2018-03

Traffic congestion analysis for a software-defined network

Maxie, Moniqua J.

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/58337

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TRAFFIC CONGESTION ANALYSIS FOR A SOFTWARE-DEFINED NETWORK

by

Moniqua J. Maxie

March 2018

Thesis Advisor:	Murali Tummala
Co-Advisor:	John McEachen

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2018	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE TRAFFIC CONGESTION ANALYSIS FOR A SOFTWARE DEFINED NETWORK	**5. FUNDING NUMBERS**
6. AUTHOR(S) Moniqua J. Maxie	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The objective of this thesis is to implement an anomaly-detection method that can be used to detect congestion in a software-defined network. The method incorporates spectral graph theory and phantom node techniques. The experimental implementation of spectral graph theory used eigenvalue-eigenvector solutions to characterize a mathematical model of the network's topology. In this thesis, we used the phantom node technique to determine congestion in the network by using a virtual node to set the threshold for available link capacity, or the maximum amount of traffic, that can cross the links in the network before the links are considered congested. Results show that when the network is congested, a shift occurs in the eigenvalue and eigenvalue index spectrum. Prior to congestion, the virtual node has the highest nodal influence in the lowest eigenvalue index; however, when a node becomes congested and high traffic in the node crosses the threshold set by the virtual node, the congested node takes the position of the virtual node in the eigenvalue index. The virtual node shifts to having the greatest nodal influence in the next-higher eigenvalue index in the spectrum. Essentially, the results show that anomalies, such as congestion, can be detected using the anomaly-detection method developed in thesis.

14. SUBJECT TERMS cyber, software defined network, phantom node, congestion	**15. NUMBER OF PAGES** 73
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

TRAFFIC CONGESTION ANALYSIS FOR A SOFTWARE-DEFINED NETWORK

Moniqua J. Maxie
Lieutenant Commander, United States Navy
B.S., North Carolina State University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2018

Approved by: Murali Tummala
Thesis Advisor

John McEachen
Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The objective of this thesis is to implement an anomaly-detection method that can be used to detect congestion in a software-defined network. The method incorporates spectral graph theory and phantom node techniques. The experimental implementation of spectral graph theory used eigenvalue-eigenvector solutions to characterize a mathematical model of the network's topology. In this thesis, we used the phantom node technique to determine congestion in the network by using a virtual node to set the threshold for available link capacity, or the maximum amount of traffic, that can cross the links in the network before the links are considered congested. Results show that when the network is congested, a shift occurs in the eigenvalue and eigenvalue index spectrum. Prior to congestion, the virtual node has the highest nodal influence in the lowest eigenvalue index; however, when a node becomes congested and high traffic in the node crosses the threshold set by the virtual node, the congested node takes the position of the virtual node in the eigenvalue index. The virtual node shifts to having the greatest nodal influence in the next-higher eigenvalue index in the spectrum. Essentially, the results show that anomalies, such as congestion, can be detected using the anomaly-detection method developed in thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

2-D	2-Dimensional
ARM	Acorn Reduced Instruction Set Computing (RISC) Machine
API	Application Program Interface
ARP	Address Resolution Protocol
AS	Autonomous System
DDoS	Distributed Denial of Service Attack
DNS	Domain Name Server
DoS	Denial-of-Service
eBGP	External Border Gateway Protocol
GB	Gigabytes
Gbps	Gigabit per Second
IP	Internet Protocol
ISP	Internet Service Provider
MAC	Media Access Control
Mpps	Million Packets per Second
NOOBS	New out of the Box Software
OCP	Open Compute Project
OS	Operating System
RAM	Random Access Memory
SSH	Secure Shell
SDN	Software-defined Network
UDP	User Datagram Protocol
WAN	Wide Area Network

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professors Murali Tummala and John McEachen for their guidance, kindness, and oversight during the thesis process. I would also like to thank LCDR Thomas Parker for his unwavering patience, mentoring, and assistance in helping me understand and work with the testbed, program, and hardware. I have learned a lot during this process from my advisors, and I know that the knowledge I have gained will be beneficial as I move forward in my career.

Finally, I would like to thank my parents, siblings, and friends for their support and words of encouragement. They and God serve as a beacon of light to me every day.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Federal laws and confidentiality agreements prevent federal and military organizations from obtaining detailed information about the Internet Service Provider (ISP) network's backbone to which the organization is connected; therefore, the organizations' network security must not only be adaptable enough to thwart and mitigate internal security breaches but also be able to deter incoming cyber attacks. Software-defined networks (SDN) are scalable, deployable, and controllable [1]. A SDN consists of a network that uses an application to route traffic among the physical network devices. For this reason, and due to organizations being targeted through cyber attacks that leave their computer networks severely degraded, entities such as commercial enterprises, federal organizations, and U.S. internet service providers are adopting SDN technology [2].

For example, Facebook is deploying SDN technology to change the way hardware and software work together in order to reduce cost and increase the scalability and manageability of their network. To initiate this effort, Facebook introduced Open Compute Project (OCP) in 2013 [3]. By 2015, Facebook had begun using a new open source switch, known as the "Wedge" [4], to connect data servers. Traffic is forwarded among these switches using the Facebook open switching system (FBOSS), which is software-based and comprises applications compatible with the Linux operating system (OS).

A simple layered model of the Facebook SDN is shown in Figure 1. The FBOSS provides instructions to ASIC devices for forwarding and management of traffic from and to Facebook's data servers. The FBOSS acting as controller coordinates with each switch within the network [5]; the switches advance traffic. For this function, as well as to manage traffic that a switch is not designed to handle autonomously, the FBOSS uses an agent to configure the internal application-specific integrated circuit (ASIC) located within each switch. The traffic is then routed to the Facebook data servers based on the switches' routing tables located in the ASIC. The routing tables are populated using information on the routes using the network's border gateway protocol [5]. FBOSS was primarily designed to be used with Facebook data centers [5]; therefore, all routing codes are very specific and do not integrate well for use with other data centers.

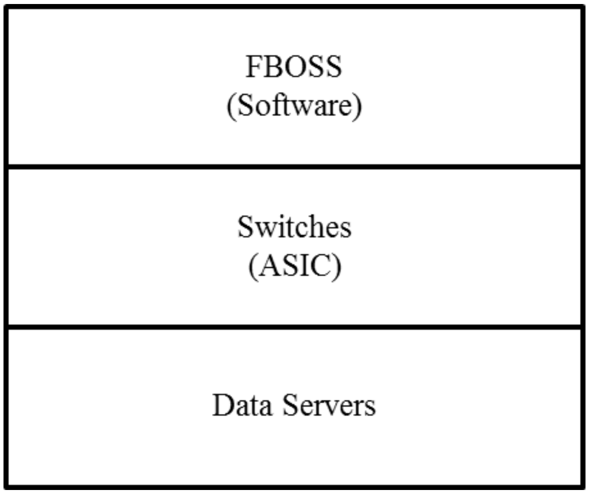


Figure 1. The Layered Model of the Facebook SDN. Adapted from [5].

The Google wide area network (WAN), known as the B_4, is another example of a SDN [6]. Unlike Facebook's FBOSS, the B_4 was designed to be operated among a wide variety of data centers. In addition, the B_4 was built to accommodate high bandwidth and flow control throughout the network. The B_4 consists of various WANs organized into three layers: global, site controller, and switch, as shown in Figure 2 [5].

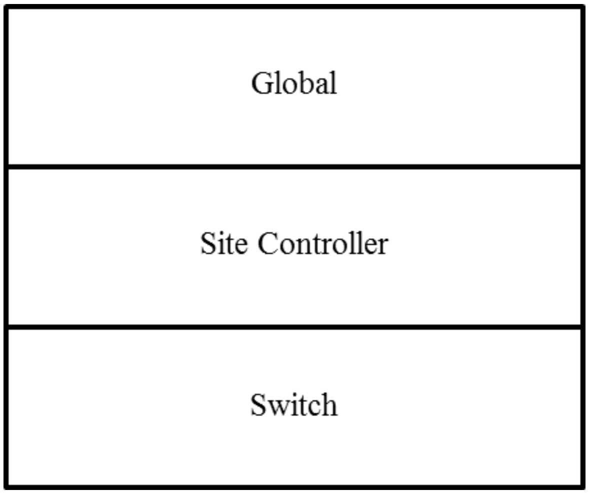


Figure 2. The Main Layers that Make up a B_4 WAN. Adapted from [6].

Inside a B_4 WAN, the global layer obtains topology about other B_4 WANs and uses aggregation to reduce routing computation to just include the WAN edges [6]. The site controller manages routing and records the state of the network [6]. When changes occur in the network state, the switch forwarding tables in the switch layer are updated, and the switches forward the packets based on information in the tables [6].

The functionality of Google's B_4 SDN mimics that of a traditional network in which a backbone is used to transport traffic between different autonomous systems. The routes for each autonomous system (AS) are aggregated at the external routers using external border gateway protocol (eBGP). In many cases, routes in a traditional network are determined by processes in the router that use a significant amount of memory and lead to increased packet processing delays. The Google WAN B_4, on the other hand, uses a software application to regulate and manage traffic. This reduces the computational, energy, and memory requirements of the routing devices. In addition, due to its centralized network state updates, B_4 enables centralized flow control that results in greater link utilization and reduction of buffering capacity required for large networks [6]. In traditional networks, traffic can be controlled via flow control and congestion windows determined by algorithms in the router, but for a SDN, traffic flow is managed and modified by the controllers. This increased controllability and efficiency of an SDN compared to traditional networks are causing more corporations to invest in software-based networks to manage traffic [7].

The manageability of the scalable network has also led Department of Defense (DoD) agencies to adopt a SDN technology. Since a SDN is expected to be critical cyber infrastructure, the DoD must not only identify the vulnerabilities and risks of operating on these SDNs but also determine ways of testing for and detecting potential threats [8]. Furthermore, DoD entities must also understand the impacts of executed threats on the network. Consequently, in this study, we focus on the testing of an anomaly detection mechanism designed to detect for network congestion in a SDN. The mechanism can be used with other cyber-threat mitigating methods to fortify the DoD's networks.

A. THESIS OBJECTIVE

The objective of this thesis is to implement and analyze a new anomaly detection method in a SDN. This method involves the use of spectral graph theory and the phantom node technique to detect congestion in a SDN. Spectral graph theory provides a mathematical model that describes the topology of the network. The phantom node is a virtual node that is added to the model but not to the physical network. This node is used to set the threshold for congestion, or the maximum limit on the amount of traffic transiting the nodes in the network. Although congestion can be attributed to various events for this study, it can also be caused by a denial-of-service (DoS) attack.

This work involves the employment of an SDN testbed. The testbed includes a controller that directs packets among switches located in the network. The controller also collects information about the packets such as the data rate transiting the switches. The recorded information is then run in the spectral graph theory-based computer codes to obtain a computational model of the network. Within this model, a phantom node is added to the network and is used to assist in the detection of congestion.

B. RELATED WORK

The controllability of the network provided by a SDN also aids in the enhancement of network security. A SDN can be used in conjunction with more traditional devices to detect abnormal behavior in the network that may be attributed to a specific cyber attack.

Ashraf and Latif [9] analyzed an anomaly detection technique in which a network's protocols are pre-determined or learned during operation of the network; they used a machine-learning technique to abate the effects of distributed DoS (DDoS) attacks against a SDN. This technique is employed by acquiring training data for a specific event and using the data to generate a model. Attributes based on this model are identified and pre-defined in order to label and categorize the training data. An attribute's category is characterized by nomenclature, such as valid or anomalous or, more specifically, a DDoS or probe; however, human interaction is necessary for determining the labels. Furthermore, acquiring a diverse catalog of training data is extremely difficult [3]. The efficiency of this method depends on the currency of the available labels and the training data in the system. While the implementation of graph theory methods and the phantom node technique on a SDN in this thesis allows for the establishment of baseline states of the network, it also enables anomaly detection in the traffic pattern of a network with minimum human interaction once the network and protocols are established.

Giotis et al. [10] examined the performance of a SDN as an entity that aids in detecting abnormal behavior in more traditional networks. An algorithm is employed to determine the internet protocol (IP) addresses that have abnormal traffic behavior. The algorithm calculates the number of packets routed between a specific IP and all other IPs at any given time. The computed value is compared to the average number of packets routed between these same IP addresses, and the resulting information is used to determine if an anomaly exists within the network [10]. In this thesis, we use the packet rate in and out of the switches to aid in determining if abnormal behavior in the network is present.

Johnson [11] introduced a new anomaly detection method, which provides the foundation for this thesis. Spectral graph theory and the phantom node technique are implemented in simulations in a virtual SDN environment in order to obtain a computational view of the state of the network and, more specifically, detect congestion in the network. The results in [11] demonstrate that the behavior of traffic-transiting devices in a SDN can be characterized using eigenvalues, which can be manipulated into more illustrative objects such as plots and graphs. As the nodes become congested, these Eigenvalues change, thereby shifting the eigenvalue spectrum of the network. In this thesis, spectral graph theory methods and phantom node technique are applied to a SDN testbed to help determine the real-time network state before and after the network experiences congestion. Results, including changes in the network eigen spectra, are expected to be similar to those seen during the MATLAB simulations presented in [11].

C. ORGANIZATION

Previously established anomaly detection methods for complete networks are summarized in Chapter II. The framework for the implementation of the spectral graph theory and the phantom node technique to detect congestion in a SDN testbed are presented in Chapter III. The results and the discussion in Chapter IV depict and explain the eigen spectrum of the network to indicate whether congestion is present. Highlights of the work reported in this thesis and recommendations for future work are presented in Chapter V. Finally, the MATLAB code used to implement a model of the network and provide the eigen spectrum is documented in the Appendix.

