BUILDING A RELIABLE AND SECURE MANAGEMENT FRAMEWORK FOR SOFTWARE-DEFINED NETWORKS

A DISSERTATION IN
Computer Networking and Communications Systems
and
Computer Science

Presented to the Faculty of the University
of Missouri-Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
FAHEED A.F. ALENEZI

B.S., Northern Border University, 2012
M.S., University of Missouri-Kansas City, 2016

Kansas City, Missouri
2021

BUILDING A RELIABLE AND SECURE MANAGEMENT FRAMEWORK FOR SOFTWARE-DEFINED NETWORKS

Faheed A.F. Alenezi, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2021

ABSTRACT

The Software-Defined Networking (SDN) technologies promise to enhance the performance and cost of managing both wired and wireless network infrastructures, functions, controls, and services (i.e., Internet of Things). However, centralized management in softwarization architecture poses new security, reliability, and scalability challenges. Significantly, the current OpenFlow Discovery Protocol (OFDP) in SDN induces substantial issues due to its gossipy, centralized, periodic, and tardy protocol. Furthermore, the problems are aggravated in the wireless and mobile SDN due to the dynamic topology churns and the lack of link-layer discovery methods.

In this work, we tackle both security and reliability management issues in SDN. Specifically, we design and build a novel multitemporal cross-stratum discovery protocol framework, which efficiently orchestrates different reliability monitoring mechanisms over SDN networks and synchronizes the control messages among various applications. It facilitates multiple discovery frequency timers for each target over different stratum instead of using a uniform discovery timer for the entire network. It supports many common reliability monitoring factors for registered applications by analyzing offline and online

network architecture information such as network topologies, traffic flows, virtualization architectures, and protocols. The framework consists of traffic-aware discovery (TaDPole), and centrality-aware protocol (CAMLE) facilities. We implemented the framework on Ryu controller. Extensive Mininet experimental results validate that the framework significantly improves discovery message efficiency and makes the control traffic less bursty than OFDP with a uniform timer. It also reduces the network status discovery delay without increasing the control overhead.

We then evaluated the security issues in SDN and proposed an SDN-based Wormhole Analysis using the Neighbor Similarity (SWANS) approach as a novel wormhole countermeasure in a Software-defined MANET. As SWANS analyses the similarity of neighbor counts at a centralized SDN controller, it apprehends wormholes not only without requiring any particular location information but also without causing significant communication and coordination overhead. SWANS also countermeasures various false-positive and false-negative scenarios generated by the Link Layer Discovery Protocol (LLDP) vulnerability. We performed extensive studies via both analysis and simulations. Our simulation results show that SWANS can detect wormhole attacks efficiently with low false-positive and false-negative rates.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled "Building a Reliable and Secure Management Framework for Software-Defined Networks," presented by Faheed A.F. Alenezi, candidate for the Doctor of Philosophy degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Sejun Song, Ph.D., Committee Chair
Department of Computer Science and Electrical Engineering

Baek-Young Choi, Ph.D.
Department of Computer Science and Electrical Engineering

Farid Nait-Abdesselam, Ph.D.
Department of Computer Science and Electrical Engineering

Cory Beard, Ph.D.
Department of Computer Science and Electrical Engineering

Praveen Rao, Ph.D.
Department of Computer Science and Electrical Engineering

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to express my deepest thanks to my advisor, Dr. Sejun Song, for his tremendous support during my Ph.D. journey. Under his supervision, I have gained many research skills. He has always been supportive and encouraging for me to achieve the best results. I am very grateful for the hours he has spent over the recent years evaluating my publications. Without his generous feedback and direction, I would not be in this great position. I would also like to thank Dr. Beak-Young Choi, my co-advisor, for her advice and support. I am very thankful for her academic help and guidance. Next, I would like to extend my thanks to the committee members Dr. Farid, Dr. Beard, and Dr. Rao for taking the time to serve on my Ph.D. committee and for their helpful advice and guidance during my research. Then, I would like to especially thank my wife, who shouldered a lot of responsibilities so that I could concentrate only on my Ph.D. research, and my kids (Rose and Faisal), who have always been my main strength and inspiration during my studies. Finally, I would like to thank my parents and my family for being my biggest supporters.

CHAPTER 1

INTRODUCTION

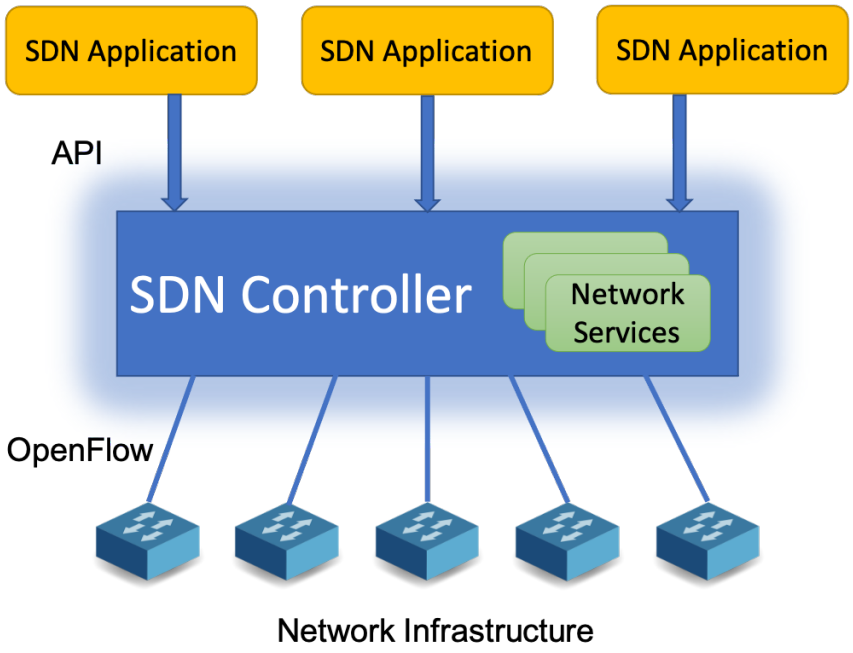


Figure 1.1: SDN Architecture

The Software-Defined Networking (SDN) [1, 2] technologies promise to simplify and reduce network management complexity and create an agile and dynamic network to enhance the performance and cost of managing both wired and wireless network infrastructures, functions, controls, and services. SDN addresses the decentralization and complexity of traditional networks by separating the network's control and data planes. The idea behind SDN architecture is to manage the entire network control traffic using a single unit

called a controller [3, 4]. As shown in Fig. 1.1, the controller works as a middle point that connects network infrastructure and applications together. It handles any communication between applications and network devices and provides a global view of the network for better control decision making using OpenFlow Discovery Protocol (OFDP). However, centralized management in softwarization architecture poses new reliability, security, and scalability challenges. Significantly, the current OpenFlow Discovery Protocol (OFDP) in SDN induces substantial issues due to its gossipy, centralized, periodic, and tardy protocol. Furthermore, the problems are aggravated in the wireless and mobile SDN due to the dynamic topology churns and the lack of link-layer discovery methods.

1.1 Reliability in Mobile, Wireless, and Wired SDN

SDN relies heavily on the OpenFlow discovery protocol (OFDP) for discovering and maintaining its network visibility [5]. OFDP is designed mainly for wired networks and utilizes the Link Layer Discovery Protocol (LLDP) to discover the network. A controller uses the Packet_Out and Packet_In messages to understand the physical connections between SDN switch nodes. The controller sends an LLDP message (Packet_Out) to all active ports of the network switches and receives two Packet_In messages per link to acknowledge the existence of a link between two switches. This process is done repeatedly (e.g., every 1 second in RYU controller) to ensure that the controller has up-to-date visibility of the network. However, current OFDP results in performance, scalability, and latency challenges when applied to wired, wireless, and mobile networks. Fast node and link failure detection can be an overwhelming task to tackle when using OFDP, as increasing the discovery frequency can saturate the network with many control traffic. SDN's discovery protocol (i.e., OFDP) also overlaps various virtualized networks when using in-band or out-of-band control paths in forwarding networks such as linear, tree, star, and wireless

topologies. It also causes many redundant control messages. Furthermore, nodes in mobile networks (i.e., mobile ad hoc networks MANETs) constantly fail and churn, which significantly decreases the control message scalability. The network size also introduces more control messages that can negatively impact the controller's performance and the network resources. Moreover, control traffic dramatically increases during the discovery period but idles otherwise, which increases the likelihood of a network collision when applied in wireless and mobile environments.

1.2 Security in Wireless and Mobile SDN

In traditional multi-hop wireless networks (e.g., Sensor Networks, MANETs, and Internet of Things (IoT)), nodes' collaboration to deliver each other's control and data traffic is essential to control and manage the network. Moving those wireless systems from distributed network architecture to a centralized one using Software-Defined Networking (SDN) can resolve many performance and management problems but not security. Even though the overhead of the control plane functionalities such as discovering neighboring nodes and applying routing algorithms moved from the wireless nodes to the centralized controller, nodes still need to exchange control traffic for helping the controller discover the network topology.

In Software-Defined Networking (SDN), nodes do not have any intelligence. They forward the control and data traffic to the appropriate location based on the controller's instructions. This process operates without any authentication method. The collaboration between the wireless nodes to form the network topology makes the network susceptible to many security challenges. In particular, one of the most challenging security concerns in wireless and mobile networks is a wormhole attack [6, 7]. Wormhole attackers leverage the open-architecture-based broadcast medium in such networks to gain unauthorized ac-

cess to the network. The wormhole attack can be deployed in the network anonymously and without being revealed. As illustrated in Fig. 1.2, Wormhole attackers collaborate to position themselves in the network in remote locations to attract control and data packets into one point and replay them to another point through the attacker's implicit direct wireless or wired communication links. The attackers can then disrupt routing decisions to perform numerous data and control traffic manipulations by selectively dropping, flooding, recording, or modifying packets without revealing their identity [8].

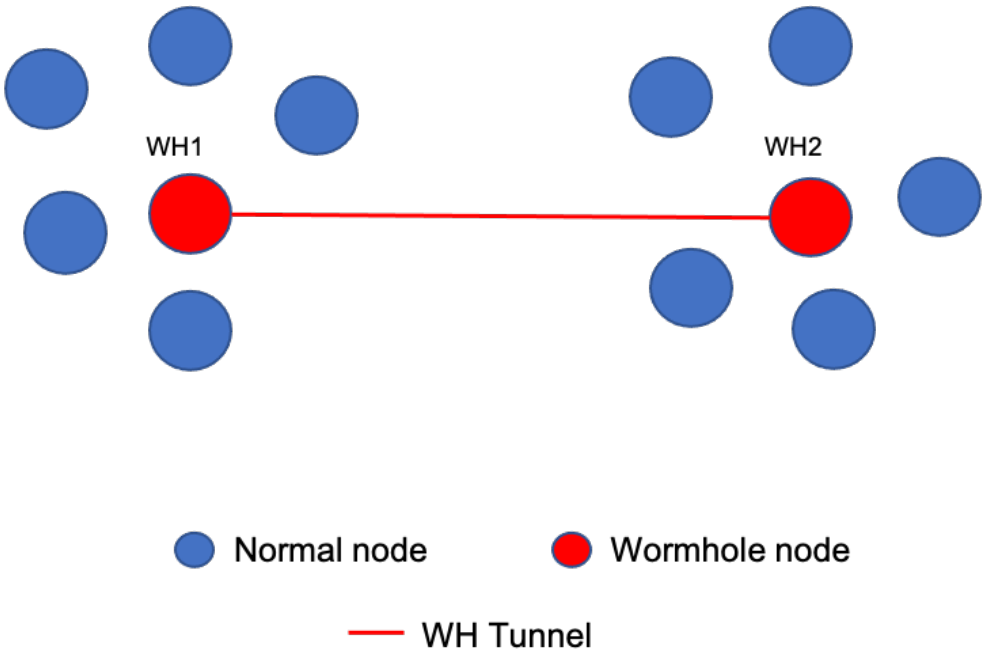


Figure 1.2: Wormhole Attack

1.3 Objective of This Study

The main direction of this work is to build a reliable and secure management framework for Software-Defined Networks (SDN). The proposed work enhances the reliability, scalability, accuracy, latency, and security issues of discovery protocols in Software-Defined Networks (SDN). Our target SDN networks include wired, wireless, and mobile networks. Specifically, we tackle three primary components in terms of the reliability management and security of the discovery protocol. With regard to reliability management, we build a Traffic-aware Discovery Protocol for Software-Defined Wireless and Mobile Networks (TaDPole) and a Centrality-Aware Multitemporal Discovery Protocol for Software-Defined Wireless and wired Networks (CAMEL). From the security perspective, we develop an SDN-based Wormhole Analysis using the Neighbor Similarity for mobile and Wireless networks (SWANS).

1.3.1 TaDPole: Traffic-aware Discovery Protocol

In TaDPole, we design and build a novel Traffic-aware Discovery Protocol for wireless and mobile SDN. It facilitates multiple discovery frequency timers for each target instead of using a uniform discovery timer for the entire network. TaDPole calculates the significance of each discovery target according to the recent network usage by assuming that the higher traffic node has more impact on the network service. It lessens discovery delay by increasing the discovery frequency to the more critical nodes. Also, it enhances the control message efficiency by reducing the discovery frequency to the less significant targets. Besides, it supports the port-neutral broadcast-based discovery method instead of using port-specific request and response approaches. Extensive Mininet experiment results validate that TaDPole improves discovery message efficiency by two times and makes the control traffic less bursty than OFDP with a uniform timer. It reduces the network status

discovery delay by three times without increasing the control overhead. we are to achieve the following objectives in TaDPole.

