NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

SPECTRAL GRAPH THEORY ANALYSIS OF SOFTWARE-DEFINED NETWORKS TO IMPROVE PERFORMANCE AND SECURITY

by

Thomas C. Parker

September 2015

Dissertation Co-Supervisors: Murali Tummala
John McEachen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2015	**3. REPORT TYPE AND DATES COVERED** Dissertation

4. TITLE AND SUBTITLE
SPECTRAL GRAPH THEORY ANALYSIS OF SOFTWARE-DEFINED NETWORKS TO IMPROVE PERFORMANCE AND SECURITY

5. FUNDING NUMBERS

6. AUTHOR(S) Parker, Thomas C.

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 dissertation 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)

Software-defined networks are revolutionizing networking by providing unprecedented visibility into and control over data communication networks. The focus of this work is to develop a method to extract network features, develop a closed-loop control framework for a software-defined network, and build a test bed to validate the proposed scheme. The method developed to extract the network features is called the dual-basis analysis, which is based on the eigendecomposition of a weighted graph that accounts for the network topology and traffic load. A software-defined network closed-loop control scheme is developed; the scheme is modeled after a closed-loop control system that includes an observer and a controller. A particle filter and phantom node are used to estimate link data rates and identify the onset of congestion. Based on the outputs of the observer, the controller is able to balance traffic throughout the network to minimize congestion. A software-defined network test bed is developed to evaluate the proposed dual-basis representation and the closed-loop control scheme. The test bed is a real-world implementation of a software-defined network that consists of 13 switches and one controller. The test bed ensures that the proposed schemes are suitable even when applied in a hardware or software implementation.

14. SUBJECT TERMS Software-defined network, networking, cybersecurity, eigenvalue, eigenvector, graph theory, spectral graph theory, control theory	**15. NUMBER OF PAGES** 175
	16. PRICE CODE

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

NSN 7540–01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SPECTRAL GRAPH THEORY ANALYSIS OF SOFTWARE-DEFINED NETWORKS TO IMPROVE PERFORMANCE AND SECURITY

Thomas C. Parker
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 2003
M.S., Naval Postgraduate School, 2007

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2015

Author: Thomas C. Parker

Approved by:

John McEachen
Professor, Electrical and Computer Engineering
Dissertation Co-Advisor

Douglas Fouts
Professor, Electrical and Computer Engineering

James Scrofani
Associate Professor, Electrical and Computer Engineering

James Newman
Professor, Space Systems Academic Group

Murali Tummala
Professor, Electrical and Computer Engineering
Dissertation Committee Chair and Dissertation Co-Advisor

Approved by: R. C. Robertson, Chair, Dept. of Electrical and Computer Engineering

Approved by: Douglas Moses, Vice Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Software-defined networks are revolutionizing networking by providing unprecedented visibility into and control over data communication networks. The focus of this work is to develop a method to extract network features, develop a closed-loop control framework for a software-defined network, and build a test bed to validate the proposed scheme. The method developed to extract the network features is called the dual-basis analysis, which is based on the eigendecomposition of a weighted graph that accounts for the network topology and traffic load. A software-defined network closed-loop control scheme is developed; the scheme is modeled after a closed-loop control system that includes an observer and a controller. A particle filter and phantom node are used to estimate link data rates and identify the onset of congestion. Based on the outputs of the observer, the controller is able to balance traffic throughout the network to minimize congestion. A software-defined network test bed is developed to evaluate the proposed dual-basis representation and the closed-loop control scheme. The test bed is a real-world implementation of a software-defined network that consists of 13 switches and one controller. The test bed ensures that the proposed schemes are suitable even when applied in a hardware or software implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS

SDN	Software-Defined Network
ONF	Open Networking Foundation
ARPANET	Advanced Research Projects Agency Network
DMZ	Demilitarized Zone
AMQ	Automated Malware Quarantine
SNR	Signal-to-Noise Ratio
LTI	Linear, Time Invariant
bps	Bits per Second
DDOS	Distributed Denial of Service
NIC	Network Interface Card
NTT	Nippon Telegraph and Telephone
C2	Command and Control
SSH	Secure Shell

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife, Leah, for supporting me as I worked to complete this research. I would also like to thank all of the Center for Cyber Warfare interns that worked on the SDN test bed over the last two summers. Their grunt work on the test bed allowed me to accomplish more hardware experiments that I had expected. Finally, I would like to thank Professors Murali Tummala and John McEachen for their patient guidance and advice. Finally, I would like to thank Michelle Pagnani from the Graduate Writing Center for helping me edit all of my writing, from my proposal through all of my conference papers and this dissertation. She was not only an outstanding editor, but also a wonderful teacher.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Current data communication networks have become too complex and too costly to continue operating and administrating them in the same basic manner as standardized by Advanced Research Projects Agency Network (ARPANET) in the late 1970s. The designers of ARPANET decided that a distributed architecture would be better because it is more resilient to failures [1]. They could not have envisioned how important networks would become to modern life, how widespread cyber espionage and cyber crime would become, and how complex these systems would turn out to be. The cost to manage and defend these complex systems needs to be reduced, while simultaneously increasing network performance to meet future demands.

A. MOTIVATION

Software-defined networking (SDN) has stepped in with the goal of reducing cost and increasing performance. This goal is achieved by simplifying the network hardware, reducing the complexity created by distributed algorithms, providing insight into the network behavior, and allowing control of all network functions from a centralized location. The simplified hardware is less expensive and consumes less power. By centralizing control, the network controller manages the network as a whole. Current networks are not able to provide the insight into the network's behavior or the flexibility to modify the network's behavior as needed.

Compare the automobile traffic on an interstate, which has no centralized monitoring or control, with airline traffic, which does have centralized monitoring and control. With automobile traffic, there is no method to prevent congestion through prioritizing certain types of traffic, rerouting traffic, or implementing any other congestion control methods throughout a city. However, air traffic is centrally controlled by air traffic controllers who can manage their local air traffic based on the needs and conditions of the system as a whole. Each air traffic controllers shares the same global traffic and weather picture. From this global picture, they are able to make decisions to proactively prevent congestion, ensure safety, and increase throughput.

Google, Facebook, AT&T, and Verizon have all decided to implement SDN as part of their core networks for some of the same reasons that air traffic is centrally controlled. They have made this choice because SDN reduces costs, boosts performance and increases flexibility [2]. Google has achieved 95% utilization in their SDN implementation [3], [4]. Facebook has automated many of its network functions by disaggregating the forwarding hardware from the control software [5]. AT&T and Verizon are in need of greater flexibility to route phone calls, texts, and data over their core networks [6]. One of their goals is to increase network throughput without adding additional hardware.

SDNs are poised to change the way networks are managed, but the transition from distributed networks will be successful only if they are built around established engineering principals. SDNs can make network measurements, decide how to route packets, and then implement those actions. The process of measurement, decisions and action is known as the observe, orient, decide, and act (OODA) loop in military strategy and tactics [7]. When applied to autonomous systems in the private sector, the OODA loop steps are renamed monitor, analyze, plan and execute [8]. Whether the OODA terms or the autonomic terms are used, the closed loop of measuring the environment, processing those measurements, making decisions based on the processed data, and acting on the decisions is a closed-loop control system [9].

Closed-loop control systems are the technical implementations of OODA loops, and SDNs are fundamentally closed-loop control systems, with the network as the object to be controlled. The main benefit of a closed-loop control system is responsiveness to the current state of the system. Open-loop control systems do not receive feedback from the system being controlled and are unable to respond to anomalies in the system. In closed-loop control systems, the controller provides feedback by changing the input to the system to improve the performance of the system.

B. OBJECTIVE

The objective of this research is to develop spectral graph theory methods to extract SDN network features and then develop a scheme that utilizes these features to influence the network behavior to improve the performance and the security of the network. These features are extracted from graphs that are derived from both the current network topology and current measured traffic; the topology may change and traffic load may fluctuate over time. These dynamics describe the behavior of the network. Because the extracted features describe the overall status of the network, they are considered to represent the *state* of the network.

One of the goals of developing a network state for SDNs is to monitor the network's behavior. If the proper features can be extracted from the network topology and traffic, the SDN controller can monitor these features to track network behavior, such as onset of congestion and malicious activity. The controller can respond at network speeds to anomalous behavior and proactively mitigate congestion. The objective here is different from the implementation of past anomaly-detection algorithms because the controller can use global information to determine the occurrence of an anomaly as opposed to attempting to determine an anomaly based only on local information or local traffic analysis. Because the controller can monitor the network-wide behavior and determine global features for the network, it may be better suited to find anomalies and detection congestion.

For the controller to effectively use these features to influence the behavior of the network, the SDN may be considered a closed-loop control system. This work adopts many control theory concepts and terminology, such as state, feedback, observer, and controller. Control theory techniques cannot be directly applied to a SDN; however, these concepts can be applied to develop a closed-loop control framework for SDN that provides a basis for future development of applications and networks.

To experimentally validate these objectives, the proposed methods must be implemented on a SDN test bed. Mathematical analysis and simulation are insufficient to fully validate methods to monitor and control networks because of the complexity of the

systems being validated. Analysis and simulation typically require the researcher to make assumptions about the network operation. It is difficult to model all of the interactions and timing issues that are present in a real-world system. For these reasons, the proposed schemes in this research are implemented on a SDN test bed to validate the effectiveness of the proposed methods.

C. RELATED WORK

SDN researchers have acknowledged that these networks are closed-loop control systems; however, specific solutions to manage the SDN as a closed-loop control system have not been proposed. The development of applications that implement the OODA loop in a SDN can benefit from the wealth of knowledge that has been developed for other closed-loop control systems, such as non-linear state estimators and optimal controllers. If these concepts can be extended to SDNs, greater confidence can be placed in the applications developed to control the network.

1. Application of Graph Theory to Optimize Network Topology

In [10], SDN was evaluated as the communication infrastructure for a smart grid implementation. It was shown that the topology to distribute power over large areas is not the same topology that is best for the communication network. This result was determined analytically using graph theory based solutions that showed which communicating nodes should be connected to reduce congestion and increase throughput. These results were experimentally validated using simulation of a real-world network and real-world traffic. The traffic was redirected based on the new communication network derived from the graph theory solution.

This solution works well in an industrial control system (ICS) like the smart grid because most ICSs have structured traffic profiles, which means that the traffic between sources and destinations in the network is known and fixed [11]. Extending this solution to arbitrary networks is ineffective because the solution is not dynamic, which does not allow it to account for changing traffic patterns, failed devices and cyber-attacks. General network traffic does have a typical profile, but it can change over time and can be

dramatically different from day to day. A more generic solution must account for the dynamic traffic profiles and network behavior.

The specific solution proposed in [10] rewired the network by keeping the number of links in a network constant and changing the directly connected nodes. This was accomplished by using an unweighted graph, which has the implicit assumption that all links are equally important to the function of the network. This assumption may not be true in all cases. Consider a graph theoretic solution that moves a link that carries no traffic from one location to a new location where it, again, does not carry any traffic. In this case, the performance of the network is unchanged after using the unweighted analysis. On the other hand, by including the traffic profiles and network behaviors in the analysis in the form of a weighted graph, better solutions may be determined.

2. SDN as a Closed-Loop Control System

In Google's B4 network [3], they demonstrated how performance can be improved in a dynamic traffic environment. They managed traffic over links that carry exceptionally large amounts of data. Link utilization was raised to nearly 100% in their test cases. Google's solution incorporated network traffic measurements locally, which were passed to the global traffic engineering server to determine the optimal path to route traffic through the network based on priority and quality-of-service (QoS) required by that specific data type. The decisions made by the global traffic engineering algorithm were passed down to the local site controllers that implemented the decisions made by the next higher level of the architecture.

Google's solution maximized throughput, but they were also able to control all aspects of their network to include when servers were able to transmit. Their solution was dynamic and achieved levels of performance that are infeasible with a distributed algorithm. They did not reveal the performance of the control network and the performance required by the centralized controller and the site controllers. Reducing the workload on these machines reduces cost and improves performance by requiring either fewer or less expensive machines. Again, a more general solution is needed that is able to

account for traffic when the network hosts are not directly controlled by the SDN controller.

By controlling the end hosts directly, a network controller has complete control over all aspects of a network, but this is not feasible in many real-world networks. A more generic solution does not include the assumption that all end hosts are controlled. The controller must accommodate the offered traffic as well as it can. Methods need to be developed that determine the network behavior as a function of the current offered traffic and then change how the traffic is routed in the network to improve the overall network performance. In many cases, this requires a load-balancing algorithm to reduce the possibility of congestion throughout the network. This research develops a method to include near real-time offered traffic in the determination of the graph theoretic representation.

3. Optimal SDN Switch Placement

In the Google B4 network, the deployment of SDN was accomplished all at once, which may not be feasible for all networks or organizations looking to transition to SDN. Many of them may end up with a hybrid of SDN and legacy routers in their networks. In [2], Agarwal, Kodialam and Lakshman examined how one would implement centralized control in a hybrid network of SDN and non-SDN devices. Again, they used network measurement techniques to measure the network data rates and make decisions based on these data rates. They developed a linear programming solution to the problem of network control and showed that even a modest number of SDN switches in a network increases performance.

They did not develop an algorithm to find the specific locations in the topology that provide the largest return on investment. Their method to determine these optimal locations was through an exhaustive search. After trying all possible locations and various traffic matrices, they were able to find the switch that provided the greatest return on investment. This location is static because their solution depended on physically replacing the forwarding device. They did not explore how one would choose which

switches to control if all switches were SDN devices. Their work, however, implies that not all switches need to be controlled to reach required levels of performance.

The research in this dissertation examines methods to find the SDN nodes that must be controlled in order to obtain the gains demonstrated in [2] and [3]; these are called control nodes in this work. The control node locations are determined dynamically in a full SDN deployment in this work. Because each node in the network is an SDN switch, they can all act as legacy routers or as SDN forwarding devices. The results in [2] demonstrate that not all of the switches need to be controlled. By taking into account current traffic patterns, network behavior, and network features, the controller can dynamically update the control node locations to improve performance and reduce the workload of the controller.

4. Optimal Controller Placement

The controller placement issue is similar to the control node placement issue. In large networks, the round-trip time from an SDN switch to the controller and back can become quite large and needs to be minimized. Two methods to minimize the round-trip time from all SDN switches to the controller and back are proposed in [12]. The topology chosen was that of a simplified Internet2 [13], which is the topology adopted for this research. Internet2 was chosen in [12] because researchers were actively debating how many and where the controllers should be placed. The analysis and results in this research are based on the Internet2 topology because there is published work with which to compare these results.

Nevertheless, no algorithm was proposed in [12] to find the optimal locations, but instead a trial-and-error approach was used. This is a simple task when there is one controller, but networks spread over large physical areas may require multiple controllers to achieve the desired performance. As the number of controllers increased, they noted that the solution required “days” of computation to determine the location. Since [12] was published, other methods, such as those proposed in [14] and [15], have been developed to determine controller locations, but almost all of them assume a one-time design choice

of controller locations. Greater performance may be achieved if this analysis is conducted periodically and the controller locations are reassigned dynamically.

5. Cybersecurity

SDNs are poised to not only increase performance but also have created a new paradigm for cybersecurity. With centralized monitoring and control, the controller is able to better monitor the network as compared to network perimeter defenses, such as firewalls and web server demilitarized zones (DMZs). The Open Network Foundation (ONF) proposed the idea of a security application called Automated Malware Quarantine (AMQ) in a white paper discussing security issues associated with SDNs [16]. The proposed application includes a method to monitor the network, detect anomalous behavior, and quarantine a portion of the network, a set of end hosts or a specific infected host to prevent the spread of the malware throughout the network.

The ONF proposed architecture of a security application stopped short of providing any specifics of the network monitoring, anomaly detection or network control features. Specific analysis and software tools developed in this research are adopted from the framework proposed by ONF. This research focuses on developing a method to determine a graph theoretic network representation, which can be used by security applications to constantly monitor the network. Because the controller can develop this representation based on global network information, the controller may be able to more accurately assess the likelihood that the network behavior is anomalous.

The objective of this work is to provide a SDN framework, which is modeled after closed-loop control systems, and to provide a method to describe a representation of the network that reveals key features of the network. To achieve this goal, a SDN needs a state estimator and a state controller. For these two to be effective, network behavior needs to be dynamically calculated based on the current offered traffic. As shown in the previous sections, many of the solutions proposed for SDNs do not account for the network behavior or offered traffic and do not account for the dynamic nature of the network. This work adds the dynamic analysis that was missing from previous research to develop better methods to monitor and control SDNs.

D. OUTLINE OF DISSERTATION

The outline of the dissertation is as follows. The background on SDNs, graph theory, spectral graph theory and closed-loop control are provided in Chapter II. The dual-basis and its role in defining the state of the network is developed in Chapter III. The closed-loop control framework around which SDN controllers can be built is discussed in Chapter IV. The methods used to validate the work from Chapters III and IV are shown in Chapter V. The results obtained from the SDN test bed that was built based on the Internet2 topology are shown in Chapter VI. The conclusions drawn from this research and areas of future work to be considered are provided in Chapter VII. The details of the limit from Eqn. (3.34) are contained in Appendix A. A sample of the Python code for the state controller application, which implemented the state control function, is contained in Appendix B. A sample of the Python code for the monitor application, which implemented the state estimation function, is contained in Appendix C.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SOFTWARE-DEFINED NETWORKING (SDN) AND ITS RELATIONSHIP TO GRAPH AND CONTROL THEORY

Most convectional networks today is accomplished through a litany of distributed algorithms and protocols, which take years to be approved. Once they go into widespread use, it is difficult and time consuming to change or improve them or even to close security vulnerabilities. This difficulty results in workarounds that reduce interoperability and security.

Software-defined networking is poised to change the way large, complex data communication networks are managed and controlled. The goal of SDN is to logically centralize network management at a device called the network controller [17]. From that centralized location, the controller provides unprecedented control over packet routes and collection of network statistics. Managing the network in a centralized manner allows for more effective traffic engineering and security. In this chapter, the background required for the remaining chapters of the dissertation is provided. First, the SDN architecture and operation are described. Next, graph theory and spectral graph theory are introduced, and then applied to the community finding problem. Finally, the basic concepts and techniques used in control theory are reviewed.

A. SOFTWARE-DEFINED NETWORKING

A software-defined network improves the network management and operation by physically separating the control of the network from the data path of the network [18]. This concept is radically different from the way networks currently operate. Networks today are distributed systems in which the devices share information to determine the best possible routes. These distributed systems can be slow to react to changes in network traffic, and routes may be sub-optimal because each router typically does not know the full topology; even protocols that share network-wide link state may not have knowledge of the full topology because of route aggregation [19]. The controller must be able to determine the current, global state. Using the current state, the controller can find globally optimal solutions to improve performance.

1. General Architecture

The Open Networking Foundation defines a three-layer SDN model as depicted in Figure 1 [20]. The infrastructure layer is the physical topology, which is composed of SDN-enabled switches and the links between them. The switches take flow rules as input from the controller and provide statistics about network traffic to the controller as an output. They are also the data forwarding devices that receive individual packets and then transmit these packets toward the intended destination.

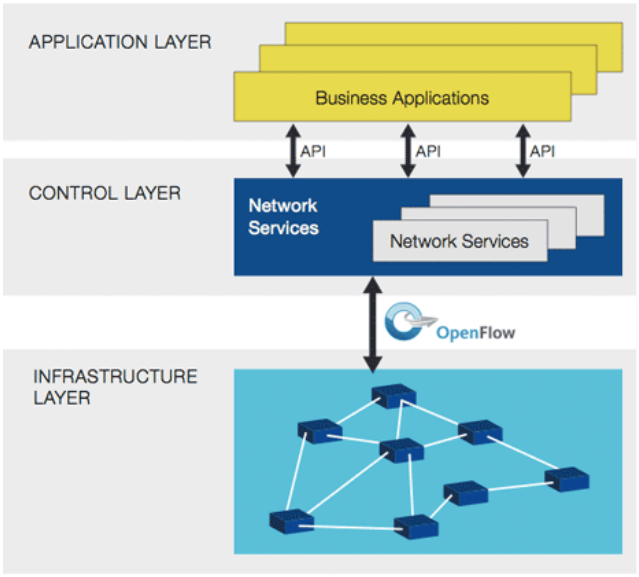

Figure 1. The three layer SDN protocol architecture includes the infrastructure layer, the control layer, and the application layer, from [20]. OpenFlow is the communication protocol between the control layer and the infrastructure layer.

The control layer develops the rules that are sent to the switches. The controller is programmable and uses the network traffic measurements to determine new routes. The controller is software that runs on a computer and communicates with the switches. The controller must be able to communicate with the switches using the OpenFlow communications protocol [21]. Examples of network services that are implemented by the controller include route determination, load-balancing, and topology discovery.

The control packets that are passed between the switches and the controller are separated from the data traffic. Typically, they are sent over a physically separate network called the control network. By having a physically separate control network potentially leads to a more secure implementation.

Network control is implemented via flow rules, which are sent to the switches from the controllers. Flow rules have two basics parts. The first part of the rule is the match, which defines which packets are processed by the rule. The second part of the rule is the action, which defines what action is taken. The flow rule matches various portions of the headers of packets that are received at the switches. The action portion of the rule tells the switch whether to change the header, drop the packet, route it out a specific port, flood the packet out all of the ports, or take some other action [20]. This ability to treat each device in a network individually provides a granularity of control that is unprecedented in traditional networks.

The interface from the application layer to the control layer, the northbound interface, has not been standardized. However, ONF has a working group actively exploring options to standardize this interface [22]. Examples of business applications are distributed denial of service (DDOS) protection, intrusion detection, and usage tracking for billing. Business applications allow an enterprise to choose which applications are required and to purchase those that are required.

In a typical SDN implementation, a single controller is communicating with multiple switches and possibly with other controllers of other domains, as shown in Figure 2. One of the drawbacks of centralization is the potential for a single point of failure for the network. Traditional networks detect a failed device and are able to recover due to the distributed nature of the system. To prevent a network failure due to a single device failure, multiple controllers must be implemented in the network. These controllers need to share information to ensure that there is a logically centralized network representation even though the controllers may be physically separated.

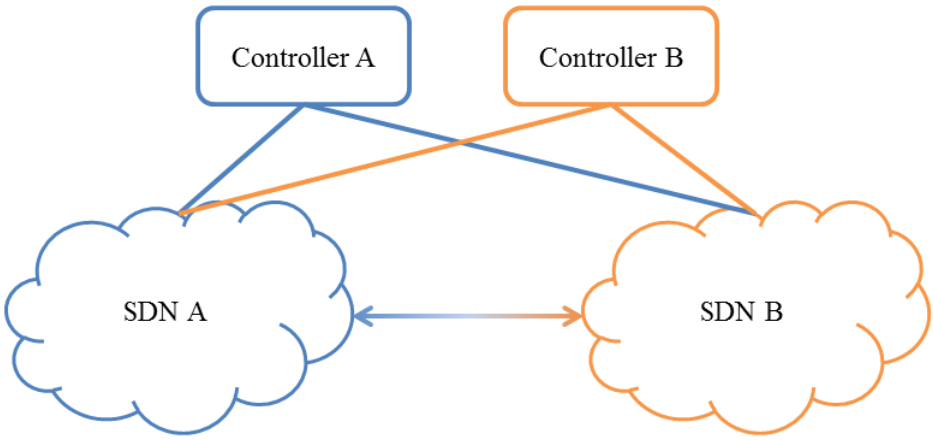

Figure 2. Typical SDNs are configured with multiple controllers to reduce the workload of any single controller or to reduce the impact of a loss of a controller.

2. Operation of the Network

When a packet arrives at the SDN switch, the packet headers are checked against the match portion of the rules that the switch already installed in a flow table. If no match is found, the switch sends the packet to the controller via the OpenFlow interface. Next, the controller determines whether or not a new flow rule needs to be sent to the switch. Typically, the controller will create a new rule. The controller then determines the correct match and the correct action. The controller then sends a flow rule to the switch, which in turn installs this rule in the flow table as a flow entry. Subsequent packets, which match this flow entry, are acted upon correctly based on the newly installed rule. The controller is free to create, modify, and delete flow entries proactively and reactively.

A method to aid the controller in determining flow rules is to develop a graph theoretic model of the network and extract features based on that model. A network is modeled by graph theory as a single entity composed of a set of devices and the connections between these devices. Based on that model, matrix representations of the network can be developed and used for feature extraction. One of the goals of this work is to develop a scheme to extract network features and to control those features.

B. APPLICATION OF GRAPH THEORY TO SDN

Graph theory provides methods to model networks as a set of nodes connected by links. These techniques can be used to model all layers of the SDN stack. The physical topology can be represented by graphs that describe which nodes communicate directly with other nodes, and the networking topology can be described by similar graphs that account for network traffic. Once this model has been developed, it can be analyzed to determine features of the network, such as nodal centrality and severity of congestion [23]. This analysis will aid the controller in the development of flow rules to maximize performance and minimize congestion.

1. Graph Theory

Graph theory is used to model interconnected objects. These interconnected objects can range from neurons in the brain to computers on a network. One of the main strengths, but also a drawback, of graph theory is that the model generated using standard graph techniques is much more abstract than the network being modeled. For instance, the communication between a client and server on the Internet is accomplished by many machines that run numerous algorithms to ensure that the web page requested by the client is properly displayed in the client's web browser. In graph theory, these complexities are reduced to nodes and links. The benefit of this analysis is that it is simpler; the drawback is that assumptions made when reducing complexity may be incorrect. These incorrect assumptions can lead to poor results.

Modeling interconnected devices requires three types of objects: nodes, links, and link weights. Nodes are the objects that are being connected by links. Link weights describe a feature of the link between nodes. The definition of link weight will vary from implementation to implementation. The robustness of a communication channel can be modeled by defining the link weights as a function of a measurable quantity. An example is the signal-to-noise ratio (SNR) of a wireless link or the utilization of a wired link [24]. For an undirected graph G, where $G = (N, L, W)$, N is the set of nodes that are connected by the set L of links with weights W. Undirected graphs are graphs in which

the links do not indicate a direction but simply indicate a connection. Directed graphs have links that indicate direction and each direction can have independent link weights.

In this work, undirected graphs are used and composed of nodes represented by SDN switches and links that represent the communication paths between them, which in this case are Ethernet cables. The link weights are determined by the data rate between the switches. The controller is able to maintain knowledge about the links between switches by querying the switches for this information. It is also able to maintain information about the data rate by periodically requesting this information from the switches.

To maintain and analyze the topology of the SDN, a matrix representation of the network is required. In graph theory, a network topology can be represented by an adjacency matrix $\mathcal{A}$ [25]. The adjacency matrix is a $n \times n$ matrix where n is the total number of nodes in the graph G. For an unweighted, undirected graph, each element in the i^{th} row and j^{th} column e_{ij} in $\mathcal{A}$ is set to one as given by [25]

$$\mathcal{A}_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in L \\ 0 & \text{otherwise} \end{cases}. \tag{2.1}$$

For a weighted, undirected graph, the adjacency matrix is obtained by assigning link weights w_{ij} according to [25]

$$\mathcal{A}_{ij} = \begin{cases} w_{ij} & \text{if } e_{ij} \in L \\ 0 & \text{otherwise} \end{cases}. \tag{2.2}$$

The degree of a node is defined as the sum of the link weights of the links attached to a node. The degree matrix is defined as [25]

$$D_{ij} = \begin{cases} \sum_{k=1}^{n} w_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{2.3}$$

If the graph is unweighted, the diagonal of the degree matrix equals the degree of each node in the graph. The degree matrix and adjacency matrix can be combined to define the Laplacian matrix $Q = D - \mathcal{A}$, equivalently the Laplacian matrix is given by

$$Q_{ij} = \begin{cases} -w_{ij} \text{ if } e_{ij} \in L \\ \sum_{k=1}^{n} w_{ik} \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}. \quad (2.4)$$

The formation of the Laplacian matrix is not unique. An $n \times n$ permutation matrix P can be used to transform one graph representation to another without changing the underlying structure [25]. For example, given a graph G_1, a new graph can be generated through permutation: $G_2 = P^T G_1 P$. The Laplacian matrix of the new graph G_2 has the same eigendecomposition as the original graph. In other words, the permutation matrix simply maps a set of nodes to another set of nodes, but does not change the topology of the graph. As a result, the first row and column of Q could be any node in graph and not necessarily a node labeled 1 [25].

Normalization is important to make a fair comparison of graphs with different numbers of links and nodes. The Laplacian matrix needs to be normalized in order to compare various graph metrics among different topologies [26]. The normalized Laplacian matrix is defined [27] as

$$Q^{norm} = I - D^{-1/2} A D^{-1/2} = D^{-1/2} Q D^{-1/2} \quad (2.5)$$

$$D_{i,i}^{-1/2} = diag\left(\frac{1}{\sqrt{d_i}} \right) . \quad (2.6)$$

The off diagonal terms of $D^{-1/2}$ remain zero.

Using the above matrices, a model of a SDN can be obtained using a series of nodes, links and link weights. Once a model is developed, it can be analyzed to determine useful characteristics of the network. The characteristics include congestion, underutilization, nodal centrality, and general health of the network. These are all important features that the controller must have to increase the performance of the network.

2. Spectral Graph Theory

Spectral graph theory is a subfield of graph theory that utilizes the eigendecomposition technique to derive characteristics of the modeled network.

Eigendecomposition yields two matrices: the eigenvector matrix and eigenvalue matrix. Eigenvector analysis, which is a part of principal component analysis, is often used for dimensionality reduction [28]. It is also used to find the fundamental frequencies and shapes of vibrating structures. Spectral graph theory uses the concepts of frequencies and shapes to analyze Laplacian matrices [25]. Spectral graph theory attempts to find meaning in the eigenvalues and eigenvectors of adjacency and Laplacian matrices.

Eigenanalysis consists of solving $\mathcal{A}v_i = \lambda v_i$ where $\mathcal{A}$ is an $n \times n$ matrix, v_i is an $n \times 1$ eigenvector of $\mathcal{A}$, and λ_i is a scalar eigenvalue of $\mathcal{A}$ for $i = 1, \ldots, n$. The first step is to obtain the eigenvalues λ_i by solving the equation $\det(\mathcal{A} - \lambda I) = 0$ where I is an $n \times n$ identity matrix. The eigenvectors v_i can then be determined by solving $\det(\mathcal{A} - \lambda_i I)v_i = 0$ [29].

The eigendecomposition can be applied to both the adjacency matrix and the Laplacian matrix. The focus, however, is on the Laplacian because the eigendecomposition provides information from both the degree matrix and adjacency matrix. By adding the degree matrix to the analysis, it makes it possible to quickly order the nodes based on their degree, and this order is reflected in the eigenvalues. It will be shown in the following chapters how the degree of a node is an important factor when determining its proximity to the center of the network, which is used to develop automated methods for the controller to locate the most central nodes. The eigendecomposition of the Laplacian matrix can be rewritten in matrix form as $Q = V \Lambda V^T$ where V is an $n \times n$ matrix of eigenvectors as columns and Λ is a diagonal matrix of eigenvalues. The eigenspace is the vector space spanned by the eigenvectors, and it has been shown to capture many of the characteristics of a graph [30].

The eigenvalues derived for the Laplacian matrix can be used to better understand how the network is constructed and its current health. Any Laplacian matrix will always have at least one eigenvalue that is zero, and all the others are positive because it is positive semi-definite, i.e., the Laplacian matrix is a square, symmetric matrix, $xQx^T \geq 0$

for any $n \times 1$ non-zero vector [25]. The n eigenvalues are can be ordered from zero to largest by

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \lambda_n . \tag{2.7}$$

By using this ordering, the eigenvectors may be referred to as leading or trailing, which are the eigenvectors associated with the largest eigenvalues and the eigenvectors associated with the smallest eigenvalues, respectively.

The number of zero eigenvalues is equal to the number of non-connected subgraphs described by a single Laplacian matrix. Physically, a single network can be divided into two networks that are unable to communicate due to a failed link or node. In the case of a failed device, a single Laplacian matrix can model two separate networks, and this Laplacian matrix will have two eigenvalues that are zero. Specifically, $\text{rank}(Q)$ will be no greater than $n-1$, and $n - \text{rank(Q)}$ equals the number of disconnected subgraphs [25]. The sum of the eigenvalues is the trace of Q [25],

$$\sum_{k=1}^{n} \lambda_k = \sum_{k=1}^{n} Q_{k,k} . \tag{2.8}$$

After normalization, all eigenvalues of Q^{norm} are bounded by $0 \leq \lambda_k \leq 2$, which provides a fair comparison of graphs of different sizes [25].

The algebraic connectivity is defined as λ_2, the second smallest eigenvalue of the Laplacian matrix, and the eigenvector associated with the algebraic connectivity is called the Fiedler vector [31]. Algebraic connectivity provides an important measure of network robustness. The algebraic connectivity and the Fiedler vector have been used to determine the robustness of a network, and methods to improve robustness by maximizing algebraic connectivity have been widely documented in the literature. Large algebraic connectivity has been shown to be correlated with well-connected graphs [25], better performance when using distributed algorithms [32], and reduced bottlenecks in computer networks [33]. As the algebraic connectivity approaches zero, the graph splits into two subgraphs, which are sparsely connected. The work reflected in the literature has mainly focused on algebraic connectivity, but not on all of the other eigenvalues, which contain important information.

The eigenvectors are equally important to the analysis of graph theoretic matrices and the underlying real-world networks. The eigenvectors of Q are mutually orthogonal. The sum of the elements of any eigenvector is zero except for the eigenvector associated with the zero eigenvalue [25]. In spectral graph theory, the eigenvector associated with the zero eigenvalue is typically denoted by a vector with elements $v^i_{\lambda=0} = 1/\sqrt{n}$ for $i = 1:n$.

The Fiedler vector along with the eigenvectors associated with the third and fourth eigenvalues can be used to create a three-dimensional view of the network [34]. In Figure 3, a simple graph that is undirected and unweighted is shown in which the x, y, and z coordinates are the second, third, and fourth eigenvectors, respectively. The three dimensional shape will change as the link weights change.

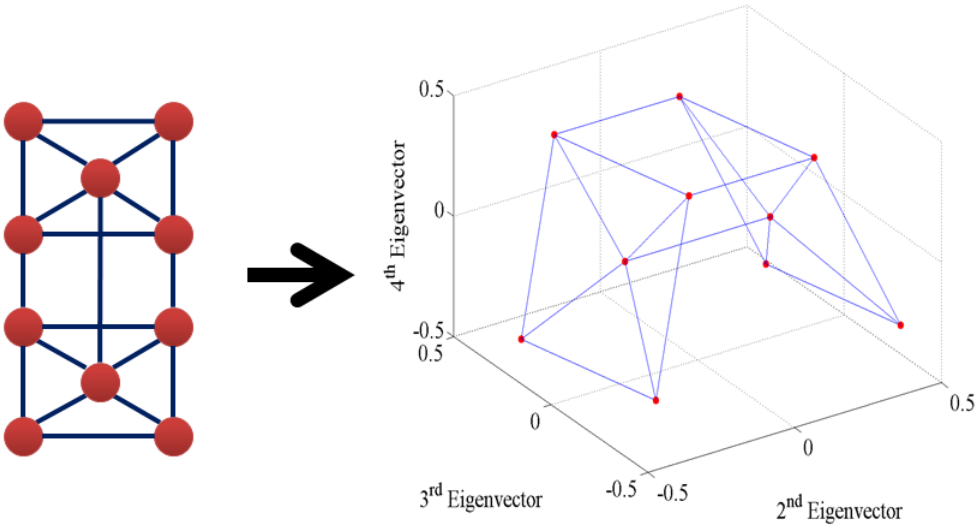

Figure 3. The three trailing eigenvectors of the Laplacian matrix are used to represent each node in the network depicted on the left. By using three eigenvectors, the network is represented in three dimensions. The red nodes are connected by the blue links.

Notice in Figure 3 the graph is divided equally between the two halves by the Fiedler vector about zero. The three links that connect the two halves are the links that cross zero on the second eigenvector's axis. The Fiedler vector has been shown to partition graphs into two separate subgraphs [35] by minimizing the number of links that

connect the two halves. This same bisection with more clarity is shown in Figure 4 by showing only the elements of the second eigenvector. The nodes that have more connections have lower eigenvector values, which are a measure of centrality [35].

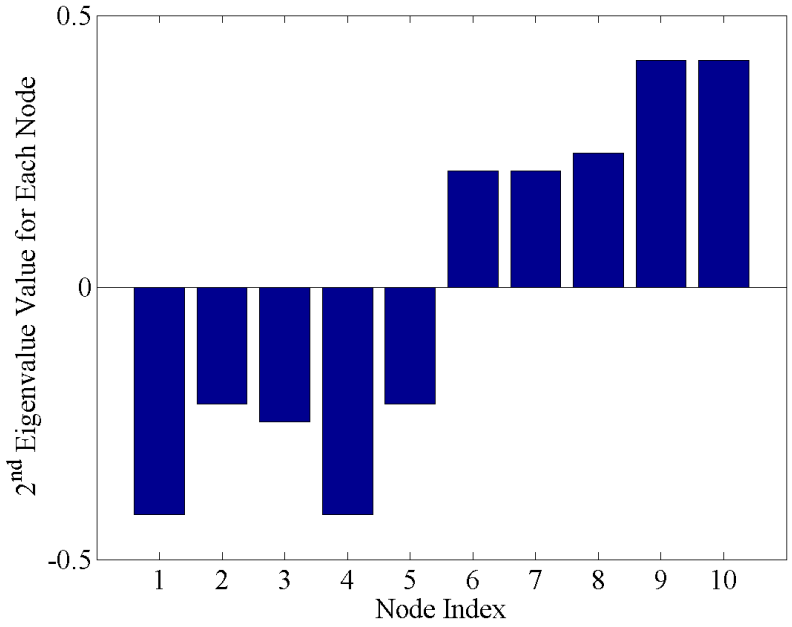

Figure 4. A bar plot of the second eigenvector of the graph from Figure 3 demonstrates how the nodes in a graph are separated into two subgraphs by using the sign of the values of each element in the Fiedler vector.

