in the flow tables, so that when the corresponding flow is matched, a specific source routing path header is formed and appended to that flow's packets to represent the path they will take to the destination.

A new action type called *OFPAT_SET_PTH_LINKS* was created and added to the list of actions supported by *OpenFlowSwitchNetDevice* module in ns-3. In OpenFlow [5], each action corresponds to a specific action structure that holds information that facilitates the execution of that action. Therefore, and new action structure was created to support the new action type, *opf_action_set_pth_links*. A new instance of this structure has to be created each time the controller receives a packet forwarded from the switch because it doesn't match an existing flow table entry. The new instance will include information about the path that this packet is supposed to take. This instance will be used along with flow matching keys to form a flow table entry to be pushed down to the switch.

Adding Path Header

When a flow is matched in the flow table and the action to be executed is found to be *OFPAT_SET_PTH_LINKS*, the switch calls a method *AddPathHeader*() and passes the packet id and the action structure found in the flow table. A new *PathHeader* class was created inheriting from ns-3 *Header* class. The class contains three members: *m_position*, *m_size*, *m_linkLabels* to reflect the design explained in *Section 4.1.1*, Figure 4.9. It also includes public methods that are used to obtain or modify its members. The information stored in the action structure is used to construct a *PathHeader* object that holds the sequence of port numbers that represent the path. The header object is then appended to the packet headers. The packet is then forwarded to the port that has the number of the first entry in the path header, following the same procedure described earlier in *Section 4.1.1*.

Path Header					Packet HEADERS	Packet PAYLOAD
2	3	3	Location	Hop Count		
			1	3		

Figure 4.9: An example of a path header inserted in a matched packet

Forwarding Using Path Header

In intermediate switches, *OpenFlowSwitchNetDevice* has to support the functionality of parsing a path header if one is found when a packet is received. Parsing the path header allows the switch to determine the interface the packet should be forwarded through. *OpenFlowSwitchNetDevice* was modified to look for a path header and use it if available before running a flow table lookup. If a path header is found, the header's information is passed to the function *UsePathHeaderToForwardPacket().* This function extracts the entry in the *m_linkLabels* array field of the class at the position held in the *m_position* field and uses that entry – which refers to a port number – to forward the packet. Before the packet is forwarded, the *m_position* is incremented and the header is rewritten into the packet.

Figure 4.10 below is a flow chart that shows a high level description of the modified model and explains the sequences of events and functions that execute starting from when a packet is received to a corresponding action is executed.

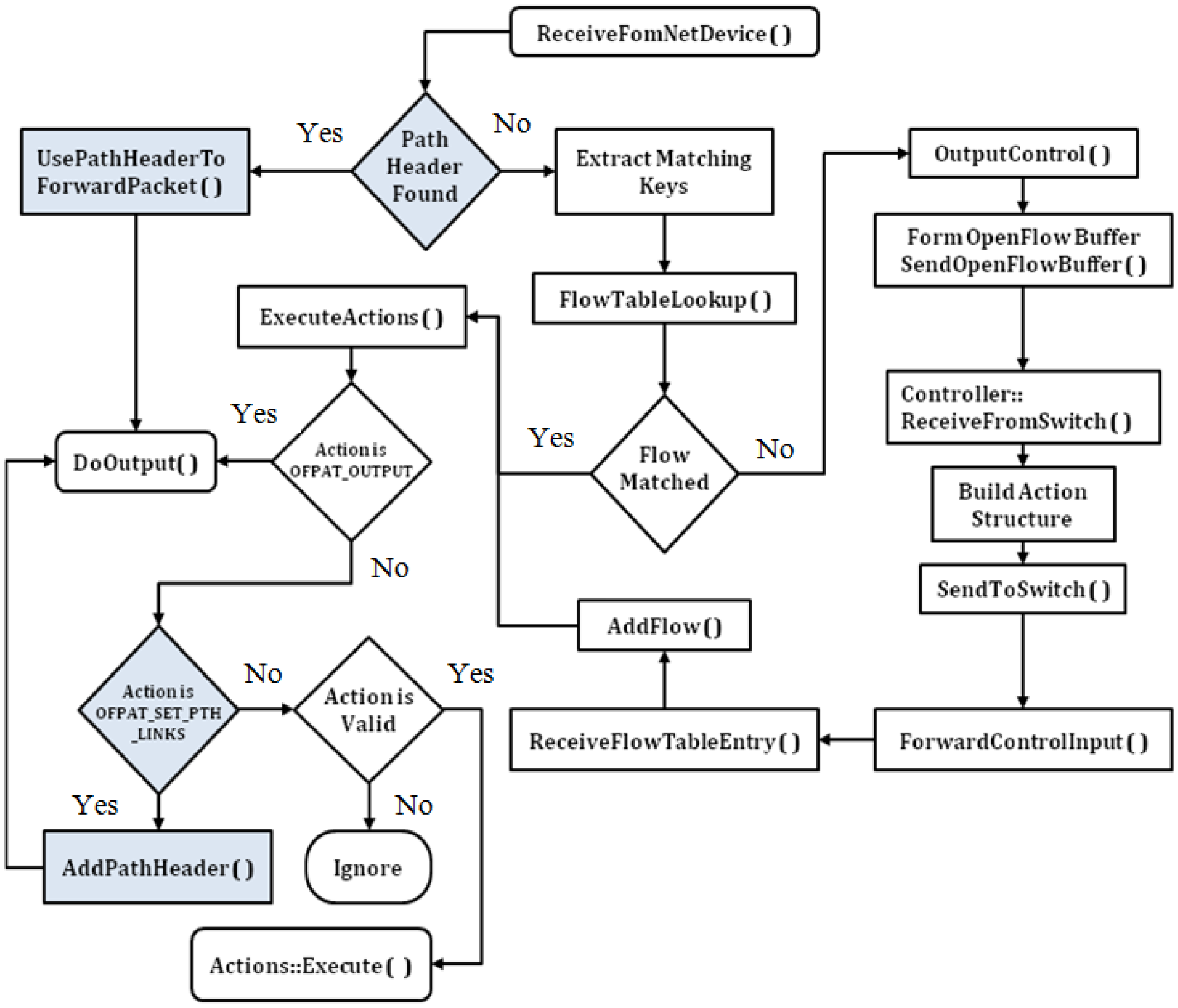


Figure 4.10: Flow chart of the OpenFlow switch functionality including Source Routing enhancements

4.4.2. Exchange of Control information

As mentioned before, one of the limitations of the original OpenFlow switch model is a lack of a proper implementation of the secure channel between the switch and the controller. In the current model implementation all OpenFlow switches in a simulated network in ns-3 can access the controller resources directly through local specific APIs (function calls). Typically, OpenFlow switches send *PACKET_IN* messages to the controller through a secure channel when unknown flow is received, to which the controller responds after processing it to install new flow entries at the switch [6]. This control traffic communication takes places over an overlay dedicated control network that ensures high priority to the control traffic, or over the same data plane network with QoS policies enforced to allow traffic prioritization. To enhance the current OpenFlow switch model to address these concerns, three main modifications were applied:

1) *OpenFlowSwitchNetDevice* was modified to have the functionality of forming and sending control packets containing OpenFlow protocol control messages to the controller through the network (*NetDevices* & *Channels*) instead of using direct local calls to reach the controller.
2) The controller architecture was changed from being an object available through APIs to *OpenFlowSwitchNetDevice* Objects, to an application that is installed on an ns-3 *Node* and available through exchange of OpenFlow protocol control packets through the simulated network.
3) In any simulation when a network of OpenFlow switches is built, an overlay control network is built. The control network objective is to transfer OpenFlow control packets between the switches and the controller. As Figure 4.11 shows, in each OpenFlowSwitchNetDevice, the control and the data traffic will be handled by separate *CsmaNetDevices*, with separate queues and higher priority given to the control traffic *CsmaNetDevice*/*Queue*.