- Avoid control traffic concentration and collision scenarios by enabling several frequency timers to send LLDP packets rather than using a single discovery timer for the whole network.
- Increase the discovery frequency only for the most critical nodes to reduce discovery delay. The importance or criticality of each target may be determined dynamically based on network traffic.
- Enhance OFDP to support the port neutral broadcast-based discovery for wireless and mobile SDN to reducing the control traffic amount.

1.3.2 CAMEL: Centrality-Aware Multitemporal Discovery Protocol

CAMEL presents a novel Centrality-Aware Multitemporal (CAMEL) discovery protocol for SDN to enhance the centralized discovery mechanism's scalability and latency issues. We use a Multi-Temporal Discovery (MTD) model that facilitates multiple discovery timers for nodes in the network and distributes each node according to the significance rather than using a single timer for the entire network. CAMEL generalizes the significance measurement for various network topologies by using degree and betweenness centrality models to identify significance nodes in the network. Applying the identified significance to a multitemporal discovery mechanism, CAMEL reduces network impact by decreasing the discovery delay to the significant nodes and enhances control message efficiency by lowering the discovery frequency to the less significant targets. We have implemented CAMEL on the RYU controller. The experimental results validate that CAMEL improves discovery message efficiency, makes the control traffic less bursty, and enhances the net-

work service quality by reducing discovery delay to the significant nodes. We are to achieve the following objectives in CAMEL.

- Enhance the centralized discovery mechanism's scalability and latency issues by facilitate multiple discovery timers for each target according to the significance instead of using a single timer for the entire network.
- Generalize a node significance measurement for various network topologies by using the centrality models such as degree and betweenness centralities.
- Reduce network impact by improving the discovery rate to the significant nodes and enhances traffic efficiency by decreasing the discovery cycle for the less critical targets.

1.3.3 SWANS: SDN-based Wormhole Analysis

SWANS proposes an SDN-based Wormhole Analysis using the Neighbor Similarity (SWANS) approach as a novel wormhole countermeasure in a Software-defined MANET. As SWANS analyses the similarity of neighbor counts at a centralized SDN controller, it apprehends wormholes not only without requiring any particular location information but also without causing significant communication and coordination overhead. SWANS also countermeasures various false-positive and false-negative scenarios generated by the Link Layer Discovery Protocol (LLDP) vulnerability. We performed extensive studies via both analysis and simulations. Our simulation results show that SWANS can efficiently detect various intelligent wormhole attacks, keeping low false-positive and false-negative rates.We are to achieve the following objectives in SWANS.

- Propose a Wormhole Analysis using the Neighbor Similarity method for SDNs.

- Design a real-time neighbor similarity analysis algorithm on a centralized SDN controller, which apprehends wormhole attackers not only without requiring any detailed topology information but also without causing significant communication and coordination overhead.

- Countermeasure various intelligent wormhole attacker models (generate false-positive and false-negative scenarios), which assesses the Link Layer Discovery Protocol (LLDP) vulnerability.

1.4 Organization

This study is organized as follows. Chapter 2 shows the background of SDN discovery protocol and Wormhole attack. In chapter 3 we present the literature review of improving the reliability management and security of SDN. Chapter 4 describes the Traffic-aware Discovery Protocol for wireless and mobile SDN (TaDPole). We discuss the Centrality-Aware Multitemporal Discovery Protocol for Software-Defined Wireless and wired Networks (CAMEL) in Chapter 5. Chapter 6 introduces the SDN-based Wormhole Analysis using the Neighbor Similarity for a Mobile ad hoc network MANET (SWANS) and Chapter 7 concludes the study.

CHAPTER 2

BACKGROUND

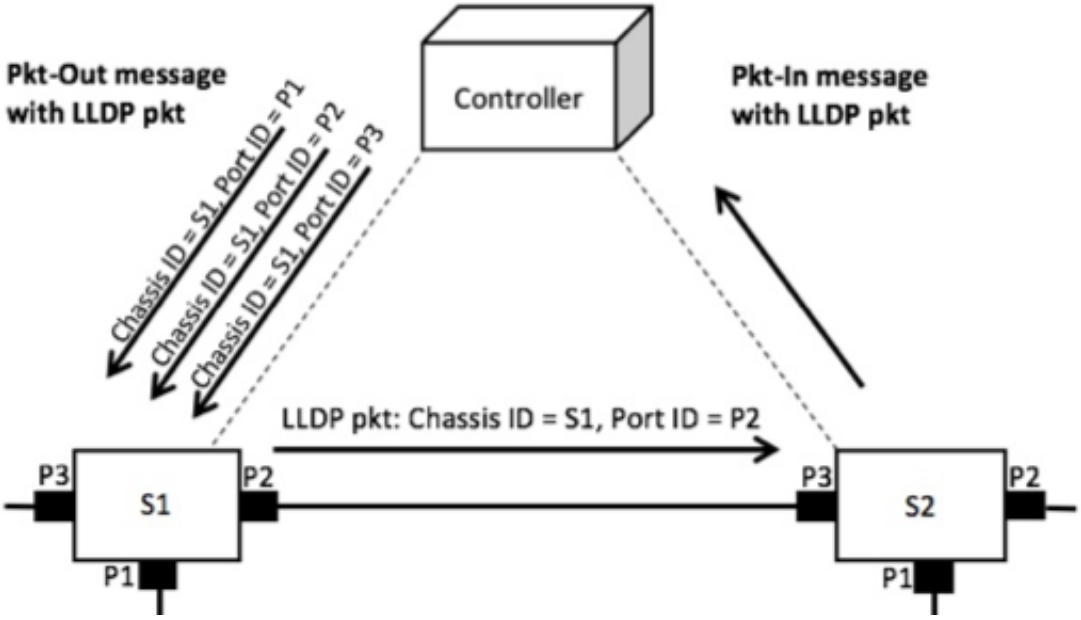


Figure 2.1: Topology discovery in OFDP

In this chapter, we present essential information about the mechanism of discovery protocol in SDN and the drawbacks of implementing it in both wired and wireless infrastructures in terms of reliability Management. We also provide an overview of the vulnerability of OFDP and how it can be easily affected by severe security attacks such as wormhole attack.

2.0.1 Discovery Protocol in SDN

SDN uses OpenFlow discovery protocol (OFDP) as a centralized discovery protocol to transmits information about the current status of a switch and the capabilities of its ports. To determine the topology, the controller uses OFDP to discover three essential components of the network switches, links, and hosts [9]. In classical OFDP, the topology discovery process begins with the switches. The switches try to establish a TCP connection with the controller. When the TCP connection begins, the controller discovers the switches by sending an OpenFlow Feature request message for each switch and waits for an OpenFlow Feature reply message. The feature request message is simply asking the switch to provide the controller its abilities, such as datapath-id (a unique identifier of the switch) and active ports in the switch. The controller uses two neighbor discovery protocols to discover the Links in the network: Link Layer Discovery Protocol (LLDP) and Broadcast Domain Discovery Protocol (BDDP). The latter protocol is used to discover multi-hop OpenFlow switches, where the OpenFlow switches are separated by one non-OpenFlow switch or more in the same broadcast domain. For a single-hop (directly connected OpenFlow switches) discovery, the controller uses LLDP. Since the controller already knows the active ports in each switch from the switch discovery process above, an LLDP speaker (i.e., switch.py in RYU) periodically sends LLDP packets to all the ports in each switch by encapsulating the LLDP packet in a Packet-out message (Chassis ID and Port ID), as shown in Fig. 2.1. When S1 receives the LLDP packets, it forwards them to their appropriate ports P1, P2, and P3. Upon receiving the LLDP message via S2 P3, S2 encapsulates the LLDP packet in a packet-in message and sends it to the controller for acknowledging a directional link between the switch ports.

The topology discovery protocol (i.e., OFDP using LLDP) [10] is an essential feature for SDN to perform many network applications such as routing, traffic engineering,

load-balancing, etc. However, LLDP is a vendor-neutral link layer protocol configured as an optional component in network management and monitoring applications in traditional networks [11]. As it was initially designed for advertising local information such as identity, capabilities, and immediate neighbors among distributed network devices, the protocol is stateless, chatty, and insecure. If OFDP is used in the wireless and mobile environment, it may cause various congestion, collision, and security problems.

2.0.2 Wormhole Attack in SDN

Due to the lack of message authentication and the simple structure of OFDP messages, OFDP is vulnerable to many security attacks [12, 13, 14], such as wormhole attacks. The wormhole attackers can utilize this vulnerability to falsely convince other nodes in the network that they are within the range of remote nodes to attract packets to go through their implicit tunnel. As illustrated in Fig. 2.2, an LLDP speaker periodically sends a port-neutral LLDP embedded in a Packet-Out message to all member nodes. Upon receiving an LLDP Packet-Out message, each node broadcasts an LLDP request message to the neighbors by embedding the node ID. The receiving node then sends a Packet-In message back to the controller with node IDs (both source and destination nodes) for acknowledging the neighbor information. When there is a wormhole attacker W_x, it can replay all the LLDP request messages within the range to its neighbors via a wormhole tunnel. For example, while node S_1 is in the communication range of a wormhole node W_1, the node S_1 receives all LLDP requests from the nodes within S_1's communication range X_{com} as well as from the nodes around W_1 and W_2 as the wormhole node W_1 replays the incoming LLDP request messages at W_2 through the virtual tunnel. Hence, when there is a wormhole attacker, the SDN controller will receive more Packet-In notifications (neighbors) from the wormhole-affected nodes for an LLDP request. When there is a packet forwarding message, the SDN controller's routing protocol identifies a route (forwarding neighbor nodes

on the path) from source to destination using the shortest hop-count algorithm according to neighbor status in the neighbor table. For example, when H_1 sends data packets to H_5, it takes the shortest route from S_1, S_2, S_n, S_4, and S_5 before any wormhole attack. However, it will take the shortest route from S_1 and S_5 via W_x after a wormhole attack. As routes via W_x become the shortest route for many forwarding cases, a wormhole attacker W_x can receive and manipulate both control and data packets.

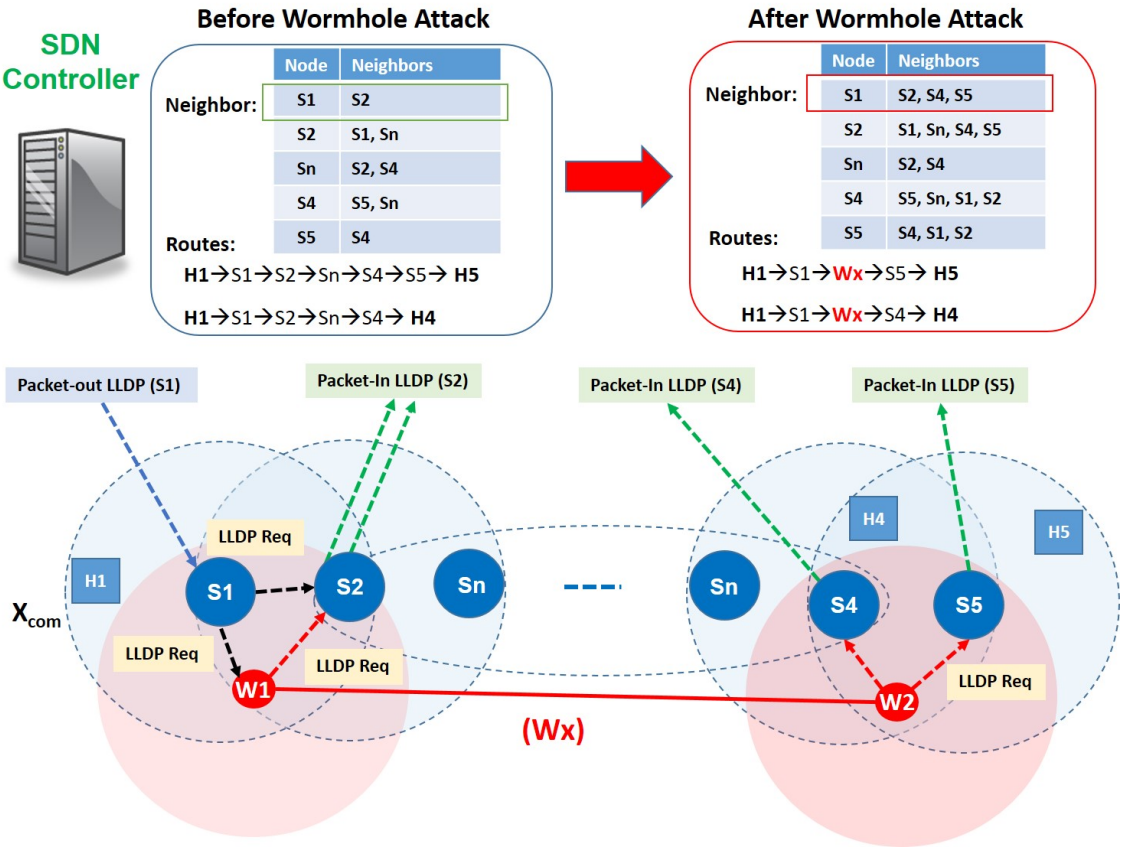


Figure 2.2: Impact of wormhole attacker on neighbors and routes

CHAPTER 3

LITERATURE REVIEW

This chapter discusses the literature review in improving the reliability management and security of software-defined Networks. We organize the chapter into two sections to represent this thesis's main two objectives: the reliability management and security of discovery protocol in software-defined networks.