3. Network Centrality

In social networks, the goal of centrality metrics is to find the person or persons that are most influential within a given community [36], [37]. Researchers in fields outside of social networks have attempted to use these same definitions of centrality to identify characteristics that are important to their research. For example, the simplest definition of centrality is degree centrality c_D, which counts the number of links connected to each node and assigns a node a centrality value based on that count.

Eigenvector centrality is a spectral graph theory metric that is used to determine the most central node in a network. This metric takes into account not only local

information about how well a node is connected, but also how well its neighbors are connected. The eigenvector centrality of node i c_e^i is defined as [38]

$$c_e^i = \frac{1}{\lambda_n} \sum_{j=1}^{n} \mathcal{A}_{ij} c_e^j \tag{2.9}$$

where λ_n is the largest eigenvalue of $\mathcal{A}$. Node $i's$ centrality is now a function of the sum of its neighbors' centrality divided by the largest eigenvalue of the adjacency matrix. In matrix form, the above equation is

$$\mathcal{A}c_e = \lambda_n c_e . \tag{2.10}$$

The $n \times 1$ vector of centrality values is the leading eigenvector of the adjacency matrix corresponding to the largest eigenvalue, which can be seen based on Eqn. (2.10). This metric is used with undirected graphs because it provides a simple method to determine centrality based on network wide information. The drawback to this metric is that it is not tied to a specific cost function that is minimized or maximized. It simply adds the values calculated and assigns that value to the node being analyzed [23]. Eigenvector centrality is used by the Google PageRank algorithm to provide the most relevant pages during web searches [39].

Betweenness centrality c_B is another metric used to quantify the importance of a node to the overall graph [23]. This centrality metric is a measure of the number of times node i is on the shortest path from a source node s to destination node d. The betweenness centrality c_B^i can be calculated using

$$c_B^i = \sum_{s \neq i \neq d \in N} \frac{\rho_{sd}^i}{\rho_{sd}} \tag{2.11}$$

where ρ_{sd}^i is the number of shortest paths that pass through node i form node s to node d, and ρ_{sd} is the total number of shortest paths form node s to node d. In short, betweenness centrality counts the number of times a node is on the shortest path divided by the total number of paths. The result is a measure that quantifies how central a node is in terms of shortest path routing, but this metric may not be significant in terms of load balancing or other routing techniques.

C. COMMUNITY FINDING IN GRAPHS AND NETWORKS

Community finding [35] or cluster finding is a significant research area within graph theory. Its applications range from finding groups within social networks [35] to finding clusters within wireless sensor networks. SDNs require similar algorithms to find communities or clusters within the network to assign switches to controllers and to find the most central nodes, which have the most influence over flows in the network.

Community finding involves dividing a graph into two or more sets of nodes [23], [31], [40], [41], [42]. The graph cut $C(A,B)$ is the number of links that are cut or removed when the set of nodes in graph A and the set of nodes in graph B are separated from one another. The ratio cut or the average cut is defined as [25]

$$\mathcal{R}(A,B) = \frac{C(A,B)}{|A|} + \frac{C(B,A)}{|B|} \tag{2.12}$$

where $|A|$ and $|B|$ is the number of nodes in the set A and B, respectively.

In image segmentation, it has been show that the eigenvectors of the normalized Laplacian matrix are an effective means to divide an image into meaningful segments [41]. The normalized cut can be shown to be related to the normalized Laplacian. The key to the normalized cut is the normalized association $\mathrm{A}_N(A,B)$ defined as

$$\mathrm{A}_N(A,B) = \frac{\mathrm{A}(A,A)}{\mathrm{A}(A,N)} + \frac{\mathrm{A}(B,B)}{\mathrm{A}(B,N)} \tag{2.13}$$

where $\mathrm{A}(A,N)$ is the association defined as the total number of links between the nodes in A and the nodes in N. From the normalized association, the normalized cut, N_{cut}, can be defined as

$$N_{cut}(A,B) = \frac{C(A,B)}{\mathrm{A}(A,N)} + \frac{C(A,B)}{\mathrm{A}(B,N)}. \tag{2.14}$$

The normalized association and normalized cut can be related to each other by

$$N_{cut}(A,B) = 2 - \mathrm{A}_N(A,B). \tag{2.15}$$

The normalized cut is related to the Laplacian as follows [41]

$$\min_x N_{cut}(x) = \min_y \frac{y^T Q y}{y^T D y} \tag{2.16}$$

where y is a vector of binary values that divide the network into two subgraphs. The key insight is that Eqn. (2.16) is now in the Rayleigh quotient form [30]. The Rayleigh quotient form allows for the calculation of the minimum and maximum normalized cut based on the eigenvectors and eigenvalues of the matrices in the numerator and denominator. The vector that minimizes or maximizes the normalized cut is the eigenvector, and the bounds on the minimum and maximum are the eigenvalues associated with the eigenvectors [30].

If y is an eigenvector of the generalized eigenvector problem

$$Qy = \lambda Dy \tag{2.17}$$

and the requirement for y to be a binary value is relaxed to include real values, the minimum is found when y is the second smallest eigenvector of the solution to the generalized eigensystem. Further, it can be shown that the generalized eigenvector problem in Eqn. (2.17) can be converted to the standard eigenvector problem as follows

$$D^{-1/2}QD^{-1/2}y = Q^{norm}y = \lambda y\,. \tag{2.18}$$

The second smallest eigenvector of the normalized Laplacian matrix is the real valued solution to minimize the normalized cut, which is shown in Eqn. (2.18).

The normalized Laplacian matrix and the normalized cut work well in segmentation of images. The definition of the normalized cut and normalized association were defined to ensure that the eigenvectors associated with the smallest eigenvalues provided the segmentation needed. It did not consider the opposite end of the spectrum of larger eigenvalues and associated eigenvectors. The normalized cut attempts to balance the number of links in subgraph A with the number of links cut between A and B. In the case where one is attempting to find the most central node, the correct answer will not be found using the normalized cut because it is attempting to balance two separate variables: the cut links between subgraphs and the links within a subgraph. The most influential node or the most central node will not balance these two measures.

In this research, the ratio cut is used as a cost function, and it is shown that the eigenvectors of the Laplacian matrix can be used to maximize or minimize the ratio cut. Combining this fact with the benefits of the eigenvector centrality from Eqn. (2.9), a

method was developed to represent the network in a form that reveals the network structure and features.

D. CONTROL THEORY

Control theory has been developed to solve the challenges presented by dynamic systems that required feedback to achieve performance goals. SDNs are a multiple-input, multiple-output (MIMO) closed-loop control system. MIMO systems can be difficult to model and control. However, by selecting a small number of inputs and outputs, a simplified model may be developed that can be used to determine observability, controllability, and stability [43]. This foundational modeling and analysis may be used as a framework to build a SDN control scheme.

In traditional control theory optimization problems, two properties must be shown to be present before a controller can be designed. First, the system must be observable; observability requires that the system's states must be determinable from the measurement of outputs [43], [44]. Second, the system must be controllable; controllability requires a controller to be able to drive any state to an arbitrary value [43].

1. State Space Representation

State space representation is one of a number of ways to model a dynamic system. It describes the dynamic system in terms of a set of vectors and matrices. The benefit of the state space representation is that there are proven methods to determine observability, and controllability. A drawback is that many of the proven methods only apply to linear or linearized non-linear systems.

The state space representation is a method to describe how a system will behave based on the system dynamics and given input. The concept of the state of a system is the basis of modern control theory. For a causal system, the state is the vector of initial conditions such that the response of the system at any time t can be uniquely determined from the state at any time $t \geq t_0$ based on the input between t_0 and t. For most physical systems the state is associated with energy storage, such as current in inductors, voltage in capacitors and position or velocity in mechanical systems.

The state space representation is a system of equations, given by [43]

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \tag{2.19}$$

where $\mathbf{A}$ is the state matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the output matrix, and $\mathbf{D}$ is the feedforward matrix; the state vector is $x(t)$, and the derivative of the state vector is $\dot{x}(t)$. The output vector is $y(t)$, and the input vector is $u(t)$.

The development of the state space representation is based on a set of first order differential equations. Clearly, this is not possible for a packet switched network. A packet switched network is non-linear, and it is difficult to formulate differential equations for a switched network without significant assumptions.

2. Observability and Controllability

For a linear time-invariant (LTI) dynamic system as described in Eqn. (2.19), observability is the feature of the state space representation that indicates whether it is possible to determine the state vector based on the output vector. Simply put, if all the state variables are directly measured, the system is always observable. Specifically, the Jacobian matrix is used to determine observability [43], [44]. In LTI dynamic systems, the Jacobian matrix reduces to the observability matrix, defined as

$$O = [C^{\mathrm{T}} \ (CA)^{\mathrm{T}} \ \ldots \ (\mathrm{C}A^{n-1})^{\mathrm{T}}]^{\mathrm{T}} \tag{2.20}$$

If O has full rank, the LTI system is observable. For systems with a large number of nodes, the observability matrix can become quite large, and it can become computationally hard to determine the rank of the matrix. In addition, the state matrix and the output matrix can change over time. The solution is particularly hard in this case because two equations may be independent at one point, but then become dependent as the system changes [44]. All SDNs are able to calculate all link data rates because they make measurements at all switches, but requesting measurements from all switches is results in redundancy. Observability in a SDN is determined by finding the minimum number of measurements required to fully describe the state.

Controllability is the second of the two requirements, and controllability is assured if the controllability matrix has full rank [43]. Controllability is a feature of the state space model that indicates whether it is possible to drive all states to an arbitrary value based on the input $u(t)$ [43]. In the terms of the matrices in Eqn. (2.19), the controllability matrix C is defined as

$$C = [B\ AB\ A^2B\ \cdots\ A^{n-1}B] \tag{2.21}$$

Similar to the observability matrix, the controllability matrix can become quite large as the number of nodes in the network grows. Again, the state and input matrices can be a function of time. Finding the correct input is a key problem in most control theory research. For a SDN, the input is the amount of traffic that is generated by the connected hosts, and this traffic is not controlled by the SDN controller. The result is that the problem is not based around controlling the input, but given an input how does the SDN controller route the traffic to maximize performance and minimize congestion.

3. State Observer

The state space formulation from Eqn. (2.19) can be used to develop the state observer, which is used in dynamic system control to estimate the state of the network in a noisy environment. The noise could come from the system or from the measuring device or, as in most cases, both. By discretizing Eqn. (2.19) a more general set of equations can be obtained as

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} + \mu_{k-1} \\ y_k &= Cx_k + Du_k + \eta_k \end{aligned} \tag{2.22}$$

where μ_{k-1} is the system process noise at time step $k-1$, and η_k is the measurement noise at time step k [45]. This updated model now includes noise, which will prevent the calculation of the state deterministically. The state must be estimated to provide the state controller with the best possible information with which to determine input required.

a. Kalman Filter

A Kalman filter is an optimal algorithm to estimate the current state based on the previous state and the current measurement. It is provably optimal in the case where the

state can be modeled with a set of linear equations, and the noise can be modeled with a Gaussian probability density function (PDF). The goal is to recursively estimate x_k using the current measurement z_k which is defined as

$$z_k = f_k\left(x_k, \eta_k\right). \tag{2.23}$$

Specifically, an accurate estimate of x_k should be based on all the previous measurements up to time k, $z_{1:k} = \left\{z_i, i = 1, \ldots, k\right\}$ [45].

This problem can be reduced to determining the probability density function (PDF) that provides the probability $p\left(x_k \mid z_{1:k}\right)$ that the state vector is a specific value given all of the measurements. At each time step k, this PDF is updated to include the next measurement. Both the optimal and sub-optimal algorithms both use a recursive process to calculate $p\left(x_k \mid z_{1:k}\right)$. The first step is to predict x_k based on the state space model in Eqn. (2.22), and then update the prediction based on the current measurement.

The Kalman filter provides a process to optimally estimate the state of a dynamic system given that the system can be modeled with a linear set of equations and the noise is modeled as a Gaussian random variable. However, in many cases these two assumptions cannot be made simultaneously. In these cases, a suboptimal algorithm must be selected; the most common suboptimal algorithms are the extended Kalman filter (EKF), approximate grid-based methods, and particle filters [45]. EKF and approximate grid-based methods are not good fits for a SDN state estimator because too many assumptions are required to effectively use those. Particle filters provide the greatest flexibility in dealing with the non-linear state equations and non-Gaussian noise.

b. Particle Filter

Particle filters are a specific type of state estimators that are based on a Monte Carlo simulation [46]. Particle filters follow the same estimate and update process as the Kalman filter, but the method they use is based on the selection of particles from a random population and then each particle is given a weight to determine the most likely state given the current measurement. Particle filters were developed specifically for

systems that cannot be linearized, have non-Gaussian noise, and must be calculated in real-time [45].

The scheme starts with the previous set of particles that were used to estimate x_{k-1} [45]. New particles are generated by updating each of these particles using the non-linear state model. This process results in an updated set of particles x_k and updated observations y_k. Next, each particle is assigned a weight based on a given PDF. In many cases a Gaussian distribution is acceptable. If that is the case, the weight for particle i at time k is assigned by

$$w_k^i = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(z_k - y_k^i)^2 / 2\sigma^2} \tag{2.24}$$

where σ is the standard deviation of the system noise. These particle filter weights are then normalized to ensure that a PDF is obtained for all of the particles. The scheme is depicted in Figure 5.

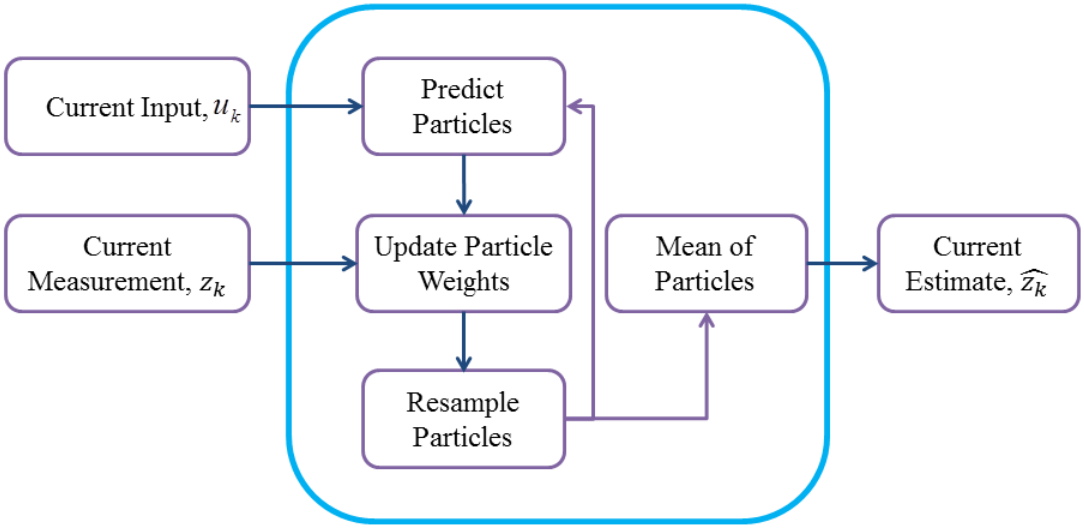

Figure 5. The particle filter process is demonstrated contained within the blue box. The first stage predicts the particle values based on the prior set of particles and the current input to the system. Next, the particle weights are updated using the current measurement. Finally, the particles are resamples and the mean of the new sample is the current estimate.

From this new PDF, a new set of particles is selected based on the weights assigned in Eqn. (2.24). By randomly sampling from this new PDF, the particles with larger weights are more likely to be chosen. Based on this new set of particles, one can estimate the state, which is typically done by finding the mean of the new set of particles. The process will start over using this set of particles and a new measurement [45].

4. State Controller

The state controller is responsible for taking the state estimate from the observer and determining the necessary control inputs to move the state towards the desired value. In a SDN, the controller is unable to control the input to the network, which is the offered traffic. It is able to control the flow of traffic that is generated by the hosts. In the Google B4 network [3], the controllers were able to control both end hosts and the network flows. That is not the scenario in this work. The problem is more difficult and more general if the controller is unable to control the end hosts' offered traffic.

In summary, SDN is a new networking technology that provides network administrators with greater visibility into the behavior of the network and control over those behaviors than in the past. There is a great deal of flexibility given to the network administrator to operate the network. Graph theory and spectral graph theory are two tools that may help identify network features. Community finding is an application of graph theory and spectral graph theory, which allows for the determination of natural partitions in the network. Finally, control theory provides many concepts and terminology that can be adopted by SDN applications to develop a closed-loop control framework.

III. DUAL-BASIS ANALYSIS AND ITS APPLICATION IN IDENTIFYING NETWORK BEHAVIOR

To maintain control over any network, one must be able to describe the behavior of the network in both a static topological sense and a dynamic traffic-aware sense. The goal of this chapter is to describe the preliminary analytical work that allows network controllers to more efficiently control these networks.

Spectral graph theory is used to develop the dual-basis representation to help determine nodal centrality and connectivity based on current network conditions. Following the dual-basis analysis, the development of a state observer and state controller for a SDN will be presented. The observer implements a state estimator that uses a non-linear state model with a non-Gaussian noise model. The SDN controller may implement any network routing algorithm, but not all nodes must implement this algorithm. As shown in [2], updating routes at a small number of nodes may be sufficient to improve performance. The dual-basis analysis is proposed to be the method to identify these nodes that provide the maximum increase in performance.

A. SPECTRAL GRAPH ANALYSIS TO IDENTIFY NETWORK FEATURES

Given an $n \times n$ matrix Q, it can decomposed as

$$Q = V \Lambda V^T \tag{3.1}$$

where V is an $n \times n$ matrix containing the right eigenvectors as columns, V^T is an $n \times n$ matrix containing the left eigenvectors as columns, and Λ is an $n \times n$ matrix with the eigenvalues along the main diagonal [25]. Together the bases formed by the column vectors of V and V^T are known as the dual-basis representation [47]. The following derivation shows how the dual-basis representation of the Laplacian can be used to optimize the ratio cut from Eqn. (2.12).

The ratio cut is a standard metric used in graph theory to find communities or partitions in a graph [41]. The dual-basis analysis of the Laplacian matrix is a method to find the optimal ratio cut of a graph. This optimization leads to the observation that one

can use this approach to obtain a set of metrics based on the dual-basis analysis to find the principal nodes in the network.

1. Optimization of Ratio Cut Using Rayleigh Quotient

The ratio cut is minimized by minimizing the cut between two subgraphs and maximizing the number of nodes in each subgraph. The opposite is true to maximize the ratio cut. From Eqn. (2.16) and based on the derivation in [41], the ratio cut can be put into Rayleigh quotient form; the eigenvectors of the Laplacian matrix can then be used to determine the optimal solution to the ratio cut problem.

By letting $k = |A|/|N|$ and $1-k = |B|/|N|$, the ratio cut can be rewritten as

$$\mathcal{R} = \frac{C(A,B)}{\frac{|A|}{|N|}|N|} + \frac{C(A,B)}{\frac{|B|}{|N|}|N|} = \frac{C(A,B)}{k|N|} + \frac{C(A,B)}{(1-k)|N|}. \tag{3.2}$$

The cut can now be expressed as a function of the Laplacian matrix Q

$$\mathcal{R} = \frac{(\mathbf{1}+x)^T Q(\mathbf{1}+x)}{k|N|} + \frac{(\mathbf{1}-x)^T Q(\mathbf{1}-x)}{(1-k)|N|} \tag{3.3}$$

where x is an $n\times 1$ vector with elements of ± 1 and $\mathbf{1}$ is an $n\times 1$ vector of 1's.

By combining the terms, Eqn. (3.3) becomes

$$\mathcal{R} = \frac{\left(\mathbf{1}^T Q\mathbf{1} + x^T Q\mathbf{1} + \mathbf{1}^T Qx + x^T Qx\right)(1-k) + \left(\mathbf{1}^T Q\mathbf{1} - x^T Q\mathbf{1} - \mathbf{1}^T Qx + x^T Qx\right)k}{(1-k)k|N|}. \tag{3.4}$$

By expanding further, grouping like terms and simplifying, one obtains

$$\mathcal{R} = \frac{x^T Qx + \mathbf{1}^T Q\mathbf{1} + (1-2k)\mathbf{1}^T Qx}{(1-k)k|N|}. \tag{3.5}$$

Substituting $\alpha = x^T Qx$, $\beta = \mathbf{1}^T Qx$, and $\gamma = \mathbf{1}^T Q\mathbf{1}$ into Eqn. (3.5) results in

$$\mathcal{R} = \frac{(\alpha+\gamma) + 2(1-2k)\beta}{(1-k)k|N|}. \tag{3.6}$$

Adding and simplifying leads to

$$\begin{aligned}\mathcal{R} &= \frac{\alpha+\gamma}{(1-k)k|N|}+\frac{2(1-2k)\beta}{(1-k)k|N|}-\frac{2(\alpha+\gamma)}{|N|}+\frac{2\alpha}{|N|}+\frac{2\gamma}{|N|}\\ &= \frac{(\alpha+\gamma)+2(1-2k)\beta-2k(1-k)(\alpha+\gamma)}{k(1-k)|N|}+\frac{2\alpha}{|N|}\\ &= \frac{(1-2k+2k^2)(\alpha+\gamma)+2(1-2k)\beta}{k(1-k)|N|}+\frac{2\alpha}{|N|}\end{aligned} \tag{3.7}$$

Further algebraic manipulation of Eqn. (3.7) yields

$$\begin{aligned}\mathcal{R} &= \frac{\frac{1}{(1-k)^2}}{\frac{1}{(1-k)^2}}\frac{(1-2k+2k^2)(\alpha+\gamma)+2(1-2k)\beta}{k(1-k)|N|}+\frac{2\alpha}{|N|}\\ &= \frac{\left(\frac{k^2+(1-k)^2}{(1-k)^2}\right)(\alpha+\gamma)+2\left(1-\frac{k^2}{(1-k)^2}\right)\beta}{\frac{k}{(1-k)}|N|}+\frac{2\alpha}{|N|}.\end{aligned} \tag{3.8}$$

For $b=k/1-k$ and $\gamma=0$, Eqn (3.8) can be rearranged to obtain

$$\begin{aligned}\mathcal{R} &= \frac{(1+b^2)(\alpha+\gamma)}{b|N|}+\frac{2(1-b^2)\beta}{b|N|}+\frac{2\alpha b}{b|N|}-\frac{2\gamma b}{b|N|}\\ &= \frac{(1+b^2)(x^TQx+\mathbf{1}^TQ\mathbf{1})}{b|N|}+\frac{2(1-b^2)(\mathbf{1}^TQx)}{b|N|}+\frac{2bx^TQx}{b|N|}-\frac{2b\mathbf{1}^TQ\mathbf{1}}{b|N|}.\end{aligned} \tag{3.9}$$

This result can be simplified by expanding and grouping like terms as follows

$$\begin{aligned}\mathcal{R} &= \frac{(\mathbf{1}+x)^TQ(\mathbf{1}+x)}{b|N|}+\frac{b^2(\mathbf{1}-x)^TQ(\mathbf{1}-x)}{b|N|}-\frac{2b(\mathbf{1}-x)^TQ(\mathbf{1}+x)}{b|N|}\\ &= \frac{\left[(\mathbf{1}+x)-b(\mathbf{1}-x)\right]^TQ\left[(\mathbf{1}+x)-b(\mathbf{1}-x)\right]}{b|N|}.\end{aligned} \tag{3.10}$$

By setting $y=\left[(\mathbf{1}+x)-b(\mathbf{1}-x)\right]$, the ratio cut simplifies to

$$\mathcal{R}=\frac{yQy}{b|N|}. \tag{3.11}$$

To finally present in the Rayleigh quotient form, the denominator of Eqn (3.11) must be shown to be equal to y^Ty. Since $b=|A|/|B|$, it can be shown that:

$$b|N| = b\left(|A| + |B|\right) = |A| + b|A|, \tag{3.12}$$

$$b|A| = \frac{|A|}{|B|}\frac{|A|}{|B|}|B| = b^2|B|, \tag{3.13}$$

and

$$|A| + b^2|B| = (\mathbf{1}+x)^T(\mathbf{1}+x) + b^2(\mathbf{1}-x)^T(\mathbf{1}-x) = y^T y. \tag{3.14}$$

From Eqn. (3.11) and Eqn. (3.14), the result is

$$\mathcal{R} = \frac{C(A,B)}{|A|} + \frac{C(B,A)}{|B|} = \frac{y^T Q y}{y^T y}. \tag{3.15}$$

This result is similar to that from [41] except matrix D is not included in the denominator; see Eqn. (2.16). With the ratio cut in Rayleigh quotient form, the optimal binary solution is determined by using the leading and trailing eigenvectors of the Laplacian matrix.

2. Example: Optimal Binary Solution to Ratio Cut

To further develop this idea, one must examine how to use the real valued eigenvectors to obtain the maximum or minimum ratio cut. As an example, consider the graph of Internet2 shown in Figure 6 [12], [13]. The ratio cut is minimized when nodes 1 through 17 are assigned to one subgraph and all others are placed in the other. In this case, four links are cut, and the ratio cut is equal to 0.47. This solution is found by assigning all nodes with values less than 0.005 in the second eigenvector to one subgraph and all others to another subgraph. Notice that one could exchange nodes 18 and 17 between the two subgraphs and the ratio cut does not change. There is more than one correct answer to the binary minimization.

When considering the maximization, a similar observation is made. There are four correct answers. In Figure 6, nodes 2, 7, 13, and 16 all have four links, and all will produce the same ratio cut maximization, which is equal to 4.12. All of these nodes are identified in the leading three eigenvectors. Identifying the four correct answers using fewer than four eigenvectors supports the assertion that one must use more than a single eigenvector to achieve a full representation of the most central nodes in the network.

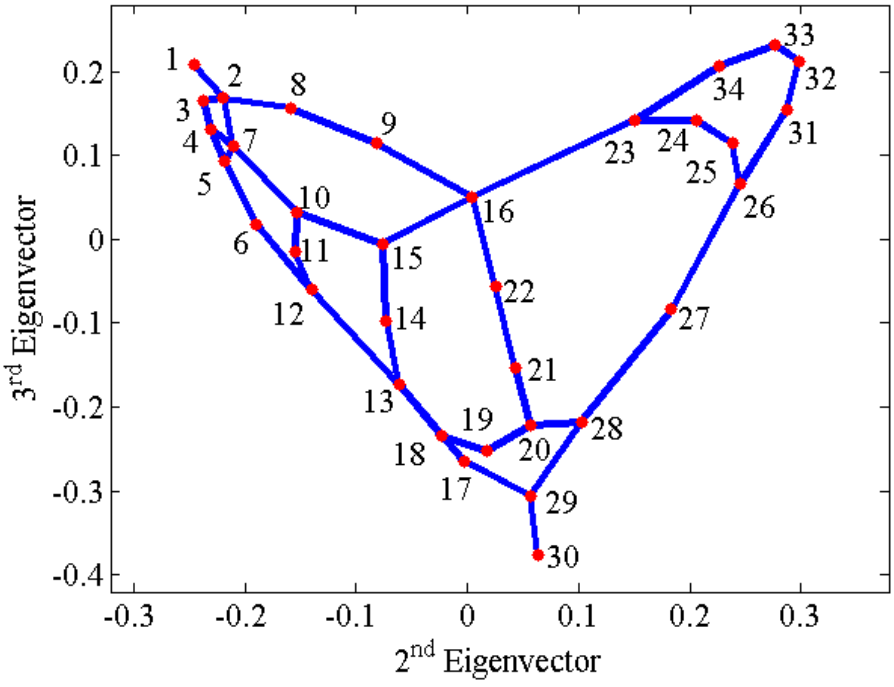

Figure 6. The two-dimensional representation of Internet2 in eigenvector space places the least connected nodes on the edge and the most central nodes in the center. Each red node is a city on Internet2, and the blue links show the connectivity between cities.

For the binary solution, the ratio cut is optimized if the y vector is constrained to include two values because networks are discrete entities. In the development of the dual-basis representation, this constraint is relaxed and the real values are used. The use of the real-valued vectors is consistent with network science research, image processing, and graph theory [23], [25], [41]. The use of real values allows the use of the full range of values, which provides greater specificity when attempting to determine which eigenvectors can be used to identify the most central nodes.

3. Principal Eigenvectors of the Dual-basis

The eigenvector centrality as defined by Eqn. (2.9) does not only take the connectivity of a node into account, but also the connectivity of its neighbors and the neighbors of neighbors. This allowed the analysis of the network as a whole as opposed to focusing too narrowly on a given node or portion of the graph.

The concept of principal eigenvectors is the idea that one can use multiple leading or trailing eigenvectors of the Laplacian matrix to describe network features. An example is Figure 6 in which the two trailing eigenvectors of the Laplacian matrix are used to represent the network. Using these two eigenvectors presents the network in a way that the most central nodes are in the center of the image and the least connected nodes are on the edges of the image. The principal eigenvector approach uses this concept to determine a suitable number of eigenvectors to use to extract the features needed for the users' specific application.

Using the principal eigenvectors of the dual-basis representation leverages all of the benefits of the eigenvector centrality except for the fact that the leading eigenvector of the adjacency matrix will always only contain positive values [23]. The major difference between the eigenvector centrality assignments and the use of the dual-basis analysis is the use of multiple eigenvectors to determine the principal vectors. As defined in Eqn. (2.10), centrality values are assigned based on the leading eigenvector of the adjacency matrix. This definition of centrality is too simplistic to fully capture the centrality of a large network. The image segmentation community recognized that multiple eigenvectors provided a more accurate segmentation of the image over the use of a single eigenvector [41]. By treating the centrality value for each node as a vector, a more complete description of centrality is provided.

Using multiple eigenvectors solves the problem presented by the eigenvector centrality that weights neighbors of the most central node more heavily than others. The result of this weighting skews the centrality of the network to be localized to one section of the graph. In large real-world networks, there is not a single node or section of a graph that can control the entire network. That is why a localized definition of centrality is not sufficient to describe the most central nodes.

The number of principal eigenvectors required to fully describe the centrality of the network is determined by calculating the angles between nodes. The centrality of the network has been fully described when all of the nodes that are near orthogonal to each other are located. This notion of nodal orthogonality will be discussed in future chapters.

Based on the use of multiple eigenvectors and eigencentrality, the dual-basis network representation can be explained and understood.

B. DUAL-BASIS NETWORK REPRESENTATION

In light of multiple correct answers to the maximization of the ratio cut problem provided by the eigenvectors of the Laplacian, one could use all of the eigenvectors of the Laplacian to build an n-dimensional space to characterize the network. The eigenvectors can be ordered according to their associated eigenvalues; the eigenvectors provide n orthogonal vectors that contain one value for each node in the network. The first eigenvalue is always zero and therefore, its eigenvector does not provide any information. The second eigenvalue, algebraic connectivity, is associated with the Fiedler vector, which is a good approximation of the minimum cut or, as shown previously, a means to estimate the minimum ratio cut. As the eigenvalues increase, the associated eigenvectors span a set of vectors that vary between highlighting the most connected nodes and the least connected nodes.

This set of orthogonal vectors is a self-dual basis that is orthogonal with itself. One set provides a range of centrality vectors while the other provides a set of nodal vectors [25]. Each is an important component when considering the static design phase and dynamic monitoring phase of the dual-basis analysis. First, the analysis is developed for a static network and then extended for a dynamic network.

1. Spectral Graph Theory Development of the Dual-basis Representation with Static Link Weights

Simply put, the dual-basis representation is the eigenvector matrix of the Laplacian matrix. Spectral graph theory provides a method to decompose the modeled network into its constituent pieces. The graph cut minimizations and maximizations are examples of a network's constituent pieces. The dual-basis analysis is based on a set of eigenvectors that form two orthogonal bases for the network where the centrality vectors are the columns of V as

$$V = \begin{bmatrix} v_1^1 & \cdots & v_n^1 \\ \vdots & \ddots & \vdots \\ v_1^n & \cdots & v_n^n \end{bmatrix} \tag{3.16}$$

and V^T contains the nodal vectors as

$$V^T = \begin{bmatrix} v_1^1 & \cdots & v_1^n \\ \vdots & \ddots & \vdots \\ v_n^1 & \cdots & v_n^n \end{bmatrix}. \tag{3.17}$$

Additionally, the null space and reachability space are formed by these same matrices. The reachability and null spaces define which nodes are reachable in a routing sense within the network and which are not [48]. The size of the null space n_{null} is determined by the number of zero eigenvalues of the Laplacian matrix, and once sorted according to Eqn. (2.7), the eigenvectors of V^T indicate which nodes are in the null space. The eigenvectors corresponding to the null space are

$$v_i = \begin{bmatrix} \ldots & 1 & 0 & \ldots & 0 \end{bmatrix}^T$$

where the indices of the value 1 are the indices of the nodes in the null space; each eigenvector contains $n-1$ zeros, and one 1. The remaining nodes are contained in the reachability space.

2. Eigencentrality Basis

The eigencentrality basis defines how influential a specific node is at a given eigenvalue. Consider the network shown in Figure 7. Nodes 1 through 6 are representative of an access network. Nodes 7 through 16 are representative of a core network. Node 17 is disconnected from the larger network to demonstrate how a disconnected node behaves in the dual-basis. A three-dimensional representation of the eigencentrality basis of the network from Figure 7 is shown in Figure 8. The third, fourth and fifth eigenvectors are used in the three-dimensional representation because they are the three eigenvectors associated with the three smallest, nonzero eigenvalues.

Each eigencentrality vector is an n-dimensional vector. In this example, the eigencentrality vectors are 17×1; one value is associated with each network node. Plotting the three trailing eigencentrality vectors typically produces a good visual

representation of the network because they place the least connected nodes at the edge and the most connected nodes at the center of the plot [48]. Networks are typically drawn this way; the core of the network is in the center of the diagram, and the access network is at the edge. Any disconnected nodes are placed at the origin, which in Figure 8 is denoted in red.

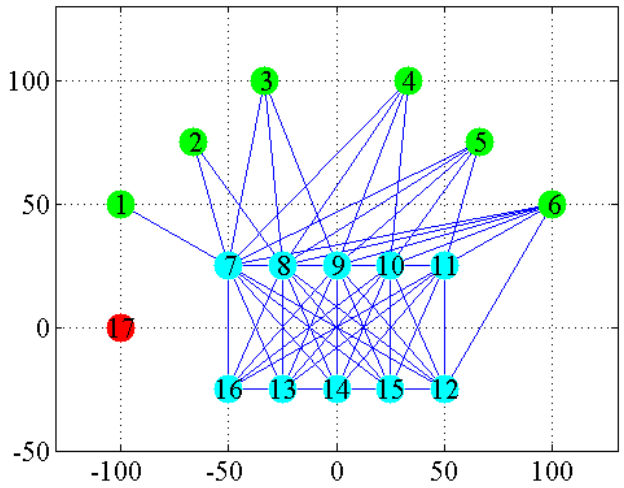

Figure 7. The three types of nodes above are represented by the green access nodes, the blue core nodes, and the one red disconnected node.

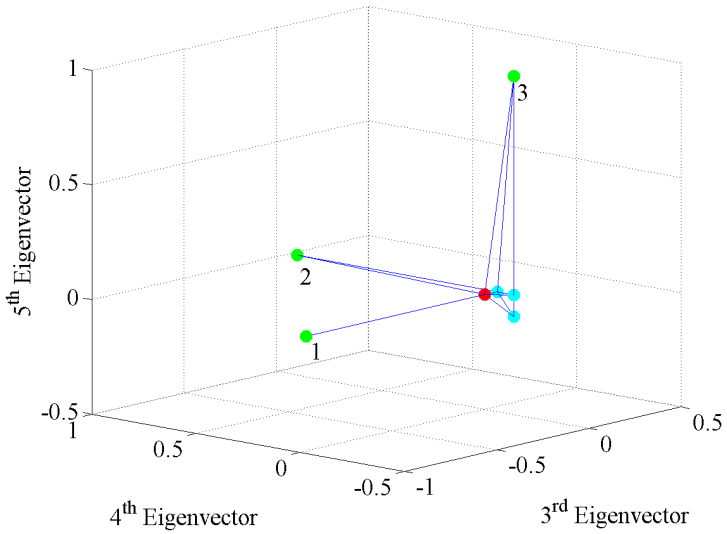

Figure 8. Eigencentrality basis plotted using the three eigenvectors associated with the three smallest, non-zero eigenvalues with nodes 2, 3, and 3 as the least central nodes in the network.

The network graph could be as easily plotted using the three leading eigencentrality vectors associated with the three largest eigenvalues. This representation places the most connected nodes at the edge of the graph and the least connected in the center as shown in Figure 9.