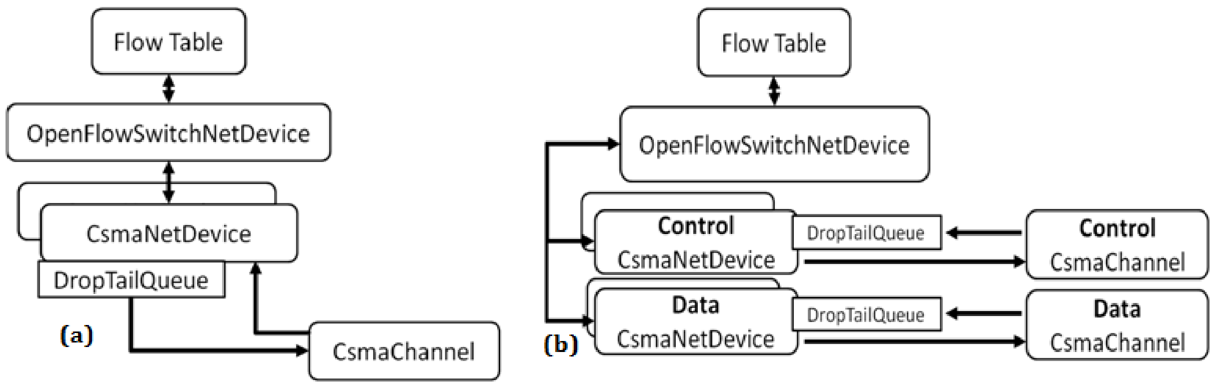


Figure 4.11: (a) Original switch model, (b) Enhanced model with separation of control and data traffic.

4.4.3. OpenFlow Controller Model

Controller Architecture

One of the major enhancements to the current OpenFlow switch model in ns-3 is changing the architecture of the OpenFlow controller. The controller in the modified model is implemented as an application that is installed on a *Node* that uses *OpenFlowSwitchNetDevice* module for underlying communications, Figure 4.12. This makes the controller only reachable through exchange of control packets containing OpenFlow messages over the control network. This enhancement allowed conducting simulations taking into account flow installation times, and network convergence delays which are important factors in studying SDN implementations in WANs.

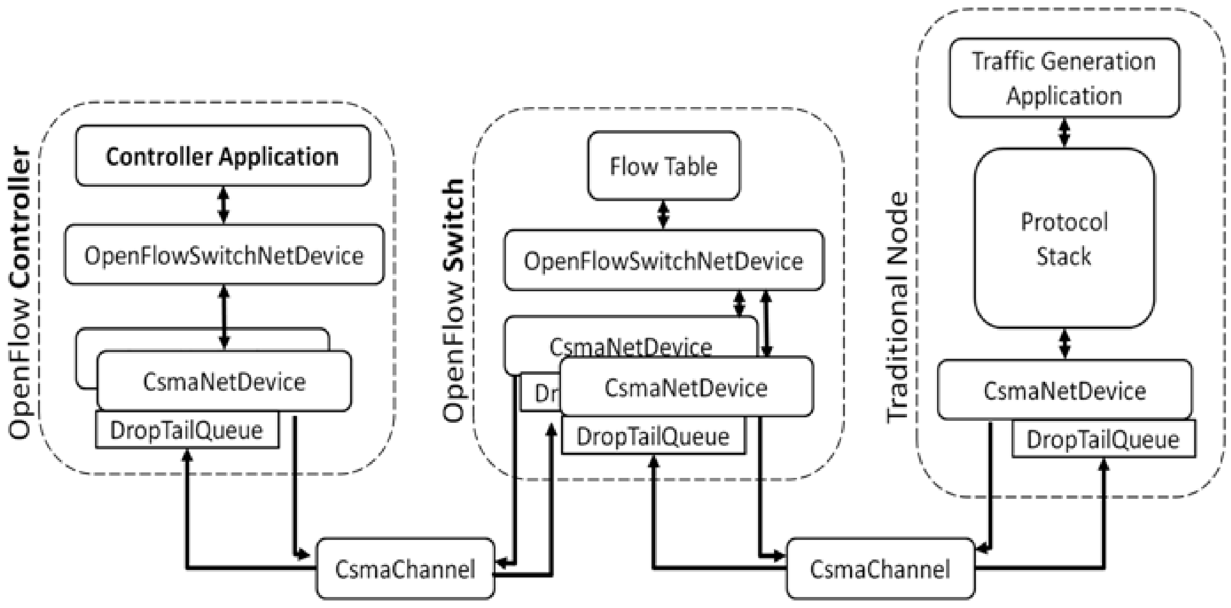


Figure 4.12: The new architecture including the separate Controller Model

Controller Customized Functionality

In ns-3, OpenFlow controllers can be created by the users internally in the (ofi namespace) by extending the base Class (Ofi::Controller).

The controller can be customized to responds to packets coming from the switches indicating unknown flows. The controller's response is by determining the right actions that should be applied on the unknown flow's packets at the switch. The controller creates action structures that hold information like the action type and some additional information to help executing the action at the switch.

In our implementation, the controller has full knowledge of the network topology. Using this knowledge the controller can determine the path between any source and destination using algorithms similar to the ones described in *Section 3.3.2*. The controller maintains a network-wide forwarding database for all the switches in the simulated network.

To carry out our simulations and to be able to compare between source routing and traditional hop-by-hop routing, two types of controller applications were developed:

<u>Hop-By-Hop Routing Controller</u>

When the controller receives a packet from a switch indicating an unknown flow:

1) Controller extracts information to determine the source and destination nodes for the unknown flow.
2) Controller uses source and destination pair and the forwarding database to find:
    a. The sequence of switches along the path from source to destination
    b. The forwarding port at each intermediate switch along the path
3) Controller forms an *OFPAT_OUTPUT* action structure for each intermediate switch that carries the port where the flow packets should be forwarded at each switch.
4) Controller sends the new flow table entry addition messages to all switches along the path concurrently except for the ingress switch that it received the original request from.
5) Controller waits for acknowledgement messages from all switches along the path before sending the flow table entry to the ingress switch to ensure the path is setup before releasing the packet at the ingress switch.

   During this wait, the controller can process packets arriving from other switches.
6) Once all acknowledgements are received, the controller sends the new flow table entry addition message to the ingress switch.

Source Routing Controller

When the controller receives a packet from a switch indicating an unknown flow:

1) Controller extracts information to determine the source and destination nodes for the unknown flow.
2) Controller uses source and destination pair and the forwarding database to find:
    a. The sequence of switches along the path from source to destination
    b. The forwarding port at each intermediate switch along the path
3) Controller forms an *OFPAT_SET_PTH_LINKS* action structure that carries a sequence of port numbers corresponding to a source route along the intermediate switches from source to destination.
4) Controller sends the new flow table entry addition messages to the ingress switch only.

4.5. Traffic Generation Model

To validate the source routing OpenFlow switch model, and compare source routing performance to traditional routing, a form of realistic traffic modeling was needed. ns-3 natively supports few basic traffic generators, however, a traffic generator that reproduces real-life traffic is needed to simulate the SDN-based WAN reliably.

The work in [38] introduces a new ns-3 implementation of a traffic generation tool that accurately mimics real-life IP networks data traffic. The tool relies on the Poisson Pareto Burst Process [39], and simulates statistical properties of real internet data traffic such as long range dependency [40].

The Poisson Pareto Burst Process (PPBP)

The PPBP is a process that relies on superimposing multiple bursts that arrive according to a Poisson process with rate λ_p, and their lengths are independent and identically distributed random variables following a Pareto distribution [39]. A Pareto distribution is characterized by Hurst parameter H, usually between 0.5-0.9, and a mean length/duration T_o [38]. If each burst is modeled by a constant bit rate (r) flow, then the overlapping bursts form a Long-Range dependent traffic [39]. Empirical studies and statistical analysis of internet core traffic have shown that internet traffic can be characterized as long range dependent and self similar [41]. Long range dependence implies that across large time scales a time dependent process behavior shows statistically significant correlations. Long range dependence is often associated with self-similarity which implies that the behavior of the process stays preserved when scaling in space or time.

This makes the PPBP a much more reliable modeling choice than regular Poisson based traffic models.