3.1 SDN Reliability Management

There are a few research papers published to overcome the performance, scalability, latency, and reliability challenges in OpenFlow Discovery Protocol (OFDP). Most of these papers focus on enhancing the switch logic to handle topology discovery messages efficiently and overcome the control message overhead of discovery protocol in weird SDN. [15] discusses the limitations of using OFDP in the aspect of security and scalability. They provided a precise analysis of the limitations of using OFDP and how using a fixed period to discover the network topology can be problematic when considering the dynamicity and size of the network. They proposed sOFTD protocol to enhances security and efficiency issues in OFDP by delegating the controller's security functions into the forwarding switches. [16] enhances the OpenFlow Discovery Protocol (OFDP), named OFDPv2, by sending a single Packet-Out message to each switch in the network instead of sending the message to the entire active ports. OFDPv2 limits the number of LLDP Packet-Out messages sent to each switch by using an OpenFlow OFPT FEATURES REQUEST at the connection estab-

lishment message between OF and the switches. When the switches receive the Packet-Out message, they send a copy of this message to every active port. While it reduces the amount of Packet-Out messages, it does not lessen the number of LLDP messages exchanged between the switch nodes. Also, the Packet-In messages to acknowledge the existence of a link between switches are still the same. OFDPv2's main objective is to reduce the CPU load imposed by the SDN controller's topology discovery service [17].

[18] propose another way of discovering network topology. Their approach is to gather LLDP messages directly from network devices. When a new switch is added or removed, the neighboring switches are responsible for refreshing their local tables and reporting the topology changes to the controller. [19] improves the topology discovery protocol by changing how the switches react when they receive an LLDP message from other switches in the network. Instead of sending the LLDP directly to the controller as a packet-in, the switch sends the LLDP packet back to the source switch, and the source switch sends the packet to the controller as a packet-in to form the bidirectional link between the switches.[20] proposed a different scheme for discovering network topology. The idea of SD-TDP is to lighten the controller's load by dividing the entire network into father nodes (FN) and active nodes (AN). The FN node sends the topology information received from all the AN associated with it to the controller. [21] propose LADP to discover the network topology. Their approach is to make the controller select a core switch randomly and send the packet-out message to it. The core switch can then broadcast the LLDP message to the neighboring switches. The neighboring switch then sends the LLDP to their neighbors and so on. Each switch is also responsible for sending the received LLDP message to the controller as a packet-in.

[22] uses multiple frequencies to represent different zones in the network. It is designed for a fixed tree topology. They divided the tree topology into three separate zones (core, aggregation, and edge). The core zone has more frequent discovery messages than

aggregation and edge zones, as a failure in the core zone has a higher impact on the network than the other zones. By paying more attention to the core zone, they can achieve fast detection and recovery to the failure. Also, the edge zone has less attention because a failure in an edge node has little effect on the network service. However, it is a hierarchy-based that maintains different control frequencies according to the static network topology, which cannot be used in a wireless and mobile SDN.

Our approach is novel because it facilitates a general significance identification method by using the recent network usage for wireless and mobile networks and centrality models for wired and static wireless networks. Various network topologies, including tree, star, linear, grid, and wireless networks, can use the proposed method without any manual configuration. It normalizes the network usage and centrality value so that nodes with higher network usage or centrality value have more impact on the network. Applying the identified significance to a multitemporal discovery mechanism enhances the network service quality by reducing discovery delay to the significant nodes without increasing the overall control overhead. It gives more attention to the critical components (i.e., node with more impact to the network service), while the overall control messages can be reduced.

3.2 SDN Security

In this section, we examine the existing solutions in the literature towards the wormhole attack. Many countermeasures techniques have been proposed to mitigate wormhole attack for wireless and mobile networks (i.e., MANET). Most of these techniques focus on utilizing devices such as GPS, directional antennas, and timing devices.

[23] proposed two types of leashes geographic and temporal leashes to detect the wormhole attack. The geographic leash is used to ensure that the packet's receiver is within the sender's range, which requires each node to know its location. The temporal leash

is used to set an expiration time for each packet to prevent the packets from traveling too far in the network. [24] proposed three machine learning-based Intrusion Detection Systems (IDSs) to detect wormhole attacks in IoT networks. The three systems are K-means clustering, decision tree, and a compilation of K-means and decision tree systems named hybrid-IDs. The K-means clustering system groups the nodes in the network into different clusters (safe zones) based on their locations from the root node. The root node accepts neighbor update requests from any two nodes that belong to the same safe zone and rejects requests between nodes in different safe zones as they are considered an attack. The decision tree approach uses a threshold (safe distance) between two directly connected nodes to detects wormhole attacks. The threshold is calculated by taking the mean of the distances between all directly connected nodes using a trained data set. When the distance between two nodes exceeds the threshold, it is considered an attack. Lastly, the hybrid-IDs uses both the safe zones and safe distance concepts to detect a wormhole attack. Combining the two approaches gives two layers of verification. If the two nodes do not belong to the same safe zone, they check the safe distance between them. If the safe distance is less than the threshold, the neighbor update request is accepted.

[25] proposed a detection technique to the wormhole attack based on fuzzy logic and artificial immune system. The fuzzy logic phase distinguishes the efficient routes from the other routes based on different parameters such as residue energy, the distance between nodes, and hop count. The immune phase is then used to select the immune routes from the selected efficient routes in phase one. [26] proposed a statistical approach using neighbors to detect the wormhole attack in mobile WSNs. They identify the wormhole attack by counting the recent number of neighbors of each node and compare it with the previous count. The node will experience a rapid change in neighbors' count when passing a wormhole zone. [27] proposed a detection method for wormhole attack based on round trip time (RTT) and hop count. The detention process consists of two phases. In the first phase, the

source node calculates RTT from itself to all its direct neighbors. If the RTT value of a node is higher than the average RTT of all neighbors, it is considered a suspicious node and added to a suspicious list. The RTT value may differ because of some factors such as congestion and processing delay, which may introduce many false-positives when applied in wireless and mobile networks.

[28] proposed a hybrid method (WRHT) to detect the wormhole attack. The proposed method is based on two popular techniques used to identify the wormhole attack. The two techniques are Watchdog and Delphi. WRHT uses the packet drop Information from Watchdog technique and RTT delay probability from Delphi to calculate whether a route has a wormhole attack or not. [29] proposed a detection method based on an artificial neural network using neighborhood count. They use a mobile node as a detector node. The detector node randomly visits locations in the network and count neighbors in each site. the information collected by the detector node is then used as a dataset to train the neural network and detect a sudden increase in neighbors' numbers. [30] proposed a trust-based approach to avoid wormhole attackers in MANETs. The parameters used to decide whether a path is trusted or not are round trip time (RTT) threshold, packet drop ratio(PDR) threshold, and rate of energy consumption threshold. They randomly select three pairs of nodes and calculate their average threshold of RTT, PDR, and energy consumption rate and use the average threshold value to evaluate the network nodes. Nodes with less RTT, PDR, and energy consumption threshold than the average threshold value considered legitimate otherwise, they are malicious.

[31] They proposed a Wormhole attack detection technique in WSNs by identifying different routes between wireless nodes. They set a threshold for the number of two-hop neighbors for each node in the network. when the number of two-hop neighbors for a node exceeds the threshold value, the algorithm then concludes that the network under a wormhole attack. [32] proposed a detection technique for mobility-based WSNs based on

the rate of neighbors change for a node and the length of paths from the source to the destination node. They introduce an upper and lower threshold for the rate of change. Any node that has a change rate higher than the upper limit is considered malicious. Nodes with a change rate lower than the lower bound are normal. Nodes between the upper and lower threshold are suspicious and are added to a suspicious list. The path length approach determines whether the path is fake or not using a pre-defined path length threshold. [33] proposed a mitigation approach for the Wormhole attack in wireless Adhoc networks by electing a coordinator node. The coordinator sends a special coordinator message to all nodes in the system. After that, each node has to respond with an acknowledgment message and the path from itself to the coordinator. The coordinator then examines the routes to determine the existence of a wormhole tunnel.

Our approach differs from the existing literature because we propose to counter-measure the wormhole attack in Software-defined MANET using a light-weight neighbor counting algorithm that recognizes various intelligent wormhole attacks efficiently with low false-positive and false-negative rates. It also does not require any special devices, location information, and significant communication and coordination overhead.

CHAPTER 4

TRAFFIC-AWARE DISCOVERY PROTOCOL FOR SOFTWARE-DEFINED WIRELESS AND MOBILE NETWORKS

4.1 Introduction

The Software-Defined Networking (SDN) technologies [34, 35, 36] for network functions, controls, and applications are promising, as traditional configuration-based closed-loop control approaches are inefficient, unreliable, and unscalable for the highly dynamic wireless and mobile network infrastructures (i.e., Internet of Things). In SDN, where the control plane responsibility is moved to the logically centralized controller, the controller maintains up-to-date network membership by using the remote node's status notifications and periodic discovery messages. For example, a controller identifies its member nodes when they initiate a TCP connection to the controller according to the initial configuration. Besides, it periodically discovers the network membership by using discovery protocols such as the Link Layer Discovery Protocol (LLDP) [37], Broadcast Domain Discovery Protocol (BDDP) [10], and OpenFlow Discovery Protocol (OFDP) [15, 36]. In SDN, OFDP is an essential centralized discovery protocol that transmits information about the current status of a device and the capabilities of its interfaces. SDN applications rely heavily on OFDP to discover and maintain their network visibility.

However, due to the gossipy, repetitive, and slow protocol nature, the centralized discovery protocols pose both scalability and latency challenges in SDN architecture. Each controller maintains *a uniform period timer* for discovery protocol and periodically requests

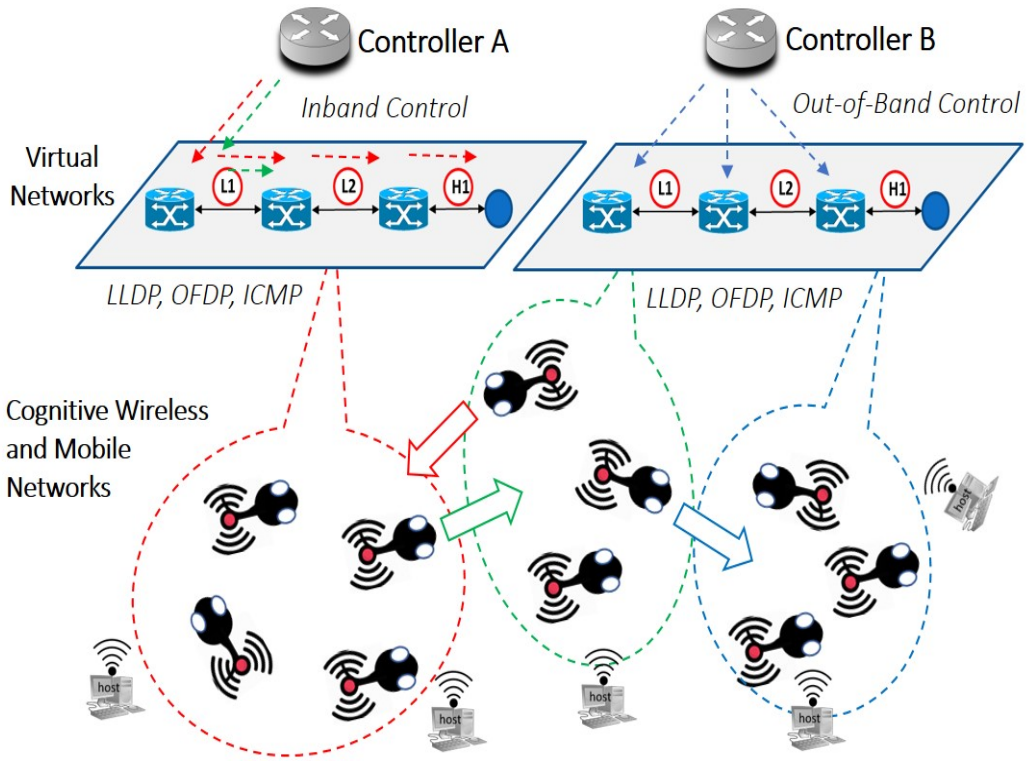


Figure 4.1: Software-defined wireless and mobile networks

network status for each port. The total number of LLDP messages for each discovery period is three times more than the entire amount of network switch ports, including the inter-SDN switch port, host port, and non-SDN switch port. Hence, the control message scalability decreases significantly if the network size and the discovery frequency increase. As illustrated in Fig. 4.1, if the forwarding network is used by various virtualized networks as well as is overlapped by forwarding the control path via linear networks (an inband SDN control), many redundant control messages will be introduced to the front (i.e., L1 which is near to the controller) of the network switches. Furthermore, unlike the wired static SDN, the wireless and mobile SDN depends solely on the discovery protocol, as it cannot support the link-layer detection protocols such as Bidirectional Forwarding Detection (BFD) [38] and OpenFlow Fast Failover [39]. As wireless links introduce frequent status changes due to various potential glitches and node churns (in Fig. 4.1), the control message scalability decreases significantly if the network size and the discovery frequency increase in highly dynamic wireless environments. Besides, SDN is periodically congested by the control traffic while the network is idling most of the time. It increases the chance of the message collision. We present initial experimental results of control message scalability and latency issues and discuss the motivation and objectives in Section 4.2. Although there have been a few studies that address the problems of failure detection and recoveries in the wired static SDN data plane [40] and control plane [41, 42] networks, respectively, very little work is done for the discovery protocols in wireless and mobile SDN.