II. BACKGROUND

Network security has become important as communities of hackers, cyber spies, and cyber terrorists expand. These communities generate and release computer viruses, DoS mechanisms, and other computer network degradation entities to conduct destructive operations against electronic machines and networks [12]. These events can result in physical and financial damages.

The cyber community is looking for ways to mitigate these threats. To mitigate the effects of these organic or inorganic threats to computer networks, the potential complications that may result from their execution must be known. Events, such as congestion of traffic, unexplained failure of network devices, or compromised administrator accounts may be anomalies that arise in the affected network [10]. Once the possible outcomes are considered, detection methods must be incorporated into the network to distinguish between the normal and abnormal function of such an entity.

One of the anomalies that occurs in a network is congestion, which may be due to a large burst of packets from a source, a downed link in the network, or a network attack. A well-documented cyber attack technique, namely, DoS and, more specifically, DDoS is known to cause a buildup of traffic in the network [9]. A DDoS attack is described in this chapter. Additionally, methods of detecting congestion resulting from these attacks are considered.

A method tested in this thesis that involves the employment of a SDN, as well as spectral graph theory and phantom node techniques, has the ability to enhance a network's security through monitoring the network and detecting congestion.

A. CONGESTION

Congestion can occur on a link when a router reaches its buffering capacity. Congestion could be due to the amount of traffic surpassing the link data rate. This may happen when data that is transiting a faster network link is routed over to a slower link [13]. Furthermore, a buildup of traffic may be present at a router when the amount of inbound traffic to a router exceeds the amount of outbound traffic. This may occur when the traffic load is not balanced in the network. More traffic is diverted through one router in the network than through other routers, or a router may experience a burst of high volume traffic in a short period of time. As a result, the incoming traffic at the router surpasses the router's buffer capacity, and, essentially, the rate of incoming packets outpaces the rate of outgoing packets of the router.

This event leads to packets being dropped or lost in the network [13]. As the volume of packets transiting the network increases, the throughput rises as well. As the network traffic begins to reach the network's capacity, the rate at which the throughput increases begins to slow down and congestion occurs. To mitigate this event, the router utilizes traffic management schemes to prioritize and drop packets of lesser importance [14]. As a result, the throughput of packets with higher priority increases at the expense of other traffic.

Given the above information, a network's vulnerability to congestion can be exploited by attackers as a way to disrupt the network and prevent the flow of data. Attacks can be used to deliberately cause congestion at a router or in a specific area of a network. One type of attack that can generate a buildup of traffic and create congestion is a DoS attack, which is discussed further in Section B.

B. DENIAL OF SERVICE

A DoS attack ultimately prevents authorized users from using sections of a network. This type of attack can be implemented when a host is compromised and maliciously used to attack the network, such as through the transmission of a large quantity of packets to hosts in the targeted network [15]. This significant transmission load results in congestion [15]. The congestion reduces the network's capability to continue to effectively transmit information. A specific DoS attack, termed a DDoS attack, employs multiple hosts to deliver a coordinated attack against the network [15]. One way to conduct a DDoS attack, which is demonstrated in this research, is by attacking the network using a botnet.

A botnet is a network of bots or hosts infected with malicious code that originates from a botmaster. The code is known as bot binary code and is transmitted to a target host on the botnet. Once the code has been injected, the botmaster is informed via an action known as call-home or rallying. The proliferation of the binary code to other bots within the net is accomplished through two methods of propagation, active and passive [16]. The propagation simulated in this thesis is active.

In the active mode, a bot conducts a scan on other hosts within the net to determine a door into the system and acquires high-level privileges to the host, enabling the injection of the malicious code. Another mechanism of infecting other hosts of the target network is by using a worm; the worm replicates and spreads without human interaction and provides the backdoor necessary to implant malicious code [16]. Once the bot binary code is installed on a host, the infected host is now a bot.

The ultimate objective of creating a botnet is to enable the botmaster to increase the arsenal used to attack and control systems within a network. Bot binary code enables the botmaster to gain control of hosts to gather information, deny service, and conduct other injurious activity; therefore, the employment of botnets is advantageous to botmasters [16]. The frequency of DoS and DDoS attacks is growing, and as a result, researchers are looking for efficient ways to manage and monitor physical networks.

C. SOFTWARE-DEFINED NETWORK

A SDN is programmable and routes traffic in a network by using an OpenFlow interface that enables transmission of information between a SDN controller and associated network devices. OpenFlow incorporation into a SDN is further explained below.

A SDN consists of three layers (top down): 3) application layer, 2) control layer, and 1) infrastructure layer, as outlined in Figure 3 [11].

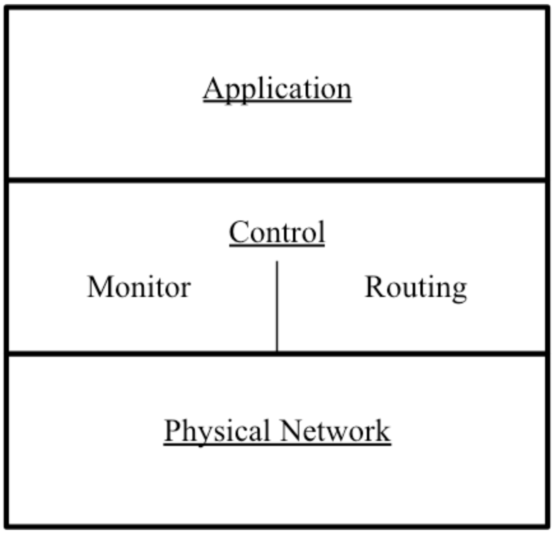


Figure 3. Three-Layer SDN Architecture. Adapted from [11].

A SDN controller houses the routing instructions and protocols. The controller revises the routing tables in the routers using the OpenFlow interface. Utilizing a secure channel, a SDN controller communicates with each switch to update flow entries in the switches. The switches rely on the flow tables to correlate flow rules for inbound packets in order to ensure that each packet is appropriately forwarded to its next destination [11]. In brief, each packet is forwarded based on the rules listed in the switch's flow table.

A SDN controller also includes a monitoring function. Here, data such as the types and number of packets transiting the network, the status of the switches in the network, and the network configuration are provided to the controller by the switches [11]. This data is then compiled in a read-state message that provides a statistical view of the network.

The statistical data within the read-state message can be utilized to determine the connectivity within the network at each switch. The information can also assist in determining if traffic congestion exists. When a packet reaches a switch, it is queued, processed, and then transmitted. If the number of inbound and queued packets is greater than the number being processed and transmitted, then congestion occurs [10]. This situation can be deliberately generated using the previously discussed DDoS attack. In order to mitigate DDoS attacks and manage their effects, congestion must first be detected in the network using network security utilities to prevent network performance degradation.

D. SDN FOCUSED CONGESTION DETECTION METHOD

One way to implement network security tools and manage DDoS attacks is through the use of the OpenFlow protocol in an SDN. To determine the network state and detect anomalies in a SDN, Sahri and Okamura [17] introduced the concept of OpenSec, in which security protocols can be executed within the network. OpenSec utilizes the OpenFlow scheme as a frame of reference, adopting the same method of operation in which the protocols run above the network devices. These policies produce information that yields traffic flow statistics, applicable security attributes that apply, and the selection of reactive pre-planned responses for constituents determined to be malicious [17]. Traffic flow statistics can be used to generate models to obtain detailed information about the network.

Once the topography of a network is known, traffic flow statistics are recorded to obtain additional information regarding the state of a network. The link capacity and maximum data rate between nodes in a network enable the operator to analyze the operation of the network for the purpose of distinguishing between normal and abnormal operating behavior of the network [11].

For this reason, in [11] it was suggested to utilize spectral graph theory to obtain a model and determine the state of a SDN. Graph theory enables the characterization of a network by considering the adjacency, degree, and Laplacian matrices to model the network; therefore, the work in [11] involves utilization of graph theory and the phantom node or switch as essential components in examining the health and the level of connectedness among nodes in a network to determine the existence of congestion. In the following subsections, we describe the adjacency matrix, degree matrix, and Laplacian matrix and explain the concept of the phantom node.

1. Adjacency Matrix

The adjacency matrix A represents a connectivity model of the physical network [11]. Each element W_{ij} in the matrix represents a link weight. A network consisting of n nodes or switches can be represented as

$$A = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1n} \\ W_{21} & W_{22} & \cdots & W_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nn} \end{pmatrix}. \tag{1}$$

The link weights are initially designated with a 1 or 0 to show if a link exists between nodes in the network, providing a matrix model of the network's topology. The 1s are subsequently replaced with pre-defined link weights or link capacities $(0 \le W_{ij} \le 1)$ between the nodes to simulate the available channel capacity between the nodes in an actual network.

2. Degree Matrix

The degree matrix is an integer diagonal matrix and is represented as

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} \tag{2}$$

where a diagonal entry d_n indicates the number of links connected to node n.

3. Laplacian Matrix

The Laplacian matrix is generated as the difference between the diagonal matrix and the adjacency matrix:

$$L = D - A. \tag{3}$$

The eigenvalues λ and eigenvectors x of the Laplacian matrix are calculated using

$$(L - I\,l)x = 0. \tag{4}$$

The (λ, x) pairs are utilized to characterize behavior of each node within the network. Nodes, as determined by the corresponding link weights, influence the eigenvalues (nodal influence) [11]. An eigenvector affects an eigenvalue when it is multiplied by an eigenvalue. Nodes with greater influence on the larger eigenvalues are determined to be more connected to the network than those with larger influence on smaller eigenvalues [11]. The number of eigenvalues is equal to the number of nodes in the network and is represented by an eigenvalue index, which is referred to as the eigenvalue index spectrum of the network [11]. As the link capacity between the nodes changes, matrices A, D, and L change as well [11], causing the network's eigenvalue index spectrum to change to update the connectivity status of the network.

4. Phantom Node

Although traffic flow can be monitored within the topographical model generated by Equations (1), (2), (3), and (4), the determination of the occurrence of congestion in the network remains to be answered. To determine that congestion exists, an anomaly in the network state must first be detected. Consequently, to determine that an anomaly exists within a SDN, the introduction of the phantom node was proposed [11]. A phantom node, or reference node, is added to the computational network defined in adjacent matrix A to monitor the available link capacity to determine if an anomaly exists. The modified adjacency matrix with the addition of the phantom node is given by

$$A = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1n} & W_{1p} \\ W_{21} & W_{22} & \cdots & W_{2n} & W_{2p} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nn} & W_{np} \end{pmatrix}, \tag{5}$$

where W_{np} is the link weight due to the attachment of the phantom node to node n [11].

The phantom node is attached to the most central node, or the node with the highest degree [11]. In addition, the phantom node’s link weight is set at a value that causes the phantom node to have the greatest nodal influence on one of the smaller eigenvalue indices [11]. Given that link weight has an impact on the results from Equations (3) and (4), the phantom node can be used to set the threshold for the minimum amount of link capacity that can remain available without congestion [11].

When a specific node is flooded with a level of traffic that exceeds the threshold, the node eventually becomes saturated, and its connectivity to the network is significantly reduced. Additionally, the congested node’s nodal influence shifts within the network to a lower position in the eigenvalue index spectrum. The congested node now exercises a greater nodal influence in the eigenvalue index where the phantom node’s eigenvalue dominates; thus, the phantom node gains nodal influence in the next highest eigenvalue index [11]. Essentially, the congested node’s nodal influence in the eigenvalue index or eigenvector spectrum shifts. With the occurrence of congestion, the change in traffic flow at the node of concern results in a higher nodal influence due to the amount of traffic passing the volume set by the threshold.

The data noted in [11] presents the simulated results obtained when using the spectral graph and phantom node techniques; however, the implementation of these mechanisms within a non-ideal computational environment and the feasibility in which a timely computation to monitor network traffic and detect congestion on a real network remains unknown. The work in [11] reflects the spectral graph technique implemented in a perfect environment where link capacities between the nodes are exactly the same. In addition, delays such as queuing, processing, and transmission delays of packets do not exist. For this reason, the real-time implementation of the work in [11] on a SDN testbed is the object of thesis. The simulations in [11] involve MATLAB code to simulate congestion in a network by simply adjusting the link weights between the nodes, not by actually congesting the switches with traffic.

E. IMPLEMENTATION OF SDN CONGESTION DETECTION

In any network, delays in packet propagation, processing time, and queuing must be considered as well as the time slot required to run the traffic monitoring and congestion detection code. Other factors that affect congestion and traffic flow are duration of the congestion and the constraints of the physical devices employed in the network. The complication of real-time monitoring and detection is that anomalies occurring in traffic are of short duration [18]. Because the operational design of a SDN may limit the accuracy of the traffic flow statistics, equipment limitations must be considered [19]. If the entity to be measured is delay, then delay within the switches, the maximum link rate of the measuring unit, and the delays in the communication channel between the controller and the switches must be taken into account. In addition, the controller's data-handling capacity also affects the scalability of a SDN and, consequently, the detection mechanism within a network [19]. As the network grows beyond a specific size, more controllers may need to be added or the anomaly detection scheme modified to detect anomalies.