The ns-3 PPBP traffic generator allows the customization of the following parameters:

- Burst Intensity (r)
- Mean Burst Time Length (T_o)
- Mean Burst Arrival Rate (λ_p)
- Packet Size (default is 1470 Bytes)
- Hurst Parameter (default is 0.7)

The overall rate of the traffic generated from a PPBP source can be calculated from the variables above using Little's Law [38] as:

$$\lambda = T_o \ x \ \lambda_p \ x \ r$$

For example:

For a burst Intensity of 10 Mbps, Bursts arriving as a poison process with a rate 5 bursts/second, and a 4 second mean burst time length:

$$\lambda = 4\,\mathrm{s} \times 10\,\frac{\mathrm{Mbps}}{\mathrm{burst}} \times 5\,\frac{\mathrm{burst}}{\mathrm{second}} = 200\ \mathrm{Mbps}$$

4.6. Results Extraction

Ns-3 provides a Flow Monitor module [36] that allows the measurement of different performance parameters. The module relies on monitoring packets as they are exchanged by the nodes, using probes that are initially installed during simulation setup. The module collects statistics for each flow in the network. Since the network hosts contain IPv4 stacks and generate IP traffic, a flow is defined as the packets with the same source IP/port, destination IP/port and protocol. The collected statistics are exported in XML format at the end of the simulation. The XML file is then parsed by customized scripts written to extract the relevant information and organize it to facilitate studying the results.

5. SIMULATION & RESULTS

To show relevance of implementing a source routing approach in SDN, our study took two approaches. First, based on the computational analysis introduced in Chapter 3, a first approximation of source routing performance in a modeled SDN-based WAN was obtained. The computational analysis focused on the architectural concerns and is mostly independent of the chosen source routing implementation. Second, to support the computational analysis, our study is further extended using a detailed packet level simulation in ns-3. This simulation study is relying on the enhanced OpenFlow Switch model that was described in Chapter 4. The enhanced ns-3 models support source routing functionalities as well as traditional hop-by-hop routing. The simulation study will also utilize OpenFlow controller applications developed in ns-3 and customized for source routing and hop-by-hop routing. The simulation study also utilizes realistic traffic generation models as well as queuing models that allow better consideration of network congestion delays and effects. The simulation will be used to reveal interesting insights regarding the deeper dynamics of the source routing approach.

5.1. Simulation Network Set-up

The network topology used in the simulation study is the same topology described in the computational analysis, OS3E Network.

To simulated the OS3E network, three network node types were used:

- Traditional Nodes as network hosts. These nodes have IPv4 protocol stacks and host traffic generation application modules (PPBP traffic generators) as well as traffic sinks to receive traffic from other nodes. The PPBP traffic generators at each node were set up to produce an average of 1Mbps of total rate of traffic. Since the PPBP traffic generator simulates overlapping several traffic bursts, one traffic generator is installed in one node deployed in every city.

- OpenFlow Switches. These nodes do not have an IP protocol Stack as they represent L2 devices with no control plane functionalities. These nodes depend on the enhanced OpenFlow switch model, described in *Section 4.4*.
- An OpenFlow Controller. This node is also a L2 device that depends on the *OpenFlowSwitchNetDevice* module for communications. This node hosts the controller applications. This node can be customized to work in two modes: source routing, and hop-by-hop.

All these nodes are connected to each other's by 1GigE full duplex connections that were simulated using the enhanced *CsmaChannel* described in *Section 4.4*. The channel is configured to reflect a 1Gbps transmission speed and a propagation speed dependant on the link distances between the nodes and the speed of light in fiber. An overlay control network with dedicated bandwidth is also deployed between the OpenFlow switches to act as secure channel for transferring OpenFlow control messages.

The position of the network controller was initially chosen to be in Chicago based on the analysis results from [2] so as to minimize average latency from the network nodes to the controller. This stays the same in all simulation experiments except for the controller placement simulations were the controller location is changed iteratively.

The simulation time for all experiments was 10 minutes and the collected results were averaged over 10 simulation runs. This allowed us to measured network convergence time averages with a confidence level of 95% and a confidence interval of 7% of the different simulation runs sample mean.

Simulations were run for the two routing approaches individually: source routing and hop-by-hop routing, each time using the corresponding controller mode.

Figure 5.1 below shows an overview of the network simulation setup.

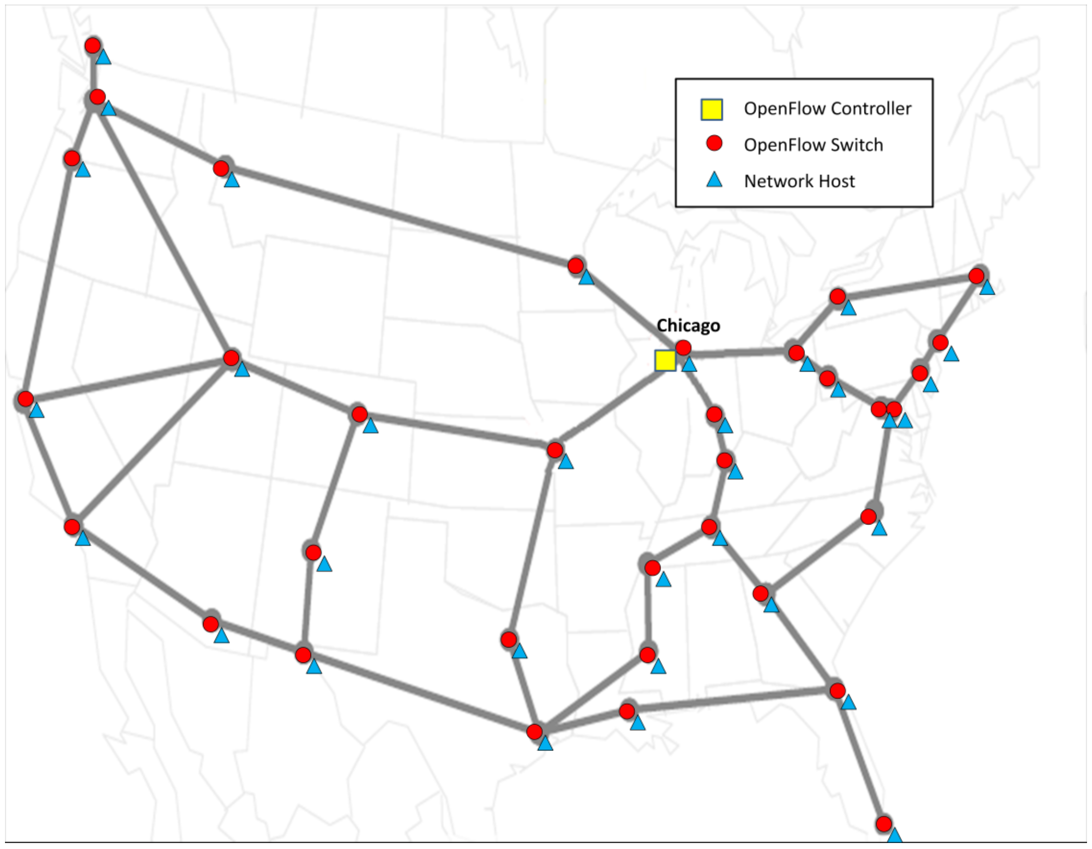


Figure 5.1: Network simulation setup

5.2. Results and Discussion

The simulation experiments focus on four major areas: network convergence times, state distribution effects, switch flow table analysis, and controller placement effects.

5.2.1. Network Convergence Time

The first set of simulation results focuses on the network convergence time. In our context, we define network convergence time as the time taken by the first packet on an unknown flow to traverse from source to destination. This includes the time taken by the switch to communicate to the controller, flow installation delays, network congestion delays, and regular propagation delays. The simulation results provide more solid grounds for comparison between the two routing approaches than the results introduced in the computational analysis, *Chapter 3*, because it has more realistic representation of traffic generation and network congestion/queuing delays.

The network convergence time for every network flow between every source and destination pair was measured. Convergence times were averaged for each source over all its destinations and the results are shown in Figure 5.2. The simulation was conducted for a moderate traffic load. 1Mbps of average total rate of traffic for each traffic source was used at each one of the 34 network nodes.

The results clearly show that using source routing achieves a lower network convergence time for all the flows in the network. Using source routing in the simulated network allows for a 40.2% reduction in the network-wide average convergence time, a 36.5% reduction in the standard deviation of network convergence times of all flows. The effect on standard deviation can be shown as the probability distribution functions for the network convergence times from both routing approaches are plotted in Figure 5.4. Figure 5.4 shows a larger dispersion around the average for hop-by-hop routing compared to source routing.