In this chapter, we propose a Traffic-aware Discovery Protocol (TaDPole) to enhance the scalability and latency issues of the centralized discovery mechanism for wireless and mobile SDN. TaDPole consists of three discovery enhancement modules, including *a multiple timer module, an impact identification module, and a wireless and mobile discovery module*.

- **The multiple timer module** maintains various discovery frequency timers for each target instead of using a uniform period for the entire network. Also, it supports a timer configuration facility in support of the wireless and mobile networks. It assigns different discovery frequencies according to the significance of the network service of each target. It decreases discovery delay by increasing the discovery frequency to the more critical nodes. Also, it enhances the control message efficiency by reducing the discovery frequency to the less significant targets.

- **The impact identification module** classifies the impact of each target to the network service by using recent network usage, including the amount of data input and output, the number of flows and the duration, protocol types on each switch port. It collects the SDN switch ports information from both off-line and on-line and assigns the impact of each target by assuming that the higher traffic node has more influence on the network service. It can also facilitate various reliability monitoring parameters such as protocol type, the heartbeat mechanism, period, and the target for the registered applications, which becomes a useful decision-making mechanism for taking corrective action against membership churns in the wireless and mobile SDN.

- **The wireless and mobile discovery module** is a modified network membership discovery protocol in support of broadcast-based wireless and mobile SDN. On the SDN controller, it sends a port-neutral LLDP request to each switch instead of sending multiple LLDP requests for the entire ports in the switch. On each wireless and mobile switches or Access Points (AP), it broadcasts an LLDP message to the neighbors by embedding the switch ID. If a switch (not a host) node receives an LLDP message, it sends an LLDP message back to its controller with both switch IDs. Instead of discovering the network topology, it classifies the neighbor membership for each wireless and mobile switches.

We implemented the proposed TaDPole modules into the RYU controller's LLDP facility (i.e., switches.py) [43]. The Mininet [44] based experimental results show that TaDPole expedites the membership churn detection on the critical network segment without impacting the network scalability. TaDPole enhances discovery message efficiency by two times and makes the control traffic less bursty than OFDP with a uniform timer. Also, it decreases the network status discovery delay by three times without increasing the control overhead.

The remainder of this chapter is organized as follows. Section 4.2 discusses the observations that motivated this work and propose the objectives of enhancing the discovery protocol.We introduce our proposed solution and the implementation details in Section 4.3. Section 4.4 provides the experimental results, and Section 4.5 concludes the chapter.

4.2 Motivational Experiments and Objectives

4.2.1 Experiments

We have conducted initial control message performance experiments using Mininet with an RYU SDN controller by varying the switch numbers from 4, 10, and 50 with a daisy chain network topology. We captured control messages for 10 mins in a default configuration without inserting any data traffic. For its controller connection, we used both *an out-of-band control SDN*, where each switch has a dedicated connection to its SDN controller and *an inband control SDN* where each switch relays the control messages toward its SDN controller without any dedicated connection.

Fig. 4.2 shows various initial control messages generated by both RYU and Cisco's OpenDayLight (ODL) [45] SDN controllers. Among the control messages TCP, ICMP, and LLDP, periodic LLDP messages are dominant (around 60% in RYU and 70% in ODL), followed by TCP (about 38% in RYU and 25% in ODL). Specifically, RYU has a relatively large number of control messages than ODL as its default LLDP timer is 1 second, which

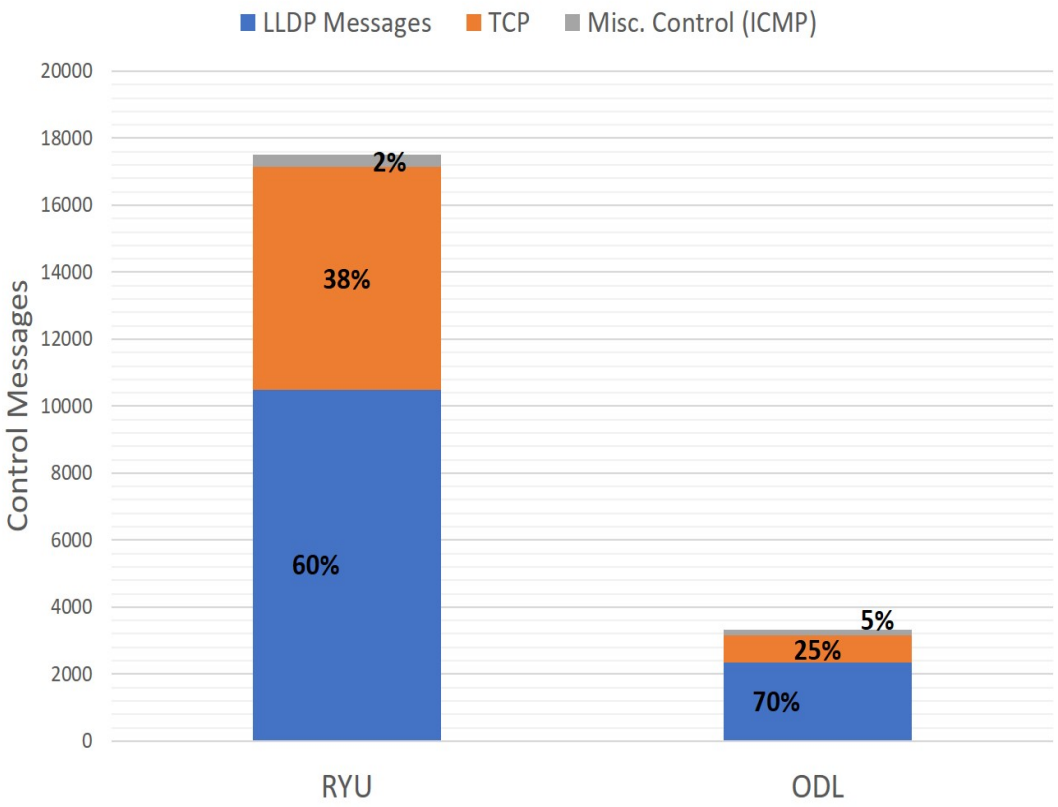


Figure 4.2: Control message types of ODL and RYU

is five times faster than the ODL controller (i.e., ODL's default LLDP timer is 5 second).

Fig. 4.3 presents accumulative LLDP message counts on the out-of-band control over daisy chain networks with 4, 10, and 50 switches, respectively. As the network size increases, the amount of control messages linearly increases. However, as shown in Fig. 4.4, if an SDN network segment uses an inband control connection over daisy chain networks, the amount of control messages dramatically increases along with the network extension. It is due to the control path overlap across the network switches. In the four switch network, the inband control creates 100% more control traffic than the out-of-band control. However, in the ten switch network, the inband control creates 400% more control traffic than the out-of-band control. Even worse, Fig. 4.5 presents that fifty node daisy chain network

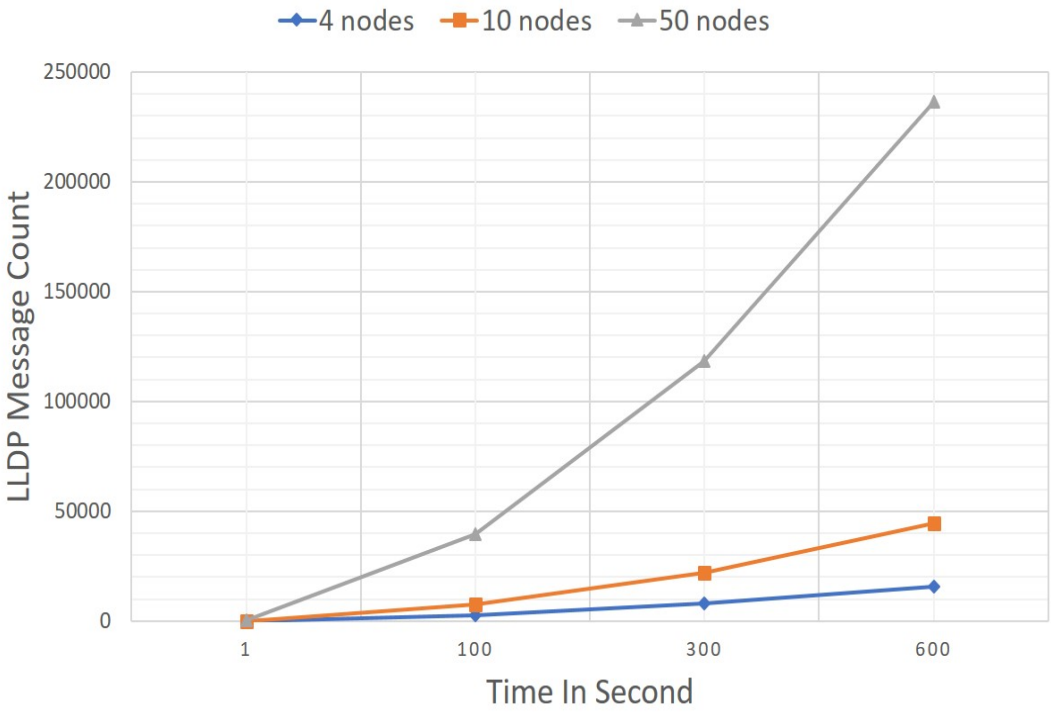


Figure 4.3: Out-of-band LLDP messages over daisy chain networks

generates twenty times more control traffic than ten node daisy chain network. Besides, Fig. 4.6 shows the control message patterns over the LLDP period for both 4 and 10 switch networks. It presents that the LLDP messages are concentrated in a timer period while the network traffic is idling most of the time. It is mainly due to a single LLDP frequency timer for the entire network.

4.2.2 Objectives

According to observation from the initial SDN control message performance experiments, we are to achieve the following objectives.

- **Subduing the control traffic burstiness:** Due to a single discovery frequency timer for the entire network, the LLDP messages are concentrated in a period while the

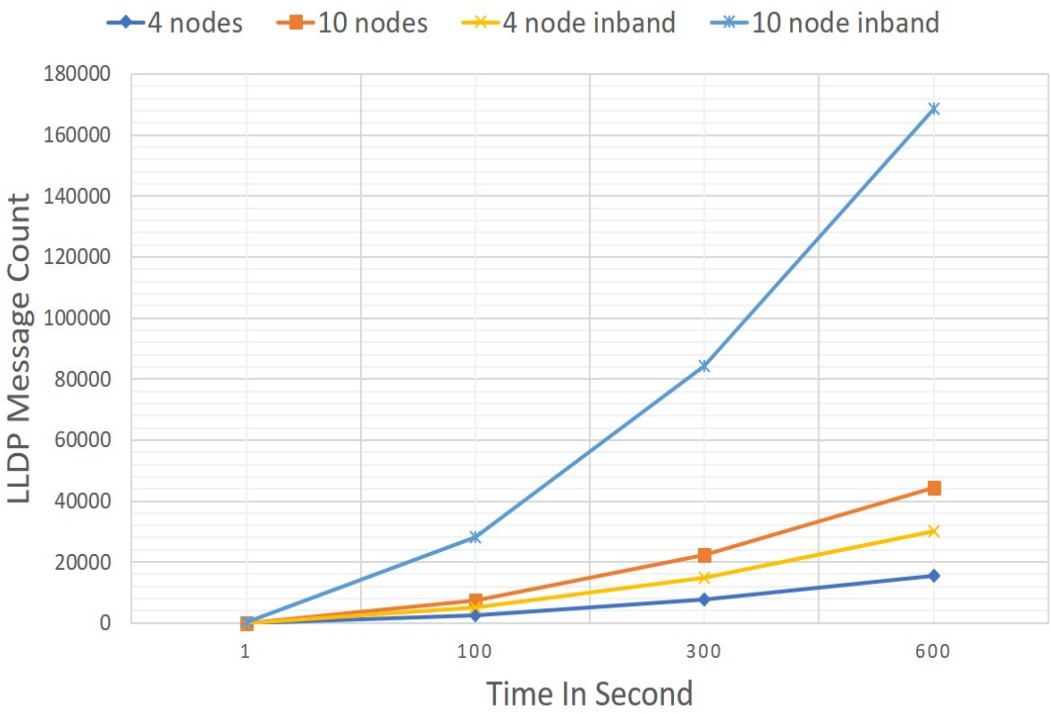


Figure 4.4: Inband LLDP messages over daisy chain networks

network traffic is idling most of the time. The control traffic concentration can cause more severe problems in wireless and mobile networks by causing various collision scenarios. We are to design an algorithm to scatter control traffic by facilitating multiple discovery frequency timers for each target instead of using a uniform discovery timer for the entire network.

- **Lessening the discovery protocol lateness:** The discovery lateness can be improved by decreasing the discovery frequency timer. However, it will increase the control traffic amount. We are to reduce discovery delay by increasing the discovery frequency only for the more critical nodes. The impact or criticality of each target can be dynamically identified according to the network traffic.