In summary, in this thesis we outlined and discussed the results obtained during the real-time execution of the proposed detection scheme on a physical SDN testbed. We compare between the original findings documented in [11] and results from the experiments during the operation of a network under normal and adverse environments and with and without a phantom node. Theoretically, results should be consistent with the simulated results in that appropriate shifts must be seen in the network's eigenvalue and eigenvalue index spectrum with the presence of congestion. In the next chapter, we lay out the proposed detection scheme for implementation of the spectral graph and phantom node methods on a SDN testbed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROPOSED SCHEME

The proposed scheme of utilizing a SDN and detecting for congestion in the network is presented in this chapter. Network traffic is collected and used to determine the link weights between the nodes. The link weights are then used to calculate the eigenvalues and eigenvectors associated with the network, which in turn can be used to determine the topological view of the network and provide the network's eigen index spectrum.

A. THE PROPOSED CONGESTION DETECTION SCHEME

The SDN testbed consists of three basic components: the controller, the switch, and the host. There exist a total of n switches, m hosts, and a controller in the network under consideration. This is displayed in Figure 4.

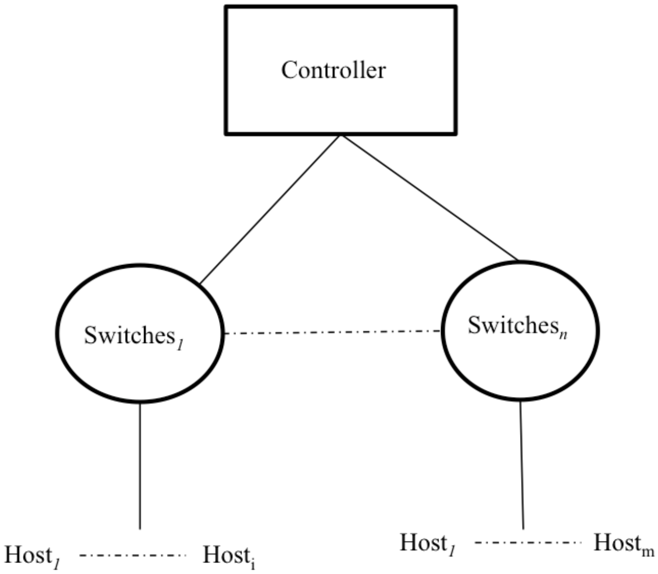


Figure 4. The SDN

The controller enables the user to configure the network devices and, thus, have full control of the network. The hosts send packets to the switch to which they are attached. Next, messages regarding the packets are sent from the switch to the controller.

The controller responds with flow rules that enable the switches to direct the packets to their respective destinations. The general operation of the SDN is illustrated in Figure 5. Fundamentally, the controller and switches work cooperatively to transport packets between the devices and provide statistical information regarding the switches and links. The proposed scheme for detection is detailed below.

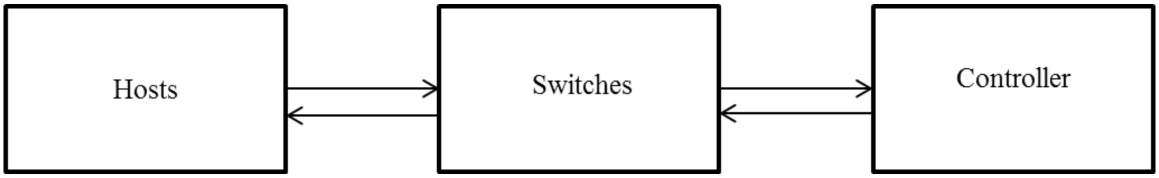


Figure 5. The SDN Operational Flow Chart

Every time the SDN is initialized, the controller must learn the location of all hosts in the network through the exchange of routine setup messages. These messages are the real-time normal traffic transiting the network. Once the network is set up, the user traffic flows through the network.

1. Traffic Data

Traffic statistics at each switch, such as the number of incoming and outgoing packets, are collected by the controller. The controller is configured to continuously estimate the link capacity between the switches. The controller also provides the up or down status of the switches. This information is logged in various files designated by the network manager. The SDN congestion detection scheme is outlined in Figure 6, laying out the process for determining the state of the network. The scheme, consisting of four steps, yields an output that is used to determine the state of the connectivity within the network.

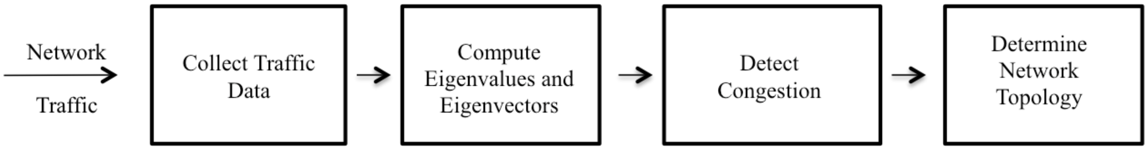

Network
Traffic
Collect Traffic Data
Compute Eigenvalues and Eigenvectors
Detect Congestion
Determine Network Topology

Figure 6. Proposed Congestion Detection Scheme

Given data rate information from the controller, the link weights are calculated and used to update the adjacency matrix. The matrix elements are initialized to a one or zero to represent the links between the switches that are connected (see Equation (1)). Each element W_{ij} in A represents the link weight or link capacity between the i^{th} and j^{th} switch. When a phantom node is added to the computational model of the network, A is updated to reflect Equation (5) in which W_{np} is the link weight due to the attachment of the phantom node to the model.

2. Eigen Characteristics

Once the link weight information is obtained, the scheme computes the eigenvectors and eigenvalues using Equation (4). The resulting eigen characteristics are used to obtain the SDN topology and the eigen index spectrum.

3. Network Topology and Eigen Plots

A comprehensive topographical model of the network, which is based on the eigen information, is formulated [11]. In addition, the eigenvalue-eigenvector solution for each switch at different instances of time are plotted and compared. Furthermore, the eigenvalues are indexed for each switch. This enables the nodal influence for each switch to be determined [11]. The total number of indices is n , the number of switches. Since eigenvalues mathematically reflect the amount of traffic transiting each switch and the connectivity and health within the network, the resulting model, plots, and graphs are interpreted to determine the status of the SDN [15].

4. Congestion Detection

Network data obtained by the controller when the SDN is not experiencing congestion is analyzed first to establish a baseline. The topological and graphical representations, which are obtained when the network is operating under normal conditions with no simulated attack, provide a baseline. The baseline results are compared to the results when anomalies are present.

B. COMPUTATION METHOD

The data rate across each switch is used to compute the network state and the level of connectivity between nodes. In this thesis, we employ the spectral graph and the phantom nodes techniques to obtain a model and the state of the network in real-time [11]. A phantom node is mathematically attached to one of the most central nodes, which is usually the node with the highest degree in the network. The phantom node in the computational model enables the determination of the network state, as well as the condition of the links and switch. The phantom node sets the threshold for congestion, and the amount of traffic transiting the nodes over time is used as the link weights. The link weights W_{ij} are calculated based on the link utilization U_{ij} [20],

$$U_{ij} = \frac{O_{ij}}{R_{ij}} \tag{6}$$

where O_{ij} is the number of bits per second transiting the links between the nodes per second and R_{ij} is the maximum datarate of the link measured in bits per second (bps); however, what is actually recorded in the data file is the number of bits going in and out of the nodes at each instance of time at each node, not the number of bits per second transiting the links as reflected by O_{ij}. Consequently, since the information that is needed from O_{ij} is not directly available from the measured data, what is provided in the measured data must be transformed in order to determine the fraction of the link capacity actually used for packets to transit the links between nodes. Equations (7), (8), and (9) contain the steps used to calculate U_{ij}.

The rate of transmission for each link between the nodes $R_{T_{ij}}$ is first computed from

$$R_{T_{ij}} = \frac{T_{t_2} - T_{t_1}}{E} \tag{7}$$

and represents the calculated transmission rate between two nodes: T_{t_1} is the number of bits through a node's port at the previous time t_1 , T_{t_2} is the number of bits transmitting at the present time t_2 , and E is the difference in time between t_1 and t_2 . Next, the rate $R_{X_{ij}}$ at which packets are received is calculated from

$$R_{X_{ij}} = \frac{X_{t_2} - X_{t_1}}{E}, \tag{8}$$

where X_{t_i} is the number of bits received through a node's port at previous time t_1 and X_{t_2} the number of bits received at the present time t_2 .

Finally, the link weight W_{ij} between the nodes is determined using

$$W_{ij} = \left(1 - \frac{R_{T_{ij}}}{R_{\max}}\right)\left(1 - \frac{R_{X_{ij}}}{R_{ij}}\right) \tag{9}$$

where $R_{\max}$ is the maximum link capacity. In Equation (9), the fraction of link capacity used to transmit and receive data, i.e., link utilization U_{ij} from Equation (6), is calculated and then subtracted from 1.0 to determine the respective link capacity available for transmission and reception of packets between nodes. An adjacency matrix as given in Equation (1) is then formed using these link weights W_{ij}. From the adjacency matrix A, a degree matrix D is produced. The degree matrix displays the number of links attached to each node. A Laplacian matrix L is then calculated using Equation (3) [21], [22].

Finally, eigenvalue and eigenvector solutions of matrix L are computed. The elements l_{ij} of the Laplacian matrix L are given as

$$l_{ij} = \begin{cases} -W_{ij} \; if \; a_{ij} \in K \\ \sum_{j=1}^{n} W_{ij} \; if \; i = j \\ 0 \; otherwise \end{cases}, \tag{10}$$

where K is the set of links between two nodes i and j and a_{ij} represents an element in A [21], [22]. This information is then used to plot the network graph G. Graph G is represented in two ways: 1) as $G(n, L, W)$, where a mathematical topology is generated without a phantom node; and 2) as $G_p(n, K, W, n_p, K_p, W_p)$, where the phantom node is included and n_p is the phantom node, K_p is the link between the phantom node and the node to which it is attached, and W_p is the link weight associated to that link [22].

Although this thesis research is based on work provided in [11], there are significant differences between the two. The spectral graph and phantom node simulations from [11] are conducted on a virtual SDN setup using a MATLAB program. Randomness that occurs in the network is mathematically accounted for in the virtual environment. The program allows the link weights to be manually altered so that the network statistics mirror that of a network experiencing congestion; however, in the testbed, randomness is caused by real traffic transiting the SDN.

As the SDN is initialized, the controller, hosts, and nodes must learn about each other. This is done by using protocols, such as address resolution and domain name server (DNS) protocols. Relearning takes time, as there exists no historical or cached information in any of these devices. In addition, the learned routes between the devices upon each SDN initiation can be different. As a result, each startup of the network may result in different data rates and, therefore, link weights in the adjacency matrix. Consequently, the node where congestion is detected may not be the same after each network startup.

The experiments conducted on the SDN testbed are based on the foundation provided in [11] and take into consideration the constraints of the equipment used. The experiment design, computation, and results are outlined in the next chapter.

IV. EXPERIMENT AND RESULTS

The implementation of the proposed scheme and the specific computation method are outlined in this chapter. The testbed and the associated devices are described. The experiment is run for various lengths of time for each shift in location of the server and of the phantom node in the adjacency matrix. Plots of the results are presented to reflect the network state and the impact of the DoS attack in relation to a node's nodal influence in the eigen spectrum. In essence, the results depict a reaction that occurs in the eigenvalue spectrum not only when the phantom node is added to the network but also when congestion occurs.

A. THE SDN TESTBED

The experiments are executed on a SDN testbed to obtain data about the network. In this experiment, a Ryu application is created and run as the controller. The hosts are Raspberry Pis. The switches, also known as the nodes, are HP 2920 switches. A traffic monitor application is built within the Ryu application. The application is employed to monitor network traffic at the controller and output flow data of incoming and outgoing packets. Meanwhile, each switch within the SDN sends messages to the Ryu application. The content within the messages pertains to traffic transiting that switch. The switch acts on the packets based on the responses received from the Ryu application. Finally, a botmaster established on a Linux computer connected to a SDN employs the Raspberry Pis to establish the botnet and conduct a DoS attack. A simple diagram of the testbed is shown in Figure 7. The roles of each of the devices in the diagram are discussed in more detail later on in this chapter.

The data about the network is extracted by the Ryu application and compiled into files using a Python computer program on a Linux system connected to the SDN via a Wi-Fi router for further analysis. The current and past packet data transmitted at each node is computationally compared and used to determine the link weights between the nodes. The link weights are used in the adjacency matrix in a computer program code that generates a graphical representation of the network topology updated at each time step.

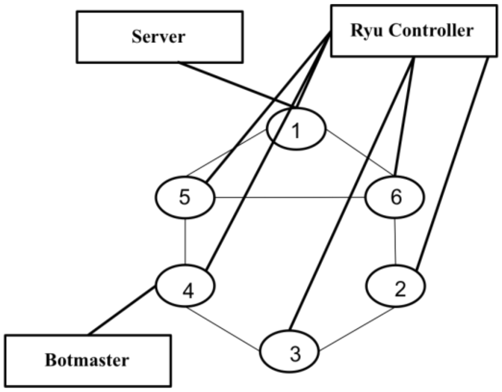


Figure 7. Testbed SDN Diagram

The code also produces a bar graph depiction, which is updated simultaneously, that reflects the connectivity of the all nodes in the network using eigenvectors and eigenvalues. The experimental design includes a testbed containing 50 Raspberry Pis that are physically connected to six SDN switches or nodes. The OpenFlow switches are also interfaced with a Ryu controller, which is the Ryu SDN application. The controller houses the Ryu-Manager responsible for calling application program interface (API) functions that directly interface with the switches.

The controller, switches, and hosts makeup the SDN. The configuration of the devices is depicted in Figure 8.