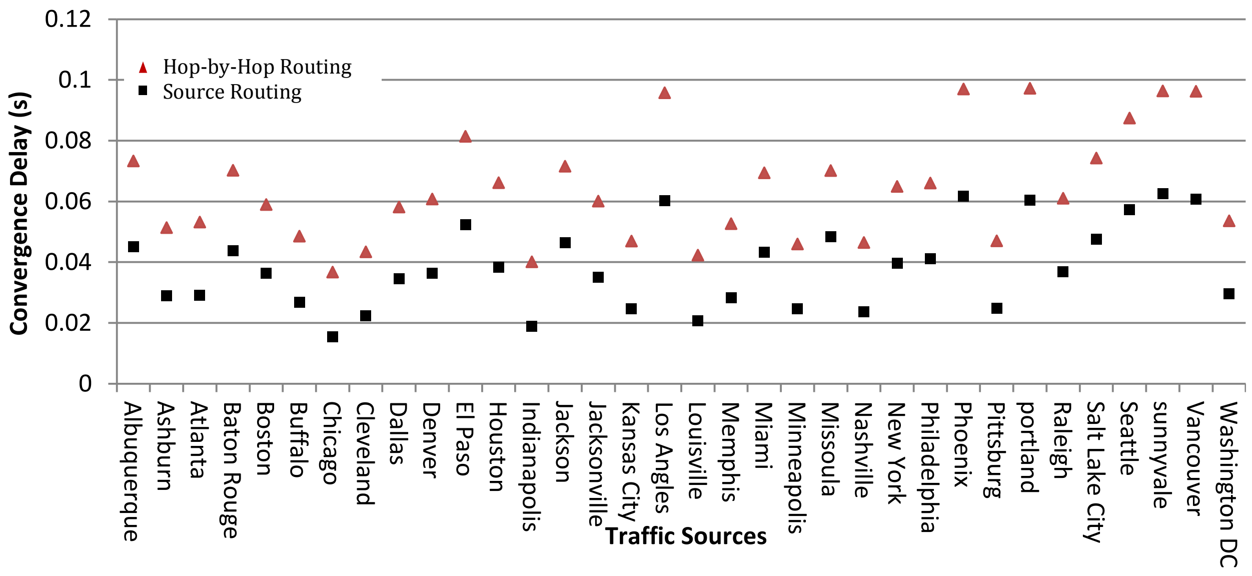


Figure 5.2: Average network convergence time for each source

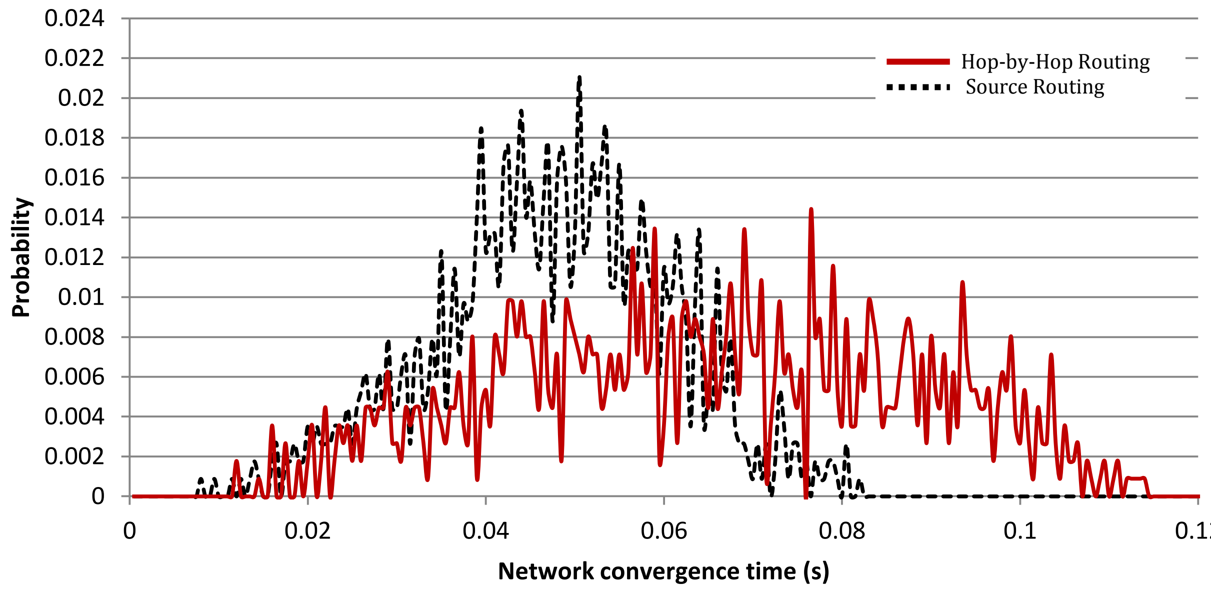


Figure 5.3: Network convergence time probability distribution

The simulation results show longer network convergence times compared to the computational analysis results as a result of taking into account the network congestion and queuing delays. However, the comparative results between source routing and hop-by-hop routing obtained from the simulations and the analysis for the average network convergence time are close. The results show 40.2% and 41% reductions in average convergence time achieved by source routing from the simulation and the computational analysis respectively. The simulation results show higher reductions in standard deviation of the network convergence times measured compared to the computational analysis. The results show 36.5% and 23% reductions from the simulation and the computational analysis respectively. This is mainly because of the nature and the randomness of traffic generation in the simulations which computational analysis didn't take into account.

The simulation was repeated iteratively with different traffic loads. The average traffic rate for each traffic source was increased incrementally and the network-wide average convergence time was measured for both source routing and hop-by-hop Routing.

The results show that network convergence time scales more efficiently when using source routing. This is reflected in the rate of increase in network convergence time as a result of increasing the network traffic load, Figure 5.4 (a). Although the performance worsens in both approaches, i.e. convergence time increases, the rate at which the network convergence time increases in source routing is lower. As mentioned before for the base traffic load, source routing achieves 40.2% reduction in network convergence time, however, as the network traffic load increases, it can be seen that source routing provides an even bigger advantage in terms of the reduction in average network-wide convergence time making it a more scalable routing alternative. The reduction can reach up to 85% in very loaded network conditions, Figure 5.4 (b).

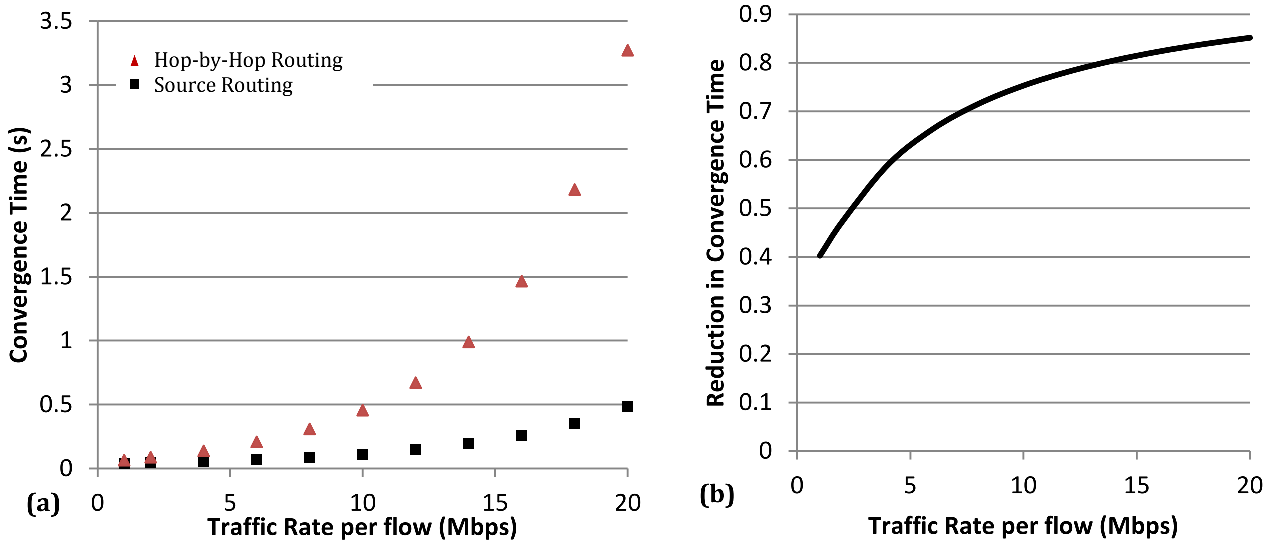


Figure 5.4: Comparison for Network Convergence time for both approaches as the rate of network traffic increases

5.2.2. State Distribution

Controller State Distribution Burden

Performing the simulations described in the previous section also allowed measuring the load on the controller. Counters were embedded inside the controller to measure the number of requests the controller receives and the number of update messages sent corresponding to each request. The ratio of the update messages sent for each request received was measured and used as an indicator of the controller state distribution burden. In the case of source routing, the ratio was 1. This is mainly because for each request received by the controller, only one update has to be sent to the ingress switch directly. On the other hand, the ratio was found to be 6.1 in the case of traditional hop-by-hop routing.