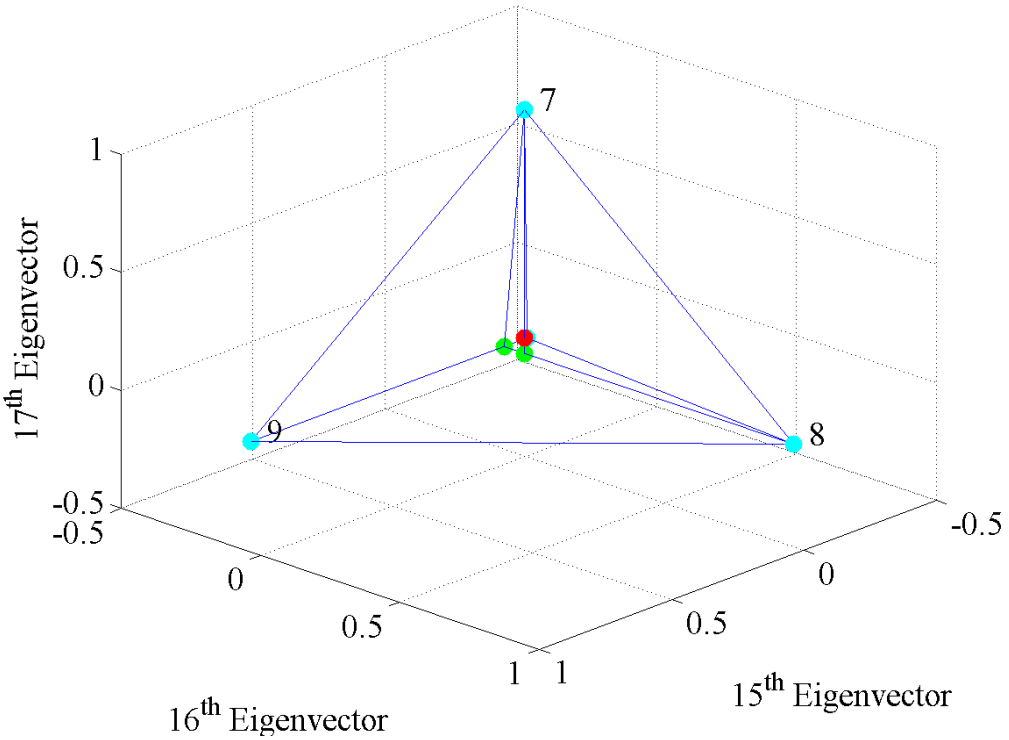

Figure 9. Eigencentrality basis plotted using the three eigenvectors associated with the three largest eigenvalues with nodes seven, eight and nine as the most central nodes in the network

The dual-basis representation reveals how coupled or isolated nodes are from one another. Notice in Figure 8 and Figure 9 that the nodes on the x, y, and z-axes are separated by 90°, which means they are isolated from each other. The result of this dual-basis analysis is that many nodes are orthogonal or near-orthogonal to the others without the use of all n eigenvectors. This means that there is a subset of the total number of eigenvectors that may be used to represent the node's centrality relative to all other nodes.

Laplacian eigencentrality provides a measure of the importance of a node to the network and the impact of its removal. The Laplacian eigencentrality is defined as

$$E_k^j = \sqrt{v_k^{j*} v_k^j} = \sqrt{\left| v_k^j \right|^2} \tag{3.18}$$

where v_k^j is the j^{th} element of the k^{th} eigenvector of the matrix V in Eqn. (3.16) [25]. This value indicates how influential each node is at each eigenvalue. This definition must be extended to include multiple eigencentrality vectors so that each node may be treated as a multi-dimensional vector in the nodal space.

By expanding Eqn. (3.18) to include multiple eigenvectors, the Laplacian eigencentrality is

$$\begin{aligned} E_{k:n}^j &= \sqrt{v_{k:n}^j v_{k:n}^{j\ \mathrm{H}}} \\ v_{k:n}^j &= [v_k^j, v_{k+1}^j, \ldots, v_n^j] \end{aligned} . \tag{3.19}$$

Eigencentrality of node j is now defined as the L_2 norm of the leading $n-k$ values of the $j's$ nodal vector. In addition to the L_2 norm, the angle between node i and j can be calculated by using the dot product

$$v_{k:n}^i \bullet v_{k:n}^j = E_{k:n}^i E_{k:n}^j \cos\theta_{k:n}^{ij} \tag{3.20}$$

where $\theta_{k:n}^{ij}$ is the angle between node i and node j. Using the Laplacian eigencentrality and angles between nodes, the most central nodes can be located.

3. Nodal Basis

The nodal basis is a set of eigenvectors that describe how influential a specific node is across the entire eigenspectrum. The eigenvectors as columns in Eqn. (3.17) are associated with a single node in the network [25]. To demonstrate the nodal basis more clearly, the Laplacian matrix can be calculated using the eigenvectors and eigenvalues. When the matrix V^T is decomposed into individual vectors and related back to Q, the vector form of the degree matrix is

$$D_{i,i} = Q_{i,i} = \left(v_i^1\right)^2 \lambda_1 + \left(v_i^2\right)^2 \lambda_2 + \cdots + \left(v_i^n\right)^2 \lambda_n \tag{3.21}$$

where $D_{i,i}$ is the i^{th} node in the degree matrix, which corresponds to the i^{th} value along the diagonal of Q, and v_i^j is the j^{th} node's value associated with the i^{th} eigenvalue as shown in Eqn. (3.17). Any node's degree is a function of one eigenvector and all eigenvalues as shown in Eqn. (3.21), which demonstrates the reason for the definition of eigencentrality in Eqns. (3.18) and (3.19). The eigencentrality norms disregard the sign of the eigenvalue element and simply use the square of the magnitude when using multiple eigenvectors. This same formulation is seen in Eqn. (3.21) in which the square of the eigenvalue is used, which disregards the sign of the eigenvector element.

For the network in Figure 7, the nodal basis of node 6 has 15 values because there are 15 non-zero eigenvalues; the eigenspectrum of nodes 6 is shown in Figure 10. The nodal basis of node 6 clearly indicates that it has the most influence over λ_8. The response shown in Figure 10 is an example of how the eigendecomposition is a tool to reveal the structure of the network. In this case, node 6 is well isolated from the other nodes, which is demonstrated by the strong response at λ_8 and small responses at all other eigenvalues. This is considered the node decoupling effect demonstrated by the eigendecomposition.

All of the nodes in the example network have similar eigenspectra to the one shown in Figure 10. The shape of the eigenspectrum is unique for each node due to the requirement that the basis vectors are mutually orthogonal. In addition, the eigenspectrum provides information about the number of connected graphs included in a single Laplacian matrix. One can extend the ideas of the eigencentrality and nodal bases to include two spaces that indicate which nodes can be used in legitimate routes in the network and which cannot be included.

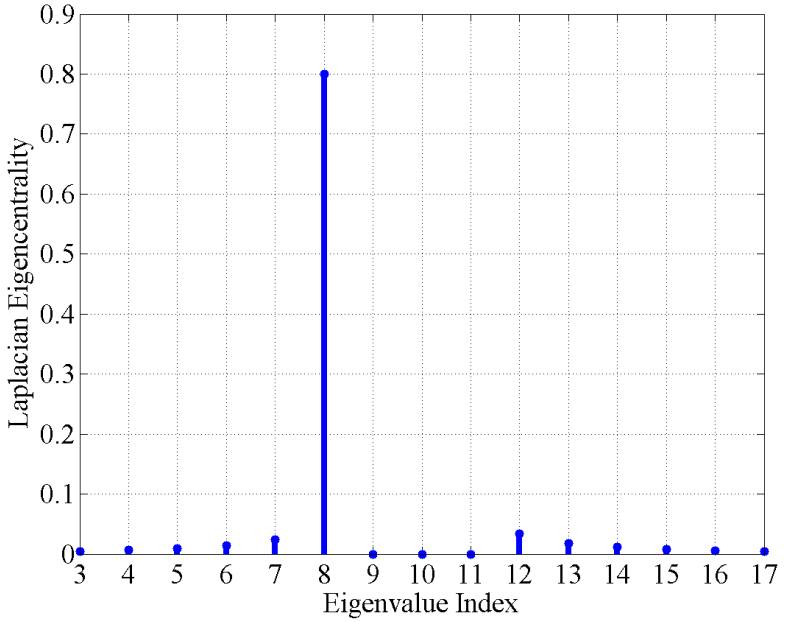

Figure 10. The eigencentrality of node 6 across the eigenspectrum of the static graph in Figure 7 demonstrates that the eigendecomposition reveals the isolation of node 6 from all other nodes in terms of the eigenresponse.

4. Null and Reachability Space

The null space is mathematically and physically interpreted as a part of the solution space or network that is unreachable. There will always be one or more zero eigenvalues of the Laplacian. The eigenvectors corresponding to the zero eigenvalues define the null space of the Laplacian matrix. By examining the elements of the null space eigenvectors, the nodes v_i^j that are unreachable by the rest of the network are indicated by the j^{th} node's value being equal to 1 in Eqn. (3.17). All other values in the null space eigenvectors are zero.

The reachability space contains the remaining eigenvectors that are not in the null space. Once the null space nodes have been identified, the remaining network is guaranteed to have a route from all nodes to all other nodes. The number of non-zero eigenvalues and the size of the reachability space is equal to $\text{rank}(Q)$ [25]. The size of the null space is equal to $n - \text{rank}(Q)$. The size of the null space determines the length of

the vectors in the nodal space; the nodal space eigenvectors will have dimensionality equal to $\text{rank}(Q)$. The eigencentrality basis vectors will always have a length equal to n.

To this point in the chapter, the focus has been on networks that have static links with weights equal to 1. This analysis is valid when considering the topology of the network. When striving to model real-world networks, the interaction between network traffic and network topology must be considered. To add the network traffic to the above analysis, the link weight will be allowed to vary, which allows the model to account for network traffic. Large network models that include varying link weights do not lend themselves to analysis with closed-form solutions. In a few specific cases, closed-form solutions for the eigenvalues and eigenvectors can be found. These closed-from solutions provide a transition from a static dual-basis analysis to simulating large, dynamic networks.

C. DYNAMIC LINK WEIGHT ANALYSIS USING THE DUAL-BASIS REPRESENTATION

The following analysis demonstrates the dynamics of mesh networks when the link weights are allowed to change. A mesh network is one in which all nodes are connected to all other nodes—similar to the core network from Figure 7. The objective is to demonstrate that dynamic, time-varying link weights are reflected in the eigenvectors and eigenvalues. Increasingly complex networks are analyzed in the following sections. The complexity quickly outweighs the usefulness of this approach because the closed-form solutions are too long to show here. Even though these are simple network graphs, the equations show patterns that can be useful in understanding more complex systems.

1. Closed-Form Solution for Algebraic Connectivity for Mesh Networks

The first step to find the closed-form solution of dynamic graphs is to find the eigenvalues of a static mesh network, which in the context of graph theory is called a complete graph. The Laplacian matrix of a mesh is constructed as

$$Q_{i,j}(u_1,\ldots,u_{n-1},n)=\begin{bmatrix} -1\times\sum_{\substack{k=1\\ i\neq k}}^{n} w_{1,k} & -1+u_1 & \cdots & -1+u_{n-1} \\ -1+u_1 & -1\times\sum_{\substack{k=1\\ i\neq k}}^{n} w_{2,k} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ -1+u_{n-1} & \cdots & \cdots & -1\times\sum_{\substack{k=1\\ i\neq k}}^{n} w_{n,k} \end{bmatrix} \tag{3.22}$$

where u_i is the link utilization, and $w_{i,k}$ is the link weight of node i's k^{th} link. The link utilization u_j is related to the weights $w_{i,k}$ by

$$w_{i,k}=-1+u_j,\ -1\le w_{i,k}\le 0.. \tag{3.23}$$

The weights w_{ij} are defined as

$$w_{ij}=\left(1-\psi^{ij}\frac{R^{ij}(k)}{R^{ij}_{\max}}-\left(1-\psi^{ij}\right)\frac{t^{ij}_{\text{RTT}}(k)}{t_{\max}}\right)\left(1-\psi^{ji}\frac{R^{ji}(k)}{R^{ji}_{\max}}-\left(1-\psi^{ji}\right)\frac{t^{ji}_{\text{RTT}}(k)}{t_{\max}}\right) \tag{3.24}$$

where $t^{ij}_{\text{RTT}}(k)$ is the round-trip time from node i to node j at time k, $t_{\max}$ is the maximum allowable round-trip time before that link is considered unusable, $R^{ij}(k)$ is the measured data rate from node i to node j at time k, $R^{ij}_{\max}$ is the link bandwidth in bits per second (bps) of the link from node i to node j and ω^{ij} is a weighting factor to bias towards one metric or the other.

For a mesh network, the characteristic equation is [25]

$$(-1)^n\lambda(\lambda-n)^{n-1}=0\,. \tag{3.25}$$

The eigenvalues of this equation are: $\lambda_1=0,\ \lambda_i=n$ for $i=2,\ldots,n$. For any complete graph, the algebraic connectivity will always be n [25]. From this foundation, the next step is to determine the effect of dynamic link weights on the eigenvalues.

a. Mesh Network with One Dynamic Link Weight

To begin examining changes in how link weights affect the results of the dual-basis analysis, a single link was assigned a varying link weight. By varying u_1 between 0 and 1 and setting $u_2,\ldots,u_{n-1}=0$, the characteristic equation was determined to be

$$(-1)^n\lambda(\lambda-n)^{n-2}(\lambda-n+2u_1)=0. \tag{3.26}$$

The solution to the above equation results in three distinct eigenvalues: $\lambda_1=0,\ \lambda_2=n-2u_1,$ and $\lambda_i=n$ for $i=3,\ldots,n$. In this case, the algebraic connectivity is always $\lambda_2=n-2u_1$ and is bound by $n-2\le\lambda_2\le n$.

This result indicates that the eigenvalues and the corresponding eigenvectors provide a method to reveal the dynamics in such a way that specific features can be isolated. Dynamic link weight behavior can be isolated to a small number of eigenvectors by using the dual-basis representation. This is not a proof, but it does provide confidence that one can decouple nodal interaction using the eigendecomposition approach. If the link weight behavior can be decoupled from node to node, then the centrality basis and nodal basis are relevant in both static and dynamic graphs.

b. Mesh Network with Two Dynamic Link Weights

The next examined was a mesh network with two links with dynamic link weights. Each link weight is not necessarily equal. When $0\le u_1\le 1,\ 0\le u_2\le 1,\ u_3,\ldots u_{n-1}=0,$ and $u_1\ne u_2$, the characteristic equation is

$$(-1)^n\lambda(\lambda-n)^{n-3}\left[\lambda^2+\lambda(2(u_1+u_2)-2n)+3u_1u_2-2n(u_1+u_2)+n^2\right]=0. \tag{3.27}$$

There are four distinct eigenvalues:

$$\begin{aligned}
&\lambda_1=0,\\
&\lambda_2=n-u_1-u_2+\sqrt{u_1^2+u_2^2-u_1u_2},\\
&\lambda_3=n-u_1-u_2-\sqrt{u_1^2+u_2^2-u_1u_2},\\
&\lambda_i=n \text{ for } i=4,\ldots,n.
\end{aligned} \tag{3.28}$$

The algebraic connectivity is always $\lambda_2=n-u_1-u_2-\sqrt{u_1^2+u_2^2-u_1u_2}$. This means that the algebraic connectivity is bound by $n-3\le\lambda_2\le n$. At this point in the analysis, adding

additional dynamic links resulted in equations that are too long to show here, but closed-form solutions do exist.

There is some interaction between the second and third eigenvalues as shown in Eqn. (3.28) because the second and third eigenvalues are a function of the two weighted links. This result is expected because there are now two links that are allowed to vary and these links are connected to the same node. Hence, nodal centrality should be coupled to both weighted links. The isolation still holds for all of the other eigenvalues. They are not affected by these varying link weights. This pattern holds as larger numbers of links are allowed to vary. The number of varying eigenvalues is equal to the number of links that vary.

c. Mesh Network with Dynamic Link Weights to One Node

In this scenario, all the links to one node are allowed to vary, but two link weights are unequal while the remaining are all equal. When $0 \le u_3 = \ldots = u_{n-1} \le 1,\ 0 \le u_2 \le 1,$ and $0 \le u_1 \le 1$, the characteristic equation is

$$\begin{aligned}(-1)^n \lambda(\lambda - n + u_3)^{n-4}[&\lambda^3 \\ &+ \lambda^2(2(u_1+u_2-u_3)+n(u_3-3) \\ &+ \lambda(n^2(3-2u_3)+4n(u_3-u_1-u_2)+u_3 n(u_1+u_2)-u_3(u_1+u_2)+3u_1u_2 \\ &+ n^3(u_3-1)+n^2(2u_1+2u_2-2u_3-u_1u_3-u_2u_3)+n(u_1u_2+u_1u_3+u_1u_2u_3-3u_1u_2)] = 0\end{aligned} \quad . \quad (3.29)$$

In this case, there are five distinct eigenvalues, and they are 0, $n-u$ and the solutions to Eqn. (3.29). There are three algebraic solutions to the third order Eqn. (3.29), but the solution is too long to include here. This demonstrates that there are closed-form solutions for the algebraic connectivity for arbitrarily large networks. Again, the pattern of varying link weights and eigenvalues continues. The eigenvalues clearly indicate that there are varying link weights in the network.

d. Mesh Network with a Node Connected by Two Links

The next set of results determines the closed-form solution of the algebraic connectivity for a mesh network with an additional node connected by two links. If $0 \le u_1 \le 1,\ 0 \le u_2 \le 1,$ and $u_3 = \ldots = u_{n-1} = 1$, the model is a mesh network with two links

to a single node with unbalanced traffic—similar to the network in Figure 7, but using only the core network and node 2. The characteristic equation is

$$\begin{aligned}(-1)^n \lambda(\lambda-n+1)^{n-4}[&\lambda^3 \\ &+\lambda^2(2(u_1+u_2-1)-2n) \\ &+\lambda(n^2+4n(1-u_1-u_2)+n(u_1+u_2)+3u_1u_2-u_1-u_2) \\ &+n^2(2u_1+2u_2-2-u_1-u_2)+n(u_1+u_2-2u_1u_2)]=0\end{aligned} \quad . \tag{3.30}$$

Again, the eigenvalues are 0, $n-1$, and the solutions to the third order Eqn. (3.30).

If the two links are traffic balanced, i.e. $0 \le u_1 = u_2 \le 1$, the characteristic equation becomes

$$(-1)^n \lambda(\lambda-n+1)^{n-4}(\lambda-n+u)[\lambda^2+\lambda(3u-n-2)+2n-2un]=0\,. \tag{3.31}$$

This case provides five distinct eigenvalues:

$$\begin{aligned}&\lambda_1 = 0, \\ &\lambda_2 = \frac{n}{2}-\frac{3u}{2}+1-\frac{1}{2}\sqrt{(n^2+n(2u-4)+9u^2-12u+4}, \\ &\lambda_3 = \frac{n}{2}-\frac{3u}{2}+1+\frac{1}{2}\sqrt{(n^2+n(2u-4)+9u^2-12u+4}, \\ &\lambda_4 = n-u \\ &\lambda_i = n-1 \text{ for } i=5,\ldots,n.\end{aligned} \tag{3.32}$$

Therefore, the algebraic connectivity will always be

$$\lambda_2 = \frac{n}{2}-\frac{3u}{2}+1-\frac{1}{2}\sqrt{(n^2+n(2u-4)+9u^2-12u+4}\,, \tag{3.33}$$

and

$$\lim_{n\to\infty} \lambda_2 = 2-2u\,. \tag{3.34}$$

The details of the algebraic manipulation to obtain the limit in Eqn. (3.34) are shown in Appendix A.

This result suggests that eigendecomposition is an effective way to examine network behavior because it reveals the structure of the network is such a way that the nodal behavior is isolated to the extent possible, and it has a natural transition from graph representation to matrix representation to dual-basis representation.

e. Mesh Network with Balanced Traffic to One Node

Finally, if all traffic is balanced to a single node in the mesh network, $0 \le u_1 = \cdots = u \le 1$, the characteristic equation simplifies significantly:

$$(-1)^n \lambda(\lambda - n + u)^{n-2}(\lambda - n + un) = 0 . \tag{3.35}$$

From this result, the eigenvalues are: $\lambda_1 = 0$, $\lambda_2 = n - nu$, and $\lambda_3 = n - u$; $n - u$ is repeated $n - 2$ times, and the algebraic connectivity is $\lambda_2 = n - nu = n(1-u)$. Therefore, if n is large and u approaches 1, the algebraic connectivity approaches $Q_{1,1}$ because

$$Q_{1,1} = -1 \times \sum_{\substack{k=j \\ j \neq k}}^{n} w_{1,k} = -1(n-1)(-1+u) = (n-1)(1-u) . \tag{3.36}$$

From these equations, one can begin to understand how the dynamic link weights affect the eigenvalues. However, the eigenvectors are the second half of the story. The combination of the two is key to understanding the network as a whole. If the SDN controller is to control the network as a whole, it must use a representation to track network behavior that isolates the network behavior to the extent possible. As shown in the previous sections, the eigenvalues can be used to isolate the dynamic link behavior because the change in link weight on a link can be isolated to a single eigenvalue. That means that the behavior that is modeled by the link weight can be tracked using the eigenvalues. This is important in a graph because the controller needs a method to isolate and locate behavior in the network, such as congestion and underutilization. The network behavior represented by the eigenvalues can be considered to be the state of the network.

2. Closed-Form Solution for the Fiedler Vector

Once the eigenvalues are known, the eigenvectors can be determined. The closed-form solution is provided for a simple case, but it can be extended to more complex cases as well. The goal here was to provide a closed-form solution for a single, simple case and illustrate more complex cases through simulation. These simulation results are shown in the next section. The simple case is one in which all of the links to a node are allowed to vary, but all of the weights are the same value.

In this case, one eigenvector is different from the rest—the Fiedler vector v_2. To solve for the elements of v_2, a set of n linear equations must be solved based on the eigenvalues determined in Eqn. (3.35). The first equation is in the form

$$v_2^1(u-1)+v_2^2(u-1)+\ldots+v_2^n(u-1)=0, \tag{3.37}$$

which reduces to $\sum_{k=1}^{n} v_2^k = 0$. The remaining equations are of the form

$$v_2^1(u-1)-v_2^2-\ldots-v_2^i[(\mathrm{n}-1)(u-1)]-\ldots-v_2^n=0 \tag{3.38}$$

for $i=3,...,n$. The solution of these n equations is

$$\begin{aligned} v_2^1 &= \chi(-1\times(n-1)) \\ v_2^2 \cdots v_2^n &= \chi \end{aligned} \tag{3.39}$$

where χ is an arbitrary constant. How well the Fiedler vector partitions node 1 from the remaining nodes in the network is evident in Eqn. (3.39). The first value v_2^1 has not only a different sign, as indicated by the -1, from the rest of the values, but it is also a much larger value.

To solve for the elements of v_2 from a slightly different approach,

$$(Q-\lambda_2 I)v_2=0 \tag{3.40}$$

must be solved for v_2. One can rearrange and expand Eqn. (3.40) to show the first linear equation in vector form, the result is

$$\begin{bmatrix} Q_{1,1} & e_{1,2} & \cdots & e_{1,\mathrm{n}} \end{bmatrix} \begin{bmatrix} v_2^1 \\ v_2^2 \\ \vdots \\ v_2^n \end{bmatrix} - \lambda_2 v_2^1 = 0. \tag{3.41}$$

where $e_{1,2},\cdots,e_{1,n}=u-1$. Rearranging Eqn. (3.41), one can isolate v_2^1 as a function of link weights, algebraic connectivity, and the remaining components of the Fiedler vector, as follows

$$v_2^1 = \frac{1}{\lambda_2 - Q_{1,1}} \left(\begin{bmatrix} e_{1,2} & \cdots & e_{1,\mathrm{n}} \end{bmatrix} \begin{bmatrix} v_2^2 \\ \vdots \\ v_2^n \end{bmatrix} \right). \tag{3.42}$$

Then, one can determine $Q_{1,1}$ as shown in

$$Q_{1,1} = -1 \times \sum_{\substack{k=1 \\ i \neq k}}^{n} w_{1,k} = (n-1)(1-u) . \tag{3.43}$$

Substituting Eqn. (3.43) back into Eqn. (3.42), the result is

$$v_2^1 = \frac{-1}{(u-1)} \left(\begin{bmatrix} u-1 & \cdots & u-1 \end{bmatrix} \begin{bmatrix} v_2^2 \\ \vdots \\ v_2^n \end{bmatrix} \right) . \tag{3.44}$$

Eqn (3.44) reduces to

$$v_2^1 = -1 \left(\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} v_2^2 \\ \vdots \\ v_2^n \end{bmatrix} \right) \tag{3.45}$$

and finally

$$v_2^1 = -1 \times \sum_{k=2}^{n} v_2^k . \tag{3.46}$$

Eqn (3.46) demonstrates that v_2^1 will have a different sign than the sum of the remaining components of the Fiedler vector. The remaining components can be shown to be all equal, which is the same as Eqn. (3.39).

To graphically demonstrate the results of Eqn. (3.39) and Eqn. (3.46), a full mesh network without and with congestion are shown in Figure 12 and Figure 13. The congested case is simulated by reducing the link weights to near zero for a node of interest. The congested node in green in Figure 13 is separated from the others as solved for in Eqn. (3.46). The congested state can be easily identified by the controller, which can monitor for the condition shown in Figure 13. Because the SDN controller is constantly updating the current network link weights, it will be able to identify that there is a congested node in the network from the eigenvalues and then it will use the eigenvectors to determine where that congestion is occurring. By knowing both that it has occurred and where it has occurred, the controller can take corrective action to relieve the congestion.

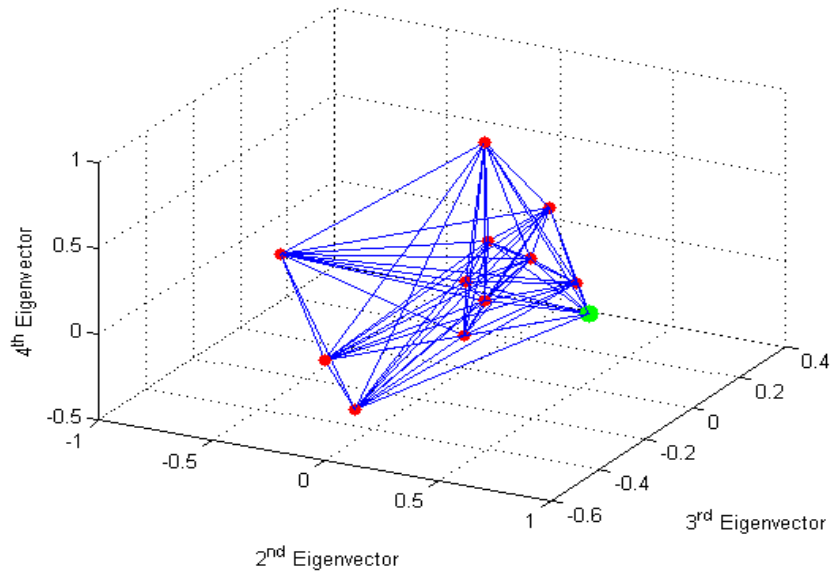

Figure 11. A random network is shown in three dimensions using the trailing three eigenvectors of the Laplacian matrix. All links in this graph are equal to 1. The green node is the node of interest, and it is not congested.

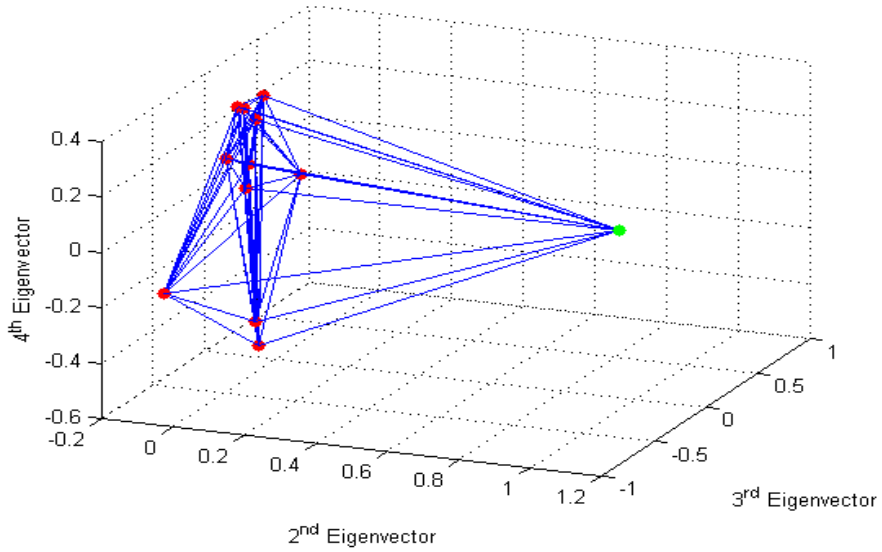

Figure 12. A random network is shown in three dimensions using the trailing three eigenvectors of the Laplacian matrix. All links in this graph are equal to 1, except for the links to the green node are reduced to near zero, which is indicative of congestion.

D. DUAL-BASIS ANALYSIS OF THE 17-NODE NETWORK

In the previous sections, simple graphs were analyzed because symmetry could be exploited to determine the closed-form solutions. The following simulations are based on the 17-node network from Figure 7.

Similar to the previous closed-form equations in Section C, large gaps between eigenvalues are indicative of distinct sets of nodes. The eigenvalues belonging to the mesh core in Figure 7 are associated with the larger eigenvalues, and the smaller eigenvalues are associated with the access network. From this perspective, the 17-node network consists of two distinct networks, but both could be managed by a single, logically centralized controller [17].

To simulate the dynamic performance of the network, all of the link weights of the links connected to node 6 are reduced from 1 to 0. The eigenvalues reflect the reduction of the link weights as shown in Figure 13. Eigenvalues 3 through 8 are all affected by this reduction, but the link behavior is isolated to a small number of eigenvalues as indicated by the small number of eigenvalues that change at any particular time in the simulation. As the eigenvalues shift down, only one eigenvalue is changing for most of the transition except right at the knee in the curves. As the node transitions between eigenvalues, it cannot be isolated from the next closest node as represented by the next lowest eigenvalue. Eigenvalues 9 through 17 also change due to the relationship between the access and core networks. They have a constant decrease because they are all connected similarly to node 6. The physical interpretation of this is that the core network's available network bandwidth capacity to the access network is constantly decreasing. In this case, the decrease in available capacity is solely related to the reduced link weights to node 6.

At the end of the simulation, all of the affected eigenvalues have shifted down by one. The number of zero eigenvalues has increased from two to three because there are now three separate networks: nodes 6, node 17, and the remaining connected nodes. The slope of the line that connects all of the transition phases of each eigenvalue can be approximated by observation as

$$\frac{d\lambda}{dw} \approx -d_k \tag{3.47}$$

where k is the node of interest, which in this case is node 6.

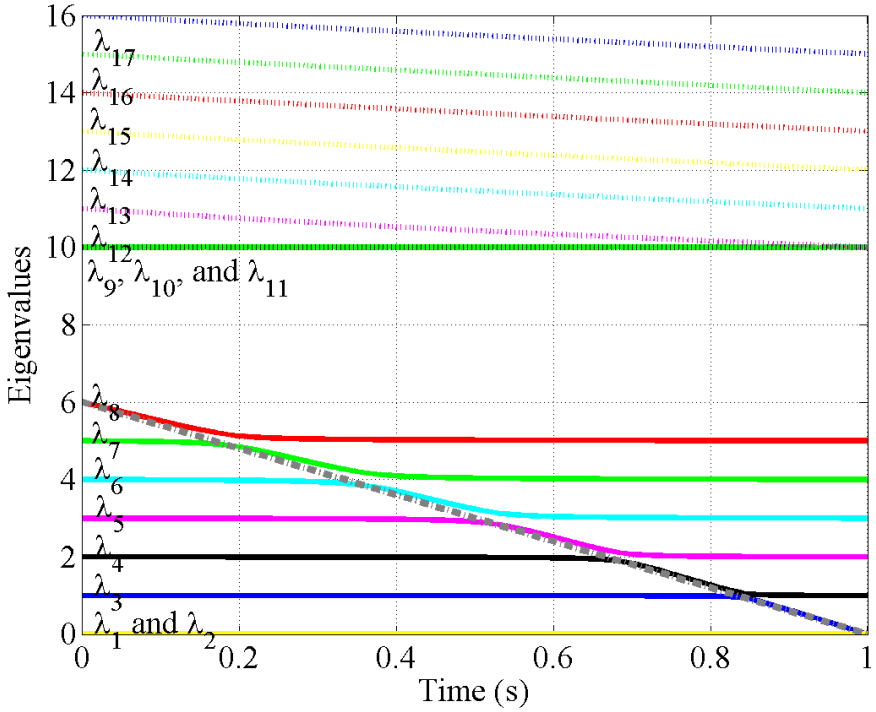

Figure 13. As node 6s links are reduced to zero from time 0 to 1 second, the eigenvalues of the 17-node network demonstrate the behavior from Eqn. (3.47) as shown by the gray dashed line.

The degree of node k d_k determines the approximate slope of each eigenvalue as it shifts from its current value to the next value down, and the starting value λ_0 determines the y-intercept. In this case, node 6 has a degree $d_k = 6$, which means that the slope is $d\lambda/dw = -6$ and the starting value is $\lambda_0 = 6$. In Figure 13, the dashed gray line's slope is based on Eqn. (3.47).

To determine if Eqn. (3.47) still holds when the links to other nodes are reduced to zero, the simulation was continued. The links of nodes 5 and 4 were reduced to zero. The result is shown in Figure 14. The pattern holds, as does Eqn. (3.47). By the end of

this simulation, the null space has increased to five; nodes 4, 5, 6, and 17 are all in the null space. This is reflected by the five zero eigenvalues.

The controller can use this information to route packets and create flow rules that avoid these links and switches. In this case, the nodes were removed from the network due to simulated congestion. The congestion could have been created by normal traffic that exceeded the capacity of the affected links, or it could be due to a failed switch. On the other hand, the congestion may also have been created by a targeted denial-of-service attack or some other cyber attack. The eigenvalues and eigenvectors do not provide the controller with sufficient information to discriminate between the two types of congestion. Nevertheless, when a node enters the null space, it could be a flag for another SDN application to determine the reason.

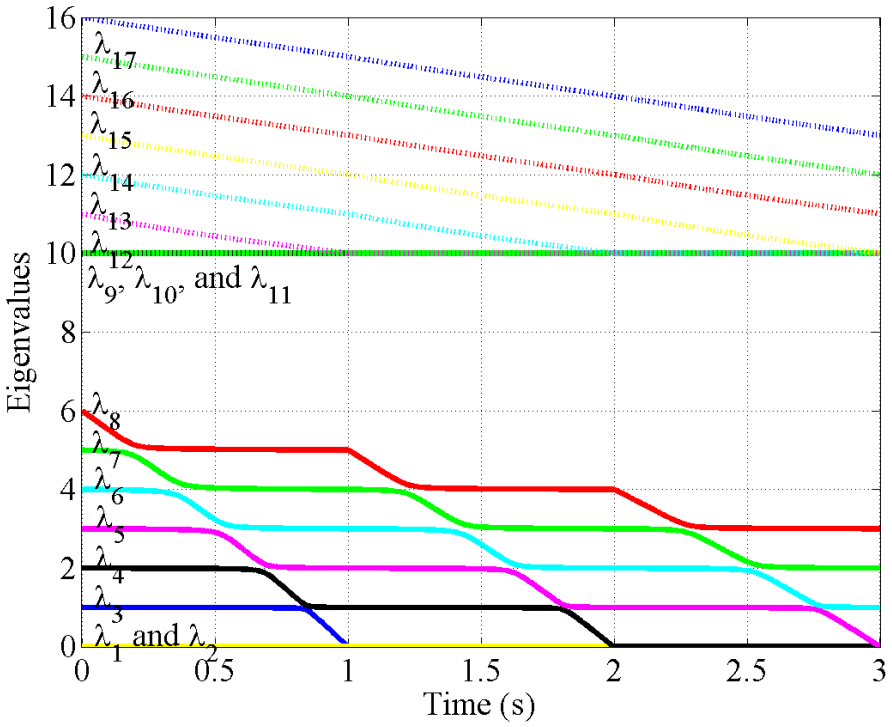

Figure 14. Eqn. (3.47) is demonstrated as three nodes enter the null space by reducing all their links to 0.

The network was simulated with complete control over which links were reduced to zero. In a real-world situation, one would not have this knowledge. The controller must know which node entered the null space or which node is congested. The eigenvectors provide the information about where in the network these dynamics are occurring. Figure 15 demonstrates the behavior of the nodes in the eigencentrality basis and is the same simulation that produced Figure 14. The third, fourth, and fifth eigenvectors are plotted as a function of time.

