9-1-2015

Solving the Software Defined Network Controller Placement Problem Using Complex Network Analysis

Husain Alyusuf

Husain Al Yusuf

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Gregory Heileman, Chairperson

Dr. Chris Lamb

Dr. Ramiro Jordán

SOLVING THE SOFTWARE DEFINED NETWORK CONTROLLER PLACEMENT PROBLEM USING COMPLEX NETWORK ANALYSIS

BY
HUSAIN AL YUSUF
B. SC. ELECTRICAL & COMPUTER ENGINEERING
KING ABDULAZIZ UNIVERSITY, 2006

THESIS

Submitted in Partial Fulfillment of the
Requirement for Degree of

Master of Science
Computer Engineering

University of New Mexico
Albuquerque, New Mexico

July, 2015

DEDICATION

To my parents who have been my largest influence.

May both live a long healthy life.

To my family who helped me wherever I am.

To my friends who encourage me to succeed my studies.

Solving the Software Defined Network Controller Placement Problem Using Complex Network Analysis

By
Husain Al Yusuf

B.Sc., Electrical & Computer Engineering, King Abdualziz University, 2006

M.S., Computer Engineering, University of New Mexico, 2015

ABSTRACT

Software Defined Network (SDN) is a new architectural design for networking that is constructed based on decoupling the control plane from the data plane in networking devices and providing programmatic interface for the control plane. The introduction of SDN has raised many new networking problems; one of the most interesting debating questions is the SDN controller placement problem. In fact, there is no single best solution for the controller placement problem, as the solution depends on the desired metrics and requirements. Our research addresses the following question: Where is the best place to attach the SDN controller in a given large-scale network? Previous work has answered this question using mathematical models or algorithms. This study proposes a new approach for solving the controller placement problem by using complex network analysis and involves finding the most central node in a given network. Our solution will focus on improving the performance of the network and will be applied to the Internet2 topology by using GENI platform to simulate our experiment.

Table of Contents

List of Figures

1.0. Introduction

Before the Software-Defined Network (SDN) was introduced, the architecture of the information technology network combined the data plane and the control plane into the same network element, thus allowing decision-making and packet forwarding to be made in the same device. The SDN architecture is based on the design concept of decoupling the control plane from the data plane, and it has programmable interfaces into the control plane. This new architecture, which utilizes programmable controllers, enhances the intelligence of the networks' operations and enables network engineers to serve their business requirements more efficiently.

SDN capabilities raise many questions related to reliability, scalability, performance, and security. However, one of the common problems associated with SDN is the placement of the controller.

In the SDN architecture, the control plane is responsible for the decision-making process, while the data plane assumes the role of the forwarding element that executes controller decisions/instructions. This mechanism makes communication between forwarding elements and controllers a very critical factor for the flow of traffics in any network. This SDN architecture illustrates the importance of SDN controller placement, as this decision will greatly affect the overall performance of any SDN network.

Previous research has proposed solutions for the SDN controller placement problem, often proposing algorithms that sought to discover where to place the minimum amount

of controllers required for a given network [11]. Other studies suggest solutions based on mathematical models [1].

In this paper, a new framework based on centrality theory through the use of complex network analysis techniques will be proposed to solve the SDN controller placement problem. This study seeks to find the best placement for a controller in a given network under the assumption that the network needs only one controller. The SDN controller placement problem will be simulated using Internet2 topology using GENI platform.

This thesis begins with a brief introduction of SDN technology. Then, a detailed explanation of the SDN controller placement problem will be presented, along with a review of previous papers that have discussed this. In Section 5, complex network analysis is described, and simulation results are provided. Finally, future work and conclusions are presented.

2.0. Software Defined Networks (SDN)

Combining the data plane and control plane in the same device is the base design concept in traditional network architecture. This architecture utilizes the same processor to forward the packets. Also, the traditional design of networking prohibits each network device from having a global view of larger networks.

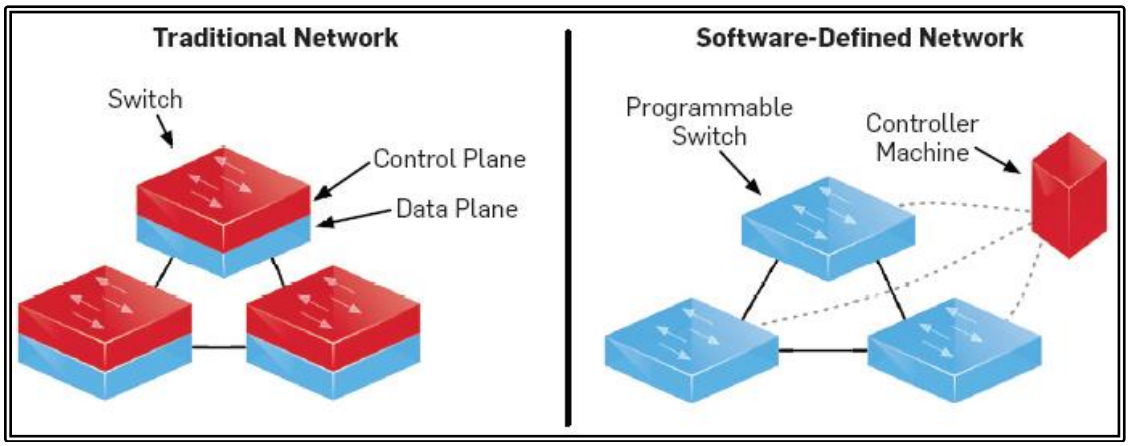

Figure 1. Traditional Network versus Software Defined Network

SDN is defined differently in many references, although all resources agree that the SDN concept of decoupling a network's control plane functions from its forwarding functions is the main concept of SDN design. The global view for the entire network infrastructure in a centralized programmable controller is SDN architecture's strongest asset, which allowed SDN to become a disruptive technology in the networking industry.

This architecture provides the controllers with a centralized global view. Forwarding elements that reside in the data plane are primarily responsible for executing the rules imposed by the controllers. More details about the control plane and the data plane interactions will be discussed in Section 2.2.

Most of the functionalities in traditional networks are implemented in dedicated appliances, which are technically referred to as middle-boxes (i.e., switch, router, monitoring tools, or application delivery controller). In addition, within the appliance, most of the functionality is implemented in hardware such as an ASIC (Application Specific Integrated Circuit) [5], which increases the complexity of change management, monitoring, and maintenance services.

From a financial point of view, this design requires engineers with different expertise to manage and operate such technologies. Eventually, traditional approaches to networking will increase the initial investments and the operational costs in comparison with SDN networking.

On the other hand, SDN innovation allows datacenters to reduce the operational cost by providing the network administrators with the ability to implement most of the functionalities into one box. This feature allows datacenters to get rid of “one function” hardware appliances, which usually cost tens of thousands of dollars. Moreover, traditional networking requires different engineers to manage different functionalities. For instance, a datacenter’s manager will hire an engineer to handle firewalls and another engineer for the routers, whereas SDN networks require only SDN expertise to manage the entire datacenter.