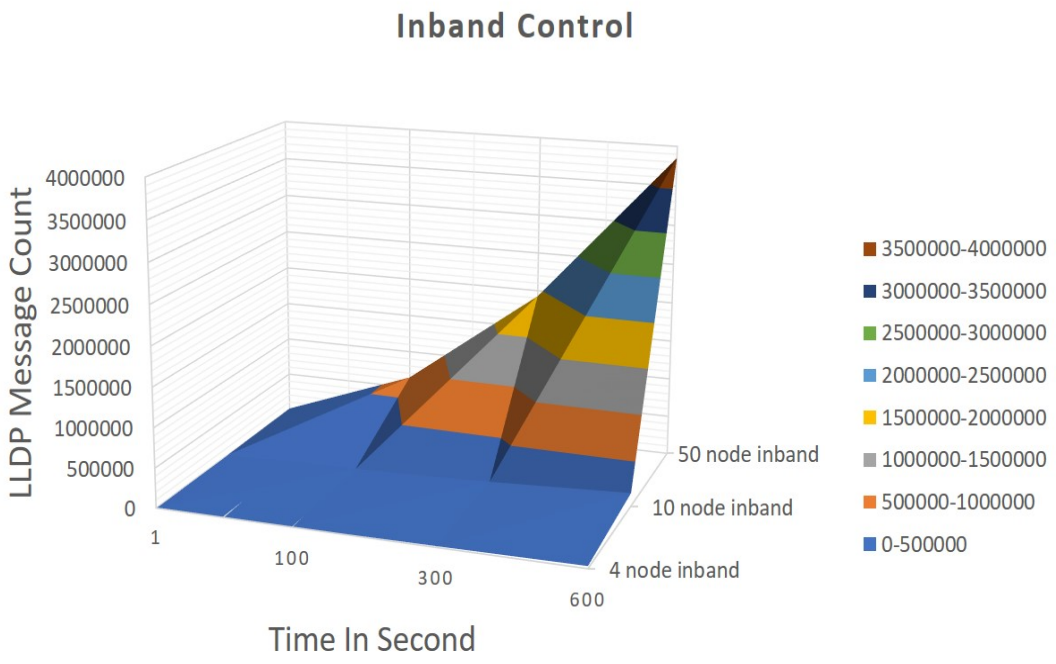


Figure 4.5: Inband LLDP messages upto 50 daisy chain networks

- **Reducing the control traffic amount:** The amount of control traffic increases along with the discovery frequency and network size. We are to enhance OFDP in support of the port neutral broadcast-based discovery for wireless and mobile SDN.

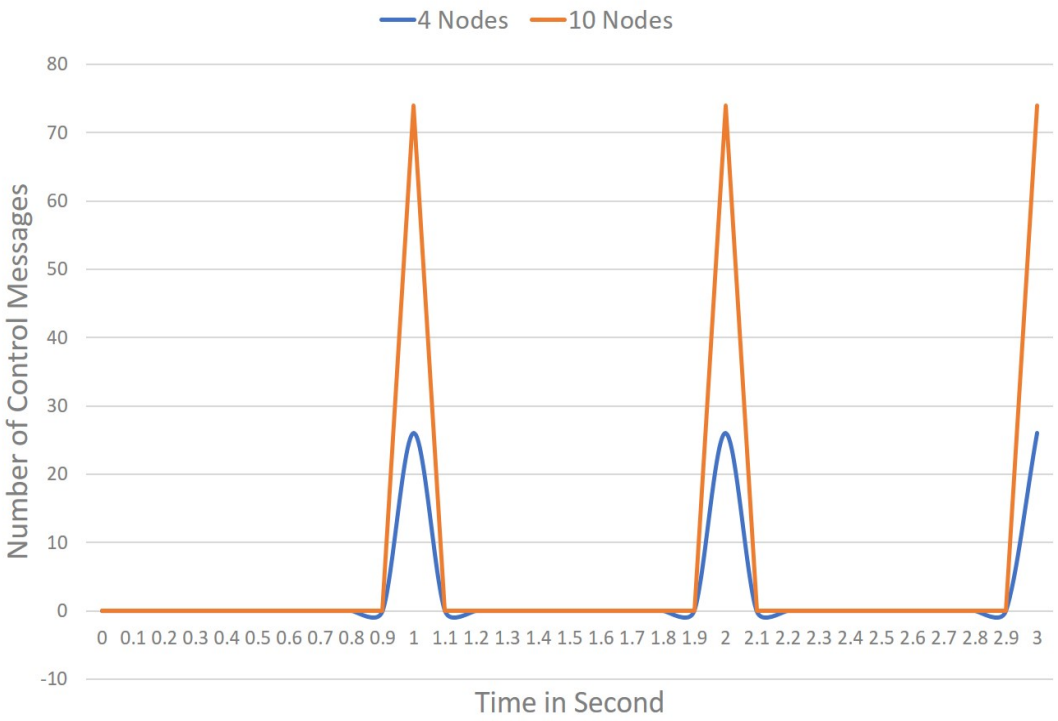


Figure 4.6: Periodic control messages

4.3 TaDPole: Traffic-aware Discovery Protocol Architecture and Implementation

4.3.1 TaDPole Architecture

The Traffic-aware Discovery Protocol (TaDPole) enhances the scalability and latency issues of the centralized discovery mechanisms for wireless and mobile SDN. It comprises three enhancement modules, including *a multiple timer module, an impact identification module, and a wireless and mobile discovery module*.

The multiple timer module improves OFDP by using multiple discovery frequency timers for each target instead of using a uniform period for the entire network and dynamically adjusts the LLDP message frequency to each destination according to the significance of the network service of each target. Algorithm 1, LLDP_Zone_Classifier()

assigns each switch node into a few different frequency zones according to the significance of the network service of each target. It initially assigns a timer on each zone and dynamically adjusts the timer frequency according to the node traffic. INITIAL-CLASSIFIER() takes the sorted node list to split them into different zones. The classifier values, including the number of zones and the range of each zone, initially use fixed values from the configuration. The average value of each zone is used to calculate the initial threshold. It assigns nodes to each zone based on the calculated initial threshold value and identifies the timer for each zone. Each zone has a normal sending rate and a threshold rate (limits the control traffic according to the network traffic). A thread of Send_lldp_packet() sends LLDP packets according to the scheduled timers to each node. TIMER-ADJUSTER() adjusts the sending rate in each zone based on the amount of control traffic to the overall data traffic. It calculates each zone traffic and checks how much is the control traffic to the data traffic. If the traffic of a node in a zone becomes out of the adjust-rate, it can be reallocated to a different zone, and its timer adjusted accordingly. If the control traffic is higher than the adjust-rate of the overall traffic, the sending rate decreases to avoid any potential network congestion. When the zone control traffic (LLDP traffic) is higher than the adjust-rate of the average of the overall traffic, the controller slows the sending rate for that zone by limiting its max traffic. It will reduce the control traffic in each zone and avoid overflowing the zone that is already saturated by the data traffic and also send less LLDP packets when the data traffic in the zone is too low. It decreases discovery delay by increasing the discovery frequency to the more critical nodes. Also, it enhances the control message efficiency by reducing the discovery frequency to the less significant targets.

The impact identification module classifies the impact of each target to the network service by using recent network usage, including the amount of data input and output, the number of flows and the duration, protocol types on each switch port. It collects the SDN switch ports information from both off-line and on-line and assigns the impact of each

Algorithm 1 LLDP_Zone_Classifier()

Input: nodes_list // a list of all switches in the network
Output: LLDP packet

```
function MAIN()
    sort (nodes_List)
    Initial-Classifier()
    Timer-Adjuster()
While True:
    // sends LLDP packets per timer
    Send_lldp_packet()
end function
function INITIAL-CLASSIFIER()
While True:
    // Split the list into initial i zones (fixed split)
    zone[i].id = set-zone(nodes_list, zone_num)
    // Get zone threshold using average usage of each zone
    zone[i].threshold = get-threshold(zone.[i].ave)
    zone[i].timer = get-initial-timer(zone[i])
end function
function TIMER-ADJUSTER()
While True:
    // Periodically adjust zone and timer values
    // Track LLDP traffic history for each zone
    zone[i].count = zone[i].count + new-lldp-traffic
    // migrate zone i to j and adjust timer using adjust-rate
    if zone[i].threshold >= adjust-rate * zone[i].count then
        zone[j] = set-new-zone(zone[i])
        zone[j].timer = adjust-timer(zone[i])
    end if
end function
```

target by assuming that the higher traffic node has more influence on the network service. Algorithm 2, Node_Impact_Identifier(), is a lightweight network traffic monitoring algorithm. Both REQUEST_NODE_TRAFFIC(nodes_list) and TRAFFIC_REPLY_HANDLER (event) threads collect the amount of data and control traffic from each switch node periodically (i.e., for every one min). Although the current algorithm design gets the data from the network nodes, it will be able to get the update directly from the controller as many applications will need the information. It maintains the most recent one min information. Although we use the amount of data (in bytes) for network usage measurement in this algorithm, it can facilitate various parameters, including duration, priority, protocol, actions (the amount of input and output), and the number of flows. The control traffic is calculated by using a counter in each zone, and every time the zone sends LLDP packets, the counter increases. It can also facilitate various reliability monitoring parameters such as protocol type, the heartbeat mechanism, period, and the target for the registered applications, which becomes a useful decision-making mechanism for taking corrective action against membership churns in the wireless and mobile SDN.

The wireless and mobile discovery module redesigns the OpenFlow discovery protocol (OFDP) in support of the port neutral broadcast-based discovery for wireless and mobile SDN. Instead of discovering the network topology, it classifies the neighbor membership for each wireless and mobile switches. The traditional wired SDN controller sends LLDP request packets to each switch ports (not a switch) embedded in a Packet-Out message. For example, as illustrated in Fig. 4.7, it sends three Packet-Out messages for each port ID (P1, P2, and P3) for a switch node k. When a switch node k receives a Packet-Out message, it punts an LLDP request message to the designated switch port. If a receiving switch port is not a host port, the switch node sends an LLDP message to the SDN controller by embedding it in a Packet-In message. However, as illustrated in Fig. 4.8, the SDN controller sends a port-neutral LLDP request to each switch in the wireless and mo-

Algorithm 2 Node_Impact_Identifier()

Input: nodes_list // a list of all switches in the network
Output: Impact

```
function MAIN()
    for all key, value ∈ GetnodesTraffic[key] do
        Impact[key] = New-Impact[key]
        New-Impact[key] = value
        Impact[key] = New-Impact[key] - Impact[key]
    end for
end function
function REQUEST_NODE_TRAFFIC(nodes_list) //thread to send a traffic information request
While True:
    for all switches ∈ nodes_list do
        req = OFPPortStatsRequest(nodes_list
        nodes_list.send_msg(req)
    end for
end function
function TRAFFIC_REPLY_HANDLER(event) //thread to listen to the replies from switches and save traffic data to GetnodesTraffic
    for all traffic ∈ event.msg.body do
        key = (event.datapath, port_no)
        value = nodeTraffic
        GetnodesTraffic[key]= value
    end for
end function
```

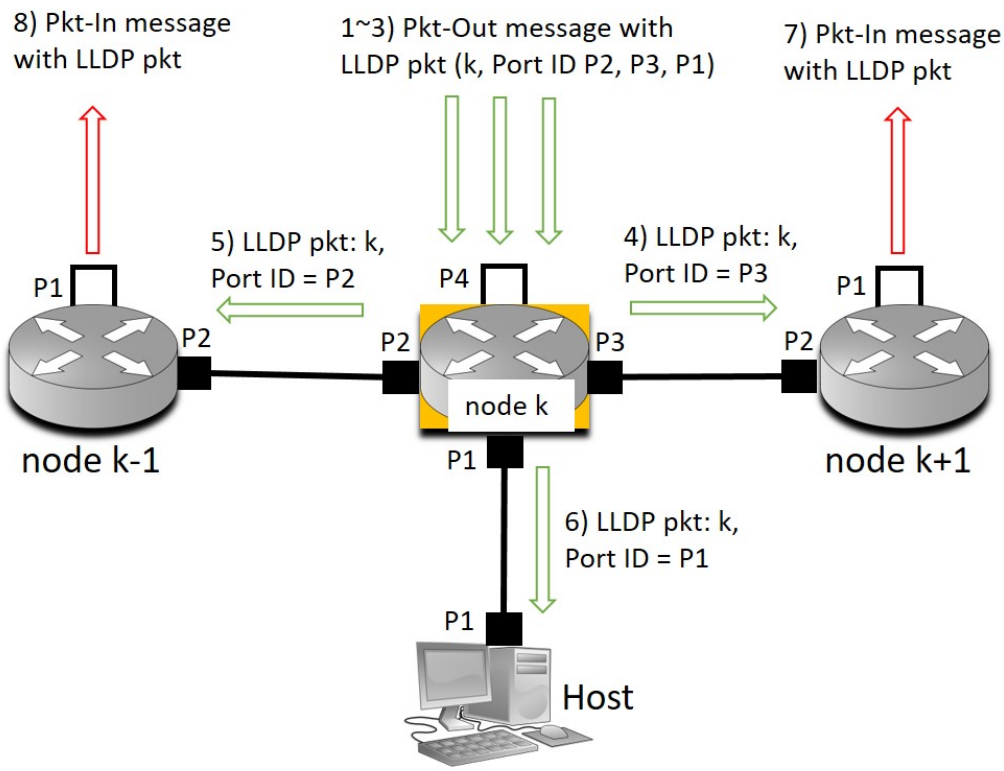


Figure 4.7: LLDP messages over wired network

bile SDN. On each wireless and mobile switches or Access Points (AP), it broadcasts an LLDP message to the neighbors by embedding the switch ID. Hence, the existing port-based OFDP approach is verbose on its Packet-Out and LLDP messages. We modified the OFDP protocol to send a single packet for both Packet-Out and LLDP neighbor discovery. If a switch (not a host) node receives an LLDP message, it sends an LLDP message back to its controller with both switch IDs (source and destination switches). The SDN controller can identify neighbor nodes for each switch by using the switch IDs.