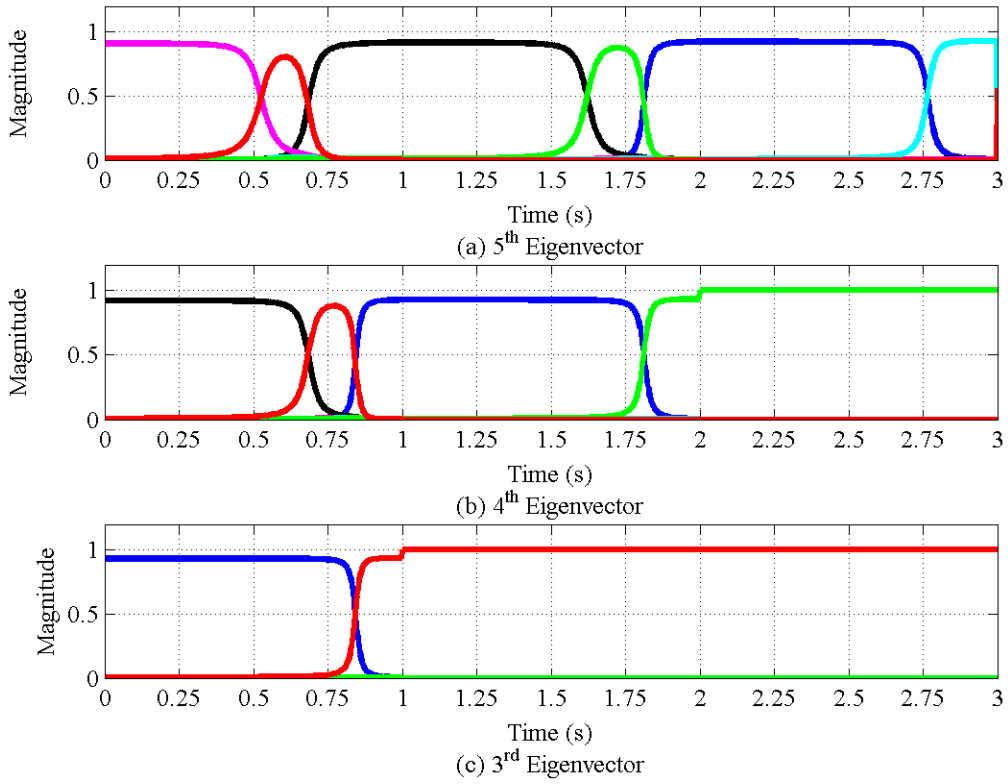

Figure 15. The third, fourth and fifth eigencentrality components are plotted versus time as all the link weights that attach to nodes 6, 5 and 4 to the core network are reduced to zero. Node 1 is blue. Node 2 is black. Node 3 is magenta. Node 4 is cyan. Node 5 is green. Node 6 is red.

The first major observation from Figure 15 is how well the nodes are isolated at the start of the simulation. Nodes 1, 2, and 3 are the dominate nodes in these three eigenvectors as they are the least central nodes as reflected in Figure 7. The first node to

have its links reduced to zero is 6. As the links are reduced, node 6 becomes less central and transitions through each of the eigenvectors until it finally becomes disconnected from the network at approximately 1 second. Similar transitions occur for node 5 and node 4.

One should notice how the isolation that is evident when the simulation starts is not present as the nodes transition. For instance, notice how node 6 transitions with node 3 at approximately 0.5 seconds into the simulation. At that point they have the same influence in the network from a centrality perspective. The eigenvectors show that these nodes are node isolated from one another. As this transition continues, the nodes are again isolated. This concept of nodal isolation and coupling will resurface in later chapters to determine how many nodes are required to control the network.

Returning to the idea that the nodes may be represented as a point in n-dimensional space, one can replot Figure 15 as a two-dimensional graph representation. The result is the transitions between eigenvectors are now a change in magnitude of the nodes and angle among nodes. For this network, there are up to 17 dimensions that can be displayed. Two of those are shown in Figure 16. The behavior of the third and fourth eigenvectors is demonstrated in Figure 16 for the first second of the simulation. The result clearly shows that as the links of node 6 are reduced to zero, the two-dimensional representation of the node 6 changes in magnitude and angle. At the beginning of the simulation, nodes 1 and 2 are orthogonal to each other. As the simulation continues, node 2 and 6 exchange places in terms of magnitude, but they are separated by 180°. Both nodes 2 and 6 remain orthogonal to node 1. Once node 6 has become the dominate node in the fourth eigenvector, it begins to rotate through the two-dimensional space to replace node 1 on the third eigenvector axis. Notice that at all times node 6 is orthogonal to node 1. This is due to the fact that the eigenvectors reveal the orthogonality between nodes to the extent possible. In this case, the nodes maintain their orthogonality throughout the simulation.

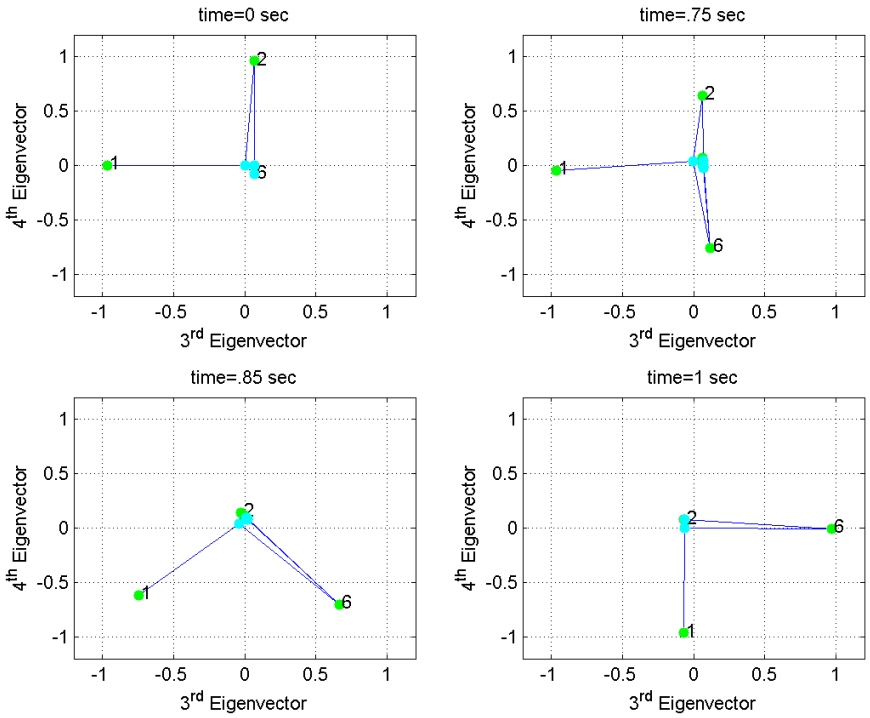

Figure 16. The nodal behavior is demonstrated for first second of the simulation in two-dimensions. The link weights of the links to node 6 are reduced from 1 to 0. The movement of nodes 1, 2, and 6 can be captured by using vector magnitudes and angles between vectors.

To complete the analysis of this 17-node network, one must observe how the leading eigenvectors behave during the transitions observed in the trailing eigenvectors. The behavior of the leading eigenvectors is demonstrated in Figure 17. Throughout the three second simulation, the leading eigenvectors remain the same. They are unaffected by the transitions in the other eigenvectors. Again, this is because the dual-basis method effectively reveals the network behavior. The leading eigenvectors are indicative of the behavior of the core network; the centrality of the core is unaffected by congestion in the access network. They are separate networks and one would expect that the centrality of these nodes to be separate from each other.

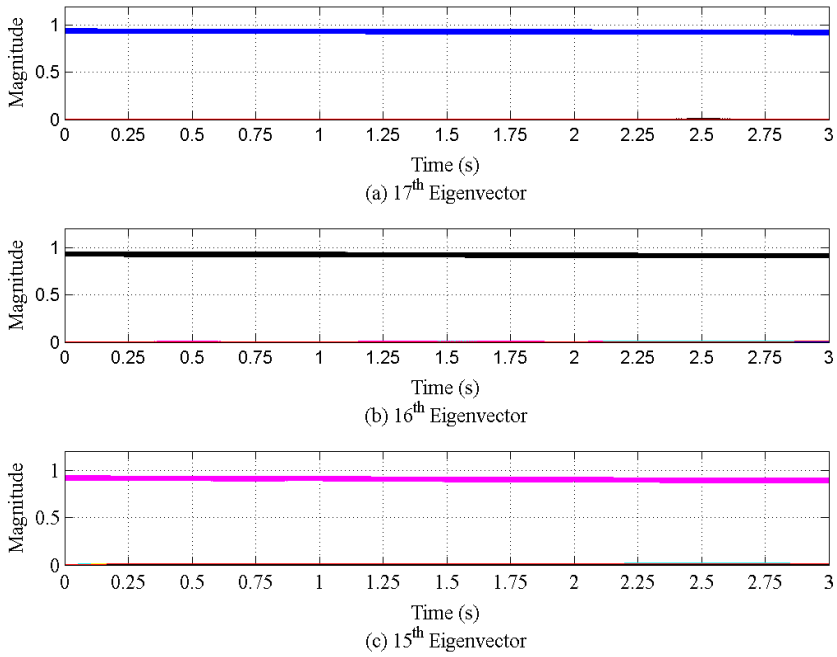

Figure 17. The 15th, 16th, and 17th eigencentrality components are plotted versus time as all the link weights that attach to nodes 6, 5 and 4 to the core network are reduced to zero. Node 14 is magenta. Node 15 is black. Node 16 is blue.

E. PHANTOM NODE

The behavior of the eigenvalues in Figures 13 and 14 and approximated by Eqn. (3.47) can be exploited by adding a virtual node or phantom node to the graph that does not exist in the physical network. This additional node is placed in such a way that it is the dominant node in the Fiedler vector. In [49], it was shown that λ_2 is bound by the node with the minimum link weight, which is the reason that the phantom node can be used as an indicator of onset of congestion. The phantom node is attached to the most central node in the network as determined by the dual-basis analysis because it will have the least effect on the dual-basis. The link weight of the phantom node can be changed to vary when congestion is indicated; smaller link weights result in a smaller algebraic connectivity, which will delay the indication of congestion because the threshold crossing will occur at smaller link weights. The opposite is true of larger link weights. The shift of the phantom node's dominance to larger eigenvalues indicates the onset of congestion,

but not where in the network congestion is beginning. The eigenvectors will indicate where in the network the congestion is occurring.

The development of the phantom node and simulations to support this hypothesis are contained in [49]. The phantom node was tested on a preliminary, six node hardware SDN to show that the hypothesis holds up when applied to a real-world network [50]. The simulations and the hardware experiments validated the development of the phantom node. A result from [50] is shown in Figure 18. Node 3 is under a DDOS attack and the congestion is indicated by node 3's nodal influence shifting to λ_2. The drawback to this approach is that it does not indicate maliciousness; by simply analyzing the phantom node behavior, the reason for the congestion cannot be determined. Deeper inspection of the packets, flows, and timing of congestion needs to be conducted to determine the root cause of the congestion.

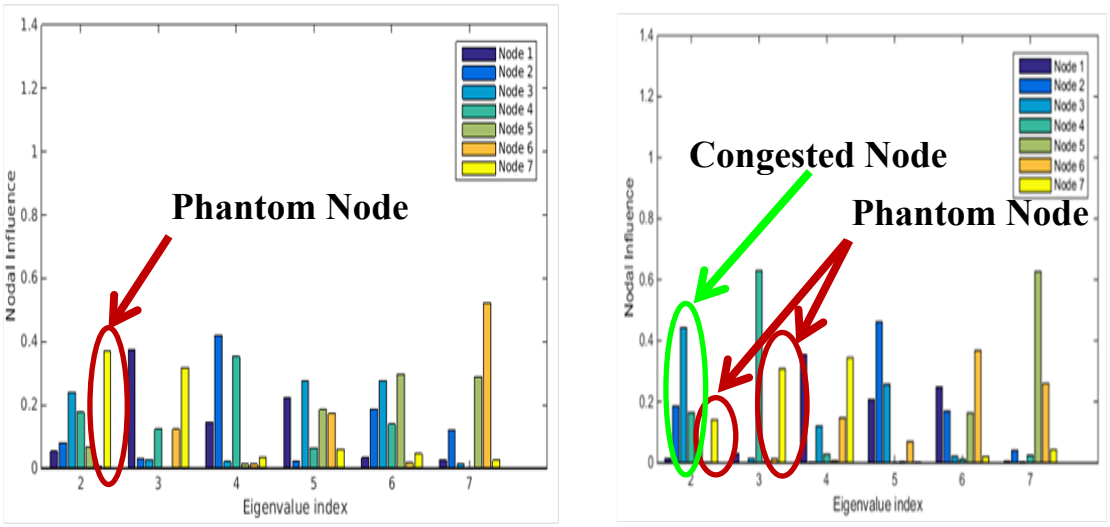

Figure 18. The phantom node is the dominant node for λ_2 until the onset of congestion in the second figure. Node 3 is congested due to a DDOS attack, which is indicated by the shift of the phantom node to dominate λ_3 and node 3 dominating λ_2, from [50].

Up to this point, the dual-basis representation has been analyzed to help determine the network behavior with both static link weights and dynamic link weights. The combination of the ratio cut optimization solution, the closed-form solutions, and the simulations provide the foundational work to demonstrate that the dual-basis analysis is an effective means to reveal the structure and behavior of the network. The link weights in the simulation were controlled directly and were not allowed to vary randomly. This will not be true in real-world networks. The SDN controller polls the switches for network traffic measurements, which will be noisy. The controller must have an effective method to estimate the link weights to ensure that the subsequent link weights used in the dual-basis analysis accurately reflect the true data rates in the network. In most control systems, a state space observer is used. In the following chapter, a state space representation of the network is explored to develop a network observer that will accurately estimate the network's link weights.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CLOSED-LOOP CONTROL OF SDN

Software-defined networks have opened the door for researchers to model and control data communication networks in a whole new way. Many real-world systems are assumed to be LTI or can be linearized. SDNs are by design non-linear because of the discrete event nature of the packets being transmitted and switching among nodes in the network. Intervals between traffic generation, the data rate of a flow, and the total amount of data transferred per flow may not be accurately characterized by a random variable with a Gaussian distribution. Both the nature of the system and the traffic force researchers to find new ways to model the network, estimate the link weights, and control the overall network.

In order to evaluate the performance of a SDN as a closed-loop control system, the model must be able to handle the non-linear behavior and non-Gaussian noise, which are inherent to large, complex data communication networks. The non-linear and non-Gaussian nature of a SDN can be modeled, estimated and controlled by using a closed-loop control system framework. The first requirement of a closed-loop control system is an observer to estimate the state of the network given system measurements and given controller feedback. Once the state is estimated, the controller then generates a feedback signal to achieve the goal set by the cost function of the overall system.

A. PROPOSED CLOSED-LOOP CONTROL SCHEME

A generalized closed-loop control system is shown in Figure 19. It includes the components that make up the SDN model: the dual-basis analysis, new packet message, and flow modification messages. SDNs are fundamentally different from traditional control systems. In a traditional control system, the entity being controlled is directly measured. For instance, many circuits have control loops that maintain voltage or current within a specification. To control the voltage or current, a direct measurement of that current or voltage is made and compared to a set point. The resulting error is passed to the controller for correction.

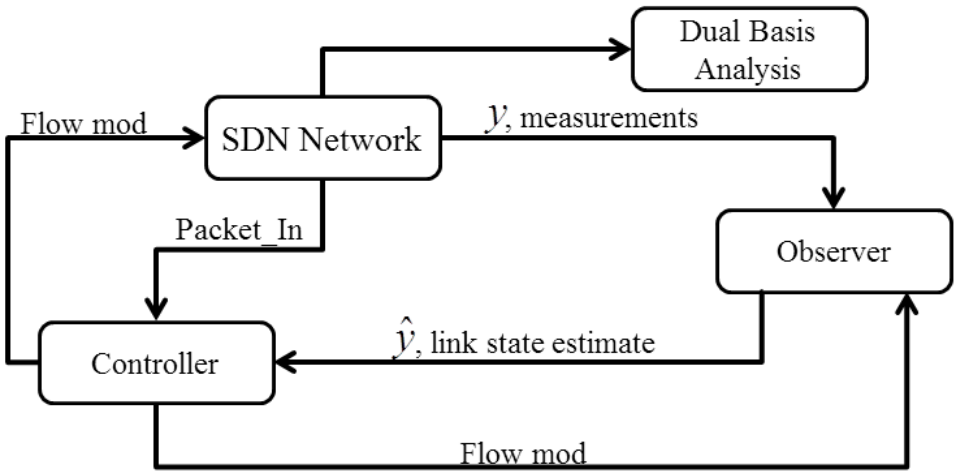

Figure 19. A SDN modeled as a closed-loop control system has an observer to estimate the link data rates, and a controller to generate flow modification messages to change the current link data rates. The dual-basis analysis is included to provide the controller with additional information in the form of network features.

In a SDN, the controller is not controlling the number of packets that enter the network or the routing of those packets but instead, directs flows through the network. The measurement y is a vector of the current, measured data rates. The SDN controller requests information about the total flow of traffic in each direction through the physical Ethernet wires from the monitor nodes. The data rate estimates $\hat{y}$ are calculated based on the current number of flows, the posterior probability, and the current measurement. The data rate estimate is provided to the controller to determine network flows, which is accomplished by sending flow modification messages to the designated network switches. The specific method used by the controller to determine the optimal route is not considered in this research; the focus here is on estimating and analyzing the network state.

The proposed closed-loop control scheme has four main parts as shown in Figure 20: the plant, the observer, the controller, and the dual-basis analysis. The observer has two significant parts: the particle filter [45] and the phantom node. The particle filter is used as described in Chapter II, but the specific inputs to the filter will be discussed in the next section. The phantom node is a method to estimate congestion in the network and to

inform the controller about the intensity and location of congestion [49], [50]. The observer provides the controller with all of the information necessary to control flows both reactively and proactively.

The controller implements any one of a number of routing algorithms from widely used algorithms like open shortest path first (OSPF) [19] to more application specific algorithms like equal cost multi-path routing (ECMP) [51]. Again, the goal of this research is not to develop new routing algorithms, but to provide a framework that can use any routing algorithm. This work seeks to demonstrate a method to minimize the number of nodes where the selected routing algorithm must be implemented, which are called the control nodes in this dissertation. Reducing the number of control nodes reduces the complexity of the overall system and reduces the computational load on the controller. Both of these reductions can be achieved without sacrificing performance in terms of the cost function used by the routing algorithm, as will be demonstrated in the next two chapters.

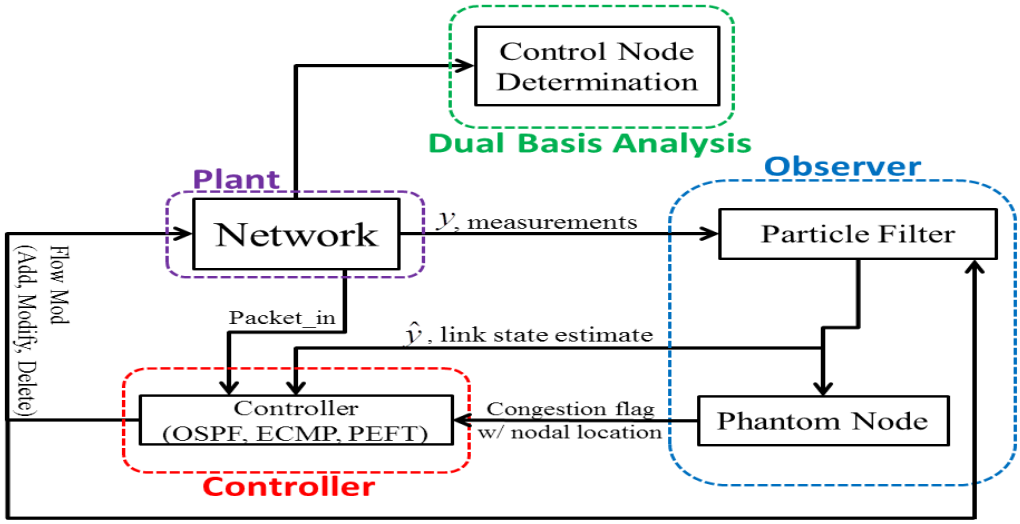

Figure 20. The proposed SDN closed-loop control scheme has an observer and a controller. The observer is the combination of the particle filter to estimate the link data rates and the phantom node to identify congestion. The controller uses the information from the observer and from the features extracted by the dual-basis analysis to generate flow modifications messages.

The following sections describe each part of the closed-loop control scheme in detail. The process to estimate the link data rates is first described. To estimate these data rates, the number of monitor nodes needs to be determined; these are the nodes that are polled to obtain the information required to calculate all link data rates and in turn the link weights. The process to identify the monitor nodes is described. The controller is described next. The method to identify the control nodes is described, which is a specific application of the dual-basis analysis. The controller's routing problem is described in terms of load balancing the network traffic.

B. LINK DATA RATE ESTIMATION

The link data rate estimate is one of the two outputs of the observer. The other is the congestion flag from the phantom node. The typical approach when designing a state estimator is to start by modeling the system with a set of linear equations. If that does not work, the system designer will try to linearize a non-linear model around an operating point. If that approach fails, a more general solution that allows non-linear models is used. The most widely used estimator is the Kalman filter [45]. It can be shown that the Kalman filter is an optimal estimator for a linear system with Gaussian noise. A SDN is not a LTI system and does not have Gaussian noise, but neither do many of the real-world systems that are modeled as linear systems. If a SDN can be modeled as an electrical circuit using linear components, then a Kalman filter can be used as an estimator. If that is unsuccessful and non-linear components must be used, it may be possible to use an EKF to achieve the link data rate estimates. If both of those paths fail, a particle filter may be used as a final option. The following sections will demonstrate the attempt to linearize a SDN to finally adopting a particle filter [45].

1. Monitor Nodes in a SDN

Monitoring network traffic in a SDN is inherent to the OpenFlow protocol. The controller can request the number of packets and bytes that have been transmitted and received on each physical port on the polled switch. The controller can also request the number of packets and bytes that have been matched to an individual flow [21]. The problem is determining the minimum number of switches that need to be polled to

calculate all link data rates throughout the network. Minimizing the number of monitor nodes reduces the workload of the controller and reduces the number of packets that are sent on the control network.

The solution of the vertex cover problem determines the minimum number of nodes required to calculate all link data rates. The vertex cover problem seeks to determine the minimum set of nodes that are required to ensure each link is incident to at least one node in the graph [52]. Various solutions to this problem have been suggested, and this work does not seek to find a new solution. Using standard techniques, the controller solves the problem to determine which switches it will transmit requests to in order to determine all data rates on all links. This solution will be updated periodically to adapt to changes in network conditions.

2. State Space Model of a SDN

Many non-electrical systems have been modeled as electric circuits, such as spring, mass, damper systems [43]. To use the Kalman filter to estimate all link data rates, a linear state space model of a SDN must be developed in a manner similar to the process used when building linear models of mechanical systems. The electrical circuit model of a simple SDN is shown in Figure 21. The equivalent electrical circuit of this SDN is shown in Figure 22. All of the components of the circuit except for the switches S_1 and S_2 are linear components. The voltage on the capacitors represents the queue size of the switches. The behavior of the energy storage in a capacitor is similar to the behavior of switches as their buffers are filled with incoming packets. The current through the resistors was used to model the data rate between switches. The resistors were used to limit the amount of current between the capacitors. The voltage source is the source node producing traffic in the network. The resistor R_4 in parallel with the capacitor C_2 represents the sink node. Traffic is being transmitted from the source to the sink via the intermediate switches.

To maintain the linearized nature of the system, each side of the circuit was analyzed separately. The right side was analyzed with S_1 closed and S_2 open. The left

side was then analyzed with S_1 open and S_2 closed. The state matrix of the right hand side of the circuit is

$$A = \begin{bmatrix} \frac{-1}{C_1}\left(\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}\right) & 0 & \frac{1}{C_1R_3} \\ 0 & \frac{-1}{C_3}\left(\frac{1}{R_5}+\frac{1}{R_6}\right) & \frac{-1}{C_3R_5} \\ \frac{1}{C_1R_3} & \frac{-1}{C_2R_5} & \frac{-1}{C_2}\left(\frac{1}{R_3}+\frac{1}{R_4}+\frac{1}{R_5}\right) \end{bmatrix} \tag{4.1}$$

and the input matrix is

$$B = \begin{bmatrix} \frac{-1}{C_1R_1} \\ 0 \\ 0 \end{bmatrix}. \tag{4.2}$$

The C matrix is an 3×3 identity matrix, and the D matrix is a 3×1 vector of zeros. The left hand side equations are similar to the matrices in Eqns. (4.1) and (4.2).

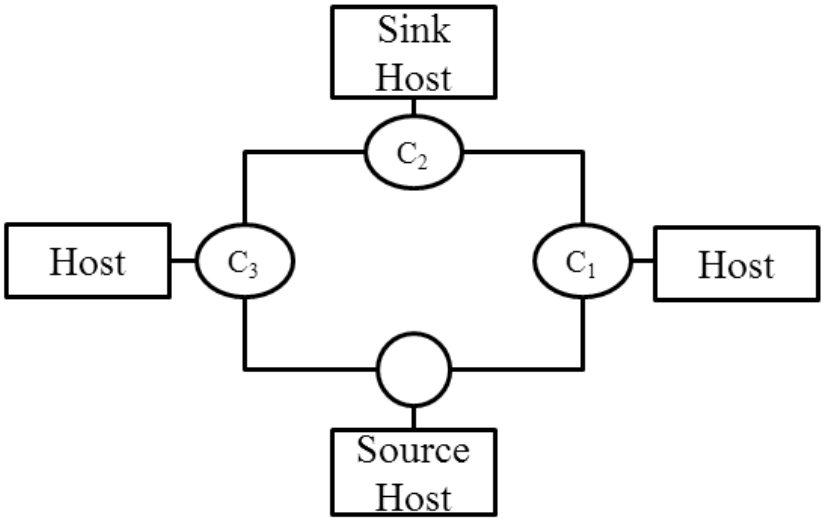

Figure 21. A simple four switch network was modeled by a circuit to determine a linear system to further develop a state space model for the system.

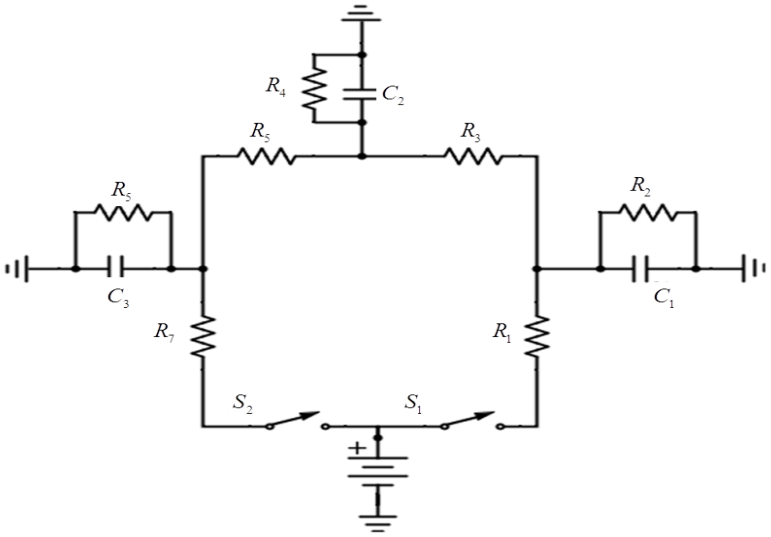

Figure 22. To model a SDN, voltage on a capacitor was used to model the queue of a switch and current through resistors was used to model the data rate between switches. This circuit is used to model the SDN in Figure 21.

After developing the state space model analytically, MATLAB scripts determined the validity of this model. Another model of the same system was created using Simulink, which was used to ensure that the state space model was correct. Two sets of simulations were run. The first set of results is based on all initial conditions being equal to 0. The second set of results is based on the initial conditions set to 0, 0, and 0.9 volts on capacitors C_1, C_2, and C_3. These results are shown in Figures 23 and 24. As shown, the results based on the state space model and the Simulink model are exactly the same.

The next step is to combine the two sets of state space equations. Using initial voltages on the capacitors, as in Figure 24, one can model the instant after a switching event, which is when one switch closes and the other opens. The voltage on each capacitor represents packets in the queue of a switch that need to be transmitted to the destination. The charge on each capacitor will generate a current that represents the continued data rate from source to sink even though the switch has redirected traffic down the other path. This is a reasonable model of how a SDN works. The controller makes decisions about which switches are open in order to allow traffic to transit to the sink.

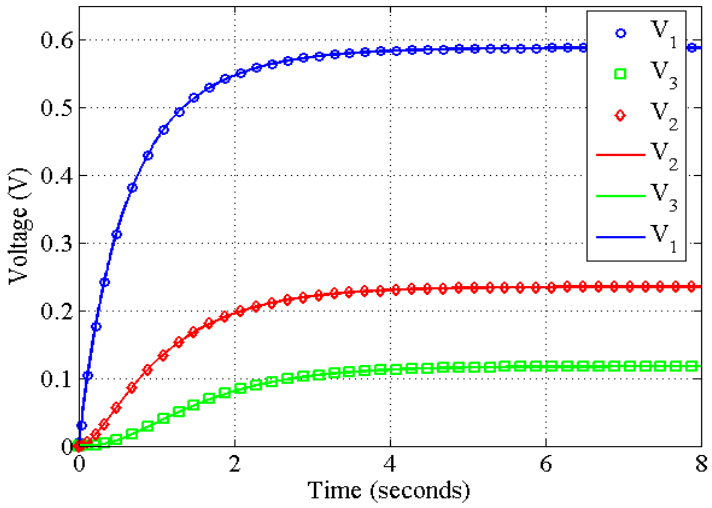

Figure 23. The result of a step input to both the Simulink model of the electrical circuit from Figure 22 that represents a SDN and state space equations from Eqns. (4.1) and (4.2). All initial conditions are set to 0. The Simulink results are the open symbols, and the state space model results are the solid lines.

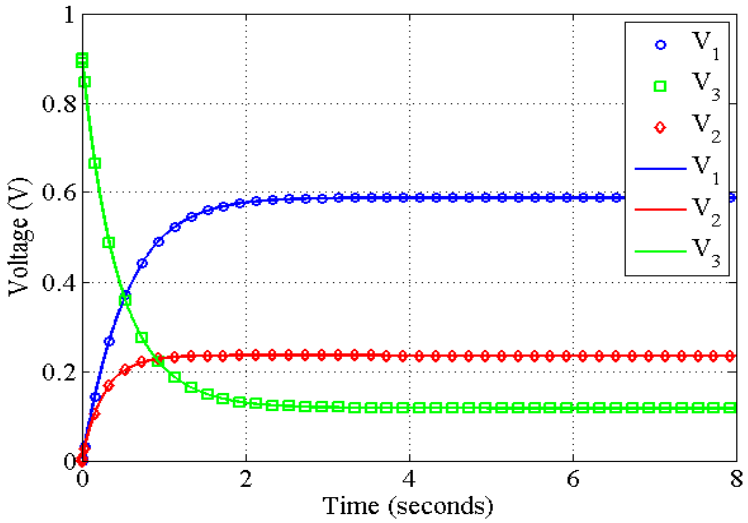

Figure 24. The result of a step input to both the Simulink model of the electrical circuit from Figure 22 that represents a SDN and state space equations from Eqns. (4.1) and (4.2). The initial conditions are 0, 0, 0.9 on capacitors C_1, C_2, and C_3. The Simulink results are the open symbols, and the state space model results are the solid lines.

However, this model includes a single source node and a single sink node. State space averaging was attempted to generalize the state space model with limited success [53]. Using a separate set of linear equations for each scenario was also considered. This approach was discarded because the controller and observer would need to know which set was being used when, and more importantly, a large number of these sets would need to be implemented for every possible case. Additionally, current in the circuit only flows in one direction at any given time, but in a SDN, data is being transmitted in both directions simultaneously. It was determined that a linear or linearized model of a SDN was not the best approach. Consequently, the particle filter method was considered [45]. Even though the particle filter does not produce an optimal solution to the link data rate estimation problem, the model it uses is often a more accurate representation of the dynamics of the network.

3. Particle Filter Estimator in a SDN

As described in Chapter II, a particle filter is an estimator that uses the same basic structure as a Kalman filter, but instead of an optimal estimation, the method uses a Monte Carlo simulation to estimate the link data rates, which were used to calculate the link weights as shown in Eqn. (3.24) [45]. The implementation of the particle filter in a SDN is a straightforward process. The particle filter runs in two steps: predict and update. The particle filter's first stage is to predict the link data rate of the next step using a non-linear model of the system. The update step uses the current measurement to update the prior PDF using the Bayes' approach. This process is the same as that shown in Figure 5, but the control input u_k is now the number of flows on each link, which is used as the input to the system.

The non-linear model implemented is [45]

$$x(k+1)=\begin{cases}0 & \text{if } x(k+1)<0\\ Ax(k)+BU(k)\mu+\mathrm{N}\eta(k) & \text{if } 0<x(k+1)<x_{\max}\\ x_{\max} & \text{if } x(k+1)>x_{\max}\end{cases} \tag{4.3}$$

where x is the state vector, A is the state matrix, B is the input matrix, U is the control matrix, μ is the input vector, V is noise matrix, and v is the system noise. The state

vector in this case is a $l \times 1$ vector where l is the number of links in the network. The state vector contains the current estimate of all of the link data rates. If the data rate limit has not been reached, the current state $x(k)$, the input and the noise are all summed to calculate the updated state $x(k+1)$. If the link is operating at maximum data rate, the model limits the traffic on that link to that value. This maximum limit is the maximum data rate of the link and is typically measured in bits per second (bps). The $l \times l$ state matrix A and the $l \times l$ input matrix B are identity matrices because the link data rate equations are not coupled. The updated and predicted state are estimated to be the same as the previous state when there is no input to the system.

The input vector μ is an $f \times 1$ vector where f is the number of types of flows that are used with the non-linear model and is equal to the mean data rate of the flow. In this case, only one type of flow was modeled because UDP was the only type of traffic used. The control matrix U is an $l \times f$ matrix that is updated to add the mean data rate to the correct link estimate based on new flows in the network. The observer is notified of these new flows by the controller. The product of B, U and μ is equivalent to the multiplication of B and u from Eqn. (2.19).

The noise matrix N is an $l \times l$ diagonal matrix because the noise in the system is considered to be independent among the links. The noise vector η is an $l \times 1$ vector that is a set of realizations selected from the random variable used to model the system noise. The random variable used was a zero-mean Gaussian random variable. The system noise can be generated in two ways. The first process that generates noise is flows that have a small number of packets associated with them, such as address resolution protocol (ARP) packets and domain name service (DNS) packets. These flows are short-lived and do not generate large amounts of traffic. The second source of noise is the variation in traffic associated with large flows, such as the large UDP flows that are used in this research.

4. Use of Phantom Node for Congestion Detection

Every control system has a dynamic range for the control input. In a SDN, the dynamic range of control is the data rate in bps of each link. When the maximum data

rate has been reached on any link and the buffer of a switch has been filled to capacity, the switch will start to drop packets because it is unable to process new incoming packets. A goal of the controller should be to prevent this condition. A goal of the observer should be to notify the controller of this condition and where it is occurring. The phantom node is a solution to this problem [49], [50].

The phantom node is added to the graph representation of the network, but it does not physically exist. The observer uses the phantom node's position in the eigenspectrum and the associated eigenvector to locate the congestion. Once the congestion is located, that congested node's location is passed to the controller. The controller then must decide how to take action to reduce the congestion. The controller can stop routing new packets to or from the affected destination. It can also remove low priority flows that are currently active in the congested area.

C. CONTROLLER

The SDN controller is an application at the heart of the closed-loop control system. No packets are transferred anywhere within the network without the direct intervention of the SDN controller. With that in mind, as the number of switches in the network increase, the workload of the controller increases. Identifying the most influential nodes or control nodes in the network using the dual-basis analysis allows the controller to reduce its workload by calculating routes from these influential nodes to all destinations. These influential nodes are known as the control nodes. These are the nodes at which the controller is able to manage the network state. The goal is to accomplish this reduction without impacting performance. The results in the coming chapters will demonstrate that this decrease in workload can be accomplished without sacrificing performance.

1. Identification of Control Nodes

In designing the controller application, one must take into account the number of switches that are going to be controlled. The hardware on which the application is running is another design criterion. The application needs to be as efficient as possible to prevent overloading. If the switches are sending too much traffic to the controller, it can

be overwhelmed and a backup of packet_in messages will occur [21], which results in long round-trip times between source and destination within the network, and it could result in dropped packets.

One method to reduce the workload of the controller is to reduce the number of switches that generate packet_in messages. The control node selection allows the network designer to focus on a subset of control node switches in the network and optimize flows for those packets that transit through the control nodes. All other nodes in the network will use static routes, which can be installed proactively or reactively. By limiting the number of nodes that determine routes dynamically, the workload of the controller is reduced.

The identification of the control nodes is a process based on the principal eigenvector analysis. Each node is represented by an n-dimensional vector as demonstrated in Eqn. (3.17). Each nodal vector is orthogonal to all other nodal vectors if, and only if all n values are used. The j^{th} $n \times 1$ nodal vector is represented by

$$v^j = \begin{bmatrix} v_1^j \\ \vdots \\ v_n^j \end{bmatrix}. \tag{4.4}$$

If less than n values are used, the angle between each nodal vector shifts away from exactly 90°. The j^{th} nodal vector using less than n values is represented by

$$v_{k:n}^j = \begin{bmatrix} v_k^j \\ \vdots \\ v_n^j \end{bmatrix}. \tag{4.5}$$

This shift indicates how isolated each node is from the other nodes. Smaller rotations away from 90° are indicative of isolated nodes. Larger rotations away from 90° are indicative of coupled nodes. For instance, a shift of 3° is a small shift and is indicative of isolation. A shift of 35° is a large shift and is indicative of nodal coupling. From Figure 7 and using the leading two of 34 eigencentrality vectors, $v_{33:34}^7$ and $v_{33:34}^{16}$, the angle between nodes 7 and 16 is shown in Figure 25. The angle is nearly 90°. The angles between the first five nodes using the leading five eigencentrality vectors are shown in

Figure 26. These five are the control nodes used in the simulation in the next section. Notice how the angle between node 5 and node 7 is 112° and is clearly no longer orthogonal to the other nodes.