Using source routing was found to achieve 81% reduction in the state distribution burden on the controller. Such a significant load reduction in state distribution translates into a reduction in the controller's processing load. This can increase the controller's performance and scalability as it can accommodate a higher number of network nodes.

Switch Control Traffic Burden

The simulations also allowed a deeper look into the switch forwarding dynamics. The ratio of the number of control packets to the number of data packets forwarded by all the switches was measured. This information was collected by processing the forwarding logs of the switches and classifying them based on the packet type (data/control). The results reveal that:

- In hop-by-hop routing: The ratio of the control traffic to the data traffic is 4%
- In source routing: The ratio of the control traffic to data traffic is 0.9%

The results conclude that source routing leads to a 77.5% reduction in the control traffic transferred in the network due to state distribution. It is important to note that this simulation result conforms to the computational results obtained in *Section 3.4.1*, where the reduction in state distribution was calculated for the OS3E network to be 77.6% using the equation below:

$$\text{State Reduction} = 100-(100/n) \quad ; n \text{ is the number of links in the path.}$$

Since control traffic in the network is usually of a higher priority when implementing QoS, source routing leads to reducing data traffic interruption due to control traffic processing at the switch level, which theoretically allows more forwarding bandwidth for data traffic.

5.2.3. Flow Table Analysis

The following set of simulation results focuses on the flow table size, which is one of the most important design considerations of an OpenFlow switch. Flow tables are typically implemented using hardware TCAM [15].

To produce this set of results, the network switches' flow table sizes were polled at random intervals during the simulation and averaged for each switch. This was repeated for both source routing and traditional hop-by-hop routing.

The results show that when using source routing flow tables contain far less entries than traditional routing, which is a direct result of the fact that intermediate switches on any network path do not store flow information and depend on embedded source routing path headers for forwarding. Figure 5.5 shows a significant reduction achieved in the flow table sizes when source routing is used that ranges from 50% to above 90%. The average network-wide reduction in flow table sizes was found to be 76.6%. It was found that the maximum reduction in flow table sizes is achieved at the nodes in the center of the network topology: Chicago, Kansas City, and Cleveland. That is because these nodes are shared nodes for a large number of flow paths in the network, and therefore would typically contain a relatively large number of flow table entries. The level of reduction in flow table sizes is found to be lower on the nodes at the edges of the topology: Miami, Vancouver, and Portland.

It is important to note that these results were collected using 34 traffic flow sources in 34 different cities producing flows between every possible source and destination pair. Using this setup, 1156 traffic flows were generated.

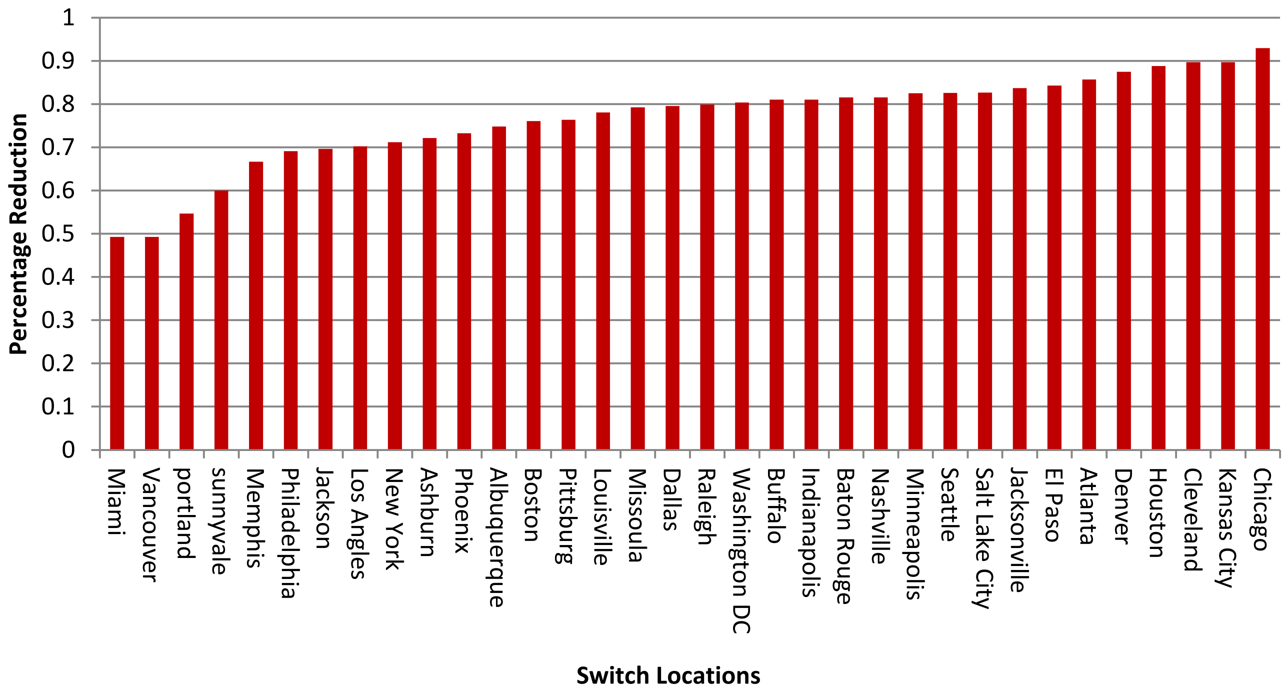
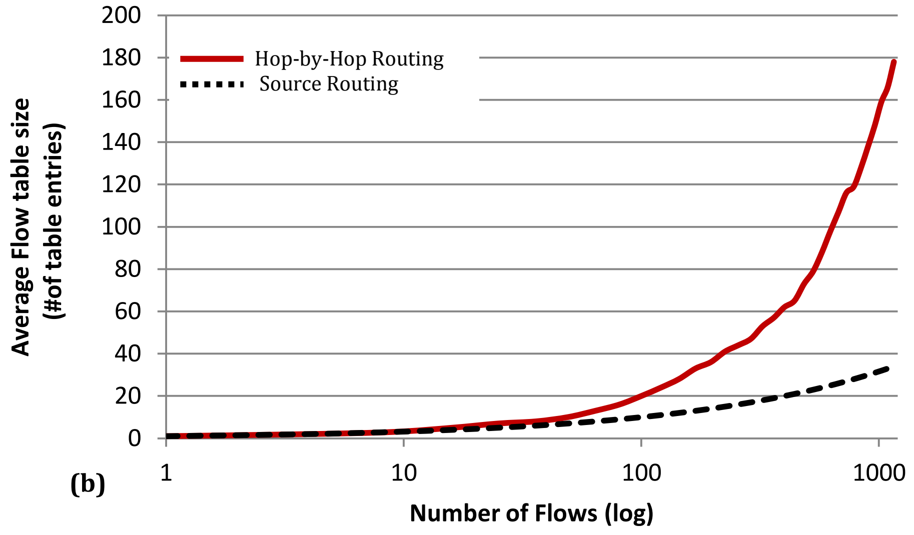


Figure 5.5: Reduction in flow table sizes using source routing

To provide a more comprehensive understanding of the effect of using source routing in such a network on the flow table size, the experiments were repeated iteratively for different number of generated flows on the network. The number of generated flows is controlled by controlling the number of nodes participating in traffic generation. The results provide an insight on how the network-wide average number of switch flow table entries increases as the total number of network flows increases when using the two different routing approaches. The results also consider another metric which is the maximum number of flow table entries among the switches considered in all scenarios.

These parameters give an idea about the memory hardware resources required to accommodate the network flows. The results are shown in Figure 5.6(a, b) below.