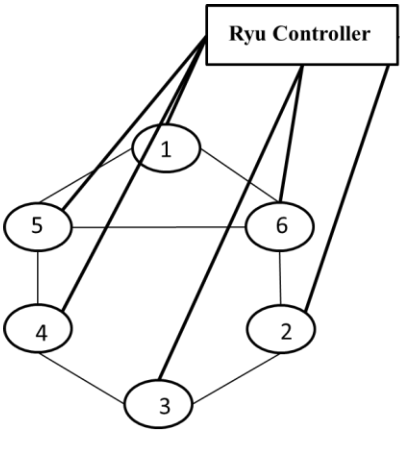


Figure 8. Diagram of the Network Connectivity

The switches are nodes 1 through 6 and interface with the Ryu controller. The hosts are not pictured, but each switch has a number of hosts attached to it. In the next section, we describe each of the component's operational connection with one another.

1. Controller

In order to setup a SDN on the testbed, the Ryu SDN application is scripted using Python and modified as necessary to control the network. The Ryu shell, or controller, initially contains the Ryu-manager. The automated application used to set up and maintain the communication between devices and the Ryu application manager reside within the Ryu-manager [23], [24]. The application manager allows the uploading of applications designed for monitoring and routing. The monitoring application is designed to continuously monitor traffic, which in this case consists of the user datagram packets (UDP) permitted to transit the network. The routing application is a switch flow code that aids in the controller's collection of flow and port data at each node. The monitoring and switch flow commands are part of the Ryu manager. In addition, the application uses the OpenFlow protocol to communicate with the switches and to manage and direct traffic within the SDN. The Ryu application also dynamically generates a media access control (MAC) table. The MAC table lists the MACs of associated devices in the network and their corresponding ports [23], [24]. Because the information in the table is dynamic, a new table is created every time the controller is reset. Information passed by the switches to the controller helps to generate this table.

2. Switches

The switches direct packet traffic among the hosts. Moreover, the switches interface with the Ryu controller to receive instruction on what action to take when packets transit the nodes. A switch examines the incoming packets to determine the host from which the packet was sent and the host's associated MAC address and port. Next, the switch checks the packet's destination address and determines if the address belongs to a "learned" host or a host within the switch's table. If so, the switch, with the use of the packet-out function, forwards the packet to the host. If not, the switch employs the same function, floods the network with an address resolution protocol (ARP) message in order to locate the destination host and port. The host of interest responds to the message, and the switch adds the information of the destination host and of the routes taken to the end host to the flow table [23]. The route that results in the fastest response to the ARP message is then added to the flow table. This flow table is associated with the switch from where the ARP message originated.

The testbed switches are HP 2920 Series switches designed to accommodate a SDN with OpenFlow protocol. The switches are rated to have 176 gigabit per second (Gbps) maximum switching capacity and 130.9 million packets per second (Mpps) throughput capability [25]. Each switch interfaces with the hosts as well as the controller. In addition, for this experiment, the link capacity is limited to 10.0 Mbps per switch.

3. Hosts

The host devices are Raspberry Pis and are connected to the OpenFlow switches. Data about the Raspberry Pi packets that transit the switches are sent to the controller from the switches. The Raspberry Pis are Acorn Reduced Instruction Set Computing (RISC) Machine (ARM) band computing devices with 512 MB random access memory (RAM). The Pis are variant Model B+ and are capable of operating with an 8.0 gigabyte (GB) New Out of the Box Software (NOOBS) installer that includes the Raspbian OS [26]. The Pis are connected via Ethernet with a maximum data rate limited to 100 megabits per second (Mbps) for UDP. The Raspberry Pis are used to execute a DoS attack on the network.

A DoS attack is conducted by a botmaster. Firstly, the botmaster runs what is considered a “malicious” traffic generation code and logs, via secure shell (SSH), into the Raspberry Pis. Secondly, the botmaster instructs each Raspberry Pi to generate UDP traffic. Thirdly, a node designated as the server is also logged into, via SSH, using a Linux computing device. The server is directed by a programmer to “listen” for UDP traffic. This latter instruction provides the destination location for the packets. Finally, changes in the network state are logged using the monitor and route applications to determine the network’s connectivity and traffic flow.

B. IMPLEMENTATION OF THE SCHEME

In this section, we explain how a MATLAB code implements spectral graph theory and the phantom node insertion to obtain a model of the state of the network at a given time. Since for the testbed the controller does not output the data rate but rather logs the packet transmit times in and out of each node, the link weight is determined by comparing traffic measured at those times using Equations (7), (8), and (9). Resulting link weights are used to solve Equation (4). Solutions for Equation (4) are then used to determine a topology of the network and the eigenvalue spectrum. The MATLAB code is documented in the Appendix.

The experiment is run with and without the phantom node. For example, for the network shown in Figure 8, without the phantom node, link weights W_{n_1,n_2} are included as

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & W_{1,5} & W_{1,6} \\ 0 & 0 & W_{2,3} & 0 & 0 & W_{2,6} \\ 0 & W_{3,2} & 0 & W_{3,4} & 0 & 0 \\ 0 & 0 & W_{4,3} & 0 & W_{4,5} & 0 \\ W_{5,1} & 0 & 0 & W_{5,4} & 0 & W_{5,6} \\ W_{6,1} & W_{6,2} & 0 & 0 & W_{6,5} & 0 \end{bmatrix} \tag{11}$$

where W_{n_1,n_2} is the link weight calculated for connecting nodes n_1 and n_2. The topology of the network is determined from the eigen solutions generated via the calculations at each instance of time using the spectral graph method. Next, the congestion at the node is detected by employing the phantom node concept.

The phantom node sets the limit for congestion and is mathematically attached to the most central node. For the six-node network in Figure 8, the attachment of the phantom node to the network in Equation (5) is expressed as

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & W_{1,5} & W_{1,6} & W_{1,p} \\ 0 & 0 & W_{2,3} & 0 & 0 & W_{2,6} & W_{2,p} \\ 0 & W_{3,2} & 0 & W_{3,4} & 0 & 0 & W_{3,p} \\ 0 & 0 & W_{4,3} & 0 & W_{4,5} & 0 & W_{4,p} \\ W_{5,1} & 0 & 0 & W_{5,4} & 0 & W_{5,6} & W_{5,p} \\ W_{6,1} & W_{6,2} & 0 & 0 & W_{6,5} & 0 & W_{6,p} \\ W_{p,1} & W_{p,2} & W_{p,3} & W_{p,4} & W_{p,5} & W_{6,p} & W_{p,p} \end{bmatrix}, \quad (12)$$

where $W_{p,n}$ and $W_{n,p}$ represent the link weight values for the nodes attached or not attached to phantom node p.

At commencement of the experiment, the phantom node, node 7, is linked with node 6. Node 6 is the most central node in the network, and the reason for this centrality is further discussed in the next section. The corresponding configuration of the SDN is depicted in Figure 9.

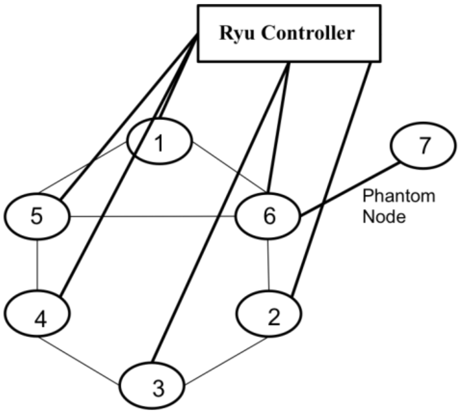


Figure 9. Initial Configuration of the SDN

It is important to remember that the phantom node is not controlled by the controller because the node does not physically exist. The node is only seen at the computational level. The MATLAB code is used to generate a computational model that determines the topology of the network and the nodes’ eigenvalue and eigenvalue index spectrums.

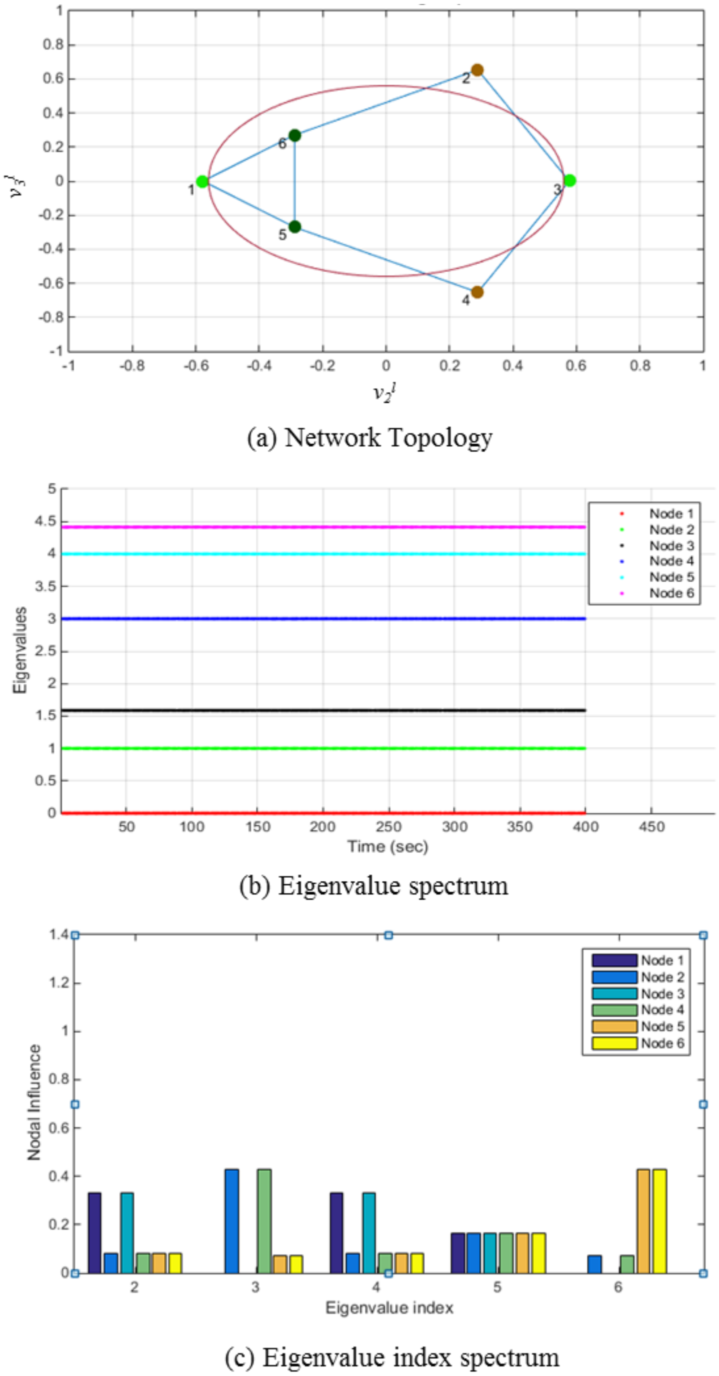
C. RESULTS

The results as discussed in the previous section are divided in two groups: 1) network topology and eigenvalue spectrums for a network not undergoing attacks and 2) network topology and eigenvalue spectrums for a network being attacked. Calculations are also made when the phantom node(s) and the server are attached to different nodes in the SDN to observe the impact on the eigenvalues and eigenvalue indices.

1. Network under Normal Conditions without Congestion

The normal traffic pattern for the SDN is determined by capturing the behavior of traffic when the network is not under attack. The computation of eigen data enables the construction of the SDN topology and the eigenvalue and eigenvector index spectrums. This normal traffic pattern represented by the computed SDN topology and spectra is used as a basis to make comparisons and to detect abnormal behavior in the network when congestion occurs. Observations recorded under normal conditions as the phantom node or servers are shifted to different nodes during each experiment are included in the following sections.

A snapshot of the network behavior without the phantom node is illustrated in Figure 10. Notice that the topology of the network is symmetrical about zero along the *y*-axis. The network's topology is determined using eigenvectors 2 and 3. The eigenvalue spectrum displays the eigenvalues of the nodes at each instance of time. Although symmetry is present in Figure 10(a), node 6 has the largest eigenvalue in the greatest index during operation without congestion in the network, as displayed by Figures 10(b) and 10(c). This makes node 6 the most central and means that the node has the greatest connectivity out of all the nodes in the network. In addition, the nodes with the highest level of connectivity, i.e., nodes 5 and 6, influence eigenvalue index 6 the most, as depicted in Figure 10(c).



(a) Network Topology

(b) Eigenvalue spectrum

(c) Eigenvalue index spectrum

Figure 10. Network Behavior Without Phantom Node

In contrast, the least connected nodes, i.e., nodes 1, 2, 3, and 4, dominate the lower indices in nodal influence. Note that in index 5, all of the nodes have the same amount of influence.