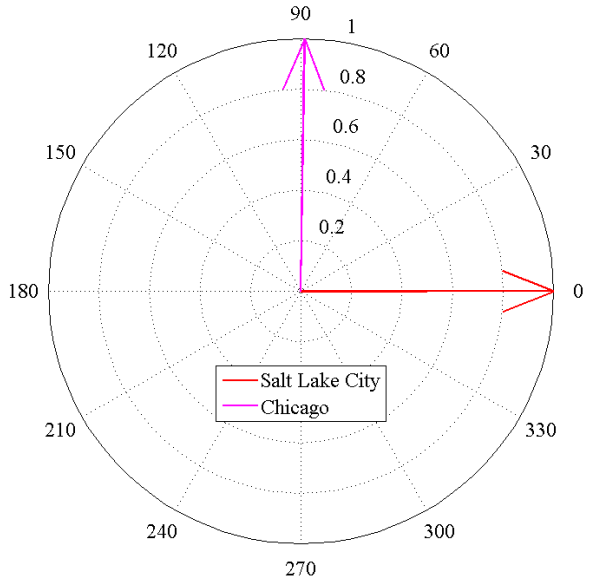

Figure 25. Chicago and Salt Lake are nearly orthogonal when using the first two eigencentrality vectors.

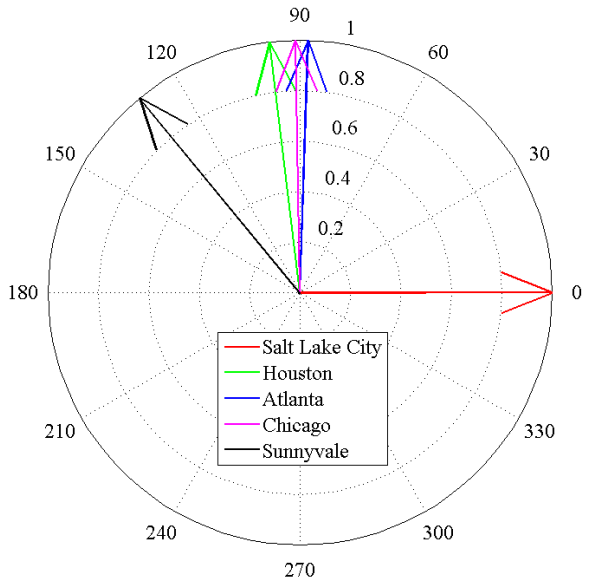

Figure 26. As more eigencentrality vectors are used, nodes will begin to drift away from 90° as Sunnyvale does in this case. All other nodes remain near orthogonal.

The control nodes are those nodes that are isolated from one another and have the largest eigencentrality $E_{k:n}^{j}$ as defined in Eqn. (3.19). The process of identifying these nodes starts with the leading eigenvector. The node with the largest eigencentrality $E_{k:n}^{j}$ value in the leading eigencentrality vector v_n is the first candidate, but it only provides a one-dimensional representation and has no angular component. The next eigencentrality vector is added so that each node has a two-dimensional representation $v_{n-1:n}$. The norm of each nodal vector and the angle between each of the nodes will indicate which two have the largest eigencentrality value and the angle closest to 90°. These two nodes are the next two control node candidates. The process continues until the nodes with the largest eigencentrality norms are no longer orthogonal. The flow chart of this process is shown in Figure 27.

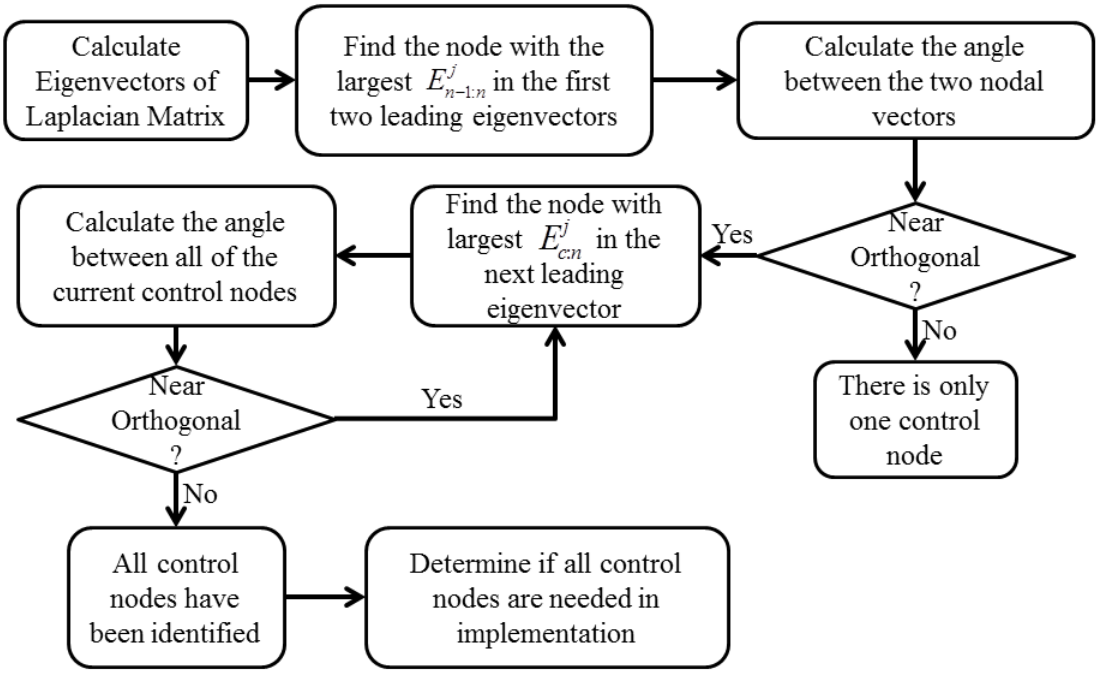

Figure 27. The process to identify the control nodes is to iteratively add centrality vectors such that the nodal vectors with largest norms are no longer orthogonal.

To combine the angles and the Laplacian eigencentrality for each node into a single metric for comparison of their centrality, the eigencentrality metric was divided by the difference between 90° and the angles. This control centrality metric χ_c is given by

$$\chi_c = \frac{E_{n-c:n}}{90c - \sum_{k=n-c}^{c} \theta_k} \tag{4.6}$$

where c is the number of control nodes being considered, and θ is the angle between the control node of interest and node k.

The process identified in Figure 27 was applied to the Internet2 topology. The results are shown in Figure 28. The colors in Figure 28 represent the relative rank of the nodes when using χ_c. Of the 34 nodes shown, four are selected as control nodes. All other nodes in the network will use static routes. At these four nodes, the controller will implement the routing algorithm to maximize the performance, which in this case the controller attempts to balance the offered load amongst all the links.

Notice that Seattle's color is light blue, which indicates it is not a control node, but its degree is four. Eigenvector centrality as defined in Eqn. (2.9) gives this node a large centrality value. In the control node identification process, this node's centrality value is diminished because its nodal vector $v^2_{31:34}$ is nearly parallel to Salt Lake City's nodal vector $v^7_{31:34}$. Salt Lake City's Laplacian eigencentrality, as given by Eqn. (3.19), is also much larger than Seattle's. The combination of these two effects diminishes Seattle and indicates that it is not a control node.

Also notice that Atlanta is identified as a control node. There are a large number of nodes with three links in this topology. If a simple degree centrality metric c_D was used, all three link nodes are given the same centrality value. Betweeness centrality and eigenvector centrality give Atlanta low values. The dual-basis analysis indicates that Atlanta is the next most influential node.

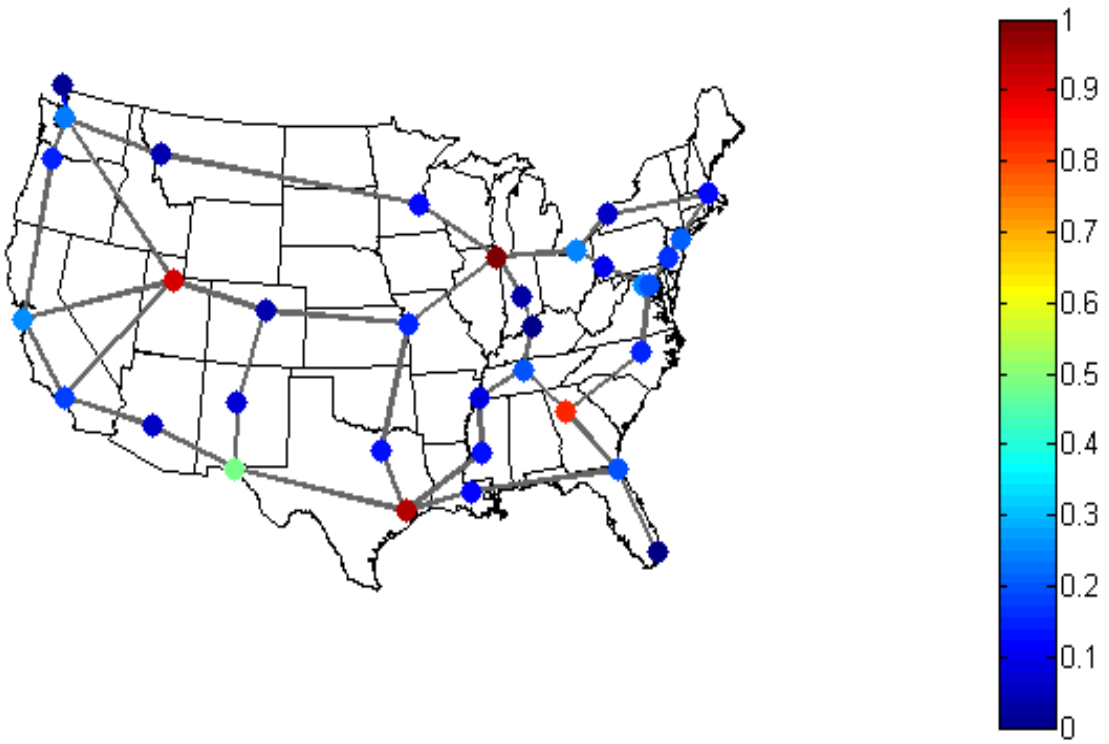

Figure 28. The control nodes are identified for the Internet2 topology using four eigencentrality vectors.

Graphs and networks are not geometric objects that can be measured to find their center. Large, complex networks can have multiple centers or control nodes and many times these nodes are not easily identified through intuition or standard centrality metrics. Once these control nodes have been found using the above analysis, the controller is able to use that information to implement the routing algorithm.

2. Load Balancing Traffic via the Control Nodes

A load-balancing algorithm was developed to demonstrate the effectiveness of the control nodes. The goal of the load-balancing algorithm is to minimize the maximum link utilization. The optimization problem the controller is attempting to solve is [2]

$$\begin{aligned} &\text{minimize } u_{1:l} \\ &\text{subject to} \\ &f_{static} + f_{dynamic} \le u_k \mathrm{c}_l \\ &f_{static} + f_{dynamic} \le I_{sd} \\ &f_{static} \ge 0 \\ &f_{dynamic} \ge 0 \end{aligned} \quad . \tag{4.7}$$

The first inequality ensures that the total routed traffic in the static and dynamic flows $f_{static} + f_{dynamic}$ is less than or equal to the maximum link utilization u_k times the link capacity c_l . The second inequality states that the total traffic in the network is less than or equal the injected traffic I_{sd} between nodes s and d . The third and fourth inequalities ensures that all flows are positive [2].

The controller application implements a discrete solution to the problem posed in Eqn. (4.7) by measuring each link data rate and determining the correct path through the network that minimizes the maximum link utilization. There is no penalty for longer paths through the network.

The results of simulations of random traffic using the Internet2 topology from Figure 27 are shown in Figure 28. Each data point in Figure 28 is the mean link weight of a Monte Carlo simulation of traffic generated by all 34 nodes directed towards node 20, Nashville. As more control nodes are added, the mean minimum link weight increases with decreasing returns after four control nodes. The zero control node case is the result of using only static routes. The order of the nodes used is based on the results of the dual-basis analysis as demonstrated in Figure 27. The control nodes in order of control centrality are: Chicago, Houston, Salt Lake City, Atlanta, El Paso, and Sunnyvale.

The load-balancing inequalities in Eqn. (4.7) are solved and that solution is implemented by the controller application using only the control nodes as source nodes. The controller is dynamically assigning flows for the packets that pass through control nodes. All other packets are routed using static flows. The simulation was conducted under a static control node analysis. A dynamic control node analysis could be implemented, and the control nodes can be updated dynamically by the controller

application. These dynamics could change which nodes are the most influential and should be assigned as the control nodes.

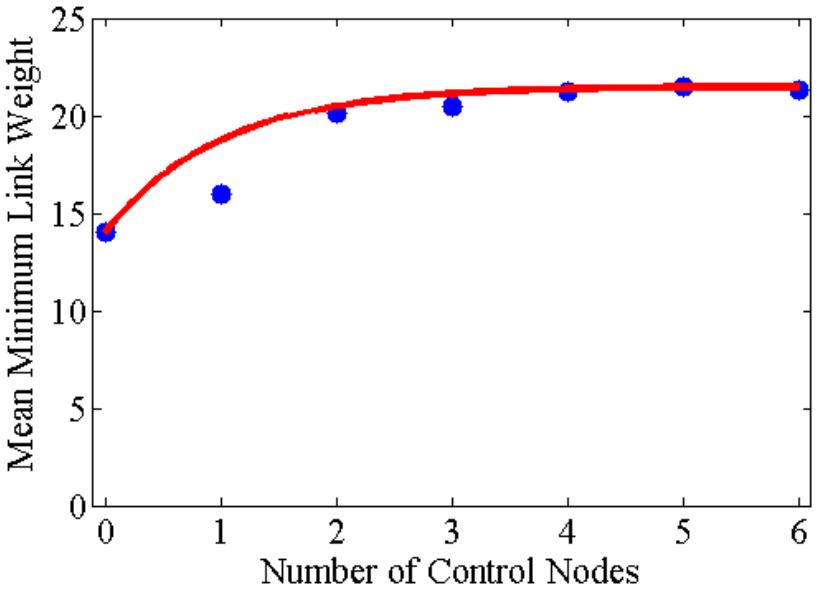

Figure 29. The mean minimum link weight increases as the number of control nodes increases and is maximized when four control nodes are used.

In summary, the development of the SDN control scheme used the closed-loop control system as a framework. Using the concepts of observability and controllability, methods were identified that can reduce the number of nodes that must be observed and controlled. Additionally, methods were identified that can be used to estimate link data rates and attempt to balance those data rates using flow control. Load balancing was selected as the objective of the routing algorithm because an objective of this dissertation is cybersecurity and preventing successful DDOS attacks. The implementation of these concepts in a SDN test bed is demonstrated in the next chapter. The experimental results obtained from the SDN test bed are shown in Chapter VI.

V. METHODS

Simulation and analysis alone are not sufficient to show the effectiveness of the dual-basis methodology in real-world situations. To move the research forward, a SDN test bed was needed. The hardware and software running in real-time with real-world inputs were required to determine the effectiveness of the proposed closed-loop control scheme. The closed-loop control scheme has been described for a generic network, but in this chapter, a specific description of the SDN test bed used to acquire the results in the next chapter is described. First, the SDN test bed is described and then the dual-basis analysis of the test bed is presented.

A. SDN TEST BED DESCRIPTION

A SDN test bed was built to add realistic complexity that was lacking in the analysis and simulations from the previous two chapters. To demonstrate that the dual-basis methodology and closed-loop control scheme would work in a real deployment of a SDN required building a SDN in hardware and software with real hosts on the network. Virtual network and virtual machines (VMs) were considered, but there were too many drawbacks. Emulation with a virtual environment is not an effective means to test implementations because all of the traffic passes through a single network interface card (NIC) on the computer being used. This feature makes it difficult to ensure repeatable data rates between experiments. Accurate data rate measurements are a requirement for the dual-basis implementation. Without repeatable experiments and accurate measurements, the results presented would not be relevant.

1. Implementation of the Proposed Closed-Loop Control Scheme in Software

To achieve closed-loop control, the controller and observer from Figure 19 were written as individual applications to be run simultaneously and interact with the SDN-enabled switches. The SDN operating system chosen was Ryu [54], an open source software package developed by Nippon Telegraph and Telephone, NTT. It was chosen because Ryu is well documented and easy to use. The applications are written in Python

2.7 [55]. Ryu is the network operating system that manages the interface between the Python applications and the hardware. It uses the OpenFlow protocol to communicate with the hardware switches. The two Python applications and the MATLAB script implement the observer and controller functions as previously described and pictured in Figure 20. The overall architecture is shown in Figure 30.

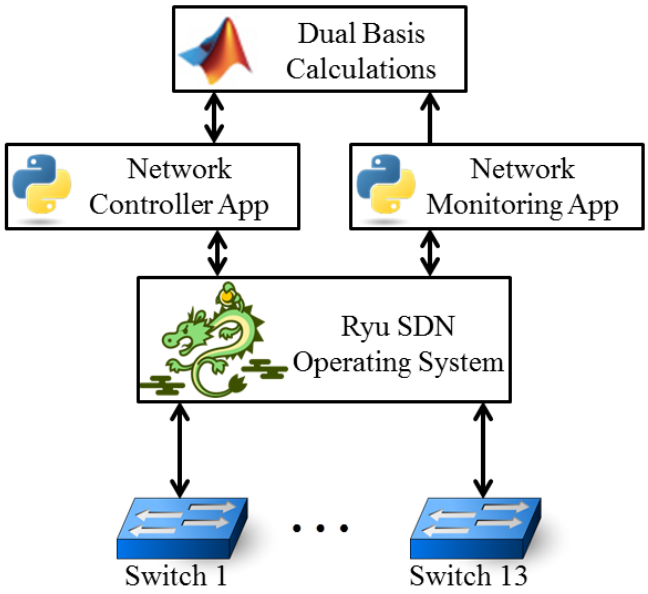

Figure 30. The implementation of the SDN test bed included 13 hardware switches, Ryu as the operating system applications written in python that directly interacted with the switches. MATLAB executed the calculation of link weights, the dual-basis, and the particle filter.

The foundation of the SDN is Ryu because it facilitates communication between the software and the hardware. The protocol running between Ryu and the switches is OpenFlow 1.0. When executed, Ryu instantiates both the controller application and the monitoring application as separate threads to run on a single machine. The monitoring application sends StatsRequest messages at fixed, one second intervals [21], [54]. The replies are parsed and sent to a MATLAB script, which uses this information as the current measurement input to the particle filter function. Once the data rates are estimated, they are passed to the controller to be used when routing packets in the network. The controller application receives packet_in messages from the switches and

generates packet_out and flow_mod messages for the switches to forward packets to the destination and to build the flow tables in the switches, respectively [21], [54].

The controller application also tracks the number and type of flows on each link. To accomplish this task, cookies were used to track each flow as it was created and removed. When a flow was created, it was assigned a cookie by using the MD5 message-digest algorithm to hash together the current time, switch datapath identification number, and destination IP address. When a flow times out due to a hard timeout or an idle timeout, a FlowRemoved message is sent from a switch [21]. The message contains the cookie that was assigned when the flow was created. This allows the controller to keep track of how many flows are assigned to each link [21], [54]. The flow count information is passed to MATLAB, which uses it as the current input in Eqn. (4.3) for the particle filter.

The observer updated the link state estimates approximately every second. The network monitoring app sent a statistics request to each monitored switch in the network once a second. The link state updates were event driven. When the network monitoring application receives a statistics reply, the application updated the link state matrix, which was passed to a MATLAB script to update the dual-basis representation. When making routing decisions, the network controller application would wait for all of the monitor nodes to reply prior to updating the routing tables for each control node.

2. Topology Modeled after Internet2

The software-defined piece of the SDN can be segregated from the underlying topology. The applications are able to learn and adapt to any given topology. The 13 node topology that was chosen was a subset of the full Internet2 topology as shown in Figure 31. The selection of these 13 nodes was based on the degree of each node. All nodes with a degree of one were removed. The next set of nodes removed were those with a degree of two. Additionally, the dual-basis analysis was compared between the 13 node topology and the 34 node topology to ensure that the eigenvalues and eigenvectors of the reduced topology were as similar as possible to the full topology. As shown in Figure 28, the most central nodes were Chicago, Houston, Salt Lake City, and Atlanta. A design choice for

the reduced topology was to ensure that the most central nodes were as similar as possible to the full topology. As will be shown, the most central nodes of the reduced topology are Chicago, Houston, and Salt Lake City. Atlanta is not included, but that is to be expected because there should be fewer control nodes in a 13 node network than in a 34 node network.

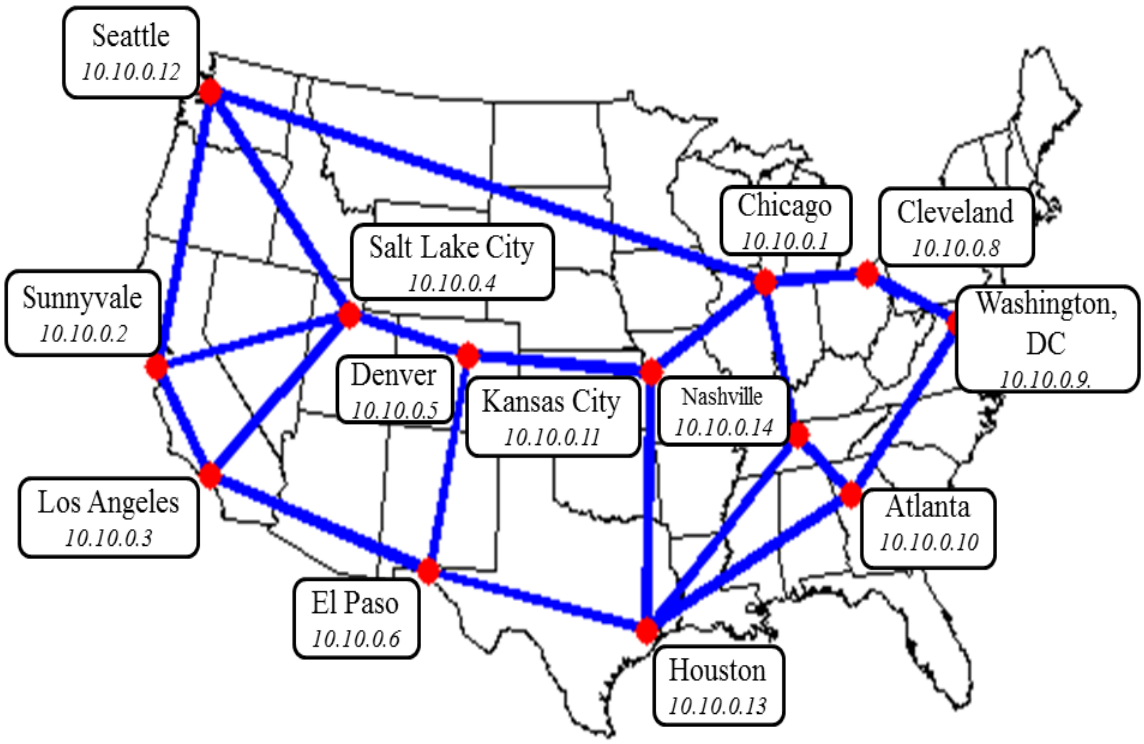

Figure 31. The reduced Interent2 topology used in the SDN test bed. Each city in the topology is listed with its associated IP address.

3. Hardware Components

The hardware in the test bed included HP switches and Raspberry Pis. Two types of switches were used in the network: HP 2920 and HP 3800 [56], [57]. The 3800 model is more capable than the HP 2920, but in the SDN test bed, there was no discernible difference between the two types. The hosts in the network are Raspberry Pis [58], which are small, inexpensive computers with 10/100 Mbps Ethernet connections. They are used to generate enough traffic to conduct DDOS attacks and more realistic day-in-the-life traffic. The Raspberry Pis ran one of four operating systems: Raspbian, ArchLinux, Kali,

and Windows 10. All of the Raspberry Pis were configured with Iperf [59], which was the program used to generate traffic between the host and server. One or more Raspberry Pis run the server side of Iperf, and the remaining Raspberry Pis are hosts sending traffic.

Iperf was used in UDP mode because this provided the most control over data rates, and it does not implement any congestion control algorithms. For this reason, UDP was used to produce the traffic profiles that are shown in the next chapter. TCP was not used because the congestion control algorithms would have been a component of the experiment that could not be controlled. The goal was to have the controller mitigate congestion by load balancing traffic as opposed to having the transport protocol mitigate the onset of congestion.

B. DUAL-BASIS ANALYSIS OF TEST BED TOPOLOGY

A full dual-basis analysis was conducted as the first step in the application development process. This research did not implement a dynamic application to update the monitor nodes and control nodes. It did, however, track congestion dynamically and attempt to maximize the minimum link weight, as described in Eqn. (4.7). The first step of the analysis was to determine the minimum number of nodes to ensure that all link data rates can be calculated.

1. Identification of Observed Nodes

For the network in Figure 31, the minimum number of monitor nodes required to calculate all data rates is eight. This result was obtained using the minimum vertex cover algorithm proposed in [60]; see Chapter IV Section B. In Figure 32, the nodes determined by the minimum vertex cover solution are highlighted in yellow. After determining the monitor nodes, the network controller must identify the control nodes.

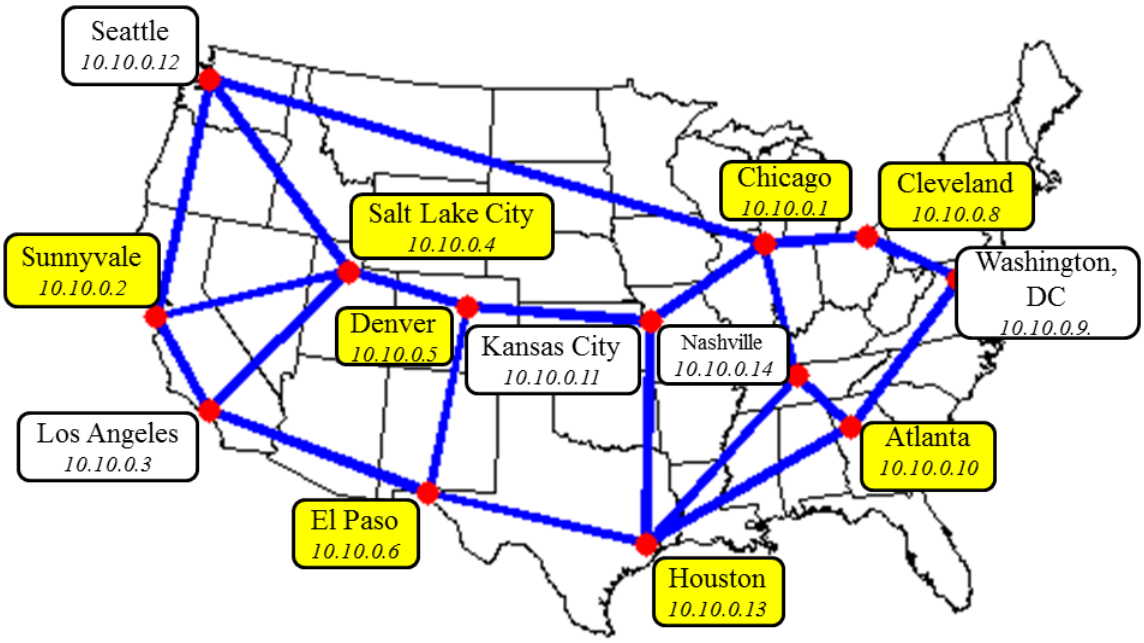

Figure 32. The set of monitor nodes in the test bed is designated by yellow. These nodes were identified by using the solution to the minimum vertex cover problem.

2. Identification of Control Nodes

The identification of the control nodes follows the process outlined in Figure 27. The first step is to determine which nodes have the largest $E_{c:n}^{j}$ in the leading eigenvectors. The first set of calculations uses the first two eigenvectors. The nodes with the largest $E_{c:n}^{j}$ in the first two eigenvectors are the first two candidate control nodes, and they are Chicago and Salt Lake City. It is worth noting the difference in the $E_{c:n}^{j}$ between Salt Lake City and Houston was 0.0576. This small difference is indicative that Houston may be added to the set of control nodes. However, the angle between Chicago and Salt Lake City is 61.57°, which is not near orthogonal as seen in Figure 33. The angle between Chicago and Salt Lake City and the small difference of the norm between Salt Lake City and Houston indicates that all of the necessary control nodes have not been found. Since they have not been found, the process was repeated with three eigenvectors.

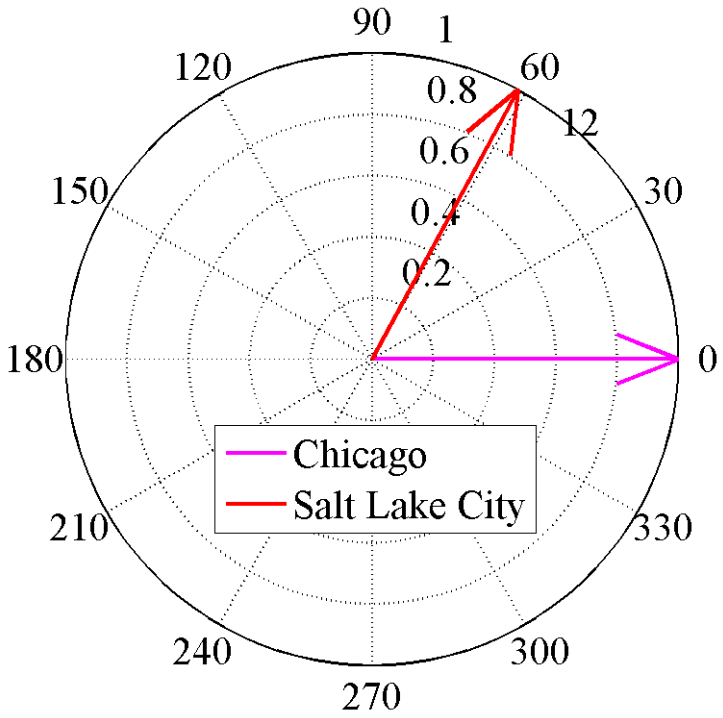

Figure 33. The angle between Chicago and Salt Lake City is shown using the leading two eigenvectors of the dual-basis representation.

When using the leading three eigenvectors, the candidate control nodes in order are Salt Lake City, Houston, and Chicago. The difference between Chicago's $E^j_{c:n}$ and the next largest $E^j_{c:n}$ is 0.2809. This large difference is a good indicator that a sufficient number of control nodes have been found. From Table 1 and Figure 34, it is clear that there Chicago is near orthogonal to Houston and Salt Lake City. The angle between Houston and Salt Lake City is not as clear. The next step is to add a fourth eigenvector and continue the process.

Table 1. The angle between the three candidate control nodes shows the degree to which the candidates are decoupled.

	Salt Lake City	Houston	Chicago
Salt Lake City	0°	76.22°	89.32°
Houston	76.22°	0°	86.23°
Chicago	89.32°	86.23°	0°

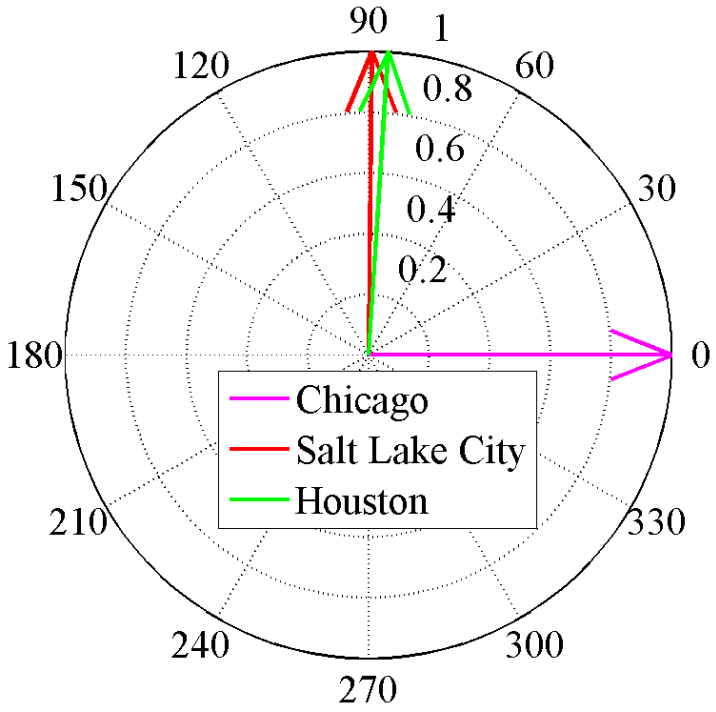

Figure 34. The angle between Chicago and Salt Lake City and between Chicago and Houston is shown using the leading three eigenvectors of the dual-basis representation.

By adding a fourth eigenvector to the analysis, four candidate control nodes are obtained: Salt Lake City, Houston, Chicago, and Sunnyvale. From Table 2 and Figure 35, it is clear that Sunnyvale is the least orthogonal node to the others. An angle of 118.08° is the farthest from 90° of all the nodes. The process stops here because Houston's angle has become closer to orthogonal with four eigenvectors, and Sunnyvale's angle is much greater than all the others in the four vector case and the three vector case.

Table 2. The angle between the four candidate control nodes shows the degree to which the candidates are decoupled.

	Salt Lake City	Houston	Chicago	Sunnyvale
Salt Lake City	0°	78.44°	87.61°	118.08°
Houston	78.44°	0°	86.99°	100.71°
Chicago	87.61°	86.99°	0°	77.31°
Sunnyvale	118.08°	100.71°	77.31°	0°

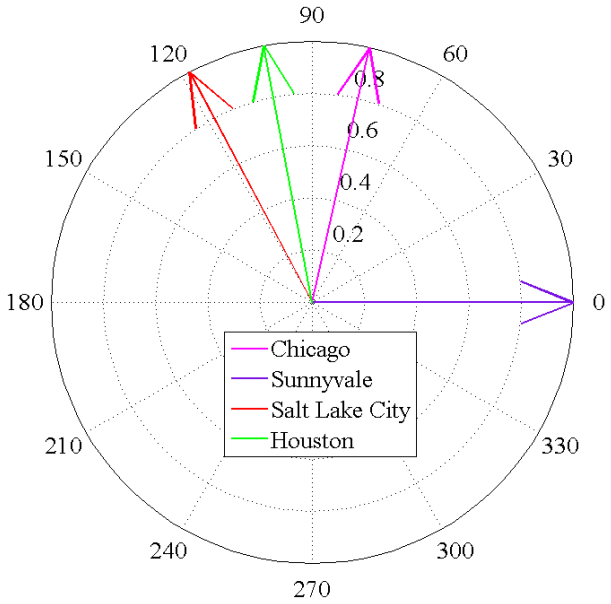

Figure 35. The angles between Sunnyvale and Chicago, Salt Lake City and Houston are shown using the leading four eigenvectors of the dual-basis representation.

The three candidate nodes are Salt Lake City, Houston, and Chicago. In this analysis, the results of Eqn. (4.6) order the nodes from most influential to least as Chicago, Salt Lake City, and Houston. The control centrality of Chicago is 0.1630, Salt Lake City's is 0.0536, and Houston's is 0.0422. The static, unweighted analysis stops here, but this control node assignment needs to be tested in the test bed to ensure that they are sufficient.

The design work that must be conducted prior to implementing any of the applications is based on the analytical work presented in Chapters III and IV. In this particular case, the monitor and control nodes were not updated dynamically; an offline analysis was done and then implemented online, in real-time. There is no reason the monitor and control nodes cannot be identified in real-time. The goal of the next chapter is to demonstrate that the assignment of monitor and control nodes as laid out in this chapter provide enough information to calculate all link weights and enough control to properly balance the offered traffic.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RESULTS

Once the monitor nodes and the control nodes have been identified, the next step is to experimentally verify the unweighted analysis. The goal of these experiments is to determine the accuracy of identification of the correct control nodes. The method to find control nodes was based on the assumption that an analysis of an unweighted graph was sufficient. However, these results challenge that assumption and suggest that a more detailed, dynamic analysis is required. Results using a static, unweighted analysis will be shown for a case with a server on the East Coast and then on the West Coast. These same cases are revisited based on a weighted, dynamic analysis to show the improvement in performance over the unweighted analysis.

A. EXPERIMENTAL RESULTS OF LOAD BALANCING CONTROL USING CONTROL NODES

In accordance with Eqn. (4.7), the network controller was programmed to maximize the minimum link weights surrounding the server node located at one location in the network and all other nodes transmitting to that node. Two server locations were chosen to show two different traffic patterns; they were Nashville and Sunnyvale. The network consisted of 50 hosts: one server, one command and control (C2) host, and 48 transmitting hosts. The server was a Raspberry Pi running Iperf as the server [59]. The C2 host logged into each transmitting host via a secure shell (SSH) and instructed them when to start transmitting to the server, for how long and at what data rate. The transmitting hosts used the client feature of Iperf.

Three traffic profiles were used to verify the behavior of the network under various traffic loads. The pyramid profile is shown in Figure 36. All of the Raspberry Pis attached to a given node were instructed to begin transmitting before the next node was initiated. This simple profile was used to ensure that the applications were working correctly and to provide a repeatable profile for each experiment. The second profile is the mountain profile as shown in Figure 37. This profile randomized the order in which the hosts were instructed to start transmitting. This profile was used to remove the bias

that was present in the first profile; the bias is evident in some of the results because multiple uncontrollable flows were initiated consecutively. The final profile was a non-deterministic profile that randomized the order in which hosts transmitted, the length of time that the hosts transmitted, and the amount of time between initiating transmissions. A single realization of the non-deterministic profile is shown in Figure 38.