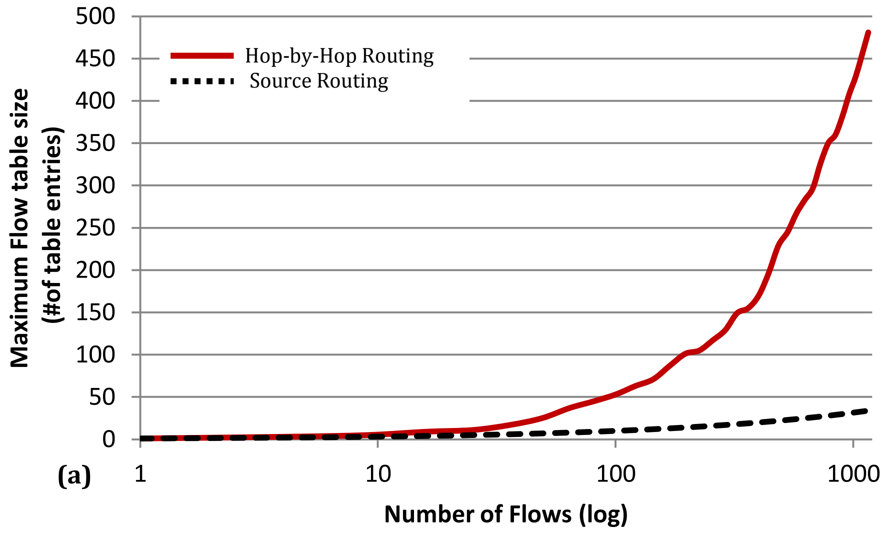


Figure 5.6: The effect of increasing the number of flows in the network on (a) Average flow table size (b) Maximum flow table size

The results show that the two flow table metrics increase at a much slower rate as the number of active network flows increase in the case of using source routing. This means that the source routing approach provides a more scalable solution in terms of the required memory hardware resources.

Figure 5.7 shows the reduction achieved in average and maximum number of flow table entries in the switches' flow tables using source routing.

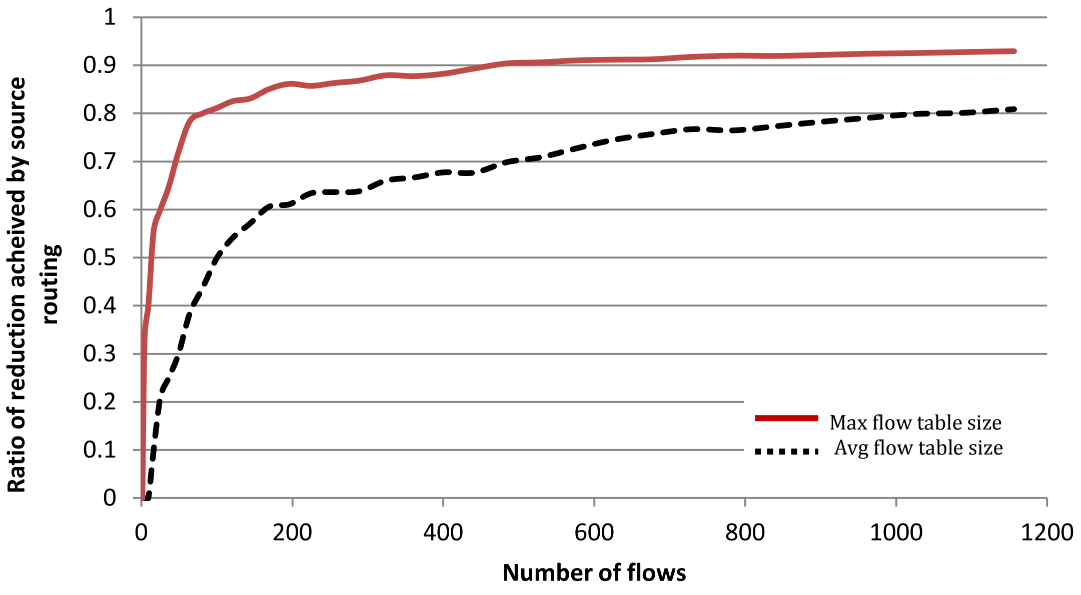


Figure 5.7: Reduction in average and maximum number of flow table entries using source routing

The results shown in Figure 5.7 are an indication of the scalability advantage of using source routing. In fact, the reduction in flow table size requirements achieved by source routing increases as the number of active flows in the network increases. Figure 5.7 shows that using source routing can achieve more than 90% reduction in the maximum required flow table size as the network becomes more loaded with active flows.

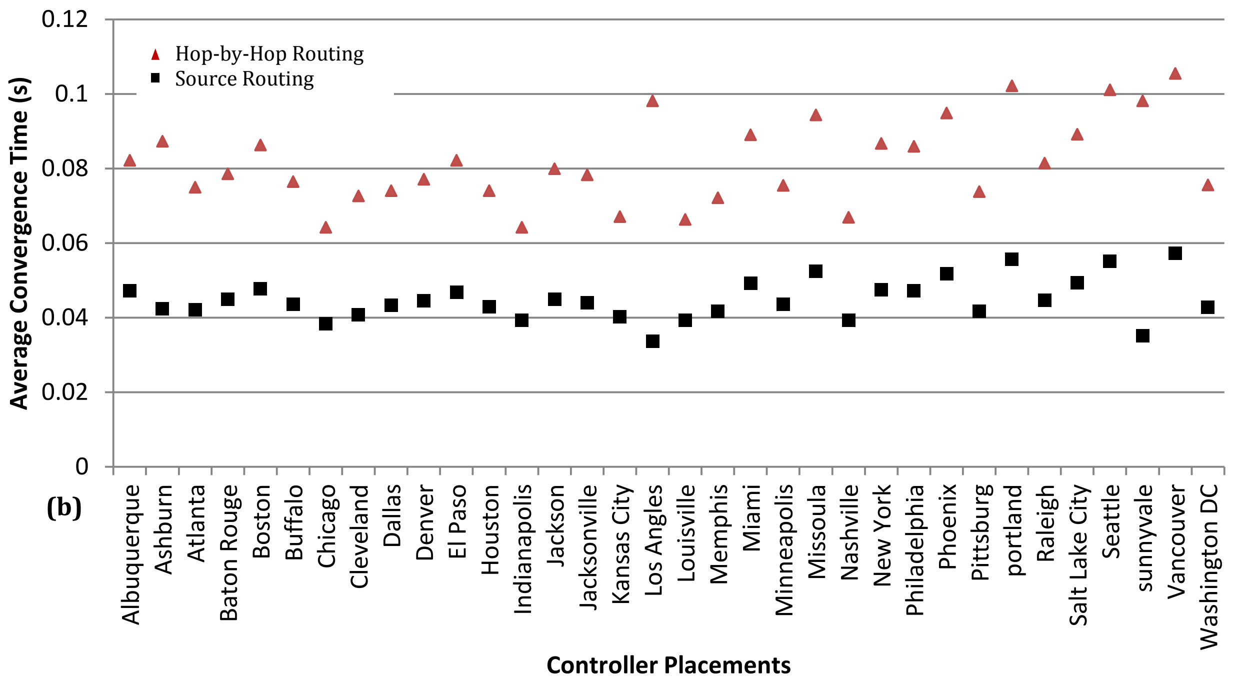
The results shown here conclude that source routing can achieve significant reduction in the memory required by each switch. The available memory in the switch is considered a very important parameter in the switch's hardware design as the TCAM modules can significantly affect the cost of the switch.

5.2.4. Controller Placement

The following set of results focuses on the controller placement problem introduced in [2], and discussed as well in the computational analysis in *Section 3.4.4*. All the different possible controller placement options were assessed iteratively using simulations focusing on two metrics, average network convergence time and worst-case network convergence time for all possible flows for each controller placement.

The simulation results in Figure 5.8 reassure the findings presented in the computational analysis. It was found that source routing achieves a 45.2% reduction in the average convergence time across all controller placement options, and a 36% reduction in the average worst-case network convergence time across all controller placement options.

To show the effect of source routing more clearly, the standard deviation of average network convergence time and worst-case network convergence time for all the controller placements was measured. It was found that source routing achieves a 51.8% reduction in the standard deviation of the average convergence time measured for each placement, and a 69% reduction for the standard deviation of the worst-case convergence times measured for each placement.