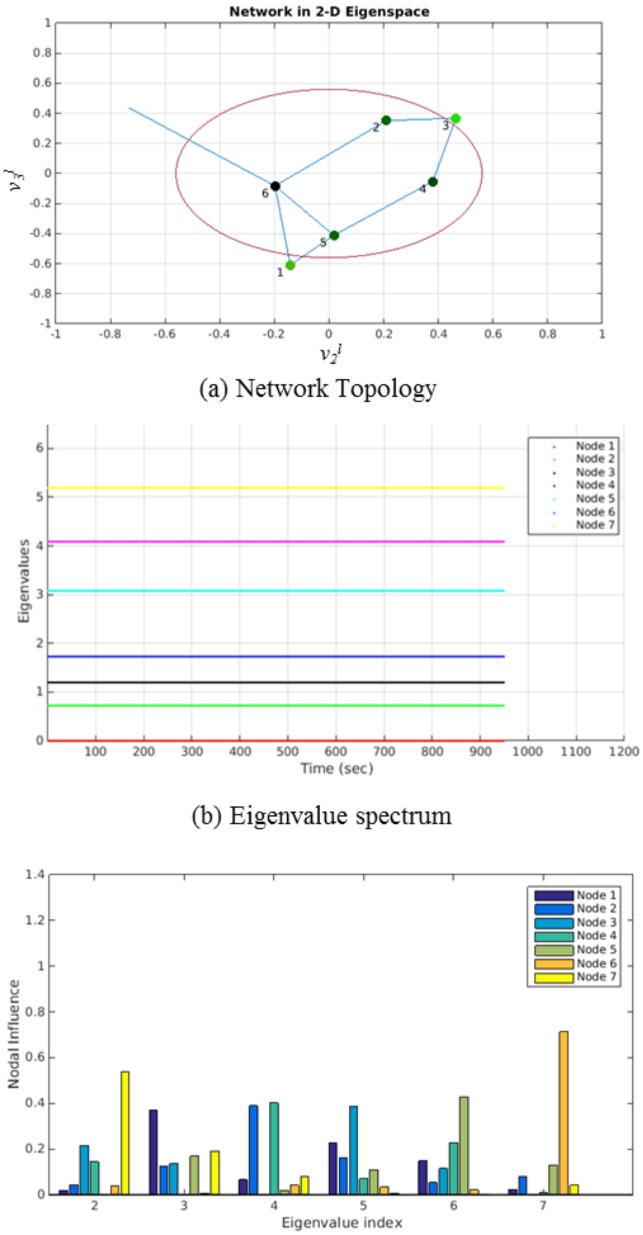
a. Phantom Node Attached to Node 6, Server at Node 1

The SDN exhibits consistent behavior under normal conditions (no attack) and, as previously discussed, node 6 was found to be the most central node based on the eigenvalue index spectrum. The phantom node (denoted node 7 here) is now attached to node 6. The resulting adjacency matrix is given by

$$A=\begin{bmatrix} 0 & 0 & 0 & 0 & W_{1,5} & W_{1,6} & 0 \\ 0 & 0 & W_{2,3} & 0 & 0 & W_{2,6} & 0 \\ 0 & W_{3,2} & 0 & W_{3,4} & 0 & 0 & 0 \\ 0 & 0 & W_{4,3} & 0 & W_{4,5} & 0 & 0 \\ W_{5,1} & 0 & 0 & W_{5,4} & 0 & W_{5,6} & 0 \\ W_{6,1} & W_{6,2} & 0 & 0 & W_{6,5} & 0 & W_{6,7} \\ 0 & 0 & 0 & 0 & 0 & W_{7,6} & 0 \end{bmatrix}, \tag{13}$$

where $W_{6,7}$ and $W_{7,6}$ reflect the communication link between nodes 6 and 7. The network's topology and eigenvalue index spectrum are then computed. The server is placed at node 1 to provide a baseline for the network's behavior under normal conditions with a phantom node attached. The impact of moving the server is seen in Subsection C.2.b.

The network behavior with the phantom node is shown in Figure 11. The topology of the network is seen in Figure 11(a). Each node, with the exception of the phantom node, is represented by a green dot. The location of the green dot reflects a node's position within the two-dimensional (2-D) eigenspace. The link with no dot depicts the point of attachment of the phantom node to the network. Along with the network topology, the eigenvalue spectrum is also generated by the MATLAB algorithm. The spectrum is illustrated in Figure 11(b). In this figure, nodes 5 and 6 are shown as having the highest eigenvalues. Both of these nodes hold the highest degrees as well. Meanwhile, nodes 1, 2, 3, and 4 occupy the lowest eigenvalues on the graph. The eigenvalue index spectrum is displayed in Figure 11(c). The spectrum also depicts node 7, the phantom node that sets the threshold for congestion, as having the greatest nodal influence in index 2, one of the lowest indices. By contrast, node 6, one of the nodes with the highest level of degree, dominates eigenvalue index 7.



(a) Network Topology

(b) Eigenvalue spectrum

(c) Eigenvalue index spectrum

Figure 11. Network Behavior with Phantom Node Attached to Node 6, Server at Node 1

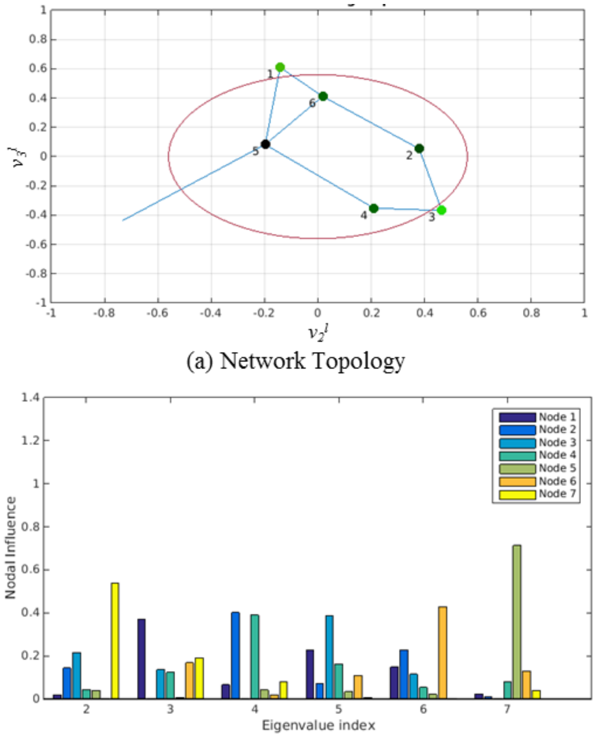
This is one of the most central nodes with the greatest connectivity.

b. Phantom Node Attached to Node 5, Server at Node 1

The phantom node is shifted to next most central node, node 5, to determine if there is a shift in the network's topology and eigenvalue spectrum. In this scenario, similar results to those previously reported are observed. The adjacency matrix is modified to reflect the node's change of position as given by

$$A=\begin{bmatrix} 0 & 0 & 0 & 0 & W_{1,5} & W_{1,6} & 0 \\ 0 & 0 & W_{2,3} & 0 & 0 & W_{2,6} & 0 \\ 0 & W_{3,2} & 0 & W_{3,4} & 0 & 0 & 0 \\ 0 & 0 & W_{4,3} & 0 & W_{4,5} & 0 & 0 \\ W_{5,1} & 0 & 0 & W_{5,4} & 0 & W_{5,6} & W_{5,7} \\ W_{6,1} & Lw_{6,2} & 0 & 0 & W_{6,5} & 0 & 0 \\ 0 & 0 & 0 & 0 & W_{7,5} & 0 & 0 \end{bmatrix}, \tag{14}$$

where the new location is notated by $W_{5,7}$ and $W_{7,5}$.



(a) Network Topology

(b) Eigenvalue index spectrum

Figure 12. Network Behavior with Phantom Node Attached to Node 5

The updated network configuration, according to this shift in the phantom node position, is depicted in Figure 12(a). Note that node 5, not 6, dominates in nodal influence in eigenvalue index 7, as seen in Figure 12(b).

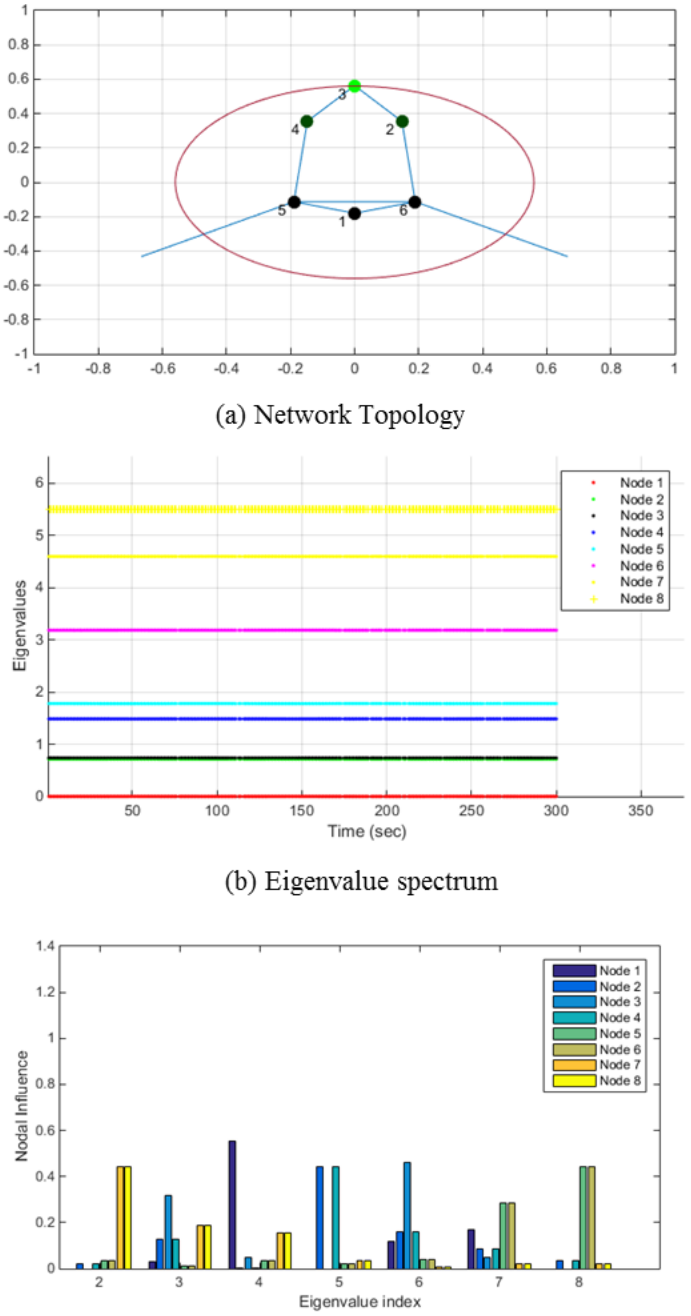
c. Phantom Nodes Attached to Node 5 and Node 6, Server at Node 1

Two phantom nodes were then added to the network to determine if this improved detection time; however, other changes were noted when this occurred. When two phantom nodes are added to the SDN configuration in the adjacency matrix A, the model provides a more symmetrical view of the network. The attachment of the phantom nodes is expressed as

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & W_{1,5} & W_{1,6} & 0 & 0 \\ 0 & 0 & W_{2,3} & 0 & 0 & W_{2,6} & 0 & 0 \\ 0 & W_{3,2} & 0 & W_{3,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & W_{4,3} & 0 & W_{4,5} & 0 & 0 & 0 \\ W_{5,1} & 0 & 0 & W_{5,4} & 0 & W_{5,6} & 0 & W_{5,8} \\ W_{6,1} & W_{6,2} & 0 & 0 & W_{6,5} & 0 & W_{6,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & W_{7,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & W_{8,5} & 0 & 0 & 0 \end{bmatrix} \qquad (15)$$

where $W_{5,8}$, $W_{8,5}$, $W_{6,7}$, and $W_{7,6}$ represent the connection between the nodes. The centrality of the network is displayed when two phantom nodes are attached to the base configuration, as shown in Figure 13.

While the eigenvalue spectrums do not change, the SDN connectivity becomes more symmetrical about zero on the *x*-axis as illustrated in Figure 13(a). Furthermore, as displayed in Figure 13(b), the phantom nodes are nodes 7 and 8. The nodes' eigenvalues result in them having the greatest nodal influence in eigenvalue index 3, shown in Figure 13(c). The most central nodes, nodes 5 and 6, dominate eigenvalue index 7 and 8. In addition, node 3 has the largest eigenvalues in indices 3 and 6, while nodes 2 and 4 have the most influence on 5.



(a) Network Topology

(b) Eigenvalue spectrum

(c) Eigenvalue index spectrum

Figure 13. Network Behavior with Phantom Nodes Attached to Nodes 5 and 6

Overall, during normal conditions, the eigenvalues of each node are consistent; however, the network topology and the eigenvalue index spectrum vary for different configurations. The node connected to the phantom node has the greatest nodal influence in the highest eigenvalue index. Furthermore, by comparing Figures 6 through 9, we can see that the supplementation of this node affects the eigenvalues and eigenvalue index spectrums. The node to which the phantom node is attached not only increases in degree but also in the level of connectivity. The eigenvalue associated with the nodes normal operation also increases; therefore, the phantom node affects the mathematical degree and connectivity of the node to which it is linked. In contrast, the phantom node has the most nodal influence in the lowest eigenvalue index, which occurs because the phantom node sets the threshold for congestion.