2.1. SDN Architecture

SDN networking is a new networking approach, and its new key attributes include: separation of control and data plane, centralized programmable controllers, and support of multiple isolated virtual networks [2]. SDN networks contain three main components: the infrastructure, controller, and application. These components interact with three type of interfaces (APIs), as shown in the following diagram:

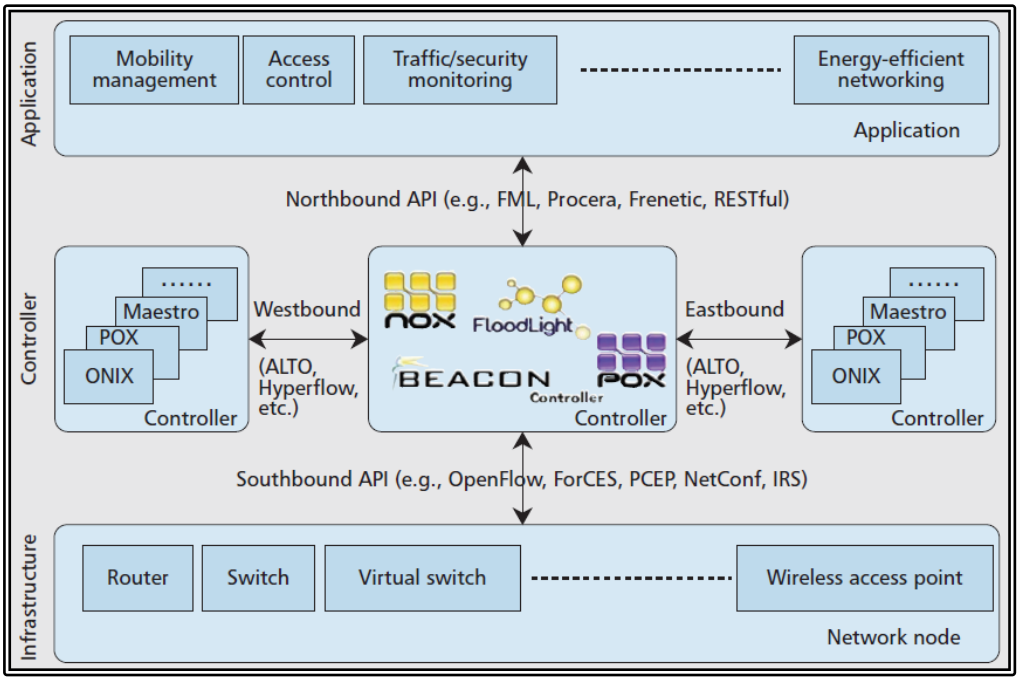

Figure 2. SDN Architecture

- *Southbound Interface*

The southbound interface is the interface between the programmable controller and the forwarding elements. This infrastructure uses many protocols, and the most popular protocol is OpenFlow. The southbound interface eliminates the need for proprietary protocols and has been adopted by many manufacturers.

The OpenFlow protocol -or any other southbound protocol- provides the controllers with the ability to communicate with any brand or type of hardware that supports OpenFlow. This design concept forces all manufacturers to implement the OpenFlow agent in most of their hardware, while also updating firmware for their current products to support OpenFlow. Southbound interface sends controller's instructions to forwarding elements. It also collects data and requests from the forwarding elements and then sends it to the controllers.

- *Northbound Interface*

The northbound interface is API between the controller and the applications running on top of the controllers' platforms. Firewall, intrusion detection systems, routing, quality-of-service (QoS), and load-balancer are examples of SDN applications that communicate with the controllers through the northbound APIs. These APIs can be deployed using many programming languages (i.e., Python, C, and C++). These applications create rules and instructions and send them to the controller, which delivers these rules into the entire infrastructure.

- *East-West Interface*

The east-west interface is the interface between controllers in the same network. Some enterprise networks have different geographical locations for their datacenters. These scenarios may require more than one controller to manage the network. East-west interfaces are used to exchange information between controllers for interdomain communication and resiliency in case of a failure.

2.2. SDN Controller Functionality

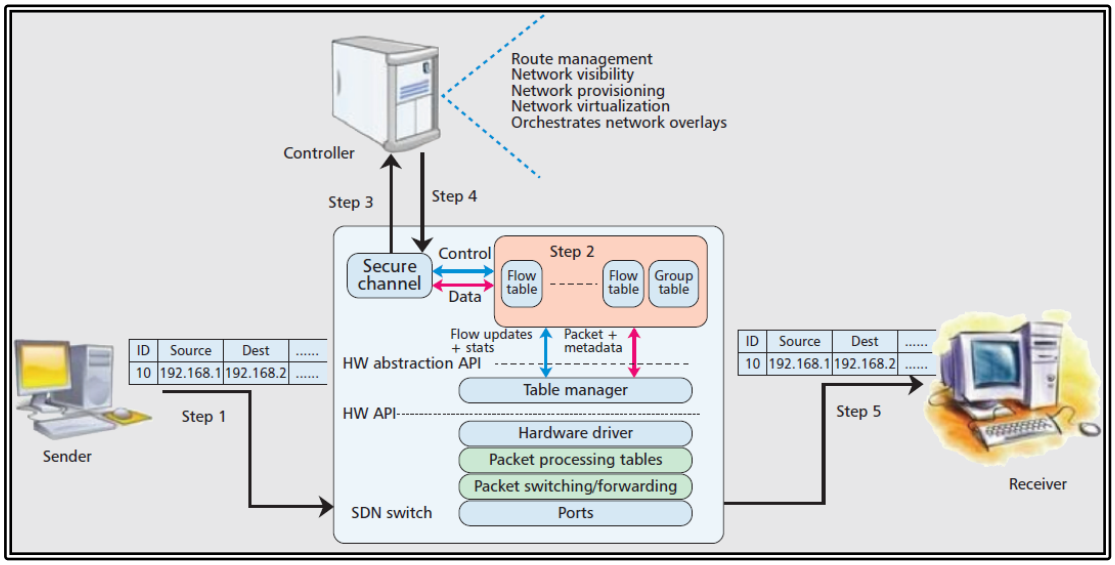

Figure 3. SDN Controller Functionality – Reactive Forwarding

SDN design architecture provides a centralized control plane that can manage all flow decisions and handle many roles in one box without the need for different types of middle-boxes. Currently, SDN networks handle most of the required datacenter's features and consolidate them into one box. SDN controllers, generally, operate in three modes [6].

- *Reactive Forwarding*

Reactive mode reacts to traffic, consults the controller, and creates rules for the forwarding elements based on the controller's instruction. As shown in Figure 3, whenever a new packet hits the switch, the controller's agent in this switch searches the contents of the inner flow tables. If no match for the flow is found, the switch creates a request, which is referred to as OpenFlow-Packet packet-in, and sends it to the controller for instructions.

- *Proactive Forwarding*

Rather than reacting to a packet, a controller populates the flow tables ahead of time for all traffic flows that could come into the switch, a process based on historical data collected by the controller. Flows also can be embedded manually into the controller by the network administrator in order to populate it toward the forwarding elements. In proactive forwarding, the flows and actions in a switch flows' tables are predefined; the packet-in event never occurs. As a result, all packets are forwarded at line rate merely by performing flow lookup in the switch flows' tables. This is the same hardware that currently populates its forwarding tables from "routing by rumor" routing protocols and "flood and spray" layer2 learning standards [6]. The flow tables that are created by proactive forwarding eliminate any latency induced by consulting a controller on every flow, as is the case in reactive forwarding.