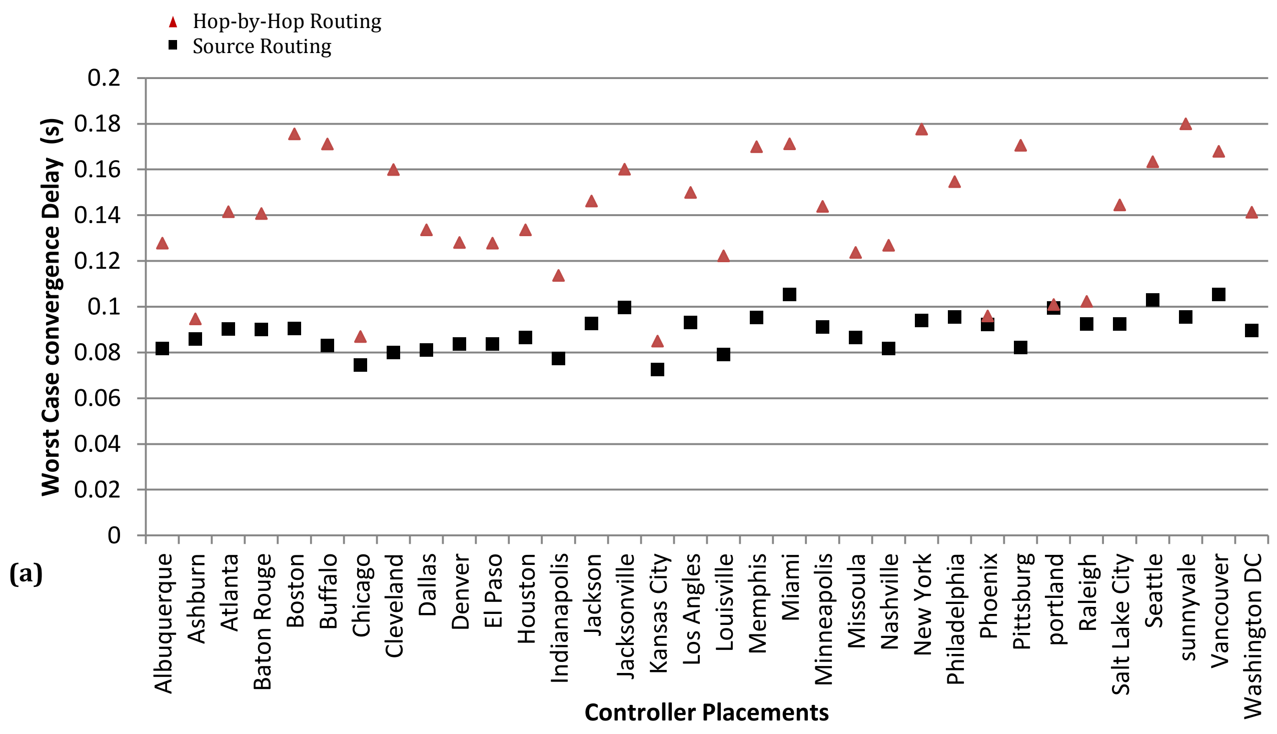


Figure 5.8: (a) Average convergence times for all placements, (b) Worst-case convergence time for all placements.

These results lead to the same conclusions presented in the computational analysis, which is that using source routing lessens the effect of changing the controller placement in the network on performance. In other words, the network convergence time is less sensitive to the location of the controller placement when using source routing, giving the network designers flexibility in designing their networks and network node locations.

Table 2.1 shows that the comparative results of the simulation experiments agree to an acceptable extent with the results obtained from the computational analysis.

Table 5.1: Comparisons between the computational analysis and the simulation results

Reduction achieved using source routing (%)				
Parameter	Average network convergence time		Worst-case network convergence time	
	Simulation Results	Computational Analysis Results	Simulation Results	Computational Analysis Results
Average	45.2	44.8	36.1	37.1
Standard Deviation	51.8	53.4	69.8	47.1

5.3. Source Routing Concerns

As mentioned before, source routing hasn't been a preferred choice in networking as a result of disadvantages such as increased packet overhead and security concerns. In addition, source routing is less useful in a distributed control environment. Given however that SDN is a centrally controlled environment it is worth reconsidering these potential concerns in light of their benefits.

The overhead of source routing comes from the path information carried in every packet. However, when using an approach that relies on local interface numbers at every intermediate switch, only a small number of bits for every hop will be sufficient to express each possible next link. 8 bits allows 255 links. Given that the average path length in a topology such as OS3E is between 5 hops and 9 hops, the packet overhead can be low. Thus, 88 bits (9 8-bit hops, 8-bit location identifier, and 8-bit hop count) are sufficient to capture all next hops for the longest path. This seems acceptable when compared to IP/Ethernet header or MPLS tunnel header sizes.

To address security concerns, in the proposed approach, source routing path headers are only added by the ingress switches. Ingress switches would be trusted devices that authenticate with the network controller during network initialization.

Another important aspect that is raised in source routing related discussions is fault management. According to the discussed source routing approach, a response to a failure event takes the form of rapidly adding the alternative source route to the packets at the ingress switch to restore the flow with no need for new state redistribution [35]. In addition, local link failure recovery can also be leveraged as described previously in *Section 4.1*, while the controller responds to the ingress switch.

6. CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

SDN has proven to be of great value in several use cases of today's networking industry and research fields. The decoupling of control and data planes that SDN relies on has brought many advantages such as increased network simplicity, programmability, vendor independence. SDN also helps bring down capital and operational expenses of network deployments. One particular use case of SDN is in WANs of service providers, large enterprises or inter-datacenter networks. The research in this study focused on the challenges of SDN deployments in WANs.

When considering SDN in a WAN, several challenges arise due to the nature of WAN that includes large number of networking devices with large propagation delays among them. The challenges include keeping network convergence time within acceptable limits in spite of the large propagation delays, controlling flow table sizes with hundreds of thousands of network flows, assuring network scalability without performance degradation, and designing the SDN with reasonable controller placements that leads to optimal performance.

This study proposes leveraging a variation of source routing as an alternative routing approach in SDN-based WANs to tackle some of their challenges. The study presented a computational analysis to show the architectural gains of implementing a source routing based routing approach in a real production SDN-based WAN. The study was also extended by developing simulation models that rely on enhancing the existing ns-3 OpenFlow switch model with source routing functionalities to be later used in performing packet level simulations to show the effectiveness of the source routing approach.

One of the contributions of this research is the enhanced OpenFlow switch simulation model that was developed in ns-3 and can serve as bases for performing simulations studying different aspects of SDN.

In addition, the main contribution of the study is illustrating that if source routing is used as an alternative routing switch in SDN-based WANs, it can bring significant improvements in performance and scalability:

- Source routing provides significant reduction in the state distribution in the network. This leads to lowering the processing load in the controller allowing for improved performance and scalability in terms of the number of the switches that one control can be managing. This also leads to lowering the amount of high priority control traffic handled by the switches, which allows more bandwidth for data plane traffic.
- Source Routing improves overall network convergence time. It also provides much lower variations in network convergence times among traffic sources, which allows better conformity to SLAs.
- Source routing provides better network convergence time scalability compared to traditional routing in SDN in the following cases:
    - ▪ Increasing the number of nodes in the network
    - ▪ Increasing network diameter
    - ▪ Increasing the overall traffic load in the network

- Source routing is also a scalable solution in terms of the required flow table sizes compared to traditional routing. With source routing, a significant reduction can be achieved in the maximum required flow table size, and the reduction has been illustrated to increase as the network scales.
- Source routing reduces network performance sensitivity to controller placements in the network. It also eases the trade-off between the analyzed controller placement optimization parameters: average and worst-case network convergence time. Consequently, source routing gives network operators and network designers more flexibility in controller placements if the optimal placement is not feasible or difficult to reach.

6.2. Future Work

One of the major areas of work that can be done to expand the presented study is addressing fault tolerance considerations in the source routing model that was developed. The use of source routing allows local link failure recover as described in *Section 4.1.* As a future step, local link failure recovery features can be implemented in the presented simulation models and analysed for different failure scenarios to compare the performance to the traditional routing approach currently used in SDN.

Another area for future work is the switch flow table analysis. Simulations can consider several other topologies with different traffic patterns and analyze the effects of using source routing on the flow table sizes. Source routing can then be compared using the presented models to flow table optimization techniques to study areas for improvements.

Moreover, the developed models can be used as a basis for studying the performance of possible controller traffic engineering applications that can utilize the advantages of source routing to achieve higher levels of link utilization in the network.

In addition, the controller placement analysis can be expanded. The analysis in this study considered the existence of only one controller in the network. As future work, the study can be expanded to consider multiple controllers in the network. The study can also be expanded to analyze several other topologies. The study can evaluate and compare the effectiveness of controller placement optimization techniques in the scope of source routing.