As presented in Equation (4.1), in the wired SDN with an out-of-band control connection, the total number of LLDP control messages (T_n) is proportional to the total number of the switch ports (P_n). For each switch port, it creates three LLDP messages (Packet-Out,

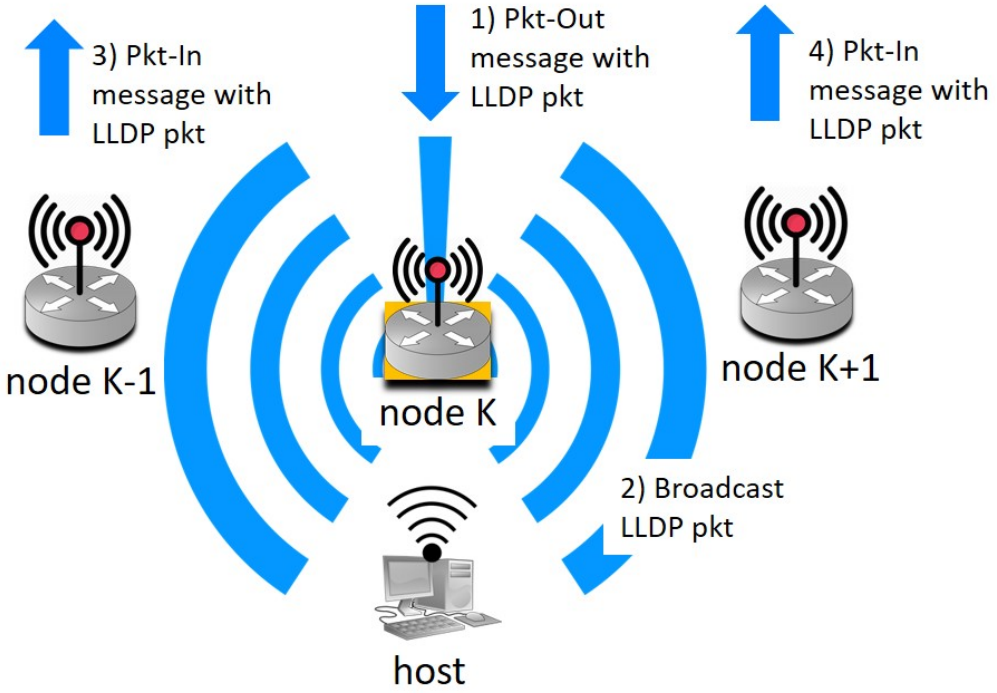


Figure 4.8: LLDP messages over wireless networks

LLDP-port, and Packet-In). As the host ports do not send any Packet-In message to the SDN controller, it creates one less LLDP packet than the switch port. In general, T_n is approximately three times of P_n ($\approx$ O(P_n)).

$$T_n = (3(P_n - H_n) + 2H_n) \approx 3(P_n) \tag{4.1}$$

However, using wireless and mobile discovery module, T_n becomes proportional to the total number of the switches (S_n) as presented in Equation (4.2). For each switch, it broadcasts three LLDP messages (Packet-Out, LLDP-port, and Packet-In). As the host ports do not send any Packet-In message to the SDN controller (less LLDP-port) and there are more than one neighbor switches (more Packet-In), T_n is approximately three times of S_n ($\approx$ O(S_n)). Considering the total number of neighbor switches is equivalent to P_n, the

port-neutral approach's T_n is three times less than the port-based T_n.

$$T_n \approx 3(S_n) \approx (P_n) \tag{4.2}$$

4.3.2 TaDPole Implementation

As illustrated in Fig. 4.9, the impact identification, and multiple timer modules are implemented in the switches(app_manager.RyuApp) class, which is responsible for the event handling functions, including port status, switch node status, and has an LLDP_loop() function to control the periodic discovery frequency and sends LLDP frames. LLDP_loop() is an application in the Openflow module for sending LLDP frames. LLDP_loop() thread also listens on DataChange events such as nodes added or removed and node-links added or removed from the network nodes. It maintains all the information of connected node IDs and node-connectors in the switches(app_manager.RyuApp) class. The node-connector consists of the node ID and port number. The LLDP-loop module runs a thread that sends the LLDP frames packaged into Openflow PACKET_OUT messages to all learned nodes for every discovery period. We intercept this routine to embed those two modules. Algorithm 2, Node_Impact_Identifier(), collects traffic information from each switch node. As many applications depend upon the information, it can facilitate the information as a common data repository. It stores the information into the Impact data class. Algo-

Table 4.1: Discovery Model Notations

Metrics	Notes
T_n	Total number of LLDP messages
P_n	Total number of switch ports
H_n	Total number of host ports
S_n	Total number of switch nodes

rithm 1, LLDP_Zone_Classifier(), presents an LLDP discovery frequency function that returns an LLDP_frequency value for each target switch (switch_ID). The LLDP_frequency determines how often the LLDP-Speaker sends a discovery request to a specific switch. First, it calls the INITIAL-CLASSIFIER() function to split switch nodes into the different frequency timer zones. It initially reads an LLDP control message period from the configuration using the zone value. Also, it gets Impact data from Node_Impact_Identifier() using switch ID. Second, using the zone number, it calls the TIMER-ADJUSTER() function. The frequency timer value and the zone assignment can be dynamically adjusted and reassigned according to the network condition. For every LLDP_frequency (period), the multiple timer module punts the LLDP packet for a switch (target) to OpenFlow Packet Processor. The wireless and mobile discovery module is implemented by modifying the Lldp_packet() function. Originally, a Lldp_packet(dpid, port_no, dl_addr, TTL) API contains dpid (chassis ID), port_no (Port ID), dl_addr (source address), and TTL (Time-To-Live) parameters. However, we replaced both dpid (chassis ID) and port_no (Port ID) with switch_id. In an Openflow packet processing module, the Lldp_packet() function calls the Send_lldp_packet(), which is, a RYU API to send the packets out to switches.

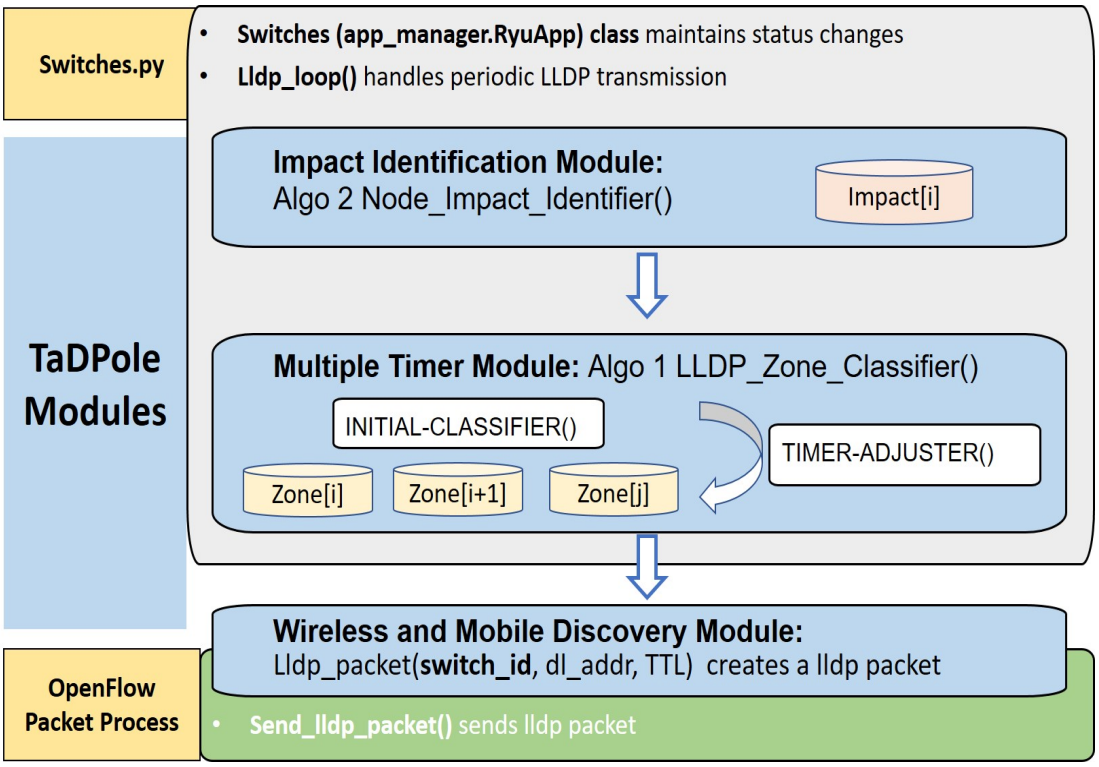


Figure 4.9: Traffic-aware Discovery Protocol (TaDPole) implementation

4.4 Evaluation

4.4.1 Experimental Setup

We investigate the performance of TaDPole using Mininet emulator [46] with real implementation on a RYU controller [43]. For the emulation environment, we used iMac with 3.0GHz 6-core Intel Core i5 and Ubuntu 19.04 OS. We have conducted a couple of experiment sets: 1) discovery protocol overheads and burstiness between RYU and TaDPole in both wired and wireless conditions, and 2) the accumulated impact from the network service outage (detection agility). For the experiments, as illustrated in Fig. 4.10, we use simple linear topologies with 4, 10, and 50 switches, and one host is connected to each switch. We test both inband and out-of-band control connection cases. We use an iperf

tool [47] to generate data traffic, and Wireshark to capture messages from the loop-back interface.

We configure simplified frequency timer sets according to three different data traffic zones (High, Mid, Low). For example, TaDPole (0.2, 1, 2) is with intervals of 0.2, 1, and 2 seconds for the high, middle, and low traffic zone switches, respectively. As the RYU controller uses a default uniform interval (one second) in sending LLDP PACKET_OUT messages to the switches, we present RYU (1, 1, 1). TaDPole (1, 1, 1) uses a default uniform interval (one second) in sending LLDP PACKET_OUT messages to the entire switches but is a discovery protocol for the wireless and mobile networks. TaDPole (0.2, 1, 2) generates five times more frequent LLDP messages than RYU (1, 1, 1) to the high traffic zone switches but sends two times less LLDP messages than RYU (1, 1, 1) to the low traffic zone switches. It also indicates that TaDPole (0.2, 1, 2) can detect the status change five times faster than RYU (1, 1, 1) in the high traffic zone but two times slower than RYU (1, 1, 1) in the low traffic zone. We use the default three consecutive LLDP message failures to change status.