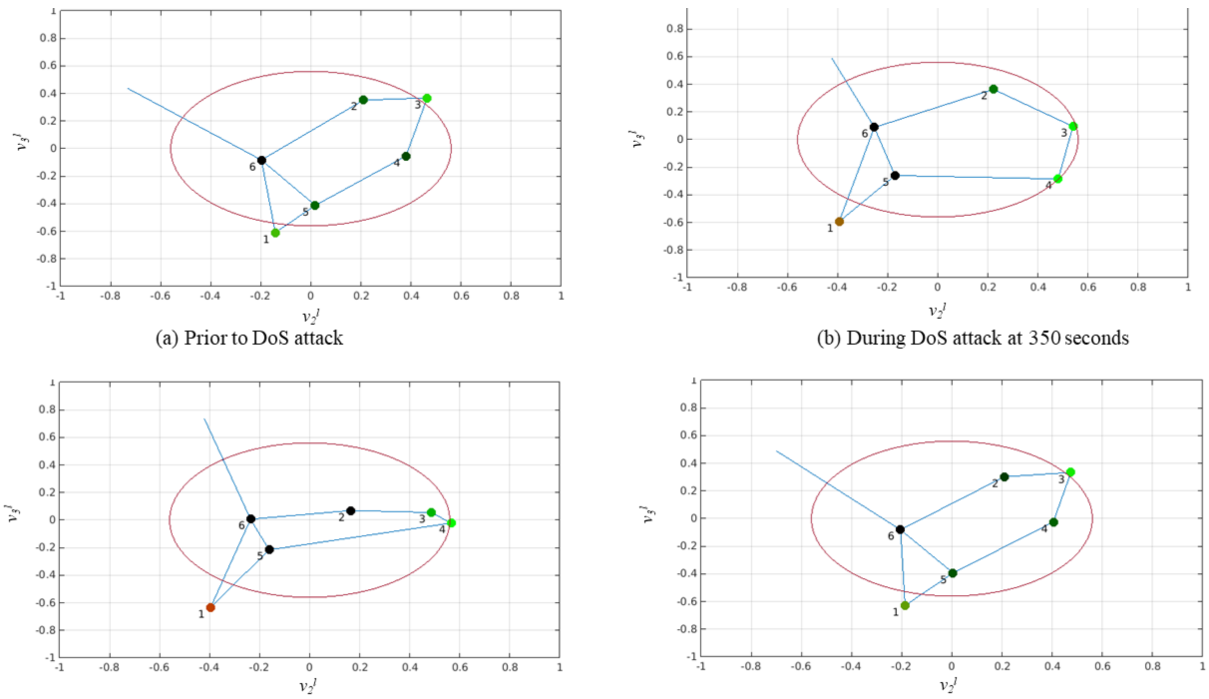
This documented behavior and the characteristics of the network provide a basis for identifying anomalies. In the next subsection, we demonstrate the behavior of the SDN testbed when it operates under a simulated DoS attack and compare the results to those provided in this subsection.

2. Network during Congestion

A DoS attack is conducted for each network configuration to observe the impact of congestion on each node. The attacks are conducted on the same configurations presented in Section 1.

a. Phantom Node Attached to Node 6, Server at Node 1

Nodes become congested as a DoS attack is conducted against the server at node 1. The network’s topology is transformed as nodes are flooded with packets, as shown in Figure 14. Note the shifting of the nodes within the eigenspace.



(a) Prior to DoS attack

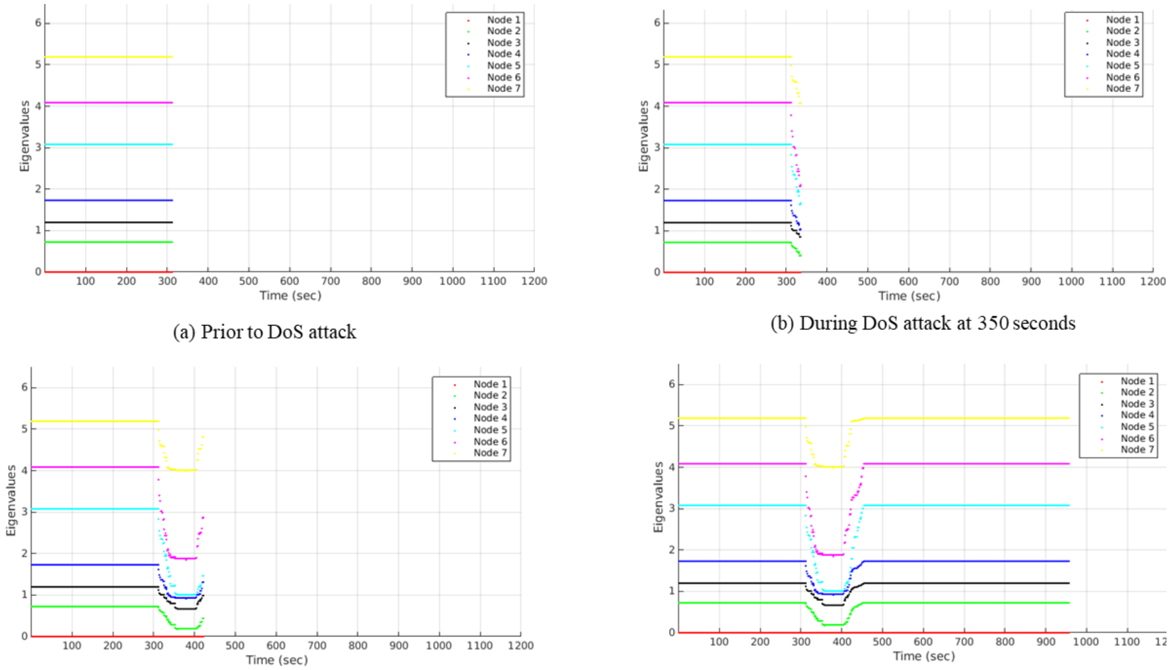
(b) During DoS attack at 350 seconds

(c) During to DoS attack at 450 seconds

(d) Post DoS attack

Figure 14. Congested Network Behavior with Phantom Node

The normal centrality of the network, shown in Figure 14(a), shifts during the DoS attack, as shown in Figures 14(b) and 14(c), then returns back to its normal configuration, shown in Figure 14(d), when the congestion has subsided. This phenomenon is due to the effects of the fall of the eigenvalues as the nodes in the network become more congested, as depicted in Figure 15. The original eigenvalues, shown in Figure 15(a), decrease with the occurrence of increasing congestion, as shown in Figures 15(b) and 15(c). The eigenvalues then return to the previous eignevalues once the congestion in the network subsides, as illustrated in Figure 15(d).



(a) Prior to DoS attack

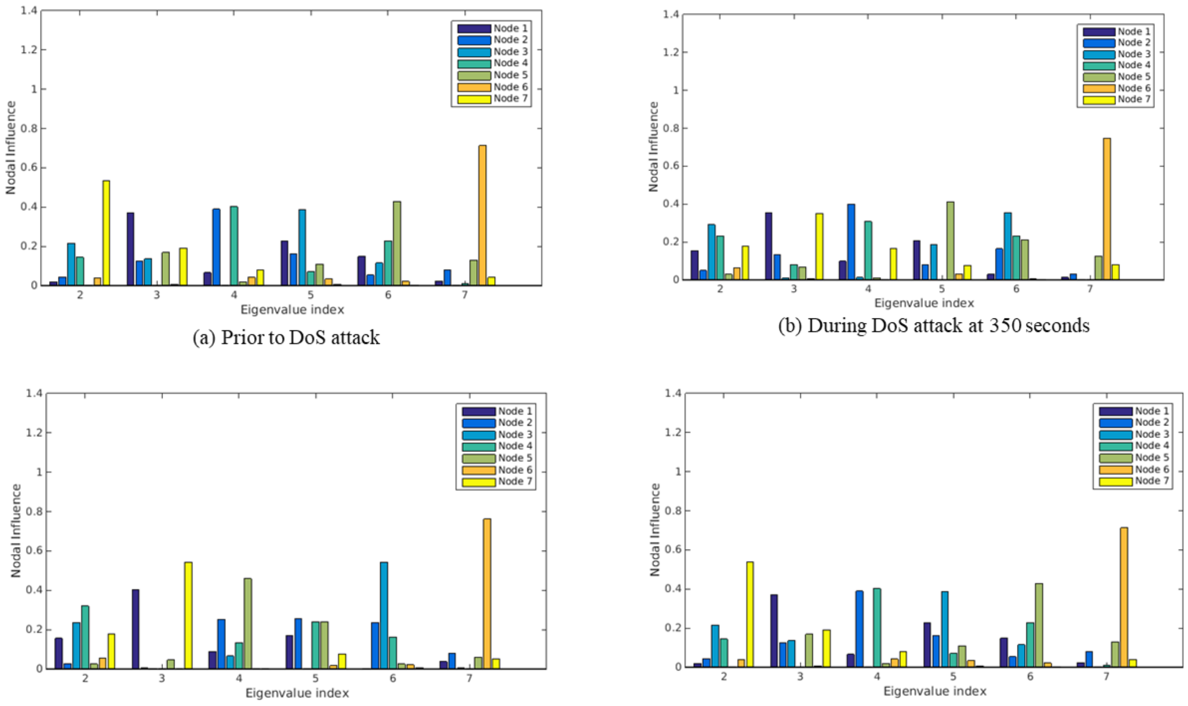
(b) During DoS attack at 350 seconds

(c) During DoS attack at 450 seconds

(d) Post DoS attack

Figure 15. Congested Network Eigenvalue Behavior: Phantom Node Attached to Node 6

The eigenvalue index also changes as the traffic transiting the nodes exceeds the congestion threshold. This threshold is set by phantom node 7, which starts off in index 2 before congestion, as shown in Figure 11(c). Traffic crosses the threshold at node 3, resulting in a shifting of the node's greatest nodal influence in the eigenvalue index spectrum. When the network is in normal conditions, as depicted in Figure 11(c), node 3 dominates in nodal influence in index 5; however, as we see in Figure 16, node 3 has the most nodal influence in eigenvalue index 2 during the DoS attack at 350 s. Node 3, initially, has very little influence in index 2, as shown in Figure 16(a). As node 3 crosses the congestion threshold set by the phantom node, node 7, node 3 plays an increased role in index 2, as shown in Figures 16(b) and 16(c), shifting the eigenvalue index spectrum.



(a) Prior to DoS attack

(b) During DoS attack at 350 seconds

(c) During DoS attack at 450 seconds

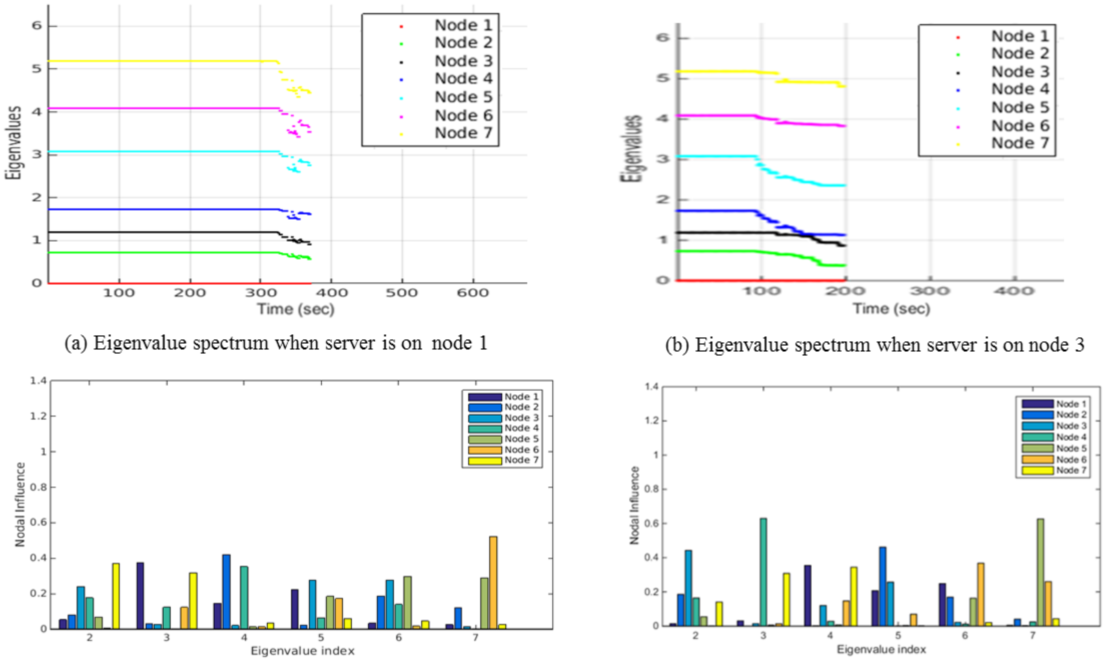
(d) Post DoS attack

Figure 16. Congested Network Behavior with Phantom Attached to Node 6

As a result, post DoS attack, node 7 dominates index 2 again as shown in Figure 16(d). Essentially, the phantom node is shifted to a higher index and node 3 to a lower index.

b. Phantom Node Attached to Node 5

When the phantom node is shifted to node 5, the configuration transforms as the node experiences congestion during the attack; however, the effects of congestion on the nodes when the server is on node 1 and node 3 vary. Congestion was seen sooner when the server was attached to node 3 because the congestion threshold set by node 7 was crossed more rapidly. This can be seen by comparing eigenvalues associated with nodes 5 and 3 in Figures 17(a) and 17(b). In addition, during the experiment, an instantaneous shift in the eigenvalue index was observed in the laboratory when the server was on node 3 as the network was attacked. While node 3 is the first to cross the threshold in both cases, node 4 was the second node to cross the threshold. This is reflected in Figures 17(c) and 17(d). Node 1 was the next node to exceed the threshold when the server was connected to node 1.



(a) Eigenvalue spectrum when server is on node 1

(b) Eigenvalue spectrum when server is on node 3

(c) Eigenvalue index spectrum when server is on node 1

(d) Eigenvalue index spectrum when server is on node 3

Figure 17. Congested Network Behavior with Phantom Node Attached to Node 5

By attaching the phantom node, node 7, to node 5 and shifting the server between node 1 and node 3, the time at which congestion was reflected in the network was impacted. Results in Section 1b of this chapter help to provide information that is used to explain the outcomes observed in this situation.