7. REFERENCES

[1] S. Jain , A. Kumar , S. Mandal , J. Ong , L. Poutievski , A. Singh , S. Venkata , J. Wanderer , J. Zhou , M. Zhu , J. Zolla , U. Hölzle , S. Stuart , A. Vahdat, "B4: experience with a globally-deployed software defined WAN". *ACM SIGCOMM 2013*, August 2013, Hong Kong, China.

[2] B. Heller, R. Sherwood, N. McKeown. "The controller placement problem". *HotSDN'12*, August 2012, Helsinki, Finland.

[3] J. Blendin. "Cross-layer optimization of peer-to-peer video streaming in openflow-based ISP networks", *Master's Thesis*, April 2013, Technische Universität Darmstadt, Germany.

[4] Open Networking Foundation (ONF). [Online] https://www.opennetworking.org/ (Accessed: January 7th, 2014)

[5] N. McKeown , T.Anderson , H. Balakrishnan , G.Parulkar , L. Peterson , J. Rexford , S. Shenker, J.Turner. "Openflow: enabling innovation in campus networks". *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, April 2008.

[6] Open Networking Foundation. "OpenFlow switch specification version 1.0.0." December 31, 2009. [Online]
URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf
(Accessed: October 29th, 2014)

[7] Open Networking Foundation. "Software-Defined Networking: The new norm for networks." ONF white paper, April 13, 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[8] SDN Central. [Online]
URL: https://www.sdncentral.com (Accessed: November 14th, 2014)

[9] R. Henrique, M. Inder, S. Abhinava, S. Sharfuddin, G.Chin, P.Eric; L. Chris, R, Tajana. "Traffic optimization in multi-layered WANs using SDN," *IEEE 22nd Annual Symposium on High-Performance Interconnects (HOTI),* August 2014, Mountain View, California, USA.

[10] A. Sadasivarao, S. Syed, P. Pan, C. Liou, I. Monga, C. Guok, A. Lake. "Bursting data between data centers: case for transport SDN," *IEEE 21st Annual Symposium on High-Performance Interconnects (HOTI),* August 2013, San Jose, California, USA.

[11] V. Pandey. "Towards widespread SDN adoption: Need for synergy between photonics and SDN within the data center", *Proc. IEEE Photon. Soc. Summer Topical Meet. Series*, 2013.

[12] Cisco: WAN automation engine. [Online]
URL:http://www.cisco.com/c/en/us/products/routers/wan-automation-engine/index.html
(Accessed: December 19th, 2014)

[13] Alcatel-Lucent: Nuage networks, virtualized network services. [Online]
URL: http://www.nuagenetworks.net/products/virtualized-network-services
(Accessed: December 19th, 2014)

[14] Ciena: Mutilayer WAN controller. [Online]
URL: http://www.ciena.com/products/multilayer-wan-controller/
(Accessed: December 19th, 2014)

[15] A. Zarek. "OpenFlow timeouts demystified", *Master's Thesis*, 2012, Department of Computer Science, University of Toronto, Ontario, Canada.

[16] S. Luo, H. Yu, L. M. Li. "Fast incremental flow table aggregation in SDN," *23rd International Conference on Computer Communication and Networks (ICCCN),* August 2014, Shanghai, China.

[17] B.-S. Lee; R. Kanagavelu, K.M.M.Aung. "An efficient flow cache algorithm with improved fairness in Software-Defined data center networks," *2013 IEEE 2nd International Conference on Cloud networking (CloudNet), November* 2013, San Francisco, California, USA.

[18] A. Xueli, D.Perez-Caparros, Q. Wei. "Consistent route update in software-defined networks," *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet),* October 2014, Luxembourg.

[19] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia. "Pareto-optimal resilient controller placement in SDN-based core networks," *2013 25th International Teletraffic Congress (ITC),* September 2013, Shanghai, China.

[20] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng. "On the placement of controllers in Software-Defined Networks," *The Journal of China Universities of Posts and Telecommunications*, , vol. 19, no. 2, pp. 92-97, October 2012

[21] H.K. Rath, V. Revoori, S.M. Nadaf, A. Simha. "Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game," *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Network,* June 2014, Sydney, Australia.

[22] J. Postel. Internet Protocol. STD 5, RFC 791, September 1981.

[23] Y.C. Hu, D.B. Johanson."Implicit source routes for on-demand ad hoc network routing". *ACM international symposium on Mobile ad hoc Networking & Computing (MobiHoc '01)*, October 2001, Long Beach, California, USA

[24] S. M. Bellovin. "Security problems in the TCP/IP protocol suite"**.** *ACM SIGCOMM Computer Communication Review*. vol. 19, no. 2, pp. 32-48, April 1989.

[25] J. Shafer, B. Stephens, M. Foss, S. Rixner, A.L Cox. Axon. "A flexible substrate for source-routed Ethernet". *2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS),* October 2010, La Jolla, California, USA.

[26] C. Guo , G. Lu , H. J. Wang , Shuang Yang , Chao Kong , Peng Sun , Wenfei Wu, Yongguang Zhang, "A data center network virtualization architecture with bandwidth guarantees". *ACM International Conference on emerging Networking Experiments and Technologies (CoNEXT),* December 2010, Philadelphia USA.

[27] S. Previdi et. Al. "Segment Routing with IS-IS Routing Protocol". Draft-previdi-filsfils-isis-segment-routing-00 (work in progress), March 2013

[28] D. Cai, A. Wielosz, W. Songbin. "Evolve carrier Ethernet architecture with SDN and segment routing," *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Network,* June 2014, Sydney, Australia.

[29] B. Stephens. "Designing scalable networks for future large datacenters". *Master's Thesis*, May 2012, Rice University, Texas, USA.

[30] R.M. Ramos, M. Martinello, C. Esteve Rothenberg. "SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow," *2013 IEEE 38th Conference on Local Computer Networks (LCN), October* 2013, Sydney, Australia.

[31] S.H. Yeganeh, A. Tootoonchian, Y. Ganjali. "On scalability of Software-Defined Networking". *IEEE Communications Magazine,* vol.51, no. 2, pp. 136-141, February 2013.

[32] Internet2's Advanced Layer 2 Service (Formerly OS3E). [Online]
URL: http://www.internet2.edu/products-services/advanced-networking/layer-2-services/
(Accessed: October 16th, 2014)

[33] M. A. Rodriguez , "Graph databases, trends in the web of data", Presentation at KRDB on Trends in the Web of Data School - Brixen/Bressanone, Italy– September 18, 2010. [Online]
URL: http://www.inf.unibz.it/krdb/school/2010/slides/krdbs-10-Rodriguez.pdf
(Accessed: March 21st, 2014)

[34] Neo4j. [Online]. URL: http://neo4j.com/ (Accessed: March 23rd, 2014)

[35] P. Ashwood-Smith. "Software Defined Networking and centralized controller state distribution reduction: Discussion of one approach". IEEE Draft, July 2012. (work in progress) [Online]. Available at:
Presentation: http://www.ieee802.org/1/files/public/docs2012/new-ashwood-sdn-optimizations-0712-v01.pdf
Text: http://tools.ietf.org/html/draft-ashwood-sdnrg-state-reduction-00

[36] ns-3 Documentation. [Online]
URL: http://www.nsnam.org/ns-3-14/documentation/
(Accessed: November 10th, 2014)

[37] OpenFlow Switch. [Online]
URL: http://openflowswitch.org (Accessed: October 11th, 2014)

[38] D. Ammar, T. Begin, I. Guerin-Lassous. “A new tool for generating realistic internet traffic in NS-3”. *4th international ICST Conference on Simulation Tools and Techniques* (SIMUTools '11). March 2011, Barcelona, Spain.

[39] M. Zukerman, T. D. Neame and R. G. Addie. “Internet traffic modeling and future technology implications”. *Proceedings of INFOCOM 2003*, April 2003, San Francisco, USA.

[40] L.Yao, M. Agapie, J. Ganbar, M. Doroslovacki. "Long range dependence in Internet backbone traffic," *IEEE International Conference on Communications (IEEE ICC '03).* May 2003, Alaska, USA.

[41] T. Karagiannis, M. Molle, M. Faloutos, "Long-Range Dependence - Ten Years of Internet Traffic Modelling", *IEEE Internet Computing*, vol. 8, no. 5, pp. 57-63, September-October 2004.