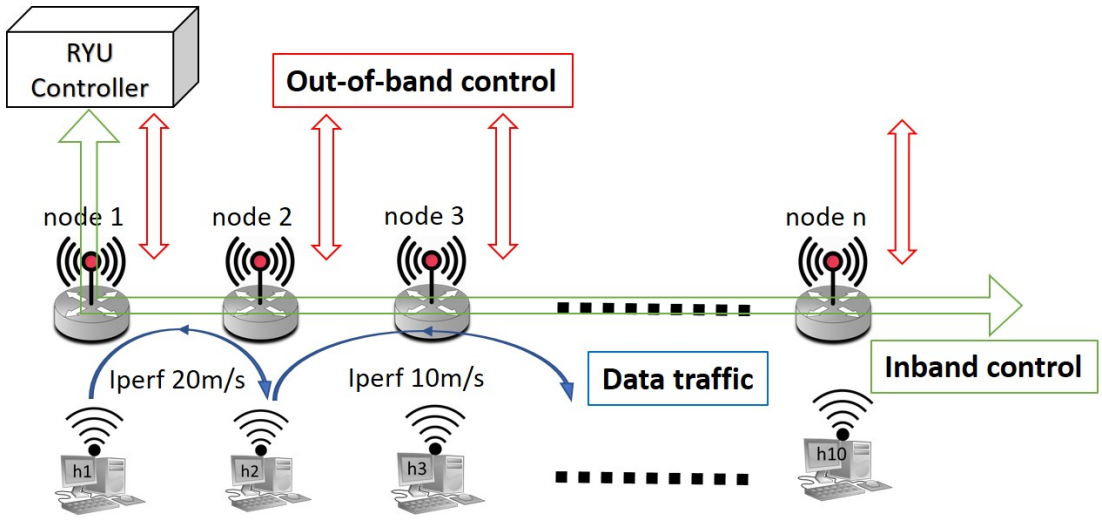


Figure 4.10: TaDPole experiment setup

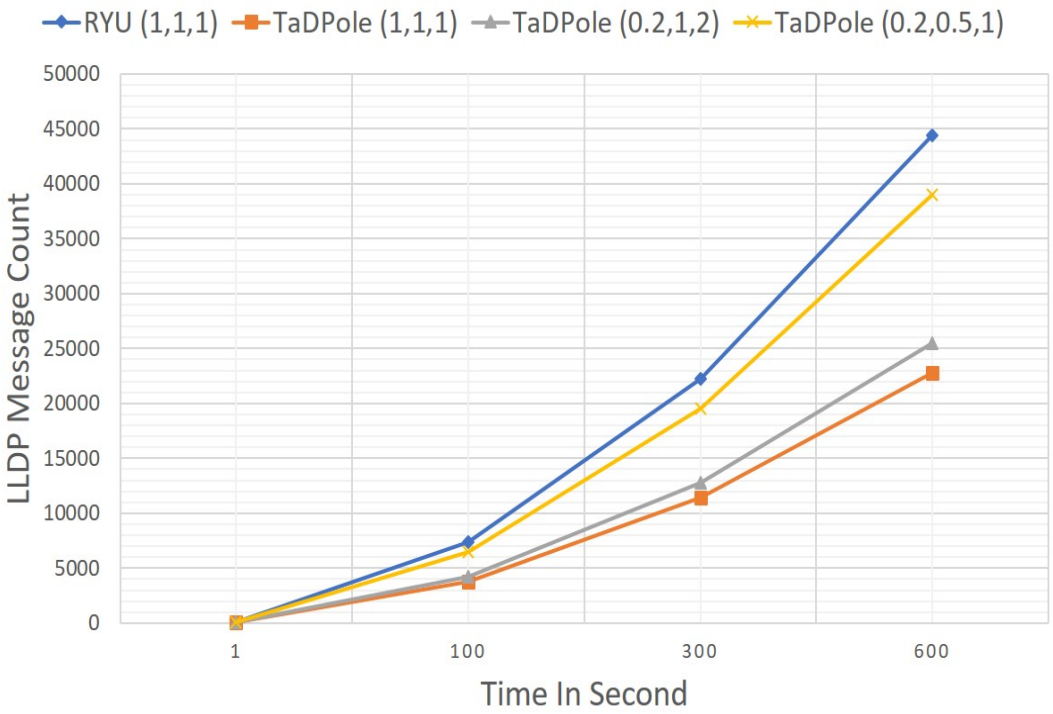


Figure 4.11: Control messages in out-of-band control network

4.4.2 Control Message Overheads

Fig. 4.11 presents the accumulative control message overhead for a ten switch out-of-band control network over 10 mins. Out of 10 switches, one switch is in the high traffic zone, three switches are in the middle traffic zone, and six switches are in the low traffic zone. All TaDPole sets create less LLDP traffic than the RYU (1, 1, 1). TaDPole (1, 1, 1), which uses a discovery protocol for the wireless and mobile networks, produces two times less LLDP traffic than RYU (1, 1, 1) for the same discovery frequency (one second). Although TaDPole (0.2, 0.5, 1) generates 70% and TaDPole (0.2, 1, 2) makes 11% more traffic than TaDPole (1, 1, 1), TaDPole (0.2, 0.5, 1) still makes 13% and TaDPole (0.2, 1, 2) creates 74% less traffic than RYU (1, 1, 1). Both TaDPole (0.2, 0.5, 1) and TaDPole (0.2, 1,

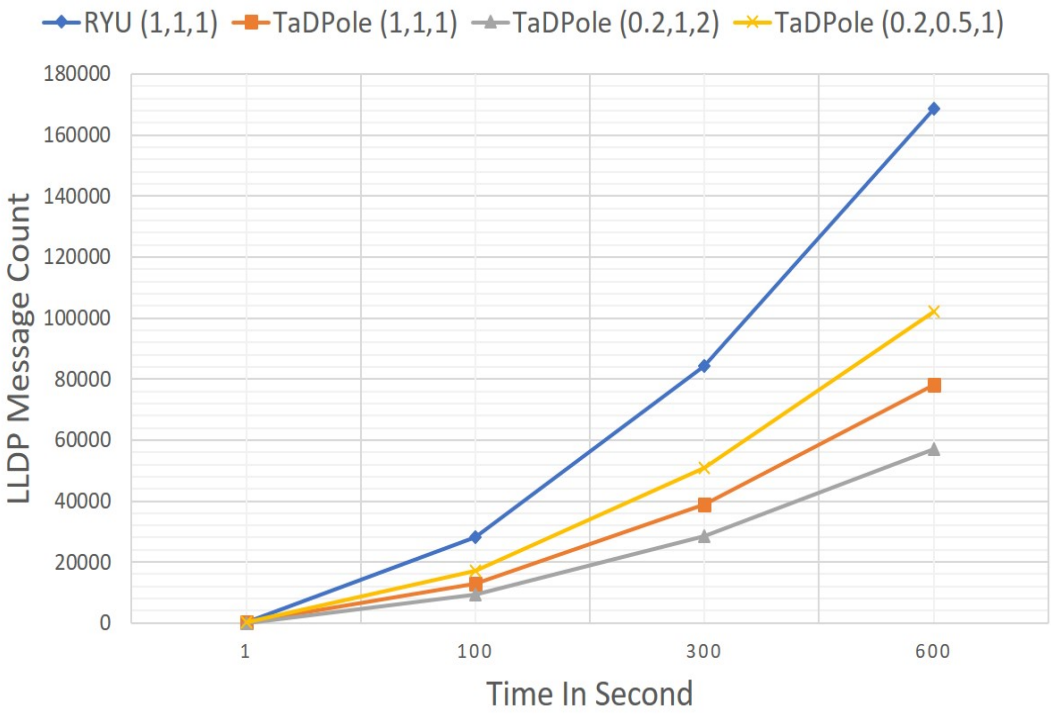


Figure 4.12: Control messages in inband control network

2) can detect the switch status changes much faster than RYU (1, 1, 1). Fig. 4.12 presents the accumulative control message overhead for a ten switch inband control network over 10 mins. Out of 10 switches, one switch is in the high traffic zone, three switches are in the middle traffic zone, and six switches are in the low traffic zone. All TaDPole sets create far less LLDP traffic than the RYU (1, 1, 1). TaDPole (1, 1, 1), which uses a discovery protocol for the wireless and mobile networks, produces two times less LLDP traffic than RYU (1, 1, 1) for the same discovery frequency (one second). Although TaDPole (0.2, 0.5, 1) generates 30% more traffic than TaDPole (1, 1, 1), TaDPole (0.2, 0.5, 1) still makes 65% less traffic than RYU (1, 1, 1). Interestingly, unlike the out-of-band case in Fig. 4.11, TaDPole (0.2, 1, 2)'s control overhead is the least and less than TaDPole (1, 1, 1) by 37%. The result implies that TaDPole can detect the switch status faster with less control overhead. Both TaDPole

(0.2, 0.5, 1) and TaDPole (0.2, 1, 2) can detect the switch status changes much faster than RYU (1, 1, 1). TaDPole (0.2, 1, 2) detects the status change faster than TaDPole(1, 1, 1).

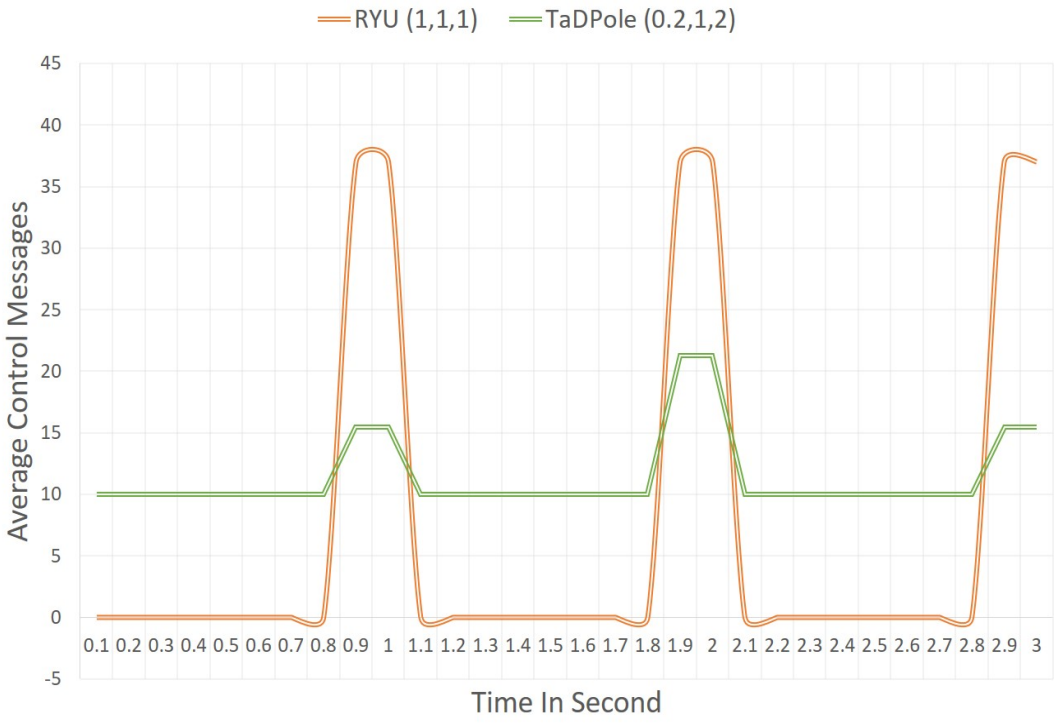


Figure 4.13: Periodic control message patterns in out-of-band network

4.4.3 Control Message Burstiness

This experiment is to show how using multiple sending interval can reduce the impact of bursting the network with LLDP messages every second. Fig. 4.13 presents the control message patterns for a ten switch out-of-band control network for 3 seconds. RYU (1, 1, 1) presents bursty control traffic patterns. Due to a single LLDP timer, the LLDP messages are concentrated in the LLDP period while the network traffic is idling most of the time. The traffic concentration may cause more serious problems in wireless and

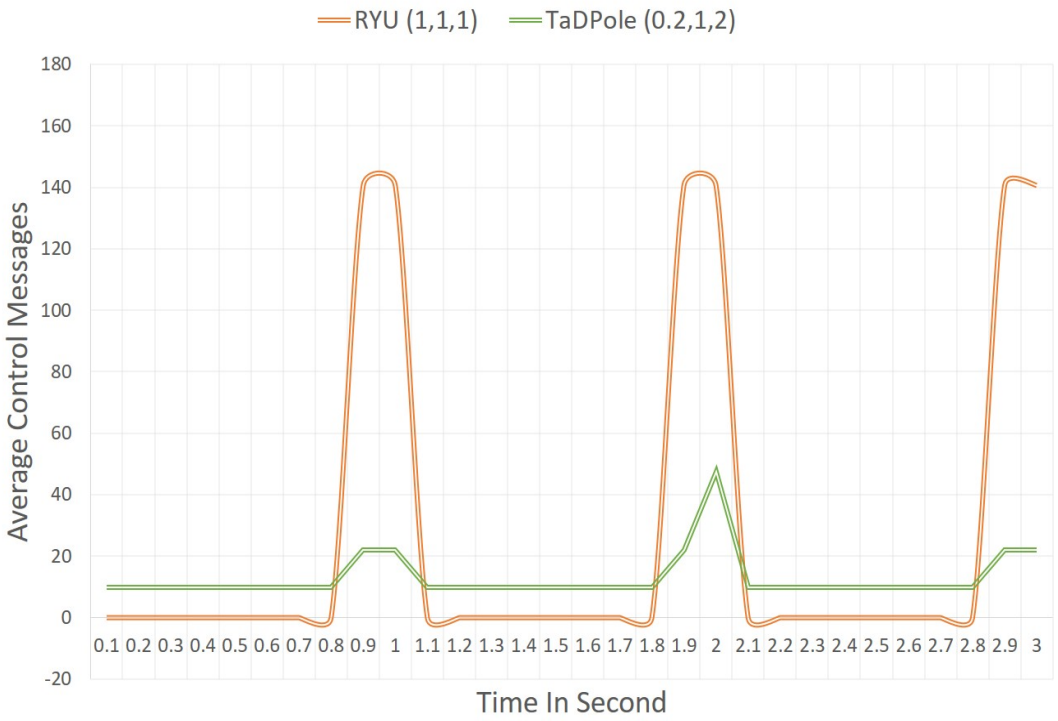


Figure 4.14: Periodic control message patterns in inband network

mobile networks by creating various collision scenarios. However, TaDPole (0.2, 1, 2) is much less bursty than RYU (1, 1, 1). Out of 10 switches, one switch is in the high traffic zone, three switches are in the middle traffic zone, and six switches are in the low traffic zone. As TaDPole (0.2, 1, 2) uses three different discovery frequencies, it can distribute the control traffic over different periods. Fig. 4.14 shows the control message patterns for a ten switch inband control network for 3 seconds. It clearly displays traffic concentration patterns in RYU (1, 1, 1). TaDPole (0.2, 1, 2) creates much smoother traffic pattern than RYU (1, 1, 1). The results are promising because creating more frequent messages does not cause a significant control message overhead. TaDPole hastens network status detection and reduces control traffic burstiness by using different time frequencies.

4.4.4 Accumulated Service Impact

The experiment quests the impact of a switch outage to the network. If it takes a long time for the controller to detect the switch outage, the outage impact is high. For example, as RYU (1, 1, 1) uses a single discovery timer, the impact of a switch outage will be the same regardless of its affected service traffic. However, as TaDPole uses different discovery timers, the impact is disparate in each traffic zone. We define an impact *(I_o)* as an accumulated service outage time in seconds, which is a product of the ratio of impaired/impacted data traffic *(D_o)* and the outage discovery frequency *(O_f)* (i.e., *(I_o)* = *(D_o)* * *(O_f)*). We got *(D_o)* values of each traffic zone by running the zoning algorithm over ten switch network. Fig. 4.15 presents *(I_o)* of the switch outages in the high, mid, and low traffic zones for TaDPole (0.2, 0.5, 1), TaDPole (0.2, 1, 2), and RYU (1, 1, 1). It shows that a switch failure in the high traffic zone causes far less impact to both TaDPole (0.2, 0.5, 1) and TaDPole (0.2, 1, 2) than RYU (1, 1, 1). It means both TaDPole (0.2, 0.5, 1) and TaDPole (0.2, 1, 2) sends more frequent discovery messages and could react much faster than RYU (1, 1, 1) for the important switch failure. Although TaDPole has more impact on the low traffic zones, it affects less to the total impact as the overall affected traffic is less than RYU (1, 1, 1).