- *Hybrid Forwarding*

Reactive forwarding adds another overhead to the end-to-end communication latency by adding the latency of the process of packet-in request, but this method provides a lot of flexibility. Proactive forwarding is faster, although network administrators find its static nature burdensome. A combination of both approaches allow more reaction flexibility for particular sets of granular traffic, while still preserving low-latency forwarding for the rest of the traffic. Some companies are required to operate in low-latency and restricted polices, so reactive-forwarding or proactive-forwarding alone would not be beneficial. Hybrid forwarding, the architecture mostly used in the SDN industry, could be reasonable in an enterprise if these polices are important.

3.0. SDN Controller Placement Problem

Some network engineers argue that the new controller-based architecture improves the network's status convergence and yields a flexible/evolvable network, while others raise concerns about decision latency and availability due to the decoupling of the control plane from the data plane. Both opinions are right to some extent. The centralized decision-making and monitoring inside a single programmable box gives the SDN network the ability to improve the convergence and flexibility of network operations.

On the other hand, adding another hop, the controller, (explained in Section 2.2) in the communication between two ends may cause additional overhead to the total delay, and any controller's failure may cause disturbance to the networks' services.

Understanding where to place controllers and how many to use is a prerequisite to answering SDN performance and fault tolerance questions and for comparing them to traditional architectures. We call this design choice the *Controller Placement Problem* [1].

In this paper, one part of the controller placement problem will be considered, with an emphasis on answering the question of where to place the controller. This study will not address the required quantity of controllers in a given network. Accordingly, we will not address the effect of controller placement from resiliency prospective, but we will focus on the performance of the network.

It is reasonable for small/medium datacenters or networks to ignore the controller placement problem, because the effect of such a problem in small-scale networks is often negligible. However, proper placement of a controller in a large-scale or wide-area

network will minimize propagation delays. In a data center or enterprise, one might instead maximize fault tolerance or actively balance controllers among administrative domains.

4.0. Background

The SDN placement controller problem is not a new issue in SDN. Every study approaches the problem from different perspectives. Previous studies have used different methodology to formulate/resolve the problem [11] [14]; some studies rely on mathematical models while other construct graphical models for the same problem [11] [15].

Sherwood et al published one of the most cited papers for the SDN controller placement problem [1]. This work formulates the controller placement problem as an optimization problem. This work does not solely consider the location of the controller, but it includes the number of controllers required for a specific network topology. Their approach is based on mathematical formulas that consider the "average-latency" and "worst-case-latency" as the main metrics. These metrics are represented mathematically for a graph G(V,E) as follows:

- *Average-latency:* $L_{avg} = \frac{1}{n} \sum_{v \in V} \min_{s \in S'} d(v, s)$
- *Worst-case latency:* $L_{wc} = \max_{v \in V} \min_{s \in S'} d(v, s)$

, where edges represent propagation latency, n is number of nodes, and $d(v, s)$ is the shortest path between $v \in V$ and $s \in V$.

The module depends mainly on the propagation delay assigned to each edge, which is the main consideration for the controller placement problem in large-scale networks. The measurements are based on propagation delay between nodes and controllers, it does

not look into the effect of the controller placement on end-to-end communication between two nodes in SDN networks. Moreover, each optimal placement shown in this paper comes from directly measuring the metrics of all possible combinations of controllers. Although this method ensures accurate results, the time complexity is exponential since the optimal placement was obtained using brute force approach where every possible location was evaluated.

By applying this to the Internet2 network, the model places the controller in Chicago for average-latency and Kansas for worst-cast-latency. The paper applied the proposed solution into WAN networks only, although network's resiliency has not been considered into the analysis, which is a very critical factor in WAN networks.

One of the most interesting conclusions from this study indicated that adding more controllers to the network will not necessarily reduce latency. After a certain amount of controllers, and depending on the topology, additional controllers will not reduce latency.

Xiao et al addresses the controller placement problem in WAN networks [11]. This research approaches the controller placement problem differently. Their methodology is based on partitioning WAN networks into smaller SDN networks (or SDN domains) and then finding the best placement for the SDN controller in each partition of the network.

A spectral clustering algorithm was used to segregate the WAN network. The success of such an algorithm depends heavily on the applied metric. The weight of the links between nodes is implemented as the metric for the spectral algorithm. The propagation latency have been considered as the weight of the edges between nodes. Accordingly, the nodes

with low weighted link are most likely used to create a group of nodes (smaller SDN network), whereas the links between groups are highly weighted (links between networks).

The decision of where to place the controller within the segregated SDN networks is based on the average-latency formula as presented in [1]. This strategy shows that the contribution of this paper is the technique of splitting the large-scale networks into smaller ones, while using previous methods to obtain the optimal placement for the SDN controller within each smaller network.

Their experimental environment contains 36 machines with *Beacon* controller and *cbench* as a tool to simulate open flow switches. Their findings are based on the extracted measurements from the simulation. This fact made this study more robust in terms of its findings, in comparison to other studies that lack of experimental measurements for their proposed resolutions. Moreover the mathematical model used considers metrics other than latency, such as load-balancing and reliability. This approach does not only improve the performance, but can significantly improve the reliability of SDN network.

There are several other previous works that discuss the SDN controller placement problem, however the motivation to resolve the problem is different than the two previous papers and my work.

Hu et al [12] developed few placement algorithms to find the best location for the SDN controller in a given network, however his goal was different than previous works. The objective of these placement algorithms was to maximize reliably, whereas other papers

focused on performance. The parameter used to solve the problem was the control path of the network, where control data transfers between the controllers and the forwarding elements. The notion was that if fewer control paths fail, then there will be less of an impact on the network. Therefore, controllers are placed based on the control paths of the given network.

Beheshti et al in [13] has proposed a similar work to [11]. This paper describes the controller placement problem in terms of resiliency. The connection resiliency between the controller and the forwarding elements was considered as the metric. This metric reflects the ability of the forwarding elements and the controller to protect the control-paths of the network. The proposed algorithms in this paper aimed to maximize the possibility of fast failover based on resilience-aware controller placement and routing of the control's traffic.

One of the most important findings in this paper was that resiliency can be improved by pre-configuring backup links/routes in the switches towards the controllers. If there is a failure in the outgoing link or upstream node of a switch, then these backup links/interfaces will re-route the control traffic to the controller.

Guo et al has proposed another work [14] that also approaches the placement problem in term of resiliency. In this research, the analysis of the controller placement are based on resiliency from the perspective of interdependent networks. They define the SDN network with two interdependent networks: the switch-to-switch networks, and controller-switch networks. The interdependence graph allows for the analysis of

cascading failure in multiple networks due to their node dependence relationships. By performing that analysis, they found a positive relationship between the fractions of nodes that survived at the steady state of the cascading of failure and network's resiliency to that failure.

A large number of variants of the controller placement problem have been proposed in the literature in an attempt to study the performance and solution quality.Most of these studies are based on mathematical optimization models. To empirically validate their proposed frameworks, most of these studies used virtual networks. The main drawback for these results is the fact that they do not simulate real life environments.