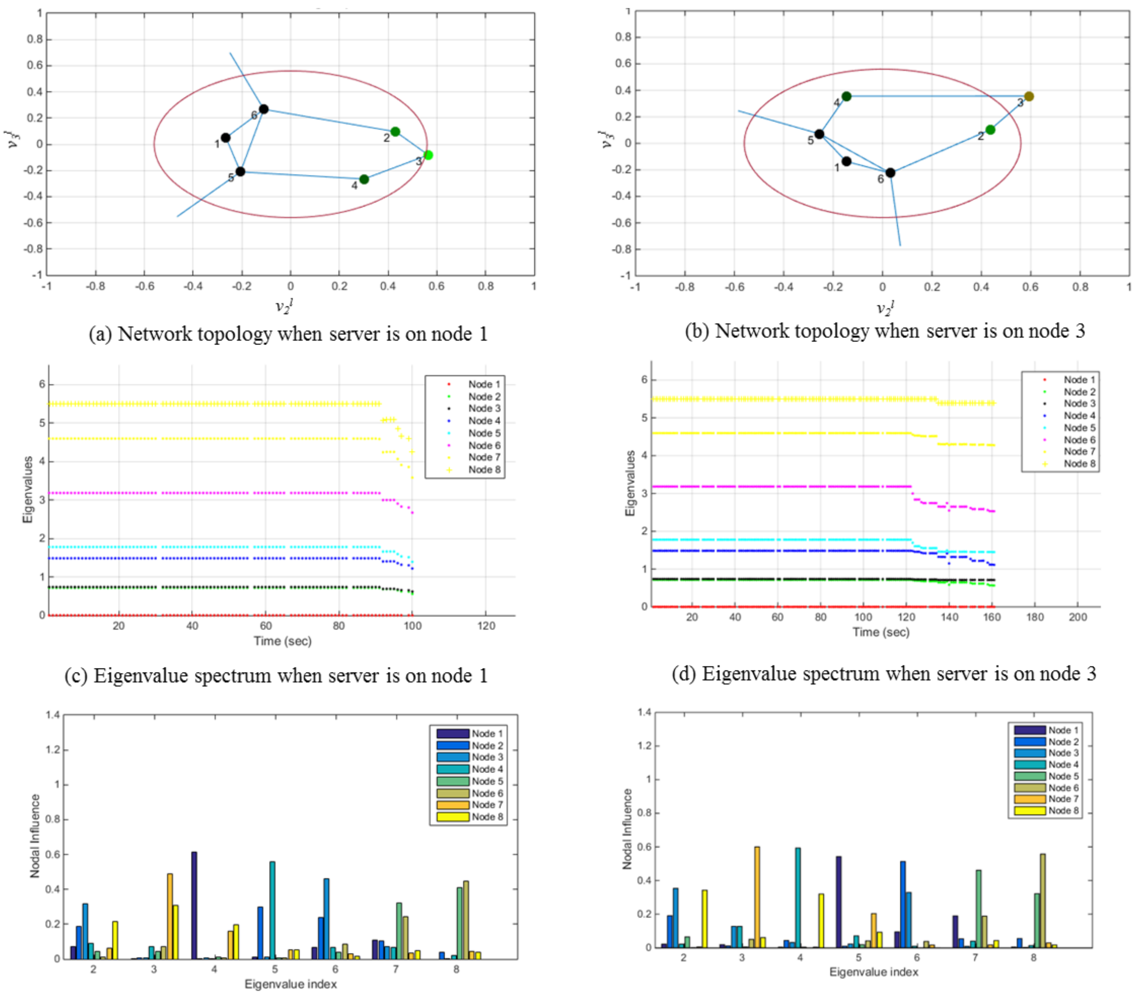
In Figure 12(a), node 1 is the node with the greatest nodal influence in eigenvalue index 3, meaning this node is one of the least connected nodes. Furthermore, node 1 is directly linked to the two most central nodes, nodes 5 and 6, the nodes with the most connectivity. This is depicted by the positions of dominance at the higher end of eigenvalue index spectrum in Figure 12(b). The connection enables the node to not become congested as quickly because nodes 5 and 6 have three nodal interfaces through which UDP traffic can be diverted; however, node 3 is linked to the two least central nodes, nodes 4 and 2. These nodes only have a sum of two interfaces. Furthermore, in Figure 12(c), these nodes have the most nodal influences on eigenindex 4, showing them to be less connected than nodes 5 and 6. Consequently, the impact to the eigenvalues happens more quickly as node 3 experiences congestion at a higher rate; therefore, we observed a shift in the eigenvalue index spectrum at a quicker rate when the server was on node 3 due to the effect of the eigenvalue characteristics, degree, and connectivity of the node. The next configuration to undergo a DoS attack is one that consists of two phantom nodes.

c. Phantom Node Attached to Nodes 5 and 6

By supplementing two phantom nodes, nodes 7 and 8, to the network configuration, we detected congestion in the network more rapidly. In Figure (18), node 5 is attached to node 8, and node 6 is attached to node 7. When comparing Figure 14, where only one phantom node is added, to Figures 18(a) and 18(b), in which only the server is moved with the network, both configurations maintain the more symmetrical topology. This topology is similar to the one exhibited when the network is not in a congested environment, as shown in Figure 13(a). Topology shifts during the DoS attack, as shown in Figure 18(a). The DoS attack commences around 95 s, as displayed by the fall in eigenvalue plots in Figure 18(b). Congestion in the network is reflected starting at 125 s, as shown in Figure 18(c).

In Figures 18(e) and 18(f), we see that the UDP traffic at node 3 is the first to exceed the congestion thresholds set by nodes 7 and 8. As illustrated in Figure 18(f), out of the two phantom nodes, only node 8 continues to shift as congestion at node 3 cause nodes 3 and 4 to dominate in nodal influence in eigenvalue indices 3 and 4.

In retrospect, the server being moved from node 1 to node 3 significantly impacts the eigenvalues at each node. This is due to node 1 being directly connected to the most central nodes and node 3 to the least. In addition, in Figure 17, only one node passes the threshold in a 50-s time interval; however, as depicted in Figures 17(d) and 18(f), the eigenvalue index, where one of the phantom nodes held the greatest nodal influence, changes twice. We observed that the addition of two phantom nodes increases the rate at which congestion is detected.



(a) Network topology when server is on node 1

(b) Network topology when server is on node 3

(c) Eigenvalue spectrum when server is on node 1

(d) Eigenvalue spectrum when server is on node 3

(e) Eigenvalue index spectrum when server is on node 1

(f) Eigenvalue index spectrum when server is on node 3

Figure 18. Congested Network Behavior with Two Phantom Nodes

Furthermore, traffic in node 3 crosses the congestion threshold regardless if one or two phantom nodes are present. Lastly, congestion is detected more quickly when the server is placed on node 3.

In summary, the results indicate that an anomaly, in this experiment congestion, can be detected in a network. The spectral graph representation can be used to provide a mathematical model of the configuration of the network. In addition, the phantom node can be used to set the congestion threshold. Congestion is reflected when the available link capacity threshold is exceeded due to traffic volume at other nodes. The eigenvalue index spectrum shifts as the congested node’s traffic exceeds the threshold and the node’s nodal influence becomes greater in one of the lowest eigenvalue indices within the eigenvalue index spectrum.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

Spectral graph and phantom node techniques are implemented on a SDN and analyzed to determine the real-time state of the network prior to and after congestion. The spectral graph methods allow mathematical modeling of the topology of the network based on eigenvalue-eigenvector solutions. The eigenvalues aid in determining the nodal influence and connectivity of each node in a network. To calculate the eigenvalues, data regarding the number of packets transiting each node are sent to the controller. A MATLAB code is used to manipulate the data into more interpretable information and compute the eigenvalues and eigenvector vectors. Experiments were conducted under two operating conditions: when the network was under DoS attack and when the network was not under DoS attack. Network behavior under no attack was used as a baseline and compared to the behavior observed under DoS attack.

The results of implementation of the spectral graph and phantom methods on the SDN are similar to those in [11]. The phantom node causes the node to which it is attached to have the highest nodal influence in the highest eigenvalue index. In the DoS attack that causes congestion, a shift occurs in the eigenvalue and eigenvalue index spectrum. The phantom node, which sets the threshold for congestion and originally had the greatest nodal influence in the lowest eigenvalue index, dominates in nodal influence in the next higher index. This is due to the amount of traffic transiting another node exceeding the traffic threshold of the phantom node. When this occurs, the node that exceeds the available link capacity threshold obtains the most nodal influence in the phantom node’s original eigenvalue index. When two phantom nodes are added, the supplemented nodes increase the rate of detection, i.e., they decrease the time that a shift in the eigenvalue index spectrum is observable. Regardless of the number of phantom nodes attached, the node with the smallest degree and directly connected to the other least central nodes is the first to surpass the congestion threshold; therefore, the spectral graph theory and phantom node technique are useful in detecting congestion in software-defined network under a DoS.

A. CONTRIBUTIONS

Several contributions were made to the overall study of using spectral graph theory and phantom node techniques to detect congestion in a SDN in this thesis. Results from implementation on a testbed in which actual traffic was used to congest the network were also presented.

The addition of two phantom nodes in the network enables the detection of congestion more quickly than with one phantom node. The second phantom node provides an additional threshold as it is placed on the second highest central node in the SDN; however, there is no change in the time rate of congestion at the least central node regardless of whether one or two phantom nodes are present.

Finally, by moving the server affected with malicious code, another anomaly was observed. When the server was placed on the least central node, congestion was detected more quickly. This is illustrated in starting at Chapter IV, Subsection C.2.b.

B. RECOMMENDATIONS FOR FUTURE WORK

Although congestion is detected when caused by a DoS, there remains room for future studies. The work in [11] and this thesis only lays the framework for detecting congestion, not for determining the cause. This study implemented the work in [11] on a small SDN testbed and showed that congestion can be detected when a DoS is simulated on the network. The drawback is that the experiment only evaluated the ability to detect congestion using spectral graph theory and phantom node methodology, not to detect the reason behind the congestion. In a future study, once congestion is detected, further analysis should be developed to determine if the packets are malicious. Computer algorithms that distinguish good from bad packets may be used to trace the path from which the packets came.

Additionally, decision-making mechanisms can be employed in future research to isolate the malicious packets to prevent them from causing significant damage in the network. These packets could also be used to trace the path back to the node from which the attack occurred.