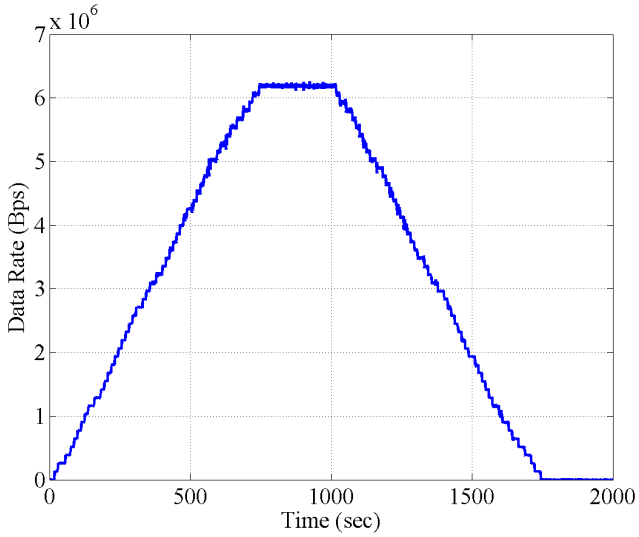

Figure 36. The pyramid traffic profile was generated by 48 hosts transmitting at 1 Mbps.

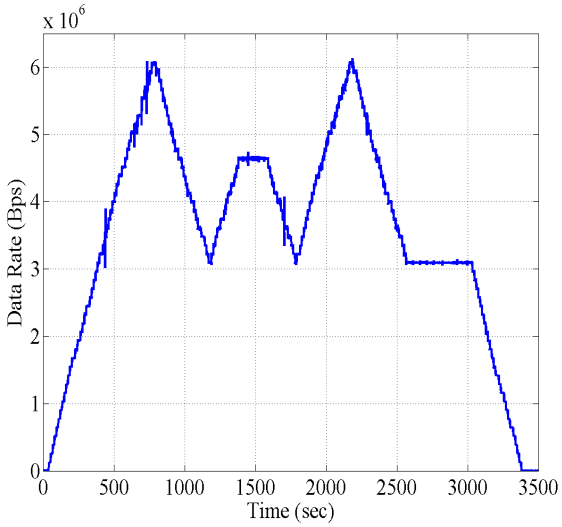

Figure 37. To generate this profile 48 hosts were used, but the transmitting order of the 48 hosts was different for each experiment.

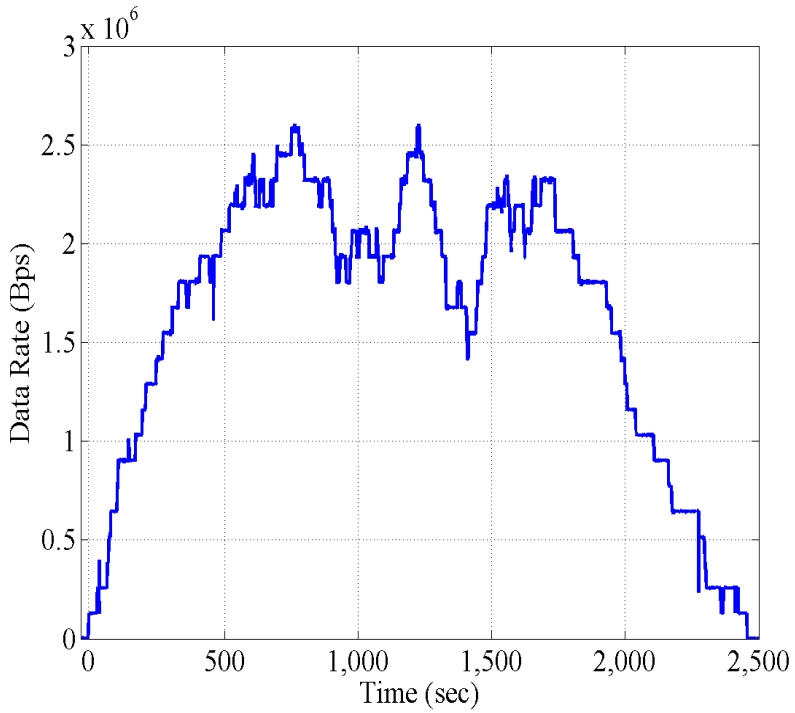

Figure 38. To generate this profile 48 hosts were used, but the transmitting order, length of transmission, and the length of time between transmitting was different.

1. Particle Filter Results

The particle filter was implemented in real-time, and the estimated data rates were used to determine the link weights of the links surrounding Nashville and Sunnyvale. Measurements were made every second, and the weighted graph was updated given the measurements. The data rate estimates for each link surrounding the server node was used to calculate the link weights, which were then used in the controller's decision to route traffic to each server location. The results of data rate estimates and data rate measurements are shown in Figures 39 and 40. The detailed performance of the particle filter is shown in the inlay. Because the noise variance used was small in both the predict and update phases, the estimates are biased towards the predict phase that uses non-linear model from Eqn. (4.3). The update phase uses the variance to determine the probability that a given realization of the Monte Carlo simulation is the actual link data rate. By using a small variance, measurements that were far from the predicted data rate were assigned a low probability of being the actual data rate. The data rate for each of the three

links surrounding the server nodes was estimated using 500 particles per iteration. This choice was made to limit the time it took the SDN controller to calculate the data rates and decide on the next set of routes, which was completed every second. Additional links could have been added in a similar manner if the SDN controller was run on a more capable computer.

The particle filter required the controller to keep track of the number of flows on each link to use Eqn. (4.3). In addition, the controller needed each switch to notify the controller when a flow had been removed. The switches and firmware that were used did not reliably transmit a flow removed message to the controller. The flow idle timeout feature did not appear to function properly and as such, it did not provide reliable feedback to the controller when flows had ended due to idling. To overcome this, the length of the transmissions was fixed, except for the non-deterministic profile. For this profile, a moving average was used instead of the particle filters to estimate the link data rate and link weights.

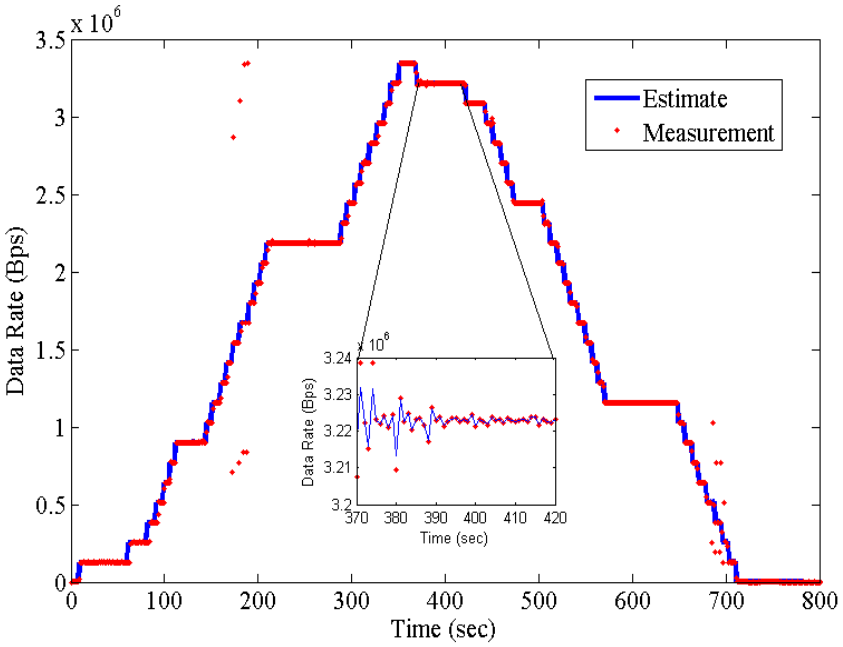

Figure 39. To estimate the data rate for each link surrounding the servers 500 particles were used, and the particle filter was successful at eliminating outlying measurements. The inset demonstrates the performance of the filter at a level that one can see the particle filter's performance.

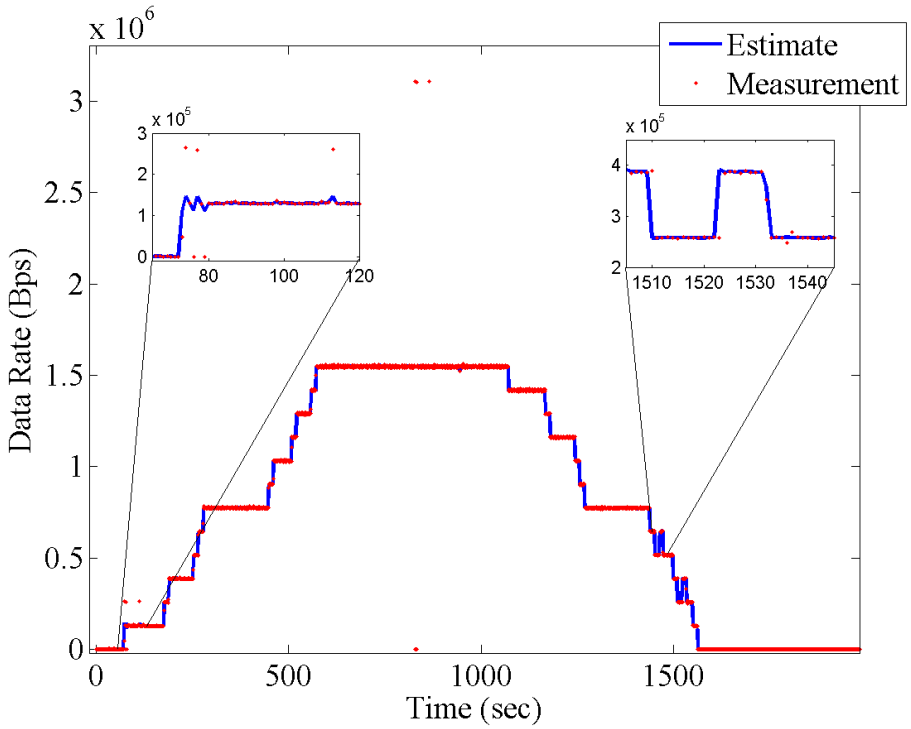

Figure 40. The particle filter was an effective means to limit the impact of outlier measurements on the routing algorithm. The inset demonstrates the performance of the filter at a level that one can see the particle filter's performance.

2. East Coast Results

The first experiment was run with the server in Nashville. The first set of results was obtained using the pyramid profile. Experiments were run using zero, one, two and then three control nodes. The results for all static flows are shown in Figure 41. with the link weights calculated using Eqn. (3.24). Larger link weight values are associated with lower link utilization and higher available capacity. The three links closest to the server are shown because those are the links which are most important to ensure the minimum link weight is maximized.

When Chicago is assigned as a control node, any traffic transiting through Chicago will be rerouted to minimize the maximum link weight. The route is instantiated using proactive routing to ensure the full path from control node to the server is established. The results are shown in Figure 42. Notice that the minimum link weight

decreases when a single control node is added. The reason for this decrease in performance is because early in the build-up of traffic the controller attempts to balance the traffic and adds traffic to the link from Houston. The controller does not have enough control input throughout the rest of the experiment to effectively minimize the link weights. This result clearly indicates that a single control node is insufficient to obtain the desired balancing of the offered traffic and missed placed control nodes can decrease performance.

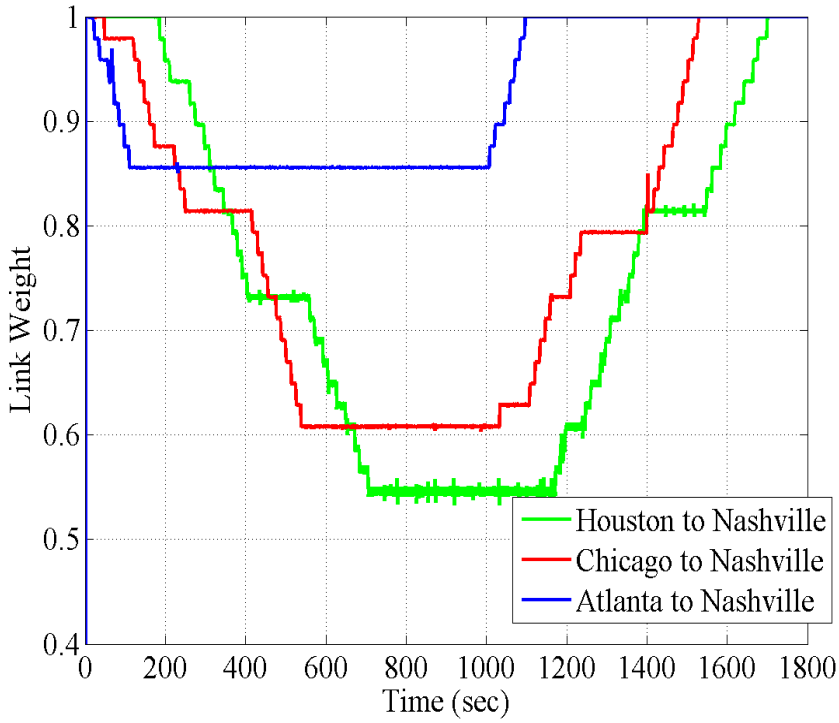

Figure 41. The link weights for the three links connected to the server node in Nashville are shown for the pyramid profile with zero control nodes. Static routes were used by all nodes in the network.

When Salt Lake City is added as the second control node, the results in Figure 42 are unchanged. Adding Salt Lake City only controls those few hosts that are directly connected to Salt Lake City; however, those same hosts were already being controlled by Chicago. The static route from Salt Lake City to Nashville transited through Chicago. Increasing the number of control nodes from one to two did not increase the ability of the controller to load balance the traffic.

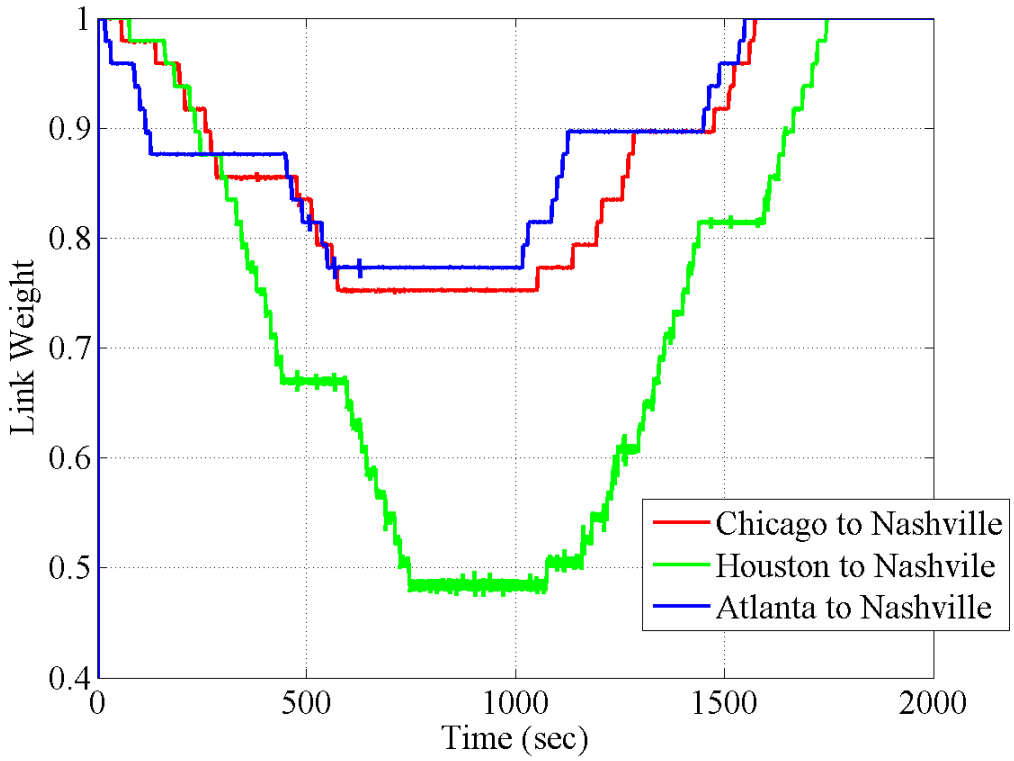

Figure 42. The plot of link weights over time is shown using Chicago alone and using Chicago and Salt Lake City as the control nodes based on an unweighted analysis. The results for the one and two control node case are identical.

For this server location, the result in Figure 42 suggests that the correct ordering of the nodes is not Chicago, Salt Lake City, and then Houston; the best ordering for this server location is Houston, Chicago, and then Salt Lake City. The results when Chicago and Houston are the control nodes are shown in Figure 43. A significant increase in performance is shown over the previous results. Because all traffic from west of Chicago and Houston must go through Chicago and Houston, all of that traffic can be used by the controller to balance the links, and the traffic from Washington, DC and Atlanta is not. Even though some traffic is not available for balancing the load, the controller is able to balance the traffic quickly because there are enough controllable flows to achieve a minimum, and the minimum link weight is increased by 13% over the static routes as shown in Figure 41.

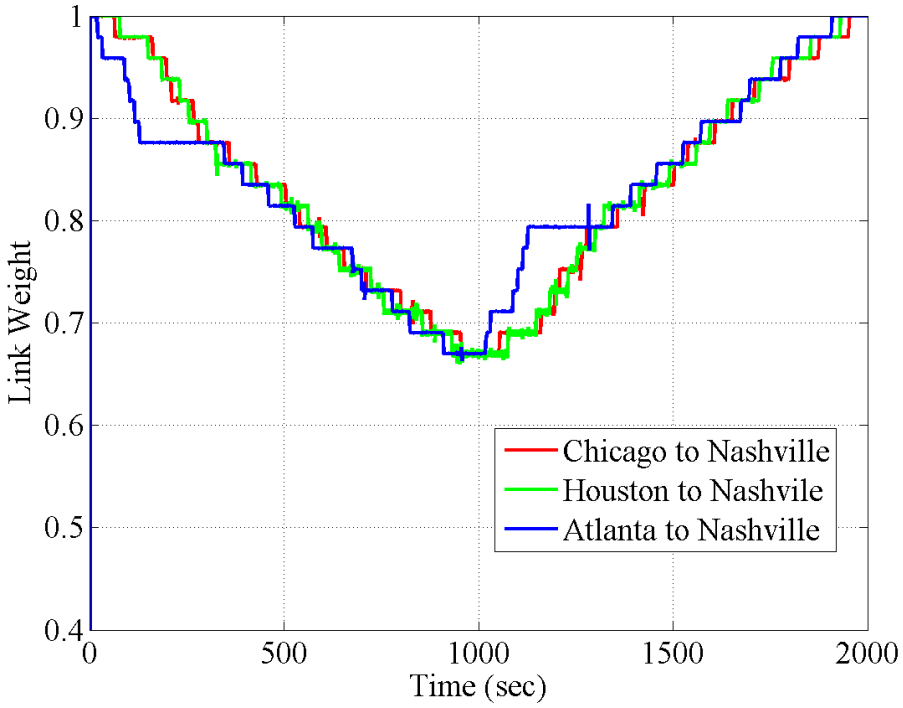

Figure 43. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City and also Chicago and Houston as the control nodes based on an unweighted analysis. The results for the two and three control node case are identical.

The second set of results was obtained using the mountain profile. The zero control node case is shown in Figure 44. The three control node case is shown in Figure 45. with Chicago, Houston, and Salt Lake City as the control nodes. Again, the addition of Salt Lake City did not add any control in terms of balancing the link weights. Because the order in which hosts begin transmitting is random, there is no bias as seen in Figure 43 where all of the uncontrollable flows are initiated in order. The traffic is well balanced throughout the profile, and the minimum link weight is increased by 11% over the static routes as shown in Figure 44.

The third set of results was obtained using the non-deterministic profile. The results when using zero control nodes are shown in Figure 46. The results when using Chicago, Salt Lake City and Houston as the three control nodes are shown in Figure 47. The traffic from Atlanta that cannot be used when balancing the load, which is routed at

approximately 700 seconds, resulted in smaller link weights than the optimal solution. However, the minimum link weight in Figure 47 is 8.5% greater than the link weight in Figure 46.

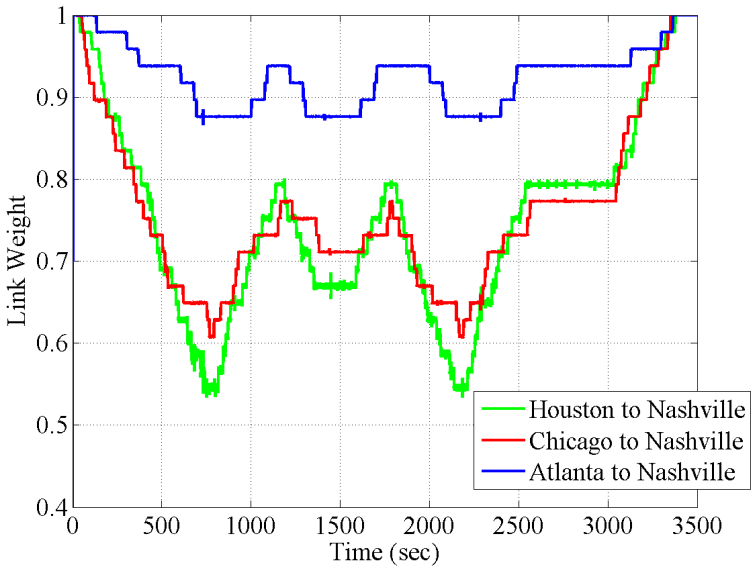

Figure 44. The plot of link weights over time is shown using zero control nodes with the mountain traffic profile.

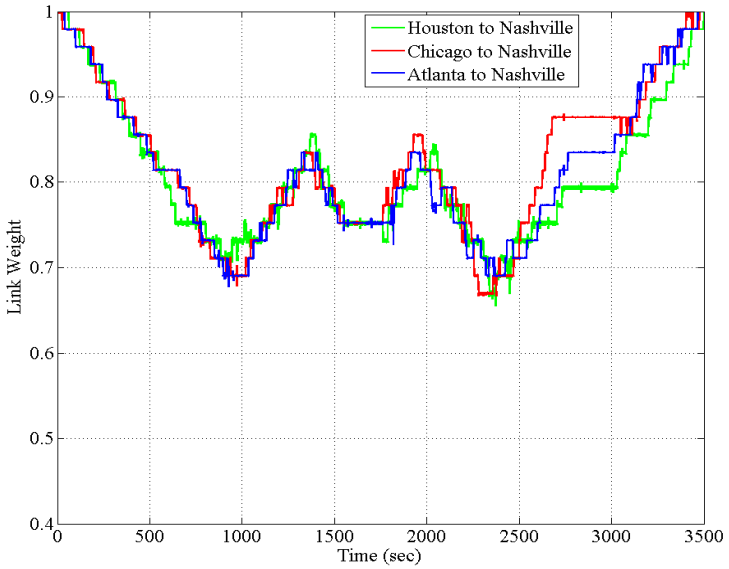

Figure 45. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City as the control nodes based on an unweighted analysis for the mountain profile.

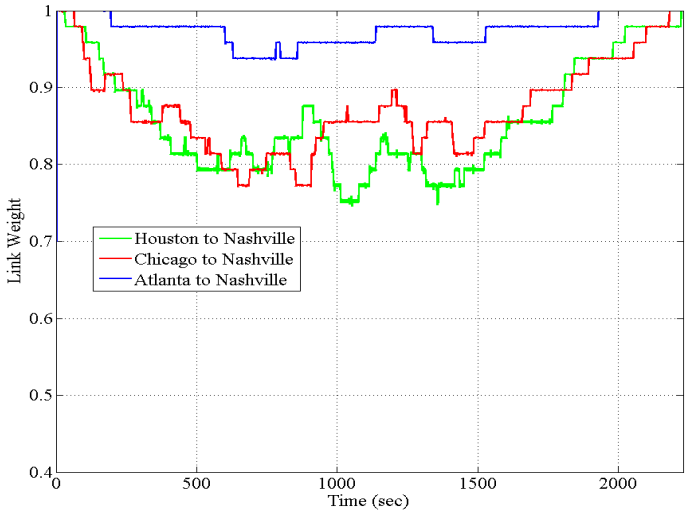

Figure 46. The plot of link weights over time is shown using zero control nodes with a non-deterministic traffic profile.

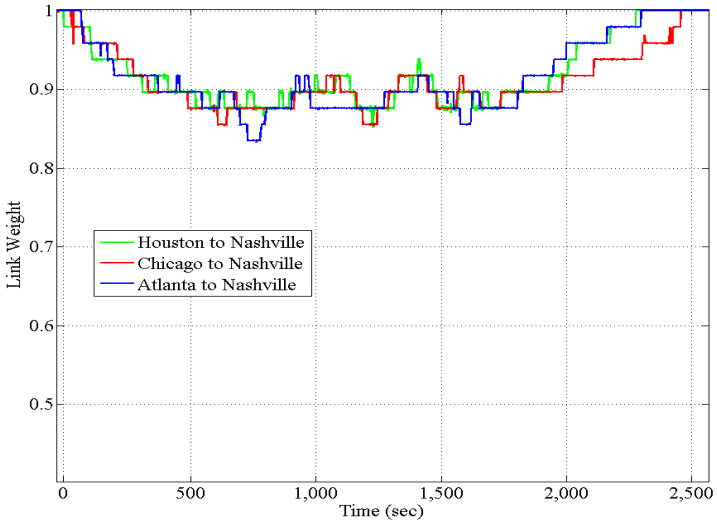

Figure 47. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City as the control nodes based on an unweighted analysis.

3. West Coast Results

Similar results were obtained for the West Coast server. The same ordering of control nodes was used as suggested by the static, unweighted analysis. The first set of results is shown in Figure 48 for the pyramid traffic profile. The results using the same three control nodes as the East Coast scenario are shown in Figure 49. Again, there is

good balancing of the link weights early in the experiment when using three control nodes, but it does not maximize the minimum link weight throughout the experiment. The links from Seattle and Salt Lake City are evenly balanced, but the link from Los Angeles ends up carrying more traffic because of the added traffic early in the experiment. This is a similar problem to that observed in the one control node experiment when the server was located in Nashville. The amount of traffic that is produced by the nodes in the southwest is a large portion of the total traffic and is not available to the controller to balance the load, and it does not enter the experiment until after much of the controllable traffic has been routed.

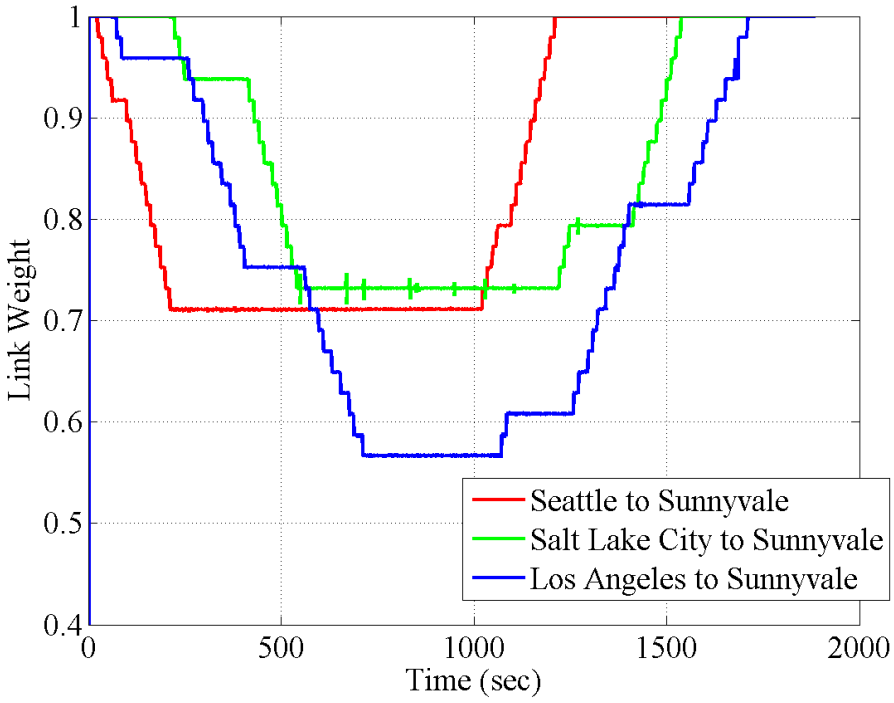

Figure 48. The plot of link weights over time is shown using zero control nodes with the pyramid traffic profile.

The second set of results for the West Coast is obtained using the mountain profile. Between the zero control node case and the three control node case, there is an increase of 8.3% in the minimum link weight. The zero control node case is shown in Figure 50, and the three control node case is shown in Figure 51.

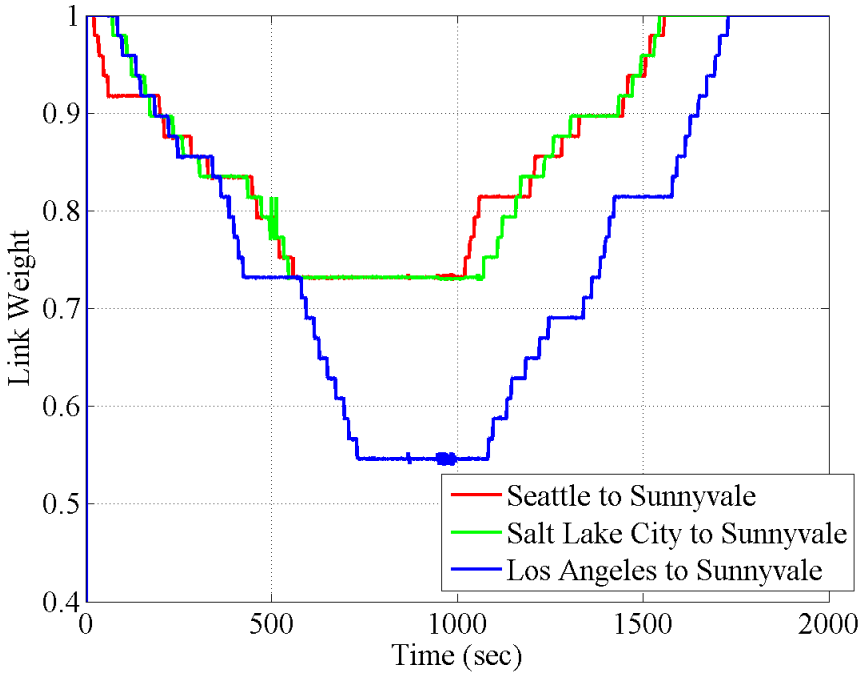

Figure 49. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City as the control nodes based on an unweighted analysis.

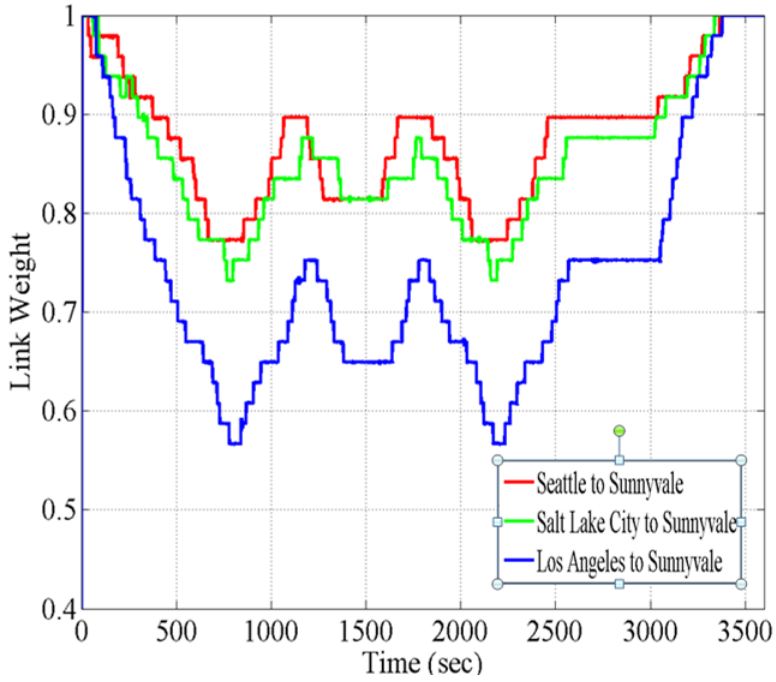

Figure 50. The plot of link weights over time is shown using zero control nodes with the mountain traffic profile.

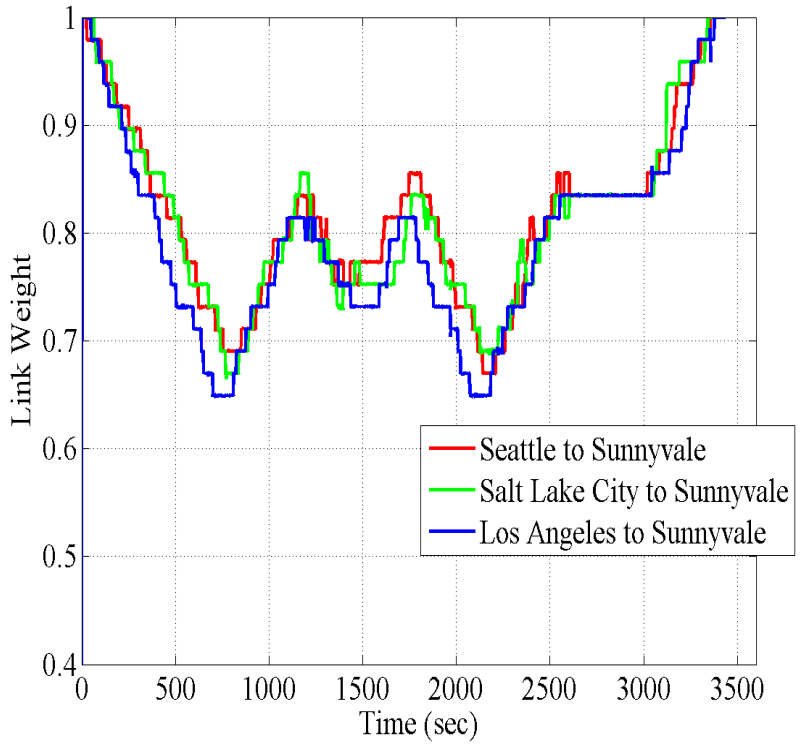

Figure 51. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City as the control nodes based on an unweighted analysis.

The next set of results for the server location in Sunnyvale is based on the non-deterministic profile. Again, the traffic from Los Angeles produces the minimum link weight as shown in Figure 52. When the three control nodes are added, the minimum link weight is increased by 8%. The balancing of the link weights in Figure 52 is not perfect. The traffic from Los Angeles is the limiting factor. All of these results suggest that Los Angeles should be added as another control node. However, the unweighted analysis did not indicate that Los Angeles should be included.

These results suggest that a better method is needed to select the control nodes based on both the topology and a traffic matrix. The locations of Chicago, Salt Lake City and Houston are near optimal choice for Nashville but not for Sunnyvale. Adding knowledge of traffic patterns as link weights to the principal eigenvector analysis will provide a more optimal solution. Additionally, if the network is able to recalculate the

principal eigenvectors and control nodes periodically based on the current traffic patterns, the controller can select the best nodes based on the current state of the network.

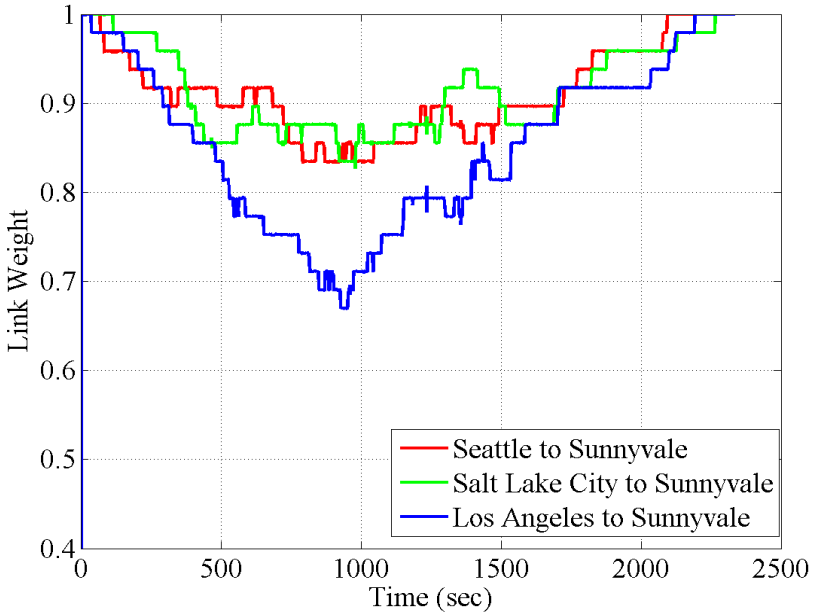

Figure 52. The plot of link weights over time is shown using zero control nodes with the non-deterministic traffic profile.