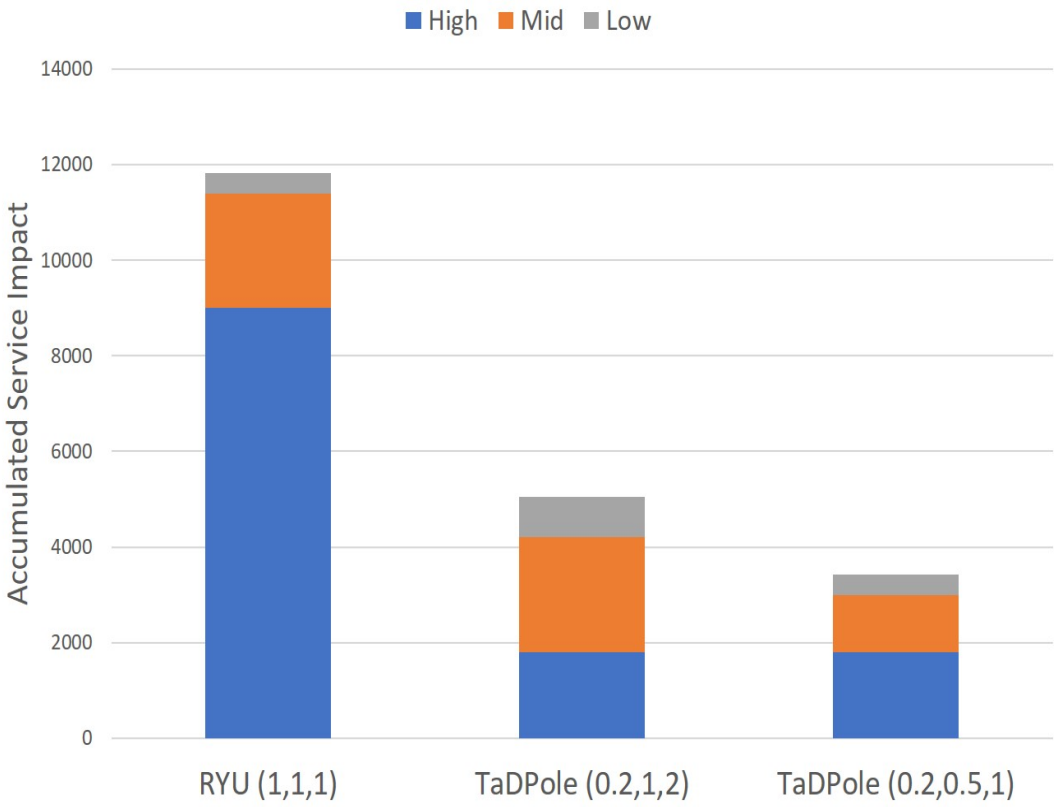


Figure 4.15: Impact of network node churns

4.5 Conclusion and Future Work

Little attention has been paid to the membership discovery protocols in wireless and mobile SDN that suffer from scalability and latency issues. We proposed a novel Traffic-aware Discovery Protocol (TaDPole), which enhances the efficiency of discovery mechanisms for wireless and mobile SDN. We support the port-neutral broadcast-based discovery protocol. We also implemented a facility to dynamically configure the discovery frequency for a specific target instead of using a uniform period for the entire network. We exploit network usage and deliberately controls the frequency of discovery messages depending on the network service impact of each node churn. Based on the impact factor, we enabled the TaDPole platform to provide fast and smart decision making information for fast node churn detection and recovery. A prototype is implemented on the RYU controller. The experimental results exhibit that TaDPole reduces discovery messages by two times and makes the control traffic less bursty. It enhances the network status discovery by three times without increasing the control overhead.

CHAPTER 5

CENTRALITY-AWARE MULTITEMPORAL DISCOVERY PROTOCOL FOR SOFTWARE-DEFINED NETWORKS

5.1 Introduction

As old configuration-based network management approaches are inefficient, unreliable, and unscalable, the virtualization and software-definition of network functions, controls, and applications become enabling technologies by improving the cost efficiency, control accuracy, and deployment flexibility of infrastructures. The Software-Defined Networking (SDN) technologies logically centralizes the network applications and controllers from the underlying distributed data planes. SDN controller uses centralized management protocols to maintain up-to-date network topologies using the remote node's status notifications and periodic discovery messages. However, the centralized discovery protocols have scalability and latency hurdles when applied in SDN wireless and wired networks due to the gossipy, repetitive, and slow protocol. Each controller regularly inquires network state for all ports using *a uniform period timer* for discovery protocol. The number of OFDP messages increases significantly if the network size, including the inter-SDN switch port, host port, and non-SDN switch port, and the discovery frequency increase. As illustrated in Figure 5.1, if the forwarding networks (linear, tree, star, and wireless topologies) overlaps various virtualized networks via control path (in-band or out-of-band), it also causes many redundant control messages. Furthermore, the network can be saturated by periodic discovery traffic while loitering most of the time. It boosts the chance of network collision.

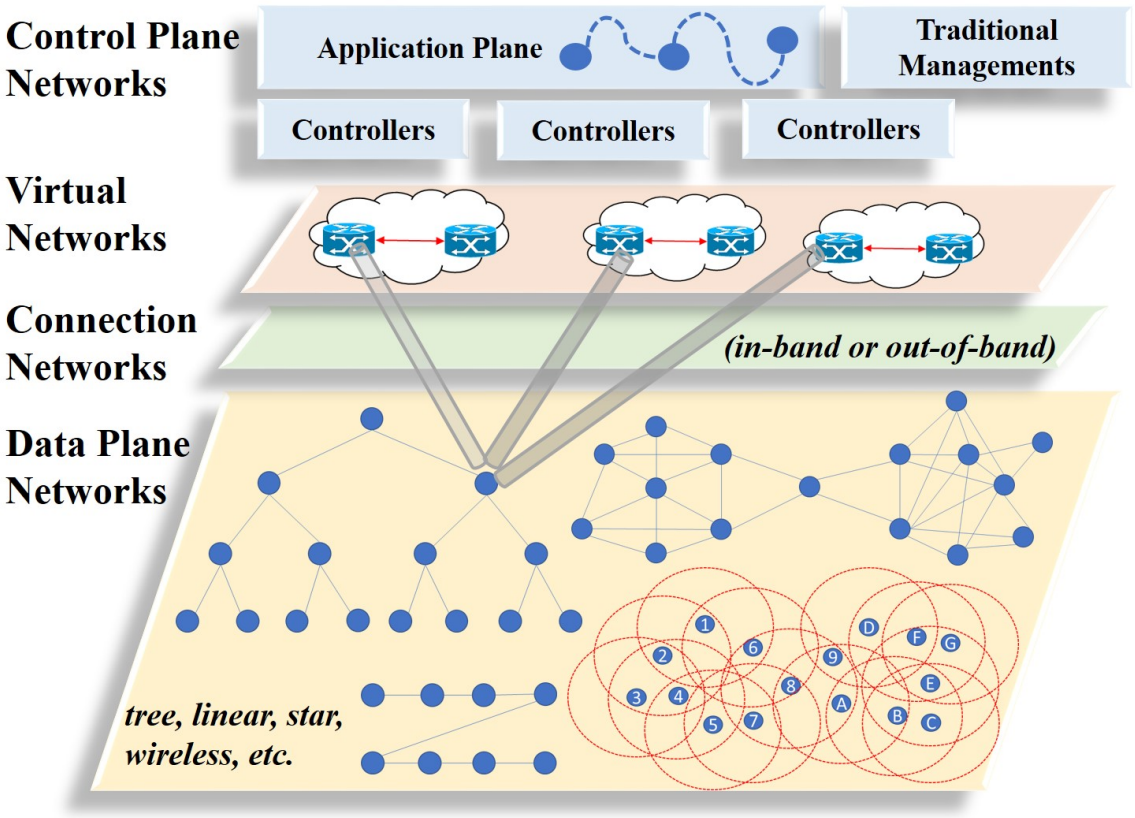


Figure 5.1: Software-defined and virtual networks

This chapter proposes a novel Centrality-Aware Multitemporal (CAMEL) discovery protocol for software-defined networks to enhance the centralized discovery mechanism's scalability and latency issues. We facilitate multiple discovery timers for each target according to the significance instead of using a single timer for the entire network. For example, a typical network architecture in a data center environment uses a three-tiered design that has a core tier in the root of the tree, an aggregation tier in the middle, and an edge tier at the leaves of the tree (i.e., Top of Rack) [48]. The impact of each target node or link failure on the network traffic (significance) is typically decreasing in order of core, aggregate, and edge according to the location, relationship, and functionality. However, the measure of a node significance for different network topologies is unexplored. CAMEL tackles to generalize a node significance measurement for various network topologies by using the centrality models [49, 50]. We combine both degree and betweenness centralities to find an unbiased impact factor of each node. It normalizes the centrality value so that nodes with higher centrality value have more impact on the network. Applying the identified significance to a multitemporal discovery mechanism, CAMEL reduces network impact by improving the discovery rate to the significant nodes and enhances traffic efficiency by decreasing the discovery cycle for the less critical targets. CAMEL contributions include *developing a node significance identification and multitemporal discovery modules*:

- **Node significance identification module** classifies each target's impact on the network service by using a combined degree and betweenness centrality value, which applies to various network topologies.
- **Multitemporal discovery module** facilitates multiple discovery timers for each target according to the significance instead of using a single timer for the entire network.

We design and develop a CAMEL algorithm into the LLDP facility (i.e., switches.py) [43] in the RYU SDN controller. Extensive Mininet [44] experiment results validate that

CAMEL can detect node or link failures on the critical channels much faster without causing any network scalability issue. Also, CAMEL enhances discovery message efficiency by making the control traffic less bursty than OFDP. Also, it decreases the network status discovery delay (making less impact on network services) without raising the control frequency. By taking a common corrective action against a failure, it acts as a useful decision-making tool.

The remainder of this chapter is organized as follows. Section 5.2 explains the proposed algorithm and the prototyping. We discuss the experimental results in Section 5.3, and offers conclusions in Section 5.4.

5.2 CAMEL: Centrality-aware Multitemporal Discovery Protocol Architecture and Implementation

5.2.1 CAMEL Architecture

The Centrality-aware Multitemporal Discovery Protocol (CAMEL) proposes a couple of enhancement modules, including **Node Significance Identification (NSI)** and **Multi-Temporal Discovery (MTD)** to tackle the scalability and latency issues in SDN.

5.2.1.1 Node Significance Identification Module

NSI module classifies each target's impact on the network service by using the centrality models, which apply to various network topologies. We combine both degree and betweenness centralities to find an unbiased impact factor of each node. Algorithm 20, $Check\_Centrality()$, is a lightweight algorithm to get the degree centrality (DC), the betweenness centrality (BC), and the total centrality (TC). Later, MTD mechanism uses the TC value to sign different discovery timers for each node group.

Algorithm 3 CAMEL()

Input: n_list // a list of all nodes in the network

Output: n_list.timezone // time zone for each node

```
function MAIN(n_list, T_sh)
    /*check centrality to identify the significance*/
    Check-Centrality(n-table)
    /*classify nodes into different timer zones using kmeans
clustering*/
    n_list.zone = Zone_Classifier(k-means(n_zone))
    /*register discovery timer for each node according to
timezone*/
    register_timer(n_list)
     return
end function
/*check BC, DC, and TC for each node*/
function CHECK_CENTRALITY(n − table)
    for i ← 0 to n − 1 do
        n-input[i]= read-neighbor-count(**n-table)
    end for
    /*get DC*/
    n_list.DC = Degree_Centrality(**n-input)
    /*get BC*/
    n_list.BC = Between_Centrality(**n-input)
    /*get TC*/
    n_list.TC = Total_Centrality(n_list.DC, n_list.BC)
     return
end function
```

Table 5.1: Notations

Notation	Explanation
E_n	The number of neighbors of a node v
T_n	The total number of nodes in the network
t_{sp}	the total number of shortest paths from a node s to a node d
$t_{sp}(v)$	the number of t_{sp} from a node s to a node d via a node v

$$DC(v) = \sum_{n}(E_n/(T_n - 1)) * 2 \tag{5.1}$$

$$BC(v) = \sum_{s \neq v \neq d} (t_{sp}(v)/t_{sp}) \tag{5.2}$$

$$TC(v) = \sum (DC(v), BC(v)) \tag{5.3}$$

As defined in equation 5.1 and in Table 5.1, the DC calculates the degree of neighbors and normalizes the degree centrality of each node to give the overall network degree centralization. In general, the node with many neighbors has a higher DC value, which indicates the node with the most connections is the most important. As defined in equation 5.2 and in Table 5.1, the BC calculates the betweenness and normalized betweenness centrality of each node and gives the overall network betweenness centralization. In general, the node that is most likely to be chosen in the shortest path between two nodes has a higher BC value. A node has high BC if the shortest paths (geodesics) between many pairs of other nodes in the graph pass through it. Thus, when a high BC node fails, it has a more significant influence on the network traffic flow and service. Although the DC value is a

good significance measure, it only indicates the importance of a node directly connected (a local measure). It does not show the global picture, such as