APPENDIX

```
% Script for Congestion Analysis Using Spectral Graph Theory and Phantom Node Technique

% Author: Thomas Parker, 2015 for calculations of link weights and generation of matrix A
% Modified by: Moniqua Maxie, 2015

close all
clear all
clc

[time3, portstats3]=importdata3('PortStats3.txt',1,26);
[time5, portstats5]=importdata5('PortStats5.txt',1,26);
[time6, portstats6]=importdata6('PortStats6.txt',1,26);

time_then=[time3(6);time5(6);time6(6)];

rx3_7_then=portstats3(8,4);
rx3_9_then=portstats3(10,4);
tx3_7_then=portstats3(8,6);
tx3_9_then=portstats3(10,6);

rx6_11_then=portstats6(12,4);
rx6_9_then=portstats6(10,4);
rx6_12_then=portstats6(13,4);
tx6_11_then=portstats6(12,6);
tx6_9_then=portstats6(10,6);
tx6_12_then=portstats6(13,6);

rx5_11_then=portstats5(12,4);
rx5_9_then=portstats5(10,4);
rx5_7_then=portstats5(8,4);
tx5_11_then=portstats5(12,6);
tx5_9_then=portstats5(10,6);
tx5_7_then=portstats5(8,6);

% the addition of a phantom node to the adjacency matrix
adj= [0 0 0 0 1 1 0 0; 0 0 1 0 0 1 0 0; 0 1 0 1 0 0 0 0; 0 0 1 0 1 0 0 0; 1 0 0 1 0 1 0 1; 1 1 0 0 1 0 1 0; 0 0 0 0 0 0 1 0; 0 0 0 0 1 0 0 0]; % phantom node 7 attached to nodes 5 and 6
L=diag(sum(adj))-adj;
[Vbase,Dbase]=eig(L);
max_r=10e6/8;

ref=Vbase(:,4:6)*Vbase(:,4:6)';
dist=sqrt(Vbase(:,2).^2+Vbase(:,3).^2);
pause(2)
t = linspace(0,2*pi);
mins=20;
delay=1;

counter=1;
FigHandle = figure('Position', [100, 100, 1049, 895]);
subplot(2,2,1)
```

```
for x=1:mins*60
 [time3, portstats3]=importdata3('PortStats3.txt',1,26);
 [time5, portstats5]=importdata5('PortStats5.txt',1,26);
 [time6, portstats6]=importdata6('PortStats6.txt',1,26);
 if isempty(portstats3) || isempty(portstats5) || isempty(portstats6) || isempty(time3) || isempty(time5) ||
isempty(time6) == 1
 %do nothing
 S = 'empty import';
 disp(S)
 else
 time_now=[time3(6);time5(6);time6(6)];
 del_time=time_now-time_then;
 if del_time(1) < 0
 del_time(1)=del_time(1)+60;
 end

 if del_time(2) < 0
 del_time(2)=del_time(2)+60;
 end

 if del_time(3) < 0
 del_time(3)=del_time(3)+60;
 end
 del_time
 x
 if abs(del_time(1)-del_time(2))>.5 || abs(del_time(1)-del_time(3)) > .5 || abs(del_time(2)-del_time(3)) > .5
 % do nothing
 S = 'del_time too large';
 disp(S)
 %time_then=time_now;
 else

 rx3_7_now=portstats3(8,4);
 rx3_9_now=portstats3(10,4);
 tx3_7_now=portstats3(8,6);
 tx3_9_now=portstats3(10,6);

 rx6_11_now=portstats6(12,4);
 rx6_9_now=portstats6(10,4);
 rx6_12_now=portstats6(13,4);
 tx6_11_now=portstats6(12,6);
 tx6_9_now=portstats6(10,6);
 tx6_12_now=portstats6(13,6);

 rx5_11_now=portstats5(12,4);
 rx5_9_now=portstats5(10,4);
 rx5_7_now=portstats5(8,4);
 tx5_11_now=portstats5(12,6);
 tx5_9_now=portstats5(10,6);
 tx5_7_now=portstats5(8,6);

 lwrx_1=(rx6_11_now-rx6_11_then)/del_time(3);
 lwtx_1=(tx6_11_now-tx6_11_then)/del_time(3);

 lwrx_2=(rx5_11_now-rx5_11_then)/del_time(2);
 lwtx_2=(tx5_11_now-tx5_11_then)/del_time(2);

 lwrx_3=(rx6_9_now-rx6_9_then)/del_time(3);
 lwtx_3=(tx6_9_now-tx6_9_then)/del_time(3);

 lwrx_4=(rx6_12_now-rx6_12_then)/del_time(3);
 lwtx_4=(tx6_12_now-tx6_12_then)/del_time(3);

 lwrx_5=(rx3_7_now-rx3_7_then)/del_time(1);
 lwtx_5=(tx3_7_now-tx3_7_then)/del_time(1);

 lwrx_6=(rx3_9_now-rx3_9_then)/del_time(1);
 lwtx_6=(tx3_9_now-tx3_9_then)/del_time(1);

 lwrx_7=(rx5_7_now-rx5_7_then)/del_time(2);
 lwtx_7=(tx5_7_now-tx5_7_then)/del_time(2);

 lw(1)=(1-(lwrx_1/max_r))*(1-(lwtx_1/max_r));
 lw(2)=(1-(lwrx_2/max_r))*(1-(lwtx_2/max_r));
 lw(3)=(1-(lwrx_3/max_r))*(1-(lwtx_3/max_r));
 lw(4)=(1-(lwrx_4/max_r))*(1-(lwtx_4/max_r));
 lw(5)=(1-(lwrx_5/max_r))*(1-(lwtx_5/max_r));
 lw(6)=(1-(lwrx_6/max_r))*(1-(lwtx_6/max_r));
 lw(7)=(1-(lwrx_7/max_r))*(1-(lwtx_7/max_r));
 lw

 % phantom node on node 5 and 6
 adj=[0 0 0 0 lw(2) lw(1) 0 0; 0 0 lw(5) 0 0 lw(4) 0 0; 0 lw(5) 0 lw(6) 0 0 0 0; 0 0 lw(6) 0 lw(7) 0 0 0; lw(2)
0 0 lw(7) 0 lw(3) 0 1; lw(1) lw(4) 0 0 lw(3) 0 1 0; 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 0 0]; L=diag(sum(adj))-adj;
 [Vnow,Dnow]=eig(L);

 % Author: Jamie Johnson's, 2014 for generating the computational model and eigen spectrums
        % Modified by: Moniqua Maxie, 2015

        % Call Nodal Basis [7]
        model=nodalbasis(n)

 vec(:,1:7,x)=[Vnow(:,2),Vnow(:,3),Vnow(:,4),Vnow(:,5),Vnow(:,6),Vnow(:,7),Vnow(:,8)];
 vals(:,x)=diag(Dnow);

 if Vnow(1,2) > 0
 Vnow(:,2)=Vnow(:,2).*-1;
 end

 if Vnow(2,3) < 0
 Vnow(:,3)=Vnow(:,3).*-1;
 end

 data1=(lwtx_1+lwtx_2)*8;
 data5=(lwrx_2+lwtx_3+lwrx_7)*8;
 dataall =
(lwrx_1+lwtx_1+lwrx_2+lwtx_2+lwrx_3+lwtx_3+lwrx_4+lwtx_4+lwrx_5+lwtx_5+lwrx_6+lwtx_6+lwrx_
7+lwtx_7)*8

 subplot(2,2,1)
```

```
gplot(L,[Vnow(:,2),Vnow(:,3)])
grid on
hold on
for y=1:6
str=sprintf('%0.0f',y);
text(Vnow(y,2)-.05,Vnow(y,3)-.05,str);
end

for y=1:6
fromavg=(.56-(sqrt(Vnow(y,2)^2+Vnow(y,3)^2)));
green(y)=1-abs(fromavg/.25);
if fromavg > 0
blue(y) = fromavg/.25;
red(y) = 0;
else
blue(y) = 0;
red(y) = -1.*fromavg/.25;
end
if red(y) > 1
red(y) = 1;
end
if green(y) > 1
green(y) = 1;
end
if red(y) < 0
red(y) = 0;
end
if green(y) < 0;
green(y) = 0;
end
plot(Vnow(y,2),Vnow(y,3),'r.','MarkerSize',30,'Color',[red(y),green(y),0])
hold on
plot(.56.*cos(t),.56.*sin(t))
end
xlim([-1,1])
ylim([-1,1])
title('Network in 2-D Eigenspace')
hold off

figure(1)
subplot(2,2,2)
hold on

title('Total Data Rate in the Network')
xlabel('Time (sec)')
ylabel('Data Rate (bps)')
grid on
plot(x,dataall,'.r','MarkerSize',5);


xlim([1,60*mins])
%ylim([0,max_r])
hold off

%figure(3)
```

```
subplot(2,2,3)
hold on

% when two phantom nodes are attached
plot(x,Dnow(1,1),'r.','MarkerSize',5);
plot(x,Dnow(2,2),'g.','MarkerSize',5);
plot(x,Dnow(3,3),'k.','MarkerSize',5);
plot(x,Dnow(4,4),'b.','MarkerSize',5);
plot(x,Dnow(5,5),'c.','MarkerSize',5);
plot(x,Dnow(6,6),'m.','MarkerSize',5);
plot(x,Dnow(7,7),'y.','MarkerSize',5);
plot(x,Dnow(8,8),'y+','MarkerSize',5);
xlim([1,mins*60])
ylim([0,6.5])
grid on
title('Eigenvalues')
xlabel('Time (sec)')
ylabel('Eigenvalues')
legend('Node 1','Node 2','Node 3','Node 4','Node 5','Node 6', 'Node 7', 'Node 8')
hold off

state=Vnow(:,4:6);
statematrix=state*state';
error=norm(ref-statematrix,'fro');

xlim([1,60*mins])
%ylim([0,max_r])
hold off


rx3_7_then=rx3_7_now;
rx3_9_then=rx3_9_now;
tx3_7_then=tx3_7_now;
tx3_9_then=tx3_9_now;

rx6_11_then=rx6_11_now;
rx6_9_then=rx6_9_now;
rx6_12_then=rx6_12_now;
tx6_11_then=tx6_11_now;
tx6_9_then=tx6_9_now;
tx6_12_then=tx6_12_now;

rx5_11_then=rx5_11_now;
rx5_9_then=rx5_9_now;
rx5_7_then=rx5_7_now;
tx5_11_then=tx5_11_now;
tx5_9_then=tx5_9_now;
tx5_7_then=tx5_7_now;

time_then=time_now;

end
end

pause(delay)
```

```
end
```

LIST OF REFERENCES

[1] M. Bouet, K. Phemius, and J. Leguay, “Distributed SDN for mission-critical networks,” in *Proc. of the IEEE Military Commun. Conf.*, 2014. [Online]. doi: 10.1109/MILCOM.2014.162.

[2] D. Kreutz, F. M. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig., “Software-defined networking: A comprehensive survey,” in *Proc. of the IEEE*, 2015. [Online]. doi: 10.1109/JPROC.2014.2371999.

[3] S. Lawson, “Facebook’s open computing project start to crack networking,” *PCWorld in IDG*, Mar. 11, 2015. [Online]. Available: http://www.pcworld.com/article/2895652/facebooks-open-compute-project-starts-to-crack-networking.html

[4] “Facebook open switching system (FBOSS),” Github: facebook/fboss. [Online]. Accessed Jun. 8, 2014. Available: https://github.com/facebook/fboss.

[5] A. Simpkins, “Facebook open switching system (FBOSS) and “wedge” in the open,” F-Code: Infra, Open Compute, Data Centers, Networking and Traffic, Open Source, March 10, 2015. [Online]. Available: https://code.facebook.com/posts/843620439027582/facebook-open-switching-system-fboss-and-wedge-in-the-open/.

[6] 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, and A. Vahdat, “B4: Experience with a globally-deployed software defined WAN,” in *Proc. ACM SIGCOMM of China*, 2013. [Online]. Available: http://cseweb.ucsd.edu/~vahdat/papers/b4-sigcomm13.pdf. Accessed Jun. 8, 2014.

[7] M. Toy, “Networks and Services Management,” in *Cable Networks, Services, and Management*, 1st ed., Hoboken, NJ, USA, Wiley-IEEE Press, 2015.

[8] B. Ferguson, A. Tall and D. Olsen, "National cyber range overview," *in Proc. of Military Commun. Conf.*, Baltimore, MD, 2014. [Online]. doi: 10.1109/MILCOM.2014.27.

[9] J. Ashraf and S. Latif, “Handling intrusion and DDoS attacks in software defined networks using machine learning techniques,” in *Proc. of Nat. Software and Eng. Conf.*, Rawalpindi, Pakistan, 2014. [Online]. doi: 10.1109/NSEC.2014.6998241

[10] K. Giotis, G. Androulidakis, and V. Maglaris, “Leveraging SDN for efficient anomaly detection and mitigation on legacy networks,” in *Proc.of Third European Workshop on Software Defined Networks*, London, UK, 2014. [Online]. doi: 10.1109/EWSDN.2014.24

[11] J. Johnson, “Software defined network monitoring scheme using spectral graph theory and phantom nodes,” M.S. thesis, Dept. Elect. And Comput. Eng. Naval Postgraduate School, Monterey, CA, 2014. [Online]. Available: http://hdl.handle.net/10945/43933

[12] J. Kizza, *Computer Network Security,* New York: Springer, 2005.

[13] X. Wei and Z. Li, “Analysis and solution of the network congestions in the local area network,” in *Proc. of Third International Conf. on Consumer Electron., Commun., and Networks,* Xianning, China, 2013. [Online]. doi: 10.1109/CECNet.2013.6703290

[14] G. Jin and H. Tang, “Control transmission pace at IP layer to avoid packet drop,” in *Proc. in IEEE Int. Workshop on IP Operations and Manag.*, Beijing, China, China, 2004. [Online]. doi: 10.1109/IPOM.2004.1547593

[15] Y. Huang and J. Pullen, “Countering denial-of-service attacks using congestion triggered packet sampling and filtering,” in *Proc. of Tenth Int. Conf. on Comput. Commun. and Networks,* Scottsdale, Arizona, USA, 2001. [Online]. doi: 10.1109/ICCCN.2001.956309

[16] S. Khattak, N. R. Ramay, K. R. Khan , A. A. Syed, and S. A. Khayam ., “A taxonomy of botnet behavior, detection, and defense,” *IEEE Commun. Surveys Tut.*, vol. 16, no. 2, pp. 898-924, Oct. 2013. [Online]. doi: 10.1109/SURV.2013.091213.00134

[17] A. Lara and B. Ramamurthy, “OpenSec: A framework for implementing security policies using openflow,” in *Proc. of IEEE Global Commun. Conf.*, Austin, TX, USA, 201. [Online]. doi: 10.1109/GLOCOM.2014.7036903

[18] D. Jiang, J. Liu, Z. Xu, and W. Qin , “Network traffic anomaly detection based on sliding window,” in *Proc. of Int. Conf. on Elect. and Control Eng.*, Yichang, China, 2011. [Online]. doi: 10.1109/ICECENG.2011.6057677

[19] M. Jarschel, T. Zinner, T. Höhn, P. Tran-Gia , “On the accuracy of leveraging SDN for passive network measurement,” in *Proc. of Australasian Telecommun. Networks and Appl. Conf.*, Christchurch, New Zealand, 2013. [Online]. doi: 10.1109/ATNAC.2013.6705354

[20] N. M. Sahri and K. Okamura, “Load sensitive forwarding for software defined networking-openflow based,” *Advances in Comput.r Sci.: an Int. J.*, vol. 3, no. 6, pp. 38-43, Nov. 2014. [Online]. Available: http://acsij.org/acsij/article/view/144

[21] P.V. Mieghem, *Graph Spectra for Complex Networks*. New York, NY: Cambridge University Press, 2011.

[22] T. Parker, J. Johnson, M. Tummala, J. McEachen, J. Scrofani, “Identifying Congestion in Software-Defined Networks Using Spectral Graph Theory,” in *Proc. of 48th Asilomar Conf. Signals, Syst., and Comput.*, Pacific Grove, CA, USA 2014. [Online]. doi: 10.1109/ACSSC.2014.7094824

[23] Component-based Software Defined Networking Framework: Build SDN Agiley. Accessed Jun. 8, 2014. [Online]. Available: http://osrg.github.io/Ryu/.

[24] RYU 3.20 Documentation: Welcome to RYU the Network Operating System (NOS). Accessed Jun. 8, 2014. [Online]. Available: http://Ryu.readthedocs.org/en/latest/index.html.

[25] HP2920 Switch Series. Accessed Jun. 8, 2014. [Online].Available: http://www8.hp.com/us/en/products/networking-switches/product-detail.html?oid=5354494#!tab=specs.

[26] Raspbian. Accessed Jun. 8, 2014. [Online]. Available: http://www.raspbian.org/RaspbianFAQ.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California