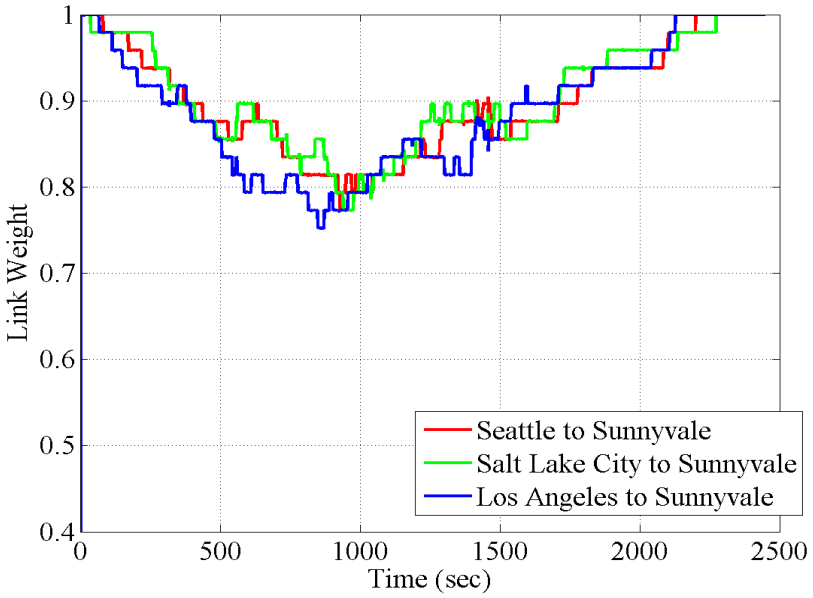

Figure 53. The plot of link weights over time is shown using Chicago, Houston, and Salt Lake City as the control nodes based on an unweighted analysis.

B. MODIFIED CONTROL NODE SELECTION METHOD

Based on the previous results, the assumption that a static, unweighted analysis is sufficient is challenged. A dynamic, weighted analysis may be more effective when attempting to identify control nodes. The procedure in Figure 27 is used to implement the weighted analysis, but this time the analysis includes information similar to that used to calculate the betweenness centrality [23].

1. Analysis of Internet2 Topology with Weighted Graph

The topology from Figure 31 does not provide any information about the location of subnets or hosts. The number of hosts at each location in the test bed topology is shown in Figure 54. Combining the location of the hosts and the location of the server, a weighted graph is developed. Each link is given a weight between 0 and 1 based on the number of flows that are transmitted over that link. Links that appear in the physical topology, but do not carry any flows are given a nominal link weight of 0.05 because by the definition of the adjacency matrix, a 0 indicates that there is no link. In any network, some small amount of traffic is carried over all links and a weight of 0.05 accounts for this.

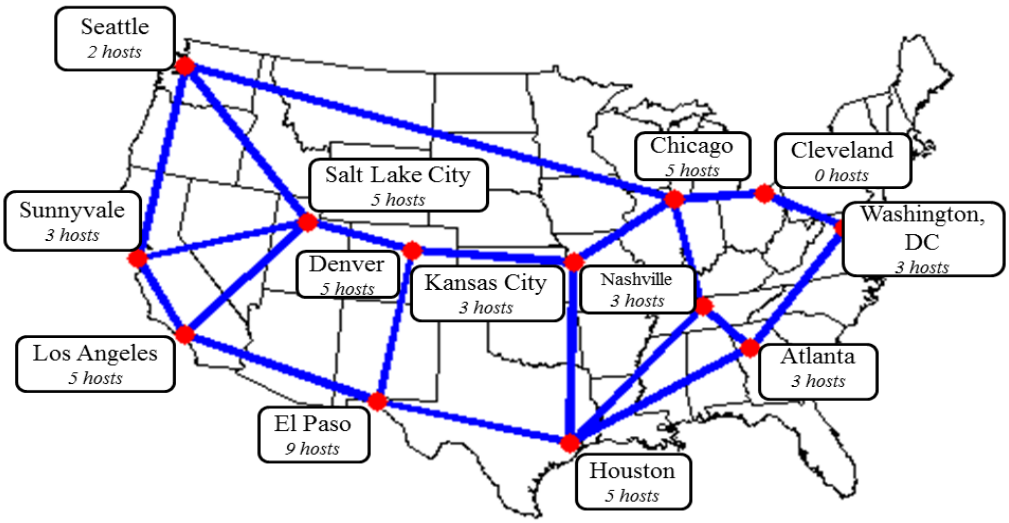

Figure 54. As in a real-world network, computers and traffic are not evenly distributed throughout the network, which is the case in the SDN test bed.

Once the traffic matrix has been developed and link weights assigned, the method to locate the control nodes can be run again. The results of the analysis indicate that the order of the control nodes when the server location is Nashville is Houston and then Chicago. This result was already observed in Figures 43, 45, and 47. Because Houston is listed first, the Nashville experiment was run again using Houston as a control node and those results are shown in Figure 55. The results show improved performance as opposed to using just Chicago, which resulted in decreased performance. The results of the weighted analysis indicate that the order of the control nodes when the server location is Sunnyvale is Salt Lake City, Los Angeles, and then Seattle. The resulting link weights when the new control nodes from the weighted analysis are used for the West Coast location are shown in Figures 56, 57 and 58. All three show much improved performance over the control nodes that are located using the unweighted analysis.

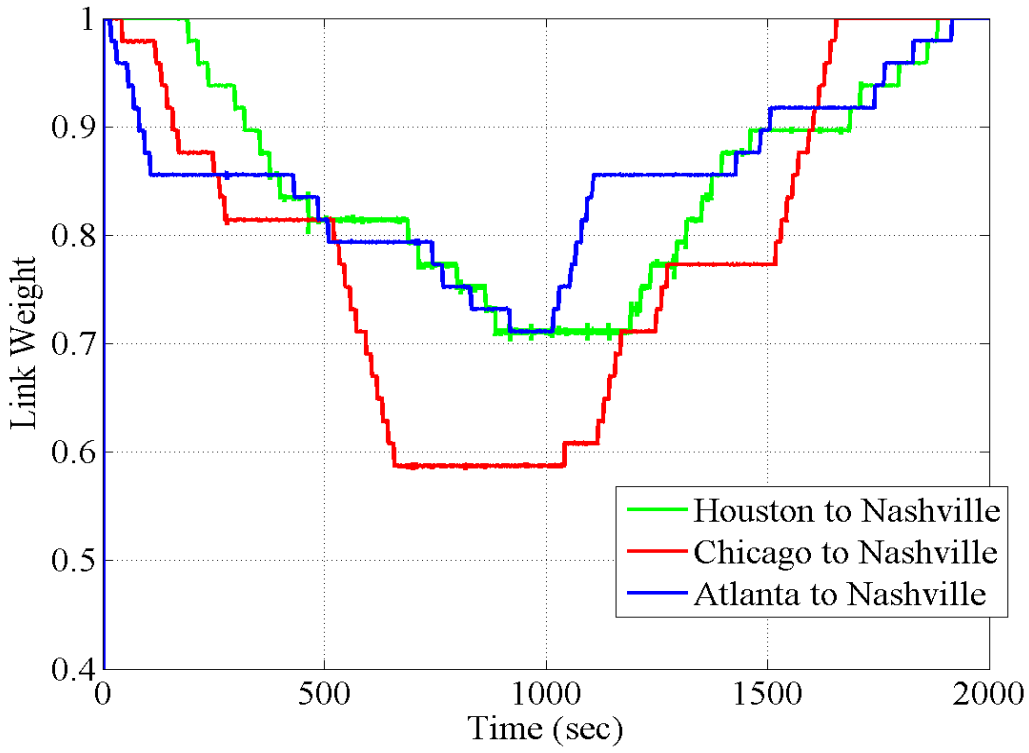

Figure 55. The plot of link weights over time is shown using Houston as a single control node based on a weighted analysis and pyramid traffic profile.

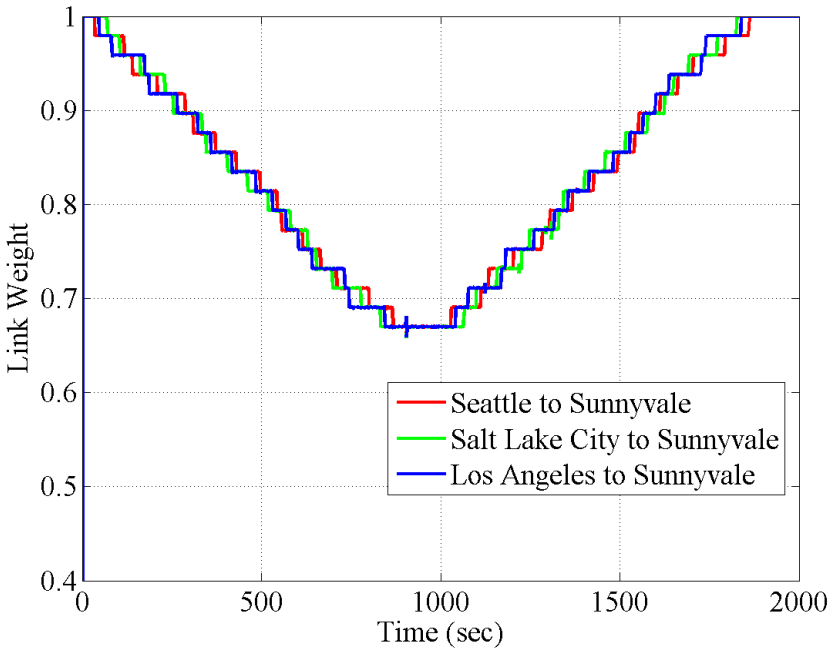

Figure 56. The plot of link weights over time is shown using Seattle, Salt Lake City, and Los Angeles as the control nodes based on a weighted analysis and pyramid traffic profile.

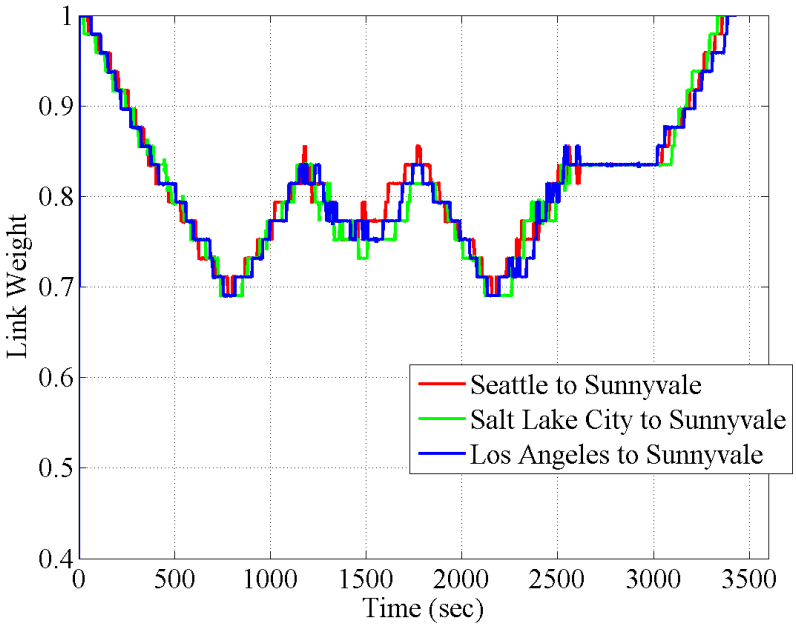

Figure 57. The plot of link weights over time is shown using Seattle, Salt Lake City, and Los Angeles as the control nodes based on a weighted analysis and mountain traffic profile.

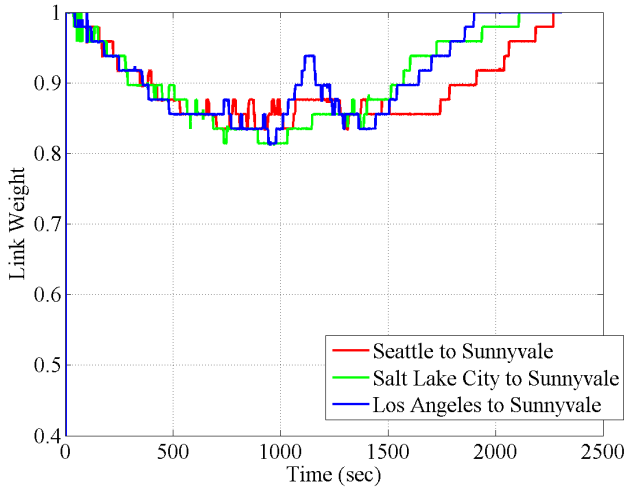

Figure 58. The plot of link weights over time is shown using Seattle, Salt Lake City, and Los Angeles as the control nodes based on a weighted analysis and non-deterministic traffic profile.

By allowing the SDN controller to recalculate the dual-basis representation and the control nodes, the network can adjust for changing and unexpected traffic conditions. The dynamic, weighted analysis is a much better solution to the control node assignment problem than the static, unweighted method.

2. Analysis of a Two-Server Network

A weighted, dynamic analysis is a more effective solution for the one server case, but in a real-world network, there are multiple destinations. In order to evaluate this weighted analysis with a more realistic scenario, the method to determine the control nodes was revisited for the two-server case. The two server locations are Nashville and Sunnyvale, but this time they will both receive traffic simultaneously. The weighted graph will take into account the flows transiting in both directions. After conducting this analysis, the resulting control nodes, in order, are Chicago, Los Angeles, Houston, and Salt Lake City.

The new list of control nodes includes a fourth control node, which is not an unexpected result in light of the previous West Coast results. Chicago, Houston, and Salt Lake City all have a degree of four and are hubs for the network. Los Angeles does not

have a degree of four, but it does have a large amount of traffic flowing through it to both Sunnyvale and Nashville, which increases its ability to control the offered traffic. The control node identification method eliminates the guess work from network design by reducing a complex network to the analysis of a small number of principal eigenvectors. The determination of the principal eigenvectors and the resulting control nodes can and should be automated in a real-world implementation.

The results were collected by running the experiments again, but this time the four control nodes were used and each profile was applied to each server location. The results show that the weighted case can work for more than one server location. There is significant increase in the link weights for each case shown, as compared to the static case. The East Coast results are not reiterated here; they are contained in Figures 43, 45, and 47. Adding control nodes to the West of Chicago and Houston does not increase the performance of the balancing of traffic to Nashville. The West Coast results are shown in Figures 59, 60, and 61. The performance here is not that much better than the one server case, but there is significant improvement over the static case.

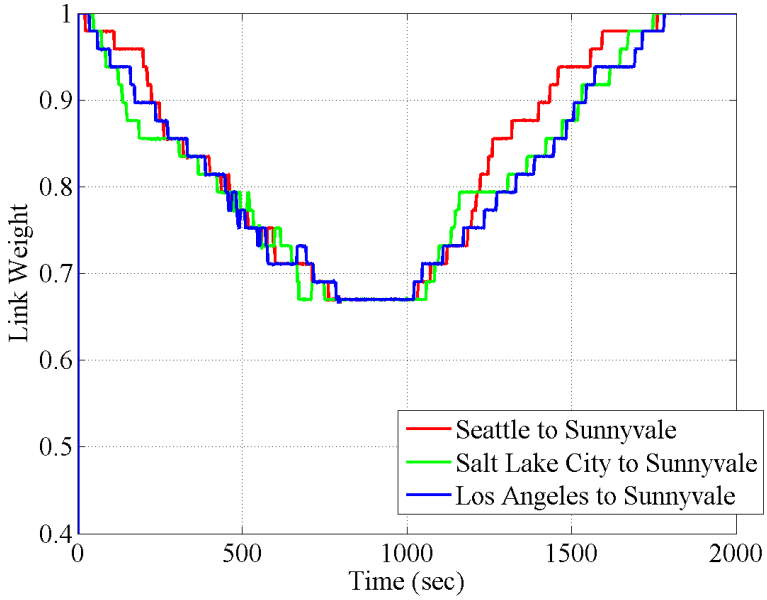

Figure 59. The plot of link weights over time is shown using Chicago, Los Angeles, Houston, and Salt Lake City as the control nodes based on a two-server, weighted analysis and pyramid traffic profile.

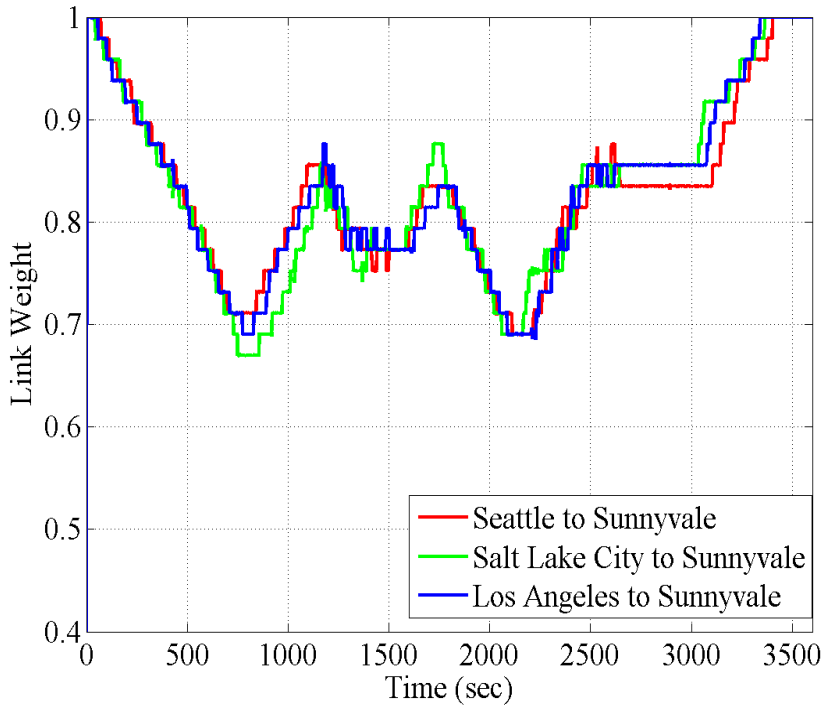

Figure 60. The plot of link weights over time is shown using Chicago, Los Angeles, Houston, and Salt Lake City as the control nodes based on a two-server, weighted analysis and mountain traffic profile

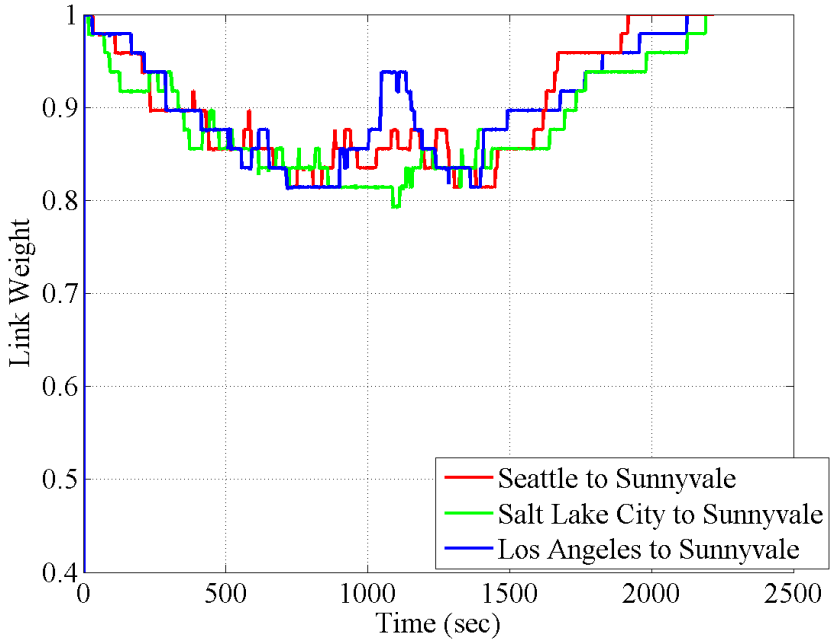

Figure 61. The plot of link weights over time is shown using Chicago, Los Angeles, Houston, and Salt Lake City as the control nodes based on a two-server, weighted analysis and non-deterministic traffic profile.

The results shown for both weighted cases were developed based on a known traffic matrix. In real-world applications, this traffic matrix may be difficult to obtain. However, the weighted graph that the controller develops for each time step to determine the phantom node's location in the eigenspectrum may be used as a substitute for the known traffic matrix. If this is true, no additional information is required in order to determine the control nodes.

The previous results are summarized in the following tables. Table 3 shows a scenario where adding control nodes did not increase performance. This is mainly because offered traffic needs to be available to the controller for it to balance the traffic. If the controller does not have traffic available to balance the link weights, it will not be able to accomplish its objective.

Two methods could be used to prevent this scenario. First, the assignment of control nodes could be dynamically updated during the experiment. This was not implemented in these experiments. Second, flows could be deleted from the switches and re-routed. This option was not implemented in these results. If the controller is too slow to re-route the flows, this option could result in dropped packets as new flow rules are sent to the switches.

In Tables 3, 4, and 5, the largest increase is 13%. The reason that the pyramid profile produces the largest increase is because it had the worst performance in terms of load balancing when using static routes. The smallest increase is in Table 5, an increase of 4.2% for the random profile. The random profile also has the smallest increase on average. The random profile is one of the many possible realizations and as such will have different performance results for each realization. The random profile also had smaller offered loads than the other profiles. This could account for the smaller performance increases.

Table 3. The change in the minimum link weight is presented when using the three control nodes identified by the unweighted analysis and one server location at a time.

	Pyramid	Mountain	Random
Nashville	13%	11%	8.5%
Sunnyvale	-2.1%	8.3%	8%

Table 4. The change in the minimum link weight is presented when using the three control nodes identified by the weighted analysis and one server location at a time.

	Pyramid	Mountain	Random
Nashville	13%	11%	8.5%
Sunnyvale	10.3%	12.4%	8.2%

Table 5. The change in the minimum link weight is presented when using the four control nodes identified by the weighted analysis and both servers simultaneously.

	Pyramid	Mountain	Random
Nashville	13%	11%	8.5%
Sunnyvale	10.3%	10.5%	4.2%

In summary, two server locations and three traffic profiles were used to validate the control nodes selection. The particle filter was used to effectively estimate link data rates. The first set of experiments showed that control nodes selected based on an unweighted graph did not produce the desired results. In a few cases, the performance degraded with the addition of control nodes. The second set of experiments used a weighted graph based on the traffic matrix. The control nodes that were identified using the weighted analysis performed better in terms of balancing the traffic load.

Additionally, it was shown that when using two server locations the control nodes based on the weighted graph provided comparable performance to the one server scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

The goal of this work was to determine a framework to model a SDN after a closed-loop control system. It was demonstrated that the standard definition of a closed-loop control system can be used as a model around which to build SDN applications. Spectral graph theory was used to develop the dual-basis representation, which is a tool to reveal the underlying structure of the graph. The observer was implemented as a non-linear state estimator. The controller was built around a cost function that would be found in any optimal controller. The dual-basis representation is also used to dynamically determine the minimum number of nodes required to monitor and control the network. These contributions can be applied to any SDN implementation and should be considered when developing new applications for SDNs. The effectiveness of the proposed ideas was demonstrated in simulation and experimentation on a test bed.

The objectives of this dissertation were accomplished through the development of the dual-basis analysis, a closed-loop control framework for a SDN, and a test bed to validate the proposed scheme. The dual-basis analysis is a new method to reveal the underlying structure of the network and dynamic network features. The closed-loop control framework includes the particle filter, the phantom node, and the load-balancing controller. Numerous experiments were run on the test bed to validate the control node selection, data rate estimation via the particle filter, and the load-balancing controller scheme.

A. SIGNIFICANT CONTRIBUTIONS

The work reported in this dissertation led to significant contribution to software-defined networking research. Specifically, three contributions are detailed in this section. The first contribution is the development of the dual-basis representation as a means to extract features from the network. The second contribution is the development of a scheme based on a closed-loop control system. The third contribution is a SDN test bed on which the dual-basis representation and closed-loop control scheme were validated.

1. Dual-basis Representation

The dual-basis representation orients the controller to current network behavior and conditions. The dual-basis representation is the real-valued solution to the ratio cut optimization problem [41]. By determining this optimal solution, the SDN controller is able to determine which node or nodes are congested, which nodes have the most control over network behavior, and which nodes have become disconnected. By extending this idea to time-varying link weights, we see that the controller is able to use the eigenvalues to determine that an event is occurring and then use the eigenvectors to determine where in the network that event is occurring. The phantom node is the implementation of the eigenvalue and eigenvector monitoring for congestion detection.

By applying concepts from image segmentation [41] and principal component analysis [28] to the dual-basis analysis, we recognize that the leading eigenvectors of the eigencentrality matrix can be used to determine the Laplacian eigencentrality and nodal angles. The controller can dynamically locate the control nodes by utilizing the Laplacian eigencentrality and nodal angles. Due to the dynamic nature of the traffic in any network, the control of the network needs to be applied in a dynamic fashion, and the network conditions need to be updated dynamically.

2. Closed-Loop Control Framework

SDNs allow for much more complex interactions between the network traffic and the network infrastructure, which is now embodied in the switches and network controller. This interaction is well-suited for modeling it after a closed-loop control system that makes observations of network parameters and topology, estimates the link data rates, and control network flows to improve performance. To accomplish the tasks of estimation and control, the SDN needs a network observer and a network controller. The link data rate estimation is the first half of the proposed closed-loop control scheme [2] [61]. Data rates can vary rapidly in any network and routing decisions should not be made using these fluctuations. The particle filter was used to estimate the link data rates to more accurately describe the weighted network graph.

Once the link weights have been estimated and passed to the controller, it can determine how to update network flows to balance the offered load. Using the dual-basis analysis, we developed a method to locate the control nodes to reduce the workload on the controller. Any traffic that was incident on a control node was used to balance the traffic load throughout the network.

3. SDN Test Bed

The analysis and simulations presented in the previous chapters are a required part of the development of any new concept, but simulations cannot fully validate the proposed scheme. The third contribution of this research is the development of a hardware test bed to validate the concepts developed analytically and in simulation. All of the concepts proposed in this research were put into practice using Python applications and MATLAB scripts that were run on a real-world SDN controller and were executed in a hardware SDN. This network was used to test and evaluate the dual-basis representation, control node assignment, and congestion detection. The fidelity provided by the test bed proved to be an indispensable component of this research. Without the test bed, the concepts and scheme proposed in this dissertation would not have been exposed to the realities of non-deterministic time delays and the issues that arise when software interacts with hardware.

B. FUTURE WORK

One goal of this work was to provide a basis on which to build future SDN research. The results presented here based on the dual-basis representation are a small subset of the information that can be extracted from the matrices V and V^T. From the results of the dual-basis analysis, the angle between nodes in the n-dimensional space can be used to determine which nodes are isolated from each other. It was observed that nodes with angles that were less than 90° were directly connected, and nodes with angles that were greater than 90° were one hop away. This nodal isolation was used to select specific switches as control nodes. Future work could include an in-depth investigation into why certain nodes are isolated and what the angles between nodes represent physically.

The SDN controller placement problem could be solved with a slight change to the control node placement solution presented here. Most SDN implementations have more than one SDN controller to provide redundancy and reduce the workload of a single SDN controller. The SDN controller placement problem is determining where the controller should be physically located. Intuition suggests that the control node locations could be similar to those of SDN controller nodes. The next step is to determine which nodes are assigned to which controllers' domain. The angle between nodes should provide a method to indicate which nodes are isolated from each other, which will help determine which nodes should be assigned to which community. Coupled nodes should be assigned to the same community, and the controller node is assigned as the most central node within the community. A scheme must be developed to use the angles to find these communities and assign controller nodes.

The test bed did not include a method to simulate the distance between the switches within the network and between the controller and the switches. An addition of the delay into the network would allow for a more faithful representation of the modeled network. The round-trip times were not considered this work. This delay means that the closed-loop control will be slower to react to changes in the network. This will have a negative effect on the ability of the controller to determine the correct routes. New methods need to be explored to minimize the delay and then work to minimize the impact of the additional delay.

This work did not include TCP packet traffic in any of the profiles because the initiation of TCP congestion control algorithms would change the expected results; however, all UDP traffic would be unusual in most networks. The closed-loop control model used here was a single loop, but adding TCP traffic would add to control loops. One of those control loops is controlled by the SDN controller and one is not. The addition of TCP traffic profiles provides another layer of complexity. The closed-loop control scheme proposed here may need to be modified to accommodate the addition of TCP congestion control algorithms.

The test bed was a built using 13 switches, but it could be expanded to 34 switches. The full 34 node network would provide a better emulation of the Internet2

topology. The 34 node network may require additional controllers to manage the larger number of nodes. Finding the correct number of controllers and determining how to assign switches to controllers can be solved in light of the dual-basis representation. The angles between nodes can be used to determine which nodes are coupled to the controller node. This work would benefit from a larger number of nodes.

SDN implementation on a Navy warship would be an excellent application of this technology. SDNs are typically implemented as closed, contained networks, such as a data center. Navy warship networks are typically closed, contained networks. In addition to the Internet2 topology, the Navy shipboard network needs to be analyzed using the dual-basis analysis. SDN could provide a new way to approach cybersecurity on a ship. The methods presented here can be applied to shipboard topologies and then test cybersecurity applications on the test bed.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ALGEBRAIC MANIPULATION TO OBTAIN THE LIMIT IN EQN. (3.34)

The following derivation verifies the result in Eqn. (3.34). First, substituting

$$A = \frac{-3u}{2} + 1,\ B = 2u - 4,\ \text{and}\ C = 9u^2 - 12u + 4\,. \tag{A.1}$$

into Eqn (3.34) and moving the constants to the outside, the limit simplifies to

$$\lim_{n\to\infty}\ \lambda_2 = A + \frac{1}{2}\lim_{n\to\infty} n - \sqrt{n^2 + Bn + C}\,. \tag{A.2}$$

By separating the variable B and C, and finding a common denominator, it can be shown that Eqn. (A.2) simplifies to

$$\lim_{n\to\infty}\ \lambda_2 = A + \frac{1}{2}\lim_{n\to\infty} \frac{-B}{1 + \sqrt{1 + \frac{B}{n} + \frac{C}{n^2}}}, \tag{A.3}$$

which further simplifies to

$$\lim_{n\to\infty}\ \lambda_2 = A + \frac{1}{2}\left(\frac{-B}{2}\right)\,. \tag{A.4}$$

By replacing the substituted equations into Eqn. (A.4), the result is

$$\lim_{n\to\infty}\ \lambda_2 = \frac{-3u}{2} + 1 + \frac{1}{4}\left(-2u + 4\right) = 2(1-u)\,. \tag{A.5}$$