To overcome this drawback, we implemented a new method that has a twofold goal: this method implements for the first time a complex network model to solve the controller placement problem and besides this method uses a real life environment to validate the proposed framework.

5.0. Solving the Controller Placement Problem by Complex Network Analysis

This research approach to solving the SDN controller placement problem is based on complex network analysis through the application of the centrality concept. Internet2 network will be used as an example problem for our solution. Before discussing the details of the solution, the centrality and methodologies used to solve the controller placement problem will be described.

5.1. Centrality

In order to discuss centrality, we must first examine the graphical representation for physical networks. Networks are usually donated as graphs {G(V,E)}. Graph G contains a set of nodes that is represented by vertices V. These nodes are connected with bidirectional links represented as a set of edges (E).

The definition of centrality can be defined by the answer to the following question: “which is the most important or central node in a given network?”[4] This is an overly generalized question that would need to elaborate on the meaning of the phrase “most important”. In the technology literature, the word "importance" has a vast array of definitions that can refer to a type of flow or transfer across the network, which allows centralities to be classified by the type of flow that is considered important. The centrality affect the cohesiveness of a network, which allows centralities to be classified based on how they measure cohesiveness. These definitions divide centralities into distinct categories [8]. Accordingly, finding centrality has many methodologies. We will use the betweenness centrality methodology and the closeness centrality methodology.

5.1.1. Betweenness Centrality

Suppose we have a network with consistent data packets (i.e. messages), with the assumption that data packets always take the shortest path (i.e. geodesic) between all vertices. In order to understand the betweenness centrality we must answer the following question: if we wait long enough for large number of messages (i.e. data traffic) to pass between each pair of vertices in the network, then how many messages have passed through each vertex in route towards their destination? The answer represents the betweenness centrality of each vertex, where the vertex with the highest number of betweenness centrality is the most central vertex for the network. Also, by assuming that all the messages are passing through each shortest path at the same rate, the number of messages passing through each vertex is proportional to the number of shortest paths on which the vertex lies.

The betweenness centrality for a node is the number of shortest paths from all vertices to all others that pass through that node. The node with the highest betweenness centrality in a network is the most central node in that network. A node with high betweenness centrality has a large influence on the transfer of items through the network. Here is a simple mathematical representation for betweenness centrality:

$$\boldsymbol{x_i = \sum_{st} n_{st}^{i}}$$

where n_{st}^{i} is equal to 1 if vertex i lies on the shortest path from s to t or 0 if it does not [4].

According to the betweenness theory, the vertices with the highest betweenness centrality will handle the largest number of messages passed through. These vertices could derive a significant amount of power from their position within the network. If these vertices were removed from the network, it would most likely disturb communication between other vertices because they have the highest amount of traffic flow. Based on the characteristics of betweenness centrality, where the node with the most traffic passing through it is the most central node. Betweenness centrality has been chosen as the methodology to solve the SDN controller placement problem because placing the controller into the most central node reduces the latency caused by the controller traffic.

5.1.2. Closeness Centrality

Closeness centrality differs from betweenness centrality in the method used to count the centrality of all vertices in the entire network. Closeness centrality measures the mean geodesic distance from a vertex to all other vertices in the same network and implies that the more central a node is, the lower its total geodesic distance to all other nodes. Closeness can be regarded as the amount of time it will take to spread information from a vertex to all other vertices sequentially. Here is a simple mathematical representation for closeness centrality:

$$\boldsymbol{C_i = \frac{n}{\sum_j d_{ij}}}$$

where d_{ij} is the length of a geodesic path from i to j (the number of edges along the path), and n is the number of vertices in the entire network. [4]

Closeness centrality is typically used in social network analysis because it measures the mean distance in relationships between people. The people with high closeness centrality have more influence on others or have access to more information than others. However, closeness centrality can be used in other types of networks.

In this study, closeness centrality is used to place the controller in a vertex that is centrally located in geodesic paths for other nodes. This study will explore the ability of closeness centrality to solve the SDN controller problem in comparison to betweenness centrality.

5.2. Controller Placement

This section presents a detailed explanation of applying complex network analysis to the Internet2 network [7]. Internet2 topology is shown in the figure below.

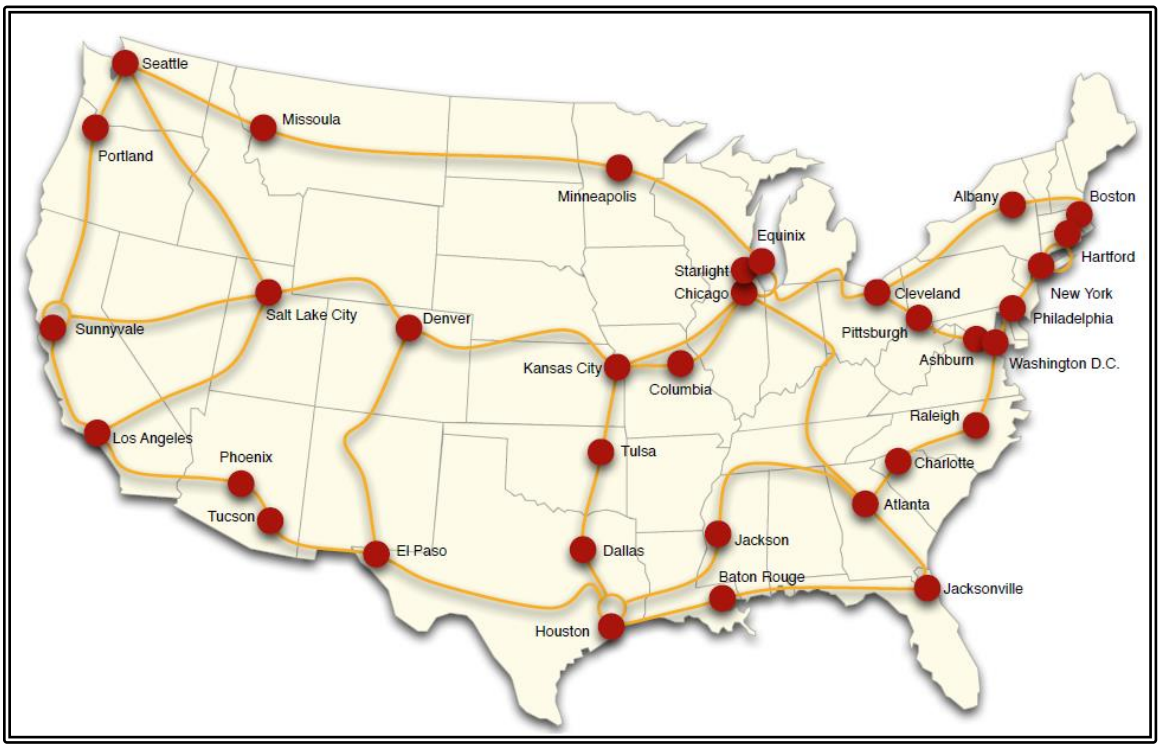

Figure 4. Internet2 Network Topology

In this work, MATLAB will be used to implement the betweenness centrality and closeness centrality on Internet2 network. In accordance with MATLAB's results, the GENI [10] platform will be used to simulate the SDN controller problem in Internet2.

In this experiment, an adjacency matrix is used to represent the Internet2 network, as shown in Appendix A. This matrix represents the distances among the nodes in the network. This study focus on the effect of the distance between nodes on the propagation delay, because the bandwidth between the nodes in Internet2 is constant. Thus, the bandwidth does not add any result to our study. While the bandwidth is constant, distance is a key factor in the propagation delay of the traffic between any two nodes.

5.2.1. MATLAB - Experiment Platform

A network topology can be represented graphically or mathematically; a weighted adjacency matrix has been used to represent Internet2 topology mathematically. MATLAB's library has been used to implement betweenness centrality and closeness centrality on the Internet2 network. The distance between nodes in Internet2 will be considered as the weight of the links. Procedure of the experiment installation appear below:

- List the nodes of the internet2 network and associate each node with a state's name and number, as shown in Appendix A.
- Build the weighted adjacency matrix, as shown in Appendix A, and save it in a variable. This example uses "internet2" as the variable for the matrix.

- Select the complex network library in MATLAB that should include the betweenness centrality and closeness centrality classes.
- Run the following commands:
  - For betweenness centrality → *[x,y]=betweenness_centrality(internet2)"*
  - For closeness centrality → *"x=closeness(internet2)"*

5.2.2. MATLAB - Experiment Outcomes

Here are the outcomes of the experiment in MATLAB,

>> [x,y]=betweenness_centrality(A)		
x =		
51.5	*403.0667*	*<--- this is Chicago*
3.6667	*0*	
2.6667	*154.9667*	
28	*10.6333*	
55	*302.5*	
200.2	*85.5*	
17.6333	*132.5*	
49.6333	*313.7*	
251.0333	*134*	
181.9333	*86*	
78.3333	*42*	
225.4	*18*	
16.6667	*40*	
20.6667	*106.5*	
34.9667	*50*	
124.1667	*95.5*	
0	*35.6667*	

>> x=closeness(internet2)		
x =		
0.0071	*0.0109*	*<--- this is Chicago*
0.0066	*0.0089*	
0.0065	*0.0094*	
0.0066	*0.0074*	
0.0073	*0.0096*	
0.008	*0.0079*	
0.0063	*0.0083*	
0.0071	*0.0101*	
0.0093	*0.0078*	
0.0088	*0.0065*	
0.008	*0.0057*	
0.01	*0.0053*	
0.0083	*0.0057*	
0.0083	*0.0066*	
0.0082	*0.0068*	
0.0093	*0.0072*	
0.0083	*0.0078*	

As shown above, the most central node in Internet2 is the Chicago exchange node. The result shows high betweenness centrality for the Chicago node; no other node has as high a score for betweenness centrality. This implies that Chicago is the node with most traffic

passing through. Interestingly, when applying closeness centrality to Internet2, it gives the same central node as betweenness centrality, which means that Chicago is the lowest total distance from all other nodes in Internet2.

5.2.3. GENI - Experiment Platform

GENI (Global Environment for Network Innovations) provides a virtual laboratory for networking and distributed systems research and education. In technical terms, GENI is the combination of several datacenters distributed over the United States that are connected by the Internet2 infrastructure. A GENI user can request resources from one or many datacenters to perform an experiment or simulation.

The aim of this experiment was to simulate a SDN network into Internet2 topology and to analyze the impact of controller placement on the network's performance in terms of total delay. Because GENI's racks are connected via Internet2 infrastructure, GENI can provide the features that allow simulating the SDN placement problem into the Internet2 real network. The following list will describe the experiment in detail:

- The project has to be created by a professor (i.e., an advisor) into GENI, where only the professor's user accounts are allowed to create projects. Students cannot start working in GENI racks without being engaged in a project.

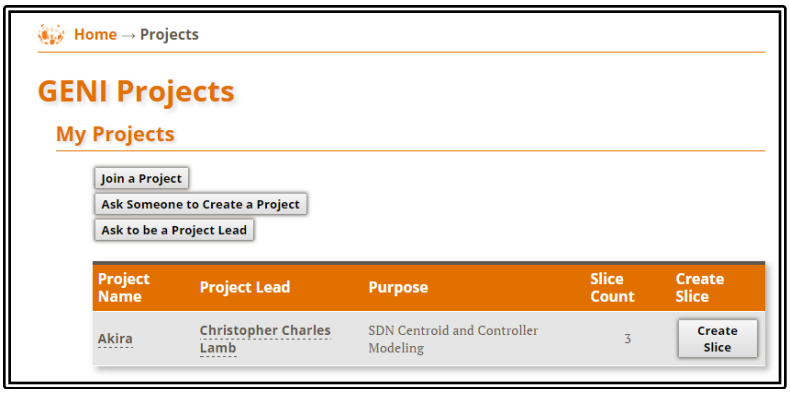

Figure 5. GENI – Create Project

- Join the created project and create a slice, a container that includes resources that can be requested.

Figure 6. GENI – Create Slice

- Add resources to the created slice as the following:
    a. VM (controller) in Chicago at University of Chicago InstaGENI rack.
    b. 2 VMs connected to ovs-switch in Los Angeles at CENIC InstaGENI rack.
    c. 2 VMs connected to ovs-switch in DC at MAX InstaGENI rack.

Many types of VMs exist in GENI and the default-vm has been used in this simulation. All VMs of the same type have similar specifications; end users cannot change the specifications of any VMs. However, a specific operating system's image has been used in the ovs switches and request to install a package into all VMs. Details about the image and the package are as follows:

a. Switches' image: https://www.geni.it.cornell.edu/image_metadata.php?uuid=06efdaf9-bd28-11e4-81ff-000000000000

b. VM's package: http://emmy9.casa.umass.edu/GEC-20/gimidev.tar.gz

- Connect the ovs-switches to the controller in Chicago using "stitched" link. Stitched link is a type of connection in GENI that allows connecting resources from different geographical location through Internet2 infrastructure, as shown in the following diagram:

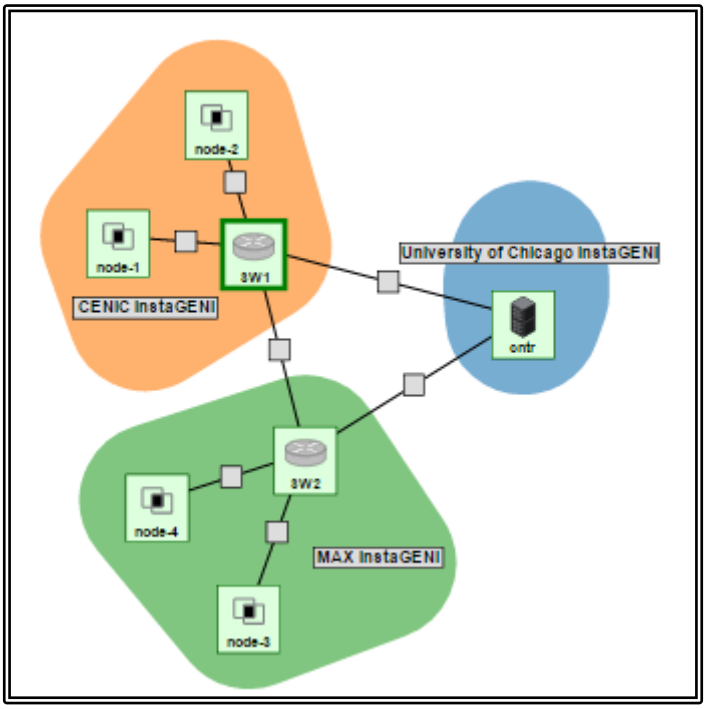

Figure 7.Experiment Network's Topology in GENI

- Configure each node as follows:

a. Node-1:
sudo ifconfig eth1 10.10.10.1/24

b. Node-2:
sudo ifconfig eth1 10.10.10.2/24

c. Node-3:
sudo ifconfig eth1 10.10.10.3/24

d. Node-4:
sudo ifconfig eth1 10.10.10.4/24

e. Cntr:
sudo ifconfig eth1 10.10.1.1/24
sudo ifconfig eth1 10.10.2.1/24
./pox/pox.py forwarding.l2_learning

f. SW1:
sudo ifconfig eth1 10.10.1.2/24 <--- connected to cntr
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 eth2
sudo ovs-vsctl add-port br0 eth3
sudo ovs-vsctl add-port br0 eth4
sudo ovs-vsctl set-controller br0 tcp:10.10.1.1:6633
sudo ovs-vsctl set-fail-mode br0 secure

g. SW2:
sudo ifconfig eth1 10.10.2.2/24 <--- connected to cntr
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 eth2
sudo ovs-vsctl add-port br0 eth3
sudo ovs-vsctl add-port br0 eth4
sudo ovs-vsctl set-controller br0 tcp:10.10.2.1:6633
sudo ovs-vsctl set-fail-mode br0 secure

- Launch Labwiki. Labwiki is a built-in tool in GENI that helps to perform an experiment with the reserved resources and extracts readings/results from the experiment. In this example, Labwiki was used to measure RTT (round-trip-time) of ICMP packets between node-1 and node-4.
- In Labwiki, use the script shown in Appendix B to perform "PINGs" operation between node-1 in Los Angeles and node-4 in DC. The output of the script will be presented graphically in Labwiki. Results will be presented in the following section.
- Repeat the same experiment twice more, using a different location for the controller in each attempt- one in Kansas at Kansas-InstaGENI rack and another in Minneapolis at Wisconsin-InstaGENI rack.

5.2.4. GENI - Experiment Outcomes

The experiment mentioned in the previous section was installed using the reactive forwarding technique as explain in section 2.2. The ICMP protocol has been used to measure the effect of the controller's location on the additional delay of controller-switch communication. RTT (Round-Trip-Time) is the period of time it takes for the ICMP packet to travel to the destination host, as well as the time it takes the acknowledgment to reach the source again. In a reactive controller design, an additional delay will appear during the first ICMP packet, because the switch consults the controller to know how to forward the packets. This delay will only occur for the first packet. This experiment measures the RTT of the ICMP's traffic between two hosts in Los Angeles and DC with three different locations for the controller. The following table shows the average RTT (ms) for ten repetition for this experiment and Figure.10 shows the average for these ten attempts. Appendix C includes more details about each attempt results

Packet#	Controller Location		
	Chicago	Kansas	Minneapolis
1	378.9	475.2	464.2
2	72.6	83.8	72.3
3	71.7	71.5	71.9
4	71.8	71.7	72
5	71.9	71.7	72

The following diagrams show samples of the output from the Labwiki tool.

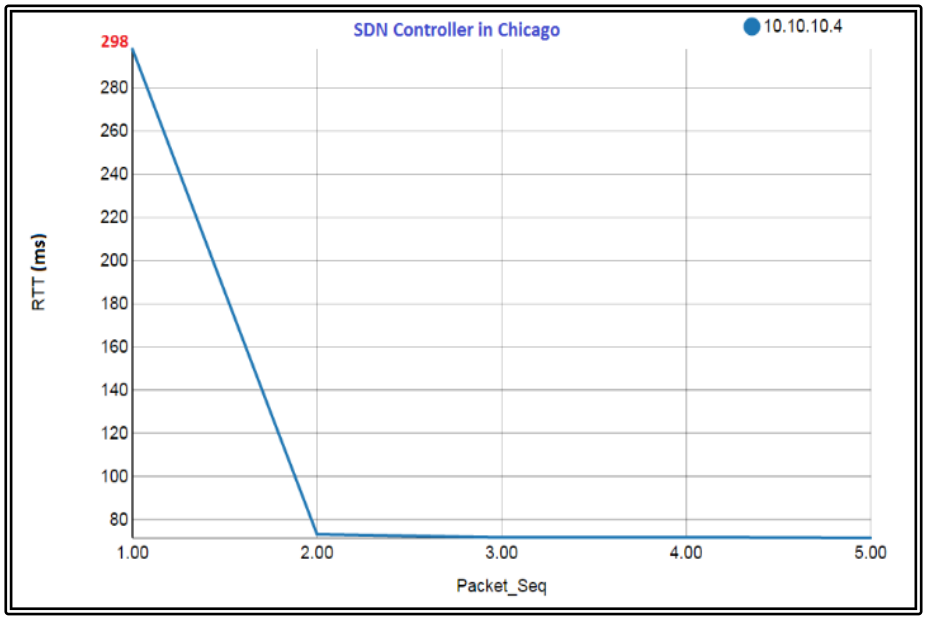

Figure 8. Labwiki - RTT for ICMP packets with SDN controller in Chicago.

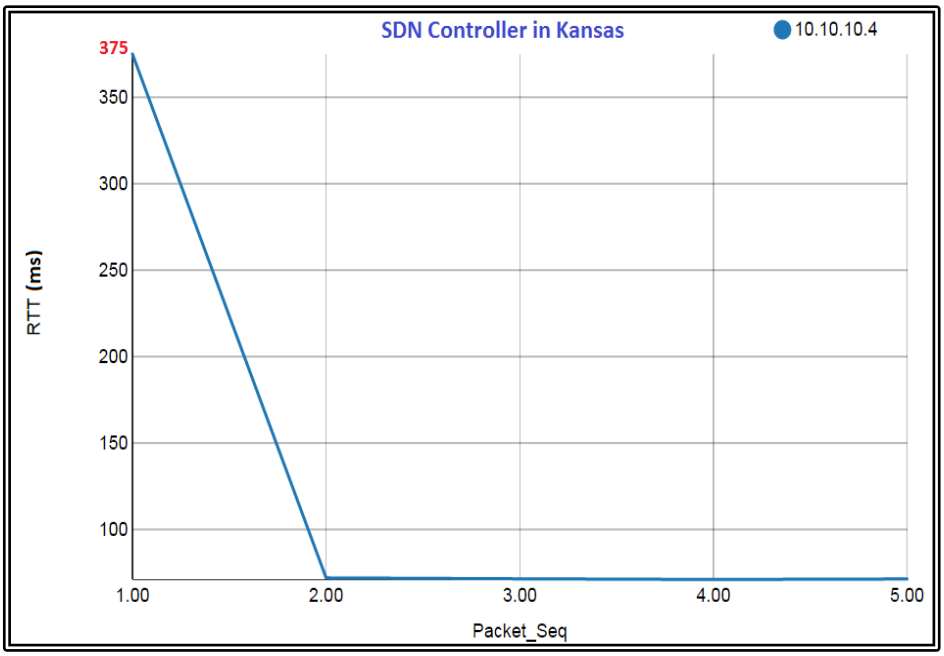

Figure 9. Labwiki - RTT for ICMP packets with SDN controller in Kansas

5.3. Result Analysis and Reflections

In this section, we will discuss the findings, outcomes, and reflections of the study in its entirely.

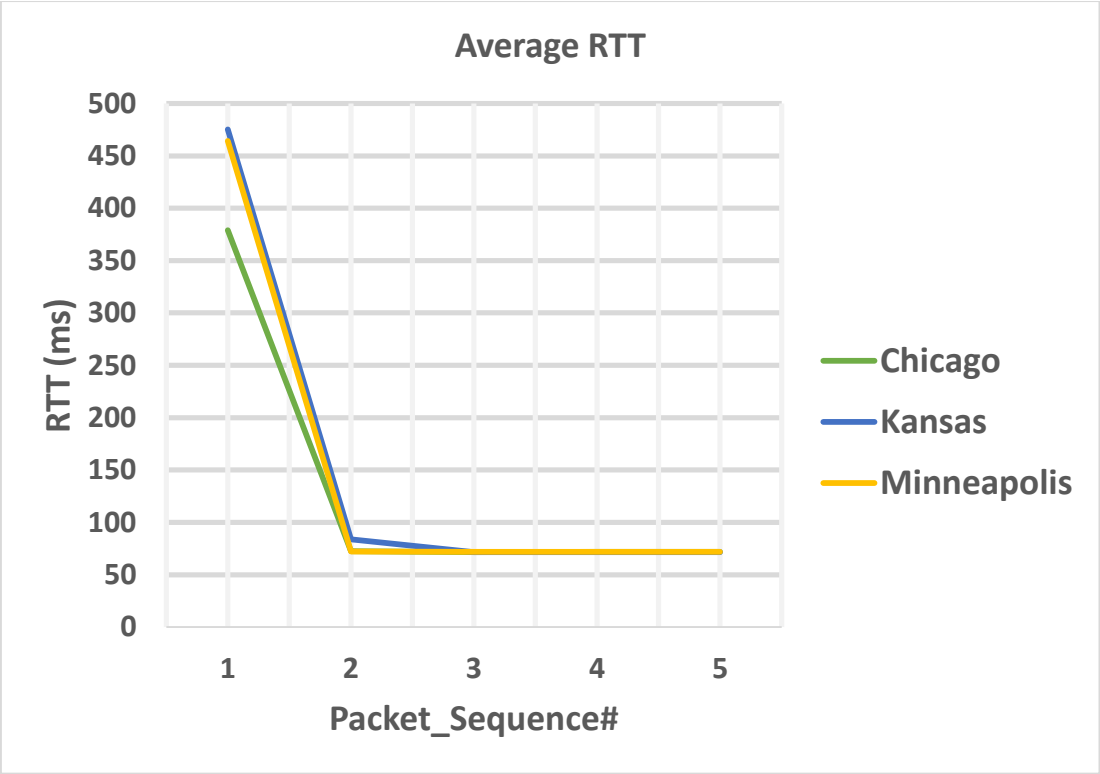

Figure 10. Compare Average RTT for Different Controller Locations

- As shown in the above charts, when the SDN controller is placed in Chicago, the average RTT for the first ICMP packet is 378.9ms, while average RTT is 464.2ms when the controller is placed in Minneapolis. Finally, the average RTT is 475.2ms when the controller is located in Kansas.
- The results indicates that placing the SDN controller in the most central node of the Internet2 network (which is Chicago according to the analysis in section 5.2.2.) will provide a better performance and will automatically lower the additional delay that is usually caused by the controller-forwarding element's communication.

- Although 85ms and 96ms may not seem significantly different numerically, this difference will impact other applications and network engineers will consider it as a key impact on the end-to-end quality of communication.
- This diagram also shows how the RTT is dramatically decreased in at the second packet. This occurs because the switch no longer need to send request to the controller as it has already installed the flow for such packets into all forwarding elements.
- Solving the controller placement problem with complex network analysis techniques provides results that are similar to past studies. Chicago was selected as the best location for Internet2 topology, and this conclusion is also seen in Sherwood's work [1]. The similarity in results occurred because the methodologies used in both works depended on the weight of the links within the shortest paths.
- The simulation findings, as explained in the previous section, also reflect the theoretical analysis. The simulation was performed on the real Internet2 network, making our methodology more reliable. In contrast to previous studies that often did not simulate their findings in a physical networks and typically conducted lab simulations only.
- The latency of end-to-end communication is an important indicator of the quality of network architecture performance, hence why PING packets were used to examine the effect of the controller's placement into total latency.

6.0. Conclusions and Future Work

SDN has begun to appear in datacenters for many industries, after several years of being exclusively implemented in research labs or technology companies. Some network engineers are still hesitant to implement it on the WAN networks, although a few technology companies have already done so [9]. Further research that identifies the solution for the SDN controller problem will allow SDN technology to gain a strong presence in industry. While the SDN architecture that decouples the control plane from the data plane provides many benefits, it also has created some problems, including the placement of the SDN controller.

Although, many previous studies have proposed solutions for the SDN controller problem through the use of algorithms and mathematical models, this study has found a new approach to solving this problem by using complex network analysis with centrality theory. This work has applied the betweenness centrality and closeness centrality analysis into a real life topology, such as the Internet2 network. Interestingly, both methods (closeness centrality and betweenness centrality) lead to the same central node for the entire Internet2 network, despite the fact that each method uses different metrics to find the centrality.

Intelligent design of the GENI platform gives us the ability to simulate the SDN placement problem in Internet2 topology, where GENI uses Internet2 infrastructure as the connection between its datacenters. The results extracted from the GENI platform reflect

the network analysis and its indication that placing the controller in Chicago will provide better performance (in terms of delay) than placing it in other states.

Controller placement is an important factor in the SDN network's performance, scalability, and resiliency. This research has benefited from using methodology that finds the most central node in a given network based on traffic flow (as in betweenness centrality) and shortest distance (as in closeness centrality) to ultimately improve performance. Placing a controller into a central location in any given network will reduce the total delay of traffic caused by proactive/reactive forwarding techniques used in SDN-forwarding elements communication.

7.0. Appendices

7.1. Appendix A. Internet2 Topology's Representation in Adjacency Matrix

List of nodes in Internet2 network as shown in Fig.4, where the number that is associated with the city will be the coordinate of the matrix, for example, node in Seattle is number 1 then element (1,1) in the adjacency matrix will be a representation for Seattle node. The list of the nodes as follow:

1- Seattle
2- Portland
3- Sunnyvale
4- Los Angeles
5- Missoula
6- Salt Lake City
7- Phoenix
8- Tucson
9- Denver
10- El Paso
11- Minneapolis
12- Kansas City
13- Tulsa
14- Dallas
15- Houston
16- Equinox
17- Starlight
18- Chicago
19- Columbia
20- Jackson
21- Baton Rouge
22- Cleveland
23- Pittsburgh
24- Charlotte
25- Atlanta
26- Albany
27- Boston
28- Hartford
29- New York
30- Philadelphia
31- Washington D. C.
32- Ashburn
33- Raleigh
34- Jacksonville

Here is the full adjacency matrix for Internet2 topology:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
1	0	173	0	0	477	689	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	173	0	666	0	0	1434	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	666	0	351	0	768	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	351	0	0	590	372	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	477	0	0	0	0	0	0	0	0	0	1183	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	689	1434	768	590	0	0	0	0	519	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	372	0	0	0	116	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	116	0	0	318	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	519	0	0	0	632	0	603	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	318	632	0	0	0	0	906	667	0	0	0	0	1110	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	1183	0	0	0	0	0	0	0	0	0	0	408	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	603	0	0	0	271	0	0	0	0	510	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	271	0	258	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	906	0	0	258	0	239	0	0	0	0	682	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	667	0	0	0	239	0	0	0	0	0	443	271	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	408	0	0	0	0	0	1	1	0	0	0	345	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	510	0	0	0	1	1	0	386	0	0	345	0	0	716	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	126	0	0	0	0	0	386	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	1110	0	0	0	682	443	0	0	0	0	0	0	0	0	0	390	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	271	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	603
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	345	0	345	0	0	0	0	134	0	0	472	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	134	0	0	0	0	0	0	0	0	0	228	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	245	0	0	0	0	0	0	0	168	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	716	0	390	0	0	0	245	0	0	0	0	0	0	0	0	0	346
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	472	0	0	0	0	168	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	0	100	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	117	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	0	94	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	94	0	139	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	139	0	32	263	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	228	0	0	0	0	0	0	0	32	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	0	0	0	0	0	0	263	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	603	0	0	0	346	0	0	0	0	0	0	0	0	0

7.2. Appendix B. GENI – Labwiki Script

```
defProperty('resource1', "node-1", "ID of a resource")
defProperty('sinkaddr11', '10.10.10.1', "Ping destination address")
peak_list = []

defApplication('ping') do |app|
 app.description = 'Simple Definition for the ping-oml2 application'
 # Define the path to the binary executable for this application
 app.binary_path = '/usr/local/bin/ping-oml2'
 # Define the configurable parameters for this application
 app.defProperty('target', 'Address to ping', '-a', {:type => :string})
 app.defProperty('count', 'Number of times to ping', '-c', {:type => :integer})
 # Define the OML2 measurement point that this application provides.
 app.defMeasurement('ping') do |m|
  m.defMetric('remote',:string)
  m.defMetric('ttl',:uint32)
  m.defMetric('rtt',:double)
  m.defMetric('rtt_unit',:string)
 end
end

defGroup('Source1', property.resource1) do |node|
  node.addApplication("ping") do |app|
   app.setProperty('target', property.sinkaddr21)
   app.setProperty('count', 5)
   #app.setProperty('interval', 1)
   app.measure('ping', :samples => 1)
 end
end

onEvent(:ALL_UP_AND_INSTALLED) do |event|
 info "Starting the ping"
 after 5 do
  group('Source1').startApplications
 end
 after 80 do
  info "Stopping the ping"
  allGroups.stopApplications
  Experiment.done
 end
end

defGraph 'RTT' do |g|
 g.ms('ping').select(:oml_seq, :remote, :rtt)
 g.caption "RTT of received packets."
 g.type 'line_chart3'
 g.mapping :x_axis => :oml_seq, :y_axis => :rtt, :group_by => :remote
 g.xaxis :legend => 'Packet_Seq'
 g.yaxis :legend => 'RTT', :ticks => {:format => 's'}
end
```

7.3. Appendix C. The Experiment Results

Location	Chicago					Kansas					Minneapolis				
Attempt	Packet_Sequence#					Packet_Sequence#					Packet_Sequence#				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	671	72	72	72	72	372	111	72	76	71	390	72	72	72	72
2	473	73	72	72	72	375	72	72	72	72	385	72	71	72	72
3	331	73	72	72	72	363	151	72	71	72	412	73	72	72	72
4	338	72	72	72	72	373	72	71	71	71	370	72	72	72	72
5	338	73	71	72	72	741	72	71	71	71	370	72	72	72	72
6	328	73	71	72	72	718	72	71	71	72	740	72	72	72	72
7	298	73	71	71	71	362	72	71	71	72	374	73	72	72	72
8	338	72	72	72	72	354	72	72	71	72	815	72	72	72	72
9	325	73	72	71	72	735	72	71	71	72	407	72	72	72	72
10	349	72	72	72	72	359	72	72	72	72	379	73	72	72	72
Average RTT (ms)	378.9	72.6	71.7	71.8	71.9	475.2	83.8	71.5	71.7	71.7	464.2	72.3	71.9	72	72

8.0. References

[1] B. Heller, R. Sherwood, and N. McKeown, “The Controller Placement Problem”, in HotSDN, New York, NY, USA, 2012.

[2] Myung-Ki Shin, Daejeon, Ki-Hyuk Nam, Hyoung-Jun Kim, “Software-defined networking (SDN): A reference architecture and open APIs”, in ICTC, South Korea, 2012.

[3] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Rao, “Are We Ready for SDN? Implementation Challenges for Software-Defined Networks”, in IEEE, USA, 2013.

[4] M. E. J. Newman, “Measure and Metrics”, in *Networks, An Introduction*, 1st ed. Oxford, UK: Oxford University Publisher, 2010.

[5] “SDN 101: An Introduction to Software Defined Networking” CITRIX, 2014. < http://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking.pdf>.

[6] Brent Salisbury. (2013, January 15). OpenFlow: Proactive vs Reactive Flows. [Online]. Available: http://networkstatic.net/openflow-proactive-vs-reactive-flows/

[7] Internet2 open science, scholarship and services exchange. <Internet2.com>

[8] Centrality. In Wikipedia. Retrieved February 10, 2015, from http://en.wikipedia.org/wiki/Plagiarism

[9] Jain, Sushant, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata et al. "B4: Experience with a globally-deployed software defined WAN." In ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 3-14. ACM, 2013.

[10] GENI (Global Environment for Network Innovations) provides a virtual laboratory for networking and distributed systems research and education. <geni.net>

[11] Xiao, Peng, Wenyu Qu, Heng Qi, Zhiyang Li, and Yujie Xu. "The SDN controller placement problem for WAN." In Communications in China (ICCC), 2014 IEEE/CIC International Conference on, pp. 220-224. IEEE, 2014.

[12] HU, Yan-nan, et al. "On the placement of controllers in software-defined networks." *The Journal of China Universities of Posts and Telecommunications*19 (2012): 92-171.

[13] Beheshti, Neda, and Ying Zhang. "Fast failover for control traffic in Software-defined Networks." *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012.

[14] Guo, Minzhe, and Prabir Bhattacharya. "Controller Placement for Improving Resilience of Software-Defined Networks." *Networking and Distributed Computing (ICNDC), 2013 Fourth International Conference on*. IEEE, 2013.

[15] Hock, David, et al. "Pareto-optimal resilient controller placement in SDN-based core networks." Teletraffic Congress (ITC), 2013 25th International. IEEE, 2013.