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SAMPLE OF PYTHON SCRIPTS FOR THE CONTROLLER APPLICATION

```
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
# Licensed under the Apache License, Version 2.0 (the “License”);
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an “AS IS” BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
“““
An OpenFlow 1.0 L2 learning switch implementation.
“““
#Update to correctly assign flows based on HP2920 ability
import logging
import struct
import hashlib
from datetime import datetime
from ryu.base import app_manager
from ryu.controller import mac_to_port
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.controller.dpset import DPSet
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import inet
from ryu.ofproto import ether
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
class SimpleSwitch(app_manager.RyuApp):
        OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
        _CONTEXTS = {
                ‘dpset’: DPSet,
                }
        def __init__(self, *args, **kwargs):
                super(SimpleSwitch, self).__init__(*args, **kwargs)
```

```
        self.mac_to_port = {} #mac to port dictionary
        self.ip_to_port = {} #ip to port dictionary
        self.cookiejar = [] #cookiejar stores all of the cookies
        self.linklist = [] #link list keeps track of which link has flows on it
        self.dpid_to_port = {} #dpid_to_port keeps track of which links are on
                               which port
        self.numflows = [0]*41 #I have 20 links, but they are directional so 40.
                              #plus 1 because python counts from 0
        self.DPSet = kwargs['dpset']
        self.a6 = datetime.now()
        self.a13 = datetime.now()
        self.a18 = datetime.now()
        self.b6 = datetime.now()
        self.b13 = datetime.now()
        self.b18 = datetime.now()
        self.c6 = 0
        self.c13 = 0
        self.c18 = 0
        self.lastip = 0
        self.lastdpid = 0
        self.lastdstip = 0
#add flow for ARP packets
def add_flow_ARP(self, datapath, in_port, dst, actions, dl_type):
        #get openflow protocol; it could be 1.0 or 1.3
        ofproto = datapath.ofproto
        #define match
        #match on ethernet type, physical incoming port, and mac
        match = datapath.ofproto_parser.OFPMatch(dl_type=dl_type,
                in_port=in_port,
                dl_dst=haddr_to_bin(dst))
        #mod builds the flow mod
        mod = datapath.ofproto_parser.OFPFlowMod(
        datapath=datapath, match=match, cookie=0,
        command=ofproto.OFPFC_ADD, idle_timeout=600, hard_timeout=3600,
        priority=ofproto.OFP_DEFAULT_PRIORITY,
        flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        #then send the flow mod to the switch
        datapath.send_msg(mod)
#add flow for IP packets
def add_flow_IP(self, out, datapath, in_port, dst, actions, dl_type, src):
        reset = 0
        with open("/home/ec4715/Documents/MATLAB/TomDissertation/reset.txt") as
          file: # Use file to refer to the file object
                reset = file.read()
        reset = int(reset)
        if reset == 1:
```

```
        self.numflows = [0]*41
#get openflow protocol; it could be 1.0 or 1.3
ofproto = datapath.ofproto
#dpid is datapath id which is a hex number assigned from the factory
dpid = datapath.id

#this code turns the dotted decimal ip address into an integer
o = map(int, dst.split('.'))
res = (16777216 * o[0]) + (65536 * o[1]) + (256 * o[2]) + o[3]
o = map(int, src.split('.'))
src_res = (16777216 * o[0]) + (65536 * o[1]) + (256 * o[2]) + o[3]
#build the match
#match is based on ethernet type, incoming port and network dest
match = datapath.ofproto_parser.OFPMatch(dl_type=dl_type,
 in_port=in_port,
 nw_src=src_res, nw_dst=res)
## Possible parameters to send to OFPMatch()
## in_port=None, dl_src=None, dl_dst=None,
## dl_vlan=None, dl_vlan_pcp=None, dl_type=None, nw_tos=None,
## nw_proto=None, nw_src=None, nw_dst=None,
## tp_src=None, tp_dst=None, nw_src_mask=32, nw_dst_mask=32)
done = 0
with open("/home/ec4715/Documents/MATLAB/TomDissertation
/DoneBuildingTable.txt") as file:    # Use file to refer to the file
                                      object
        done = file.read()
done = int(done)
cookie = 0
if out < 30 and dst == '10.10.2.6' and done == 1
 and src != '10.10.13.1':
        idletime = 1100
        if dpid == 0x00012c59e5107640:
            if out == 4:
                self.numflows[1]=self.numflows[1]+1
            elif out == 3:
                self.numflows[3]=self.numflows[3]+1
            elif out == 1:
                self.b6 = datetime.now()
                self.c6 = self.b6-self.a6
                if self.c6.seconds > 10:
                        self.numflows[6]=self.numflows[6]+1
                        self.a6 = self.b6
                        hashee = str(src) + str(dpid) +
                         str(datetime.now())
                        cookie = int(abs(hash(hashee)))
                        #I save off the cookie into the cookie jar
```

```
                #I save off the link that the cookie is assigned
                if cookie not in self.cookiejar:
                        self.cookiejar.append(cookie)
                        self.linklist.append(out)
                        print "add 1: chicago link is:"
                        print self.numflows[6]
                        print "add 1: houston link is:"
                        print self.numflows[18]
                        print "add 1: atlanta link is:"
                        print self.numflows[13]
                else:
                        cookie = 0
    elif out == 2:
        self.numflows[7]=self.numflows[7]+1
#node 2
elif dpid == 0x0001c4346b94a200:
    if out == 1:
        self.numflows[34]=self.numflows[34]+1
    elif out == 2:
        self.numflows[35]=self.numflows[35]+1
    elif out == 3:
        self.numflows[37]=self.numflows[37]+1
#node 3
elif dpid == 0x00012c59e51016c0:
    if out == 1:
        self.numflows[31]=self.numflows[31]+1
    elif out == 2:
        self.numflows[30]=self.numflows[30]+1
    elif out == 3:
        self.numflows[33]=self.numflows[33]+1
#node 4
elif dpid == 0x0001c4346b99dc00:
    if out == 1:
        self.numflows[27]=self.numflows[27]+1
    elif out == 2:
        self.numflows[29]=self.numflows[29]+1
    elif out == 3:
        self.numflows[36]=self.numflows[36]+1
    elif out == 4:
        self.numflows[39]=self.numflows[39]+1
#node 5
elif dpid == 0x0001c4346b946200:
    if out == 1:
        self.numflows[25]=self.numflows[25]+1
    elif out == 2:
        self.numflows[25]=self.numflows[25]+1
```

```
    elif out == 3:
        self.numflows[28]=self.numflows[28]+1
#node 6
elif dpid == 0x0001c4346b971ec0:
    if out == 1:
        self.numflows[21]=self.numflows[21]+1
    elif out == 2:
        self.numflows[23]=self.numflows[23]+1
    elif out == 3:
        self.numflows[32]=self.numflows[32]+1
#node 8
elif dpid == 0x0001f0921c220e80:
    if out == 1:
        self.numflows[8]=self.numflows[8]+1
    elif out == 2:
        self.numflows[9]=self.numflows[9]+1
#node 9
elif dpid == 0x0001c4346b98a200:
    if out == 9:
        self.numflows[10]=self.numflows[10]+1
    elif out == 10:
        self.numflows[11]=self.numflows[11]+1
#node 10
elif dpid == 0x0001c4346b972a80:
    if out == 9:
        self.numflows[12]=self.numflows[12]+1
    elif out == 10:
        self.numflows[15]=self.numflows[15]+1
    elif out == 11:
        self.b13 = datetime.now()
        self.c13 = self.b13-self.a13
        if self.c13.seconds > 10:
                self.numflows[13]=self.numflows[13]+1
                self.a13 = self.b13
                hashee = str(src) + str(dpid)+ str(datetime.now())
                cookie = int(abs(hash(hashee)))
                #I save off the cookie into the cookie jar
                #I save off the link that the cookie is assigned
                if cookie not in self.cookiejar:
                        self.cookiejar.append(cookie)
                        self.linklist.append(out)
                        print “add 1: chicago link is:”
                        print self.numflows[6]
                        print “add 1: houston link is:”
                        print self.numflows[18]
                        print “add 1: atlanta link is:”
```

```
                        print self.numflows[13]
                else:
                        cookie = 0
#node 11
elif dpid == 0x0001f0921c226e80:
    if out == 1:
        self.numflows[20]=self.numflows[20]+1
    elif out == 2:
        self.numflows[4]=self.numflows[4]+1
    elif out == 3:
        self.numflows[26]=self.numflows[26]+1
#node 12
elif dpid == 0x000140a8f0d12bc0:
    if out ==1:
        self.numflows[2]=self.numflows[2]+1
    elif out == 2:
        self.numflows[40]=self.numflows[40]+1
    elif out == 3:
        self.numflows[38]=self.numflows[38]+1
#node 13
elif dpid == 0x0001f0921c219d40:
    if out == 1:
        self.numflows[16]=self.numflows[16]+1
    elif out == 2:
        self.b18 = datetime.now()
        self.c18 = self.b18-self.a18
        if self.c18.seconds > 10:
                self.numflows[18]=self.numflows[18]+1
                self.a18 = self.b18
                hashee = str(src) + str(dpid)+ str(datetime.now())
                cookie = int(abs(hash(hashee)))
                #I save off the cookie into the cookie jar
                #I save off the link that the cookie is assigned
                if cookie not in self.cookiejar:
                        self.cookiejar.append(cookie)
                        self.linklist.append(out)
                        print “add 1: chicago link is:”
                        print self.numflows[6]
                        print “add 1: houston link is:”
                        print self.numflows[18]
                        print “add 1: atlanta link is:”
                        print self.numflows[13]
                else:
                        cookie = 0
    elif out == 3:
        self.numflows[22]=self.numflows[22]+1
```

```
                elif out == 4:
                    self.numflows[19]=self.numflows[19]+1
            #node 13
            elif dpid == 0x0001f0921c225480:
                if out == 1:
                    self.numflows[14]=self.numflows[14]+1
                elif out == 2:
                    self.numflows[17]=self.numflows[17]+1
                elif out == 3:
                    self.numflows[5]=self.numflows[5]+1
            else:
                print "error updating when adding to numflows"
        else:
                idletime=30
        #I write these values to a text file to be read in by MATLAB
        #Matlab uses it as input to the particle filter
        fh = open("input_to_filter.txt,""w")
        fh.seek(0)
        fh.write(str(self.numflows))
        fh.close()
        #then I generate the flow mod and send it
        mod = datapath.ofproto_parser.OFPFlowMod(
                datapath=datapath, match=match, cookie=cookie,
                command=ofproto.OFPFC_ADD, idle_timeout=idletime,
                hard_timeout=idletime,
                priority=ofproto.OFP_DEFAULT_PRIORITY,
                flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)
#catch all flow.
def add_flow(self, datapath, in_port, actions, dl_type):
        ofproto = datapath.ofproto
        match = datapath.ofproto_parser.OFPMatch(dl_type=dl_type,
                in_port=in_port)
        mod = datapath.ofproto_parser.OFPFlowMod(
                datapath=datapath, match=match, cookie=0,
                command=ofproto.OFPFC_ADD, idle_timeout=600, hard_timeout=3600,
                priority=ofproto.OFP_DEFAULT_PRIORITY,
                flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)
#This section handles all packet-in events
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
        a = datetime.now() #find current time to time this loop
        #first pull the message (msg) from the event (ev)
        msg = ev.msg
        #pull of the datapath
```

```
 datapath = msg.datapath
#check the openflow protocol
 ofproto = datapath.ofproto
#parse out the packet from the message
 pkt = packet.Packet(msg.data)
#parse out the ethernet (MAC) header and the ip header
 eth = pkt.get_protocol(ethernet.ethernet)
ip = pkt.get_protocol(ipv4.ipv4)
#the source and destination MACs are parsed out
dstMAC = eth.dst
srcMAC = eth.src
#if it is not an ip packet (i.e., arp) then ip is returned as 'None'
#if it is an ip packet parse the source and destination ip address
if ip != None:
        dstIP = ip.dst
        srcIP = ip.src
else:
        dstIP = 0xFFFFFFFF
        srcIP = 0xFFFFFFFF

#dpid is the switch ID
 dpid = datapath.id
#start building the mac to port and ip to port dictionary for each
 switch
self.mac_to_port.setdefault(dpid, {})
self.ip_to_port.setdefault(dpid, {})
done = 0
with open("/home/ec4715/Documents/MATLAB/TomDissertation
          /DoneBuildingTable.txt") as file:# Use file to refer to the
                                           file object
        done = file.read()
done = int(done)

# learn a Source mac address to avoid FLOOD next time.
if msg.in_port < 30 and done == 0:
        if srcMAC in self.mac_to_port[dpid]:
                print 'mac to port already assigned' + srcMAC
        else:
                self.mac_to_port[dpid][srcMAC] = msg.in_port
                print 'updating mac_to_port with MAC address ' +
                str(srcMAC)
if msg.in_port < 30 and done == 0:
        if srcIP != 0xFFFFFFFF:
                if srcIP in self.ip_to_port[dpid]:
                        print 'ip to port already assigned' + srcIP
                else:
```

```
                    self.ip_to_port[dpid][srcIP] = msg.in_port
#define my control nodes by their dpid (switch ID)
controlnodes = []
controlnodes = [0x00012c59e5107640]
controlon = 0
with open("/home/ec4715/Documents/MATLAB/TomDissertation/control.txt")
as file:        # Use file to refer to the file object
        controlon = file.read()
controlon = int(controlon)


#determine what to do with each packet based on current learned
 locations or flood
if dstMAC in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dstMAC]

elif dstIP in self.ip_to_port[dpid]:
        out_port = self.ip_to_port[dpid][dstIP]
else:
        out_port = ofproto.OFPP_FLOOD

#here I assign the action that the flow should take
actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
#Flood ARP packets
if dstMAC == "ff:ff:ff:ff:ff:ff" and eth.ethertype != 0x002c:
        out_port = ofproto.OFPP_FLOOD
        actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
        #call add_flow_ARP function
        self.add_flow_ARP(datapath, msg.in_port, dstMAC, actions=actions,
                dl_type=eth.ethertype)
        #Because the switches do not buffer the packet once it sends a
        packet_in
        #message, I must send that packet out using OFPPacketOut function
        call
        out = datapath.ofproto_parser.OFPPacketOut(
        datapath=datapath, buffer_id=0xffffffff, in_port=msg.in_port,
                actions=actions, data=msg.data)
        datapath.send_msg(out)
#If the packet is not an IP or ARP packet, I want to drop it
elif eth.ethertype != 0x800 and eth.ethertype != 0x806:
        #To drop a packet the action is assigned None
        #the switch reads this as an instruction to drop the packet
        actions=None
        self.add_flow(datapath, msg.in_port, actions=actions,
        dl_type=eth.ethertype)
#if it is a ip packet I execute this loop
```

```
        elif eth.ethertype == 0x800:
                #don't do anything if the in port and out port are the same
                #This happens when there are loops in the network
                #Spanning tree algorithms help solve this problem
                        #if the switch is a control node and the destination is
                        10.10.2.6
                        if dpid in controlnodes and dstIP == '10.10.2.6' and
                        controlon == 1:
                                        #the file read in below has a row for each
                                         control switch
                                        #the router uses that assigned row to route
                                         the packets
                                        row = 4
                                        if dpid == 0x0001c4346b99dc00: #if switch
                                                                        is node 4
                                                row = 0
                                        elif dpid ==  0x0001f0921c219d40: #else
                                                                switch is node 13
                                                row = 2
                                        elif dpid == 0x00012c59e5107640:
                                                row = 1
                                        #Here I read in the text file that MATLAB
                                          wrote to determine routes
                                        with open('//home//ec4715//Documents//
                                       MATLAB//TomDissertation//route.txt') as f:
                                                route = []
                                                for line in f:
                                                        line = line.split()
                                                        if line:
                                                                line = [int(i) for i
                                                                  in line]
                                                                route.append(line)
                                        f.close
                                        self.dpid_to_port.setdefault(0,{})

self.dpid_to_port[1]=[0,0,0,0,0,0,0,0,2,0,0,3,4,0,1,0]

self.dpid_to_port[2]=[0,0,0,1,2,0,0,0,0,0,0,0,3,0,0,0]

self.dpid_to_port[3]=[0,0,3,0,2,0,1,0,0,0,0,0,0,0,0,0]

self.dpid_to_port[4]=[0,0,3,2,0,1,0,0,0,0,0,0,4,0,0,0]

self.dpid_to_port[5]=[0,0,0,0,3,0,2,0,0,0,0,1,0,0,0,0]

self.dpid_to_port[6]=[0,0,0,3,0,2,0,0,0,0,0,0,0,1,0,0]
```

```
self.dpid_to_port[7]=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

self.dpid_to_port[8]=[0,1,0,0,0,0,0,0,0,2,0,0,0,0,0,0]

self.dpid_to_port[9]=[0,0,0,0,0,0,0,0,9,0,10,0,0,0,0,0]

self.dpid_to_port[10]=[0,0,0,0,0,0,0,0,0,9,0,0,0,10,11,0]

self.dpid_to_port[11]=[0,2,0,0,0,3,0,0,0,0,0,0,0,1,0,0]

self.dpid_to_port[12]=[0,1,3,0,2,0,0,0,0,0,0,0,0,0,0,0]

self.dpid_to_port[13]=[0,0,0,0,0,0,3,0,0,0,1,4,0,0,2,0]

self.dpid_to_port[14]=[0,3,0,0,0,0,0,0,0,0,1,0,0,2,0,8]
                                        counter = 0
                                        print route[row]
                                        for i in range(len(route[row])-1):
                                                if counter == 0:
                                                        in_portip = msg.in_port
                                                        counter = 1
                                                else:
                                                        first = route[row][i]
                                                        second = route[row][i-1]
                                                        in_portip =
                                                    self.dpid_to_port[first][second]
                                                first = route[row][i]
                                                second = route[row][i+1]
                                                out_portip =
                                                    self.dpid_to_port[first][second]
                                                if first == 1:
                                                        dpidflow=0x00012c59e5107640
                                                elif first ==2:
                                                        dpidflow=0x0001c4346b94a200
                                                elif first ==3:
                                                        dpidflow=0x00012c59e51016c0
                                                elif first ==4:
                                                        dpidflow=0x0001c4346b99dc00
                                                elif first ==5:
                                                        dpidflow=0x0001c4346b946200
                                                elif first ==6:
                                                        dpidflow=0x0001c4346b971ec0
                                                elif first ==8:
                                                        dpidflow=0x0001f0921c220e80
                                                elif first ==9:
```

```
                dpidflow=0x0001c4346b98a200
        elif first ==10:
                dpidflow=0x0001c4346b972a80
        elif first ==11:
                dpidflow=0x0001f0921c226e80
        elif first ==12:
                dpidflow=0x000140a8f0d12bc0
        elif first ==13:
                dpidflow=0x0001f0921c219d40
        elif first ==14:
                dpidflow=0x0001f0921c225480
        datapath=self.DPSet.get(dpidflow)
        actionscontrol =
            [datapath.ofproto_parser.
            OFPActionOutput(out_portip)]
        self.add_flow_IP(out_portip,
           datapath, in_portip, dstIP,
           actions=actionscontrol,
           dl_type=eth.ethertype, src=srcIP)

        actionscontrol =
            [datapath.ofproto_parser.
           OFPActionOutput(in_portip)]
        self.add_flow_IP(in_portip,
           datapath, out_portip, srcIP,
           actions=actionscontrol,
           dl_type=eth.ethertype, src=dstIP)

#after for loop of flows send the message
 back out on the original
#switch
first=route[row][0]
second=route[row][1]
if first == 1:
        dpidflow=0x00012c59e5107640
elif first ==2:
        dpidflow=0x0001c4346b94a200
elif first ==3:
        dpidflow=0x00012c59e51016c0
elif first ==4:
        dpidflow=0x0001c4346b99dc00
elif first ==5:
        dpidflow=0x0001c4346b946200
elif first ==6:
        dpidflow=0x0001c4346b971ec0
elif first ==8:
```

```
                                        dpidflow=0x0001f0921c220e80
                                elif first ==9:
                                        dpidflow=0x0001c4346b98a200
                                elif first ==10:
                                        dpidflow=0x0001c4346b972a80
                                elif first ==11:
                                        dpidflow=0x0001f0921c226e80
                                elif first ==12:
                                        dpidflow=0x000140a8f0d12bc0
                                elif first ==13:
                                        dpidflow=0x0001f0921c219d40
                                elif first ==14:
                                        dpidflow=0x0001f0921c225480
                                datapath=self.DPSet.get(dpidflow)

                                out_portip=self.dpid_to_port[first][second]
                                actions =
                                [datapath.ofproto_parser.OFPActionOutput
                                        (out_portip)]
                                print "packet out"
                                print first, second, out_portip, dstIP
                                out = datapath.ofproto_parser.OFPPacketOut(
                                        datapath=datapath,
                                        buffer_id=0xffffffff,
                                        in_port=in_portip,
                                        actions=actions, data=msg.data)
                                datapath.send_msg(out)
                        #if it isn't a control node, then I send it down the
                         static path
                        else:
                                self.add_flow_IP(out_port, datapath, msg.in_port,
                                        dstIP, actions=actions,
                                        dl_type=eth.ethertype, src=srcIP)
                                out = datapath.ofproto_parser.OFPPacketOut(
                                       datapath=datapath, buffer_id=0xffffffff,
                                       in_port=msg.in_port,
                                        actions=actions, data=msg.data)
                                datapath.send_msg(out)
elif eth.ethertype == 0x806:
        if msg.in_port == out_port:
                out_port = ofproto.OFPP_FLOOD

                #here I assign the action that the flow should take
                actions =
                   [datapath.ofproto_parser.OFPActionOutput(out_port)]
                print "in port equals out port for IP traffic 0x806"
```

```
                        self.add_flow_ARP(datapath, msg.in_port, dstMAC,
                                actions=actions,
                                dl_type=eth.ethertype)
                        out = datapath.ofproto_parser.OFPPacketOut(
                                datapath=datapath, buffer_id=0xffffffff,
                                in_port=msg.in_port,
                                actions=actions, data=msg.data)
                        datapath.send_msg(out)
                else:
                        self.add_flow_ARP(datapath, msg.in_port, dstMAC,
                                actions=actions,
                                dl_type=eth.ethertype)
                        out = datapath.ofproto_parser.OFPPacketOut(
                          datapath=datapath, buffer_id=0xffffffff,
                          in_port=msg.in_port,
                          actions=actions, data=msg.data)
                        datapath.send_msg(out)
        b = datetime.now()
    #Here I process the flow removed message
    @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
    def _flow_removed_handler(self, ev):
        #I do similar parsing as above
        msg = ev.msg
        dpid = msg.datapath.id
        match = msg.match
        inport = match.in_port
        #first I check to see if the cookie is zero
        done = 0
        with
open("/home/ec4715/Documents/MATLAB/TomDissertation/DoneBuildingTable.txt") as file:
    # Use file to refer to the file object
                done = file.read()
        done = int(done)
        if msg.cookie != 0 and done == 1:
                link_index = self.cookiejar.index(msg.cookie)
                out = self.linklist[link_index]

                #Knowing the link it came in on and the switch
                #I can decrement the correct flow number
                if out < 30:

                #node 1
                    if dpid == 0x00012c59e5107640:
                        if out == 4:
                                self.numflows[1]=self.numflows[1]-1
                        elif out == 3:
```

```
                self.numflows[3]=self.numflows[3]-1
        elif out == 1:
                self.numflows[6]=self.numflows[6]-1
                print “subtract 1: chicago link is:”
                print self.numflows[6]
                print “subtract 1: houston link is:”
                print self.numflows[18]
                print “subtract 1: atlanta link is:”
                print self.numflows[13]
        elif out == 2:
                self.numflows[7]=self.numflows[7]-1
#node 2
    elif dpid == 0x0001c4346b94a200:
        if out == 1:
                self.numflows[34]=self.numflows[34]-1
        elif out == 2:
                self.numflows[35]=self.numflows[35]-1
        elif out == 3:
                self.numflows[37]=self.numflows[37]-1
#node 3
    elif dpid == 0x00012c59e51016c0:
        if out == 1:
                self.numflows[31]=self.numflows[31]-1
        elif out == 2:
                self.numflows[30]=self.numflows[30]-1
        elif out == 3:
                self.numflows[33]=self.numflows[33]-1
#node 4
     elif dpid == 0x0001c4346b99dc00:
        if out == 1:
                self.numflows[27]=self.numflows[27]-1
        elif out == 2:
                self.numflows[29]=self.numflows[29]-1
        elif out == 3:
                self.numflows[36]=self.numflows[36]-1
        elif out == 4:
                self.numflows[39]=self.numflows[39]-1
#node 5
    elif dpid == 0x0001c4346b946200:
        if out == 1:
                self.numflows[25]=self.numflows[25]-1
        elif out == 2:
                self.numflows[25]=self.numflows[25]-1
        elif out == 3:
                self.numflows[28]=self.numflows[28]-1
#node 6
```

```
    elif dpid == 0x0001c4346b971ec0:
        if out == 1:
                self.numflows[21]=self.numflows[21]-1
        elif out == 2:
                self.numflows[23]=self.numflows[23]-1
        elif out == 3:
                self.numflows[32]=self.numflows[32]-1
#node 8
    elif dpid == 0x0001f0921c220e80:
        if out == 1:
                self.numflows[8]=self.numflows[8]-1
        elif out == 2:
                self.numflows[9]=self.numflows[9]-1
#node 9
    elif dpid == 0x0001c4346b98a200:
        if out == 9:
                self.numflows[10]=self.numflows[10]-1
        elif out == 10:
                self.numflows[11]=self.numflows[11]-1
#node 10
    elif dpid == 0x0001c4346b972a80:
        if out == 9:
                self.numflows[12]=self.numflows[12]-1
        elif out == 10:
                self.numflows[15]=self.numflows[15]-1
        elif out == 11:
                self.numflows[13]=self.numflows[13]-1
                print “subtract 1: chicago link is:”
                print self.numflows[6]
                print “subtract 1: houston link is:”
                print self.numflows[18]
                print “subtract 1: atlanta link is:”
                print self.numflows[13]
#node 11
    elif dpid == 0x0001f0921c226e80:
        if out == 1:
                self.numflows[20]=self.numflows[20]-1
        elif out == 2:
                self.numflows[4]=self.numflows[4]-1
        elif out == 3:
                self.numflows[26]=self.numflows[26]-1
#node 12
    elif dpid == 0x000140a8f0d12bc0:
        if out ==1:
                self.numflows[2]=self.numflows[2]-1
        elif out == 2:
```

```
                        self.numflows[40]=self.numflows[40]-1
                    elif out == 3:
                        self.numflows[38]=self.numflows[38]-1
                #node 13
                elif dpid == 0x0001f0921c219d40:
                    if out == 1:
                        self.numflows[16]=self.numflows[16]-1
                    elif out == 2:
                        self.numflows[18]=self.numflows[18]-1
                        print "subtract 1: chicago link is:"
                        print self.numflows[6]
                        print "subtract 1: houston link is:"
                        print self.numflows[18]
                        print "subtract 1: atlanta link is:"
                        print self.numflows[13]
                    elif out == 3:
                        self.numflows[22]=self.numflows[22]-1
                    elif out == 4:
                        self.numflows[19]=self.numflows[19]-1
                #node 13
                elif dpid == 0x0001f0921c225480:
                    if out == 1:
                        self.numflows[14]=self.numflows[14]-1
                    elif out == 2:
                        self.numflows[17]=self.numflows[17]-1
                    elif out == 3:
                        self.numflows[5]=self.numflows[5]-1
                else:
                    print "error updating"
            #I write this info to a text file for MATLAB to read
            fh = open("input_to_filter.txt,""w")
            fh.seek(0)
            fh.write(str(self.numflows))
            fh.close()

    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
            msg = ev.msg
            reason = msg.reason
             port_no = msg.desc.port_no
             ofproto = msg.datapath.ofproto
             if reason == ofproto.OFPPR_ADD:
                 self.logger.info("port added %s," port_no)
             elif reason == ofproto.OFPPR_DELETE:
                 self.logger.info("port deleted %s," port_no)
             elif reason == ofproto.OFPPR_MODIFY:
```

```
            self.logger.info("port modified %s," port_no)

        else:
            self.logger.info("Illeagal port state %s %s," port_no, reason)
```

APPENDIX C. SAMPLE OF PYTHON SCRIPTS FOR MONITOR APPLICATION

```
From operator import attrgetter
from datetime import datetime
import numpy as np
import May25routingApp_3ControlNode
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub
import sys
class SimpleMonitor(May25routingApp_3ControlNode.SimpleSwitch):
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if not datapath.id in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]
    def _monitor(self):
       while True:
            for dp in self.datapaths.values():
                  self._request_stats(dp)

           hub.sleep(1)

    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = datapath.ofproto_parser.OFPMatch(datapath.ofproto.OFPFW_ALL,
                                                 0, 0, 0, 0, 0,
                                                 0, 0, 0, 0, 0, 0, 0)
```

```
    #req = datapath.ofproto_parser.OFPFlowStatsRequest(datapath, 0, match,
    #                                0, datapath.ofproto.OFPP_NONE)
     #datapath.send_msg(req)
     req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_NONE)
     datapath.send_msg(req)
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
     body = ev.msg.body
    self.logger.info('flows reply from: %016x', ev.msg.datapath.id)
     self.logger.info('datapath         '
                      'in-port  eth-dst                    '
                      'out-port packets  bytes')
     self.logger.info('---------------- '
                      '-------- ----------------------------- '
                      '-------- -------- --------')
     for stat in sorted(body, key=attrgetter('packet_count')):
    #for stat in sorted([flow for flow in body if flow.cookie != 1],key=lambda):
            if body.actions:
                self.logger.info('%016x %8x %17s %8x %8d %8d',
                  ev.msg.datapath.id,
                                #stat.match.in_port, stat.actions[0].port,
                stat.match.in_port, repr(stat.match.dl_dst), stat.actions[0].port,
                  stat.packet_count, stat.byte_count)
    with open('FlowStats.txt','a') as file:
            file.writelines("%s:  , %s" % str(datetime.now()),ev.msg)
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    body = ev.msg.body
    """self.logger.info('datapath         port     '
                      'rx-pkts  rx-bytes rx-error '
                      'tx-pkts  tx-bytes tx-error')
     self.logger.info('---------------- -------- '
                      '-------- -------- -------- '
                      '-------- -------- --------')"""
    """for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                        ev.msg.datapath.id, stat.port_no,
                        stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                        stat.tx_packets, stat.tx_bytes, stat.tx_errors)"""
    if ev.msg.datapath.id == 0x0001c4346b946200:
            with open('PortStats5.txt','w') as file:
            #for item in ev.msg.datapath:
                    file.writelines("%s \n" % str(datetime.now()))
                    for stat in sorted(body, key=attrgetter('port_no')):
                         did, port, rx_b, tx_b = ev.msg.datapath.id, stat.port_no,
                         stat.rx_bytes, stat.tx_bytes
```

```
                        file.write('port {}, rx_bytes {}, tx_bytes
                        {}\n'.format(port, rx_b, tx_b))
        #node 1
        elif ev.msg.datapath.id == 0x00012c59e5107640:
                with open('PortStats1.txt','w') as file:
                #for item in ev.msg.datapath:
                        file.writelines("%s \n" % str(datetime.now()))
                        for stat in sorted(body, key=attrgetter('port_no')):
                             did, port, rx_b, tx_b = ev.msg.datapath.id,
                             stat.port_no, stat.rx_bytes, stat.tx_bytes
                             file.write('port {}, rx_bytes {}, tx_bytes
                             {}\n'.format(port, rx_b, tx_b))
        #node 2
        elif ev.msg.datapath.id == 0x0001c4346b94a200:
                with open('PortStats2.txt','w') as file:
                #for item in ev.msg.datapath:
                        file.writelines("%s \n" % str(datetime.now()))
                        for stat in sorted(body, key=attrgetter('port_no')):
                             did, port, rx_b, tx_b = ev.msg.datapath.id,
                             stat.port_no, stat.rx_bytes, stat.tx_bytes
                             file.write('port {}, rx_bytes {}, tx_bytes
{}\n'.format(port, rx_b, tx_b))
        #node 4
        elif ev.msg.datapath.id == 0x0001c4346b99dc00:
                with open('PortStats4.txt','w') as file:
                #for item in ev.msg.datapath:
                        file.writelines("%s \n" % str(datetime.now()))
                        for stat in sorted(body, key=attrgetter('port_no')):
                             did, port, rx_b, tx_b = ev.msg.datapath.id,
                             stat.port_no, stat.rx_bytes, stat.tx_bytes
                             file.write('port {}, rx_bytes {}, tx_bytes
                             {}\n'.format(port, rx_b, tx_b))
        #node 5
        elif ev.msg.datapath.id == 0x0001c4346b946200:
                with open('PortStats5.txt','w') as file:
                #for item in ev.msg.datapath:
                        file.writelines("%s \n" % str(datetime.now()))
                        for stat in sorted(body, key=attrgetter('port_no')):
                             did, port, rx_b, tx_b = ev.msg.datapath.id,
                             stat.port_no, stat.rx_bytes, stat.tx_bytes
                             file.write('port {}, rx_bytes {}, tx_bytes
                             {}\n'.format(port, rx_b, tx_b))
        #node 6
        elif ev.msg.datapath.id == 0x0001c4346b971ec0:
                with open('PortStats6.txt','w') as file:
                #for item in ev.msg.datapath:
```

```
                file.writelines("%s \n" % str(datetime.now()))
                for stat in sorted(body, key=attrgetter('port_no')):
                    did, port, rx_b, tx_b = ev.msg.datapath.id,
                    stat.port_no, stat.rx_bytes, stat.tx_bytes
                    file.write('port {}, rx_bytes {}, tx_bytes
                    {}\n'.format(port, rx_b, tx_b))
#node 9
elif ev.msg.datapath.id == 0x0001c4346b98a200:
        #print 'node 9 entered'
       with open('PortStats9.txt','w') as file:
        #for item in ev.msg.datapath:
                file.writelines("%s \n" % str(datetime.now()))
                for stat in sorted(body, key=attrgetter('port_no')):
                    did, port, rx_b, tx_b = ev.msg.datapath.id,
                    stat.port_no, stat.rx_bytes, stat.tx_bytes
                    file.write('port {}, rx_bytes {}, tx_bytes
                    {}\n'.format(port, rx_b, tx_b))
#node 10
elif ev.msg.datapath.id == 0x0001c4346b972a80:
        with open('PortStats10.txt','w') as file:
        #for item in ev.msg.datapath:
                file.writelines("%s \n" % str(datetime.now()))
                for stat in sorted(body, key=attrgetter('port_no')):
                    did, port, rx_b, tx_b = ev.msg.datapath.id,
                    stat.port_no, stat.rx_bytes, stat.tx_bytes
                    file.write('port {}, rx_bytes {}, tx_bytes
                    {}\n'.format(port, rx_b, tx_b))
#node 13
elif ev.msg.datapath.id == 0x0001f0921c219d40:
        with open('PortStats13.txt','w') as file:
        #for item in ev.msg.datapath:
                file.writelines("%s \n" % str(datetime.now()))
                for stat in sorted(body, key=attrgetter('port_no')):
                    did, port, rx_b, tx_b = ev.msg.datapath.id,
                    stat.port_no, stat.rx_bytes, stat.tx_bytes
                    file.write('port {}, rx_bytes {}, tx_bytes
                    {}\n'.format(port, rx_b, tx_b))
```

LIST OF REFERENCES

[1] K. Hafner and M. Lyon, *When Wizards Stay Up Late: The Origins of the Internet*. New York: Simon and Schuster, 1998.

[2] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, Turin, Italy, 2013.

[3] S. Jain, A. Kumar, S. Mandal, J. Org, 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 *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp 3–14, Oct. 2013.

[4] Z. Gross, Revealed: the secret gear connecting Google's online empire. (n.d.). Wired. [Online]. Available: http://www.wired.com/2015/06/google-reveals-secret-gear-connects-online-empire/. Accessed 23 July 2015.

[5] A. Andreyev, Introducing data center fabric, the next-generation Facebook data center network. (n.d.). Facebook. [Online]. Available: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network. Accessed 10 August 2015.

[6] J. Burt, Verizon outlines plan to transform network with SDN. eWeek. [Online]. Available: http://www.eweek.com/networking/verizon-outlines-plan-to-transform-network-with-sdn.html and AT&T white paper. Accessed 10 August 2015.

[7] R. Coram, *Boyd: The Fighter Pilot Who Changed the Art of War*, Boston: Little, Brown, and Company, 2002.

[8] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer,* vol. 36, no. 1, pp. 41–50, Jan. 2003.

[9] T. C. Parker, J. Jones, J. Mayberry, G. Chanman, Z. Staples, J. McEachen, and M. Tummala, "Defensive cyber operations in a software-defined network," to appear in *Proc*. *Hawaii International Conf. System Sciences*, Kauai, HI, 2016.

[10] A. Sydney, *The evaluation of software defined networking for communication and control of cyber physical systems,* Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Kansas State University, Manhattan, KS, 2013.

[11] S. Das, K. Kant, and N. Zhang, *Handbook on Securing Cyber-Physical Critical Infrastructure*, Waltham, MA: Morgan Kaufmann, 2012.

[12] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” in the *Proc*. *HotSDN*, Helsinki, Finland, 2012.

[13] Advanced Networking. (n.d.). Internet2. [Online]. Available: www.internet2.edu. Accessed 1 May 2014.

[14] F. Yonghong, B. Jun, W. Jianping, C. Ze, W. Ke, and L. Min, “A dormant multi-controller model for software defined networking,” in *China Communications,* vol. 11, no. 3, pp. 45–55, Mar. 2014.

[15] M. F. Bari, “Dynamic controller provisioning in software defined networks,” in *Proc*. *9th International Conf. Network and Service Management*, Zurich, 2013.

[16] Solution brief: SDN security considerations in the data center. (n.d.). Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/solution-brief-sdn-security-considerations-in-the-data-center. Accessed 23 July 2015.

[17] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized? State distribution trade-offs in software defined networks,” in *Proc*. *HotSDN*, Helsinki, Finland, 2012.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” in *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69–74, Apr. 2008.

[19] J. T. Moy, *OSPF: Anatomy of an Internt Routing Protocol*, Boston: Addison-Wesley Professional, 1998.

[20] Open networking foundation homepage. (n.d.). Open Networking Foundation. [Online]. Available: https://www.opennetworking.org. Accessed 27 March 2014.

[21] OpenFlow Switch Consortium, “OpenFlow switch specification version 1.3.0,” Open Networing Foundation, Palo Alto, 2012.

[22] S. Raza and D. Lenrow, Open networking foundation north bound interface working group charter. (n.d.). [Online]. Available: https://www.opennetworking.org/working-groups/northbound-interfaces. Accessed 2 May 2014.

[23] M. Newman, *Networks: An Introduction*, Oxford: Oxford University Press, 2010.

[25] T. C. Parker, J. Johnson, M. Tummala, J. McEachen, and J. Scrofani, “Analysis of the robustness dynamics of wireless mobile ad hoc networks via time varying dual basis representation,” in the *Proc*. *48th Hawaii International Conf. System Sciences*, Kauai, 2015.

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

[26] G. Bounova and O. de Weck, "Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles," in *Physical Review,* vol. 85, no. 1, pp. 016117-1, 016117-9, Jan. 2012.

[27] F. Chung, "Spectral graph theory," in *CBMS Regional Conference Series in Mathematics*, Providence, RI, 1997.

[28] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal Ccomponent analysis," in *A Practical Approach to Microarray Data Analysis*, New York, Springer US, 2003, pp. 91–109.

[29] G. Strang, *Linear Algebra and Its Applications*, Belmont, CA: Brooks/Cole, Cengage Learning, 2006.

[30] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, Philadephia: Society for Industrial and Applied Mathematics, 1997.

[31] M. Fiedler, "Algebraic connectivity of graphs," in *Czechoslovak Mathematical Journal,* vol. 23, no. 98, pp. 298–305, 1973.

[32] R. Olfati-Saber, A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," in *Proc. IEEE,* vol. 95, no. 1, pp. 215–233, Jan. 2007.

[33] A. Sydeny, C. Scoglio, and D. Gruenbacher, "The impact of optimizing algebraic connectivity in hierarchical communication networks for transmission operations in smart grids," in the *Proc*. *IEEE PES Innovative Smart Grid Technologies (ISGT)*, Washington, DC, 2013.

[34] D. Spielman, "Spectral graph theory and its applications," in the *Proc*. *48th Annual IEEE Symp. Foundations of Computer Science*, Providence, RI, 2007.

[35] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," in *Physical Review E,* vol. 74, no. 3, pp. 036104-1 - 036104-19, Sept. 2006.

[36] J. Scott, *Social Network Analysis*, Los Angeles: Sage, 2013.

[37] S. Wasserman and K. Faust, *Social Network Analysis*, Cambridge: Cambridge University Press, 1994.

[38] P. Bonacich, "Power and centrality: a family of measures," in *American Journal of Sociology,* vol. 92, no. 5, pp. 1170–1182, Mar. 1987.

[39] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: bringing order to the web," Stanford InfoLab Publication, Palo Alto, 1999.

[40] L. Donetti and M. Muñoz, "Detecting network communities: a new systematic and efficient algorithm," in *Journal of Statistical Mechanics: Theory and Experiment,* vol. 2004, no. 10, pp. P10012 - P10019, Oct. 2004.

[41] J. Shi and J. Malik, "Normalized Cut and Image Segmentation," in *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 8, pp. 888–905, Aug. 2000.

[42] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," in *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 11, pp. 1101–1113, Nov. 1993.

[43] N. Nise, *Control Systems Engineering*, Menlo Park, CA: Addison-Wesley Publishing Compnay, 1995.

[44] Y. Y. Liu, J. J. Slotine, and A. L. Barabási, "Observability of complex systems," in *Proc. the Nataional Academy of Science,* vol. 110, no. 7, pp. 2460–2465, Feb. 2013.

[45] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," in *IEEE Trans. Signal Processing,* vol. 50, no. 2, pp. 174–188, Feb. 2002.

[46] N. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proc. F (Radar and Signal Processing),* vol. 140, no. 2, pp. 107–113, Apr. 1993.

[47] P. Van Mieghem, Graph eigenvectors, fundamental weights and centrality metrics for nodes in networks. (n.d.). arXiv. [Online]. Available: http://arxiv.org/abs/1401.4580. Accessed 11 August 2015.

[48] T. C. Parker, J. Johnson, M. Tummala, J. McEachen, and J. Scrofani, "Dynamic state determination of a software-defined network via dual basis representation," in *Proc*. of *8th International Conf. Signal Processing and Communication Systems*, Gold Coast, 2014.

[49] J. Johnson, "Software defined network monitoring scheme using spectral graph theory and phantom nodes," M.S. thesis, Dept. of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA, 2014.

[50] M. Maxie, "Congestion Analysis in a SDN," M.S. thesis, Dept. of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA, 2015, in preparation.

[51] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *IEEE/ACM Trans. Networking ,* vol. 19, no. 6, pp. 1717–1730, Dec. 2011.

[52] T. Herinckx, "Dynamic and performance driven control of OpenFlow networks," M.S. thesis, Dept. of Information Technology, Ghent University, Belgium, 2013.

[53] N. Mohan, T. Undeland, and W. P. Robbins, *Power Electronics*, Hoboken: John Wiley & Sons, 2007.

[54] RYU Project Team, RYU SDN Framework. 20 April 2014. [Online]. Available: http://osrg.github.io/ryu/index.html. Accessed 21 July 2015.

[55] Python homepage. Python Software Foundation. (n.d.). [Online]. Available: https://www.python.org/. Accessed 21 July 2015.

[56] HP 2920 series switches. (n.d.). Hewlett-Packard Company. [Online]. Available: http://www8.hp.com/us/en/products/networking-switches/product-detail.html?oid=5354494. Accessed 21 July 2015.

[57] HP 3800 series switches. (n.d.). Hewlett-Packard Company. [Online]. Available: http://www8.hp.com/us/en/products/networking-switches/product-detail.html?oid=5171624. Accessed 21 July 2015.

[58] B. Smith, *Raspberry Pi Assembly Language RASPBIAN Beginners: Hands On Guide*, CreateSpace Independent Publishing Platform, 2013.

[59] Iperf homepage. (n.d.). [Online]. Available: http://sourceforge.net/projects/iperf2/. Accessed 21 July 2015.

[60] O. Ugurlu, "New heuristic algorithm for unweighted minimum vertex cover," in *Proc*. *International Conf. Problems of Cybernetics and Informatics*, Baku, Azerbaijan, 2012.

[61] Y. Kim and M. Mesbahi, "On maximizing the second smallest eigenvalue of a state-dependent graph Laplacian," in *IEEE Trans. Automatic Control,* vol. 51, no. 1, pp. 116–120, Jan. 2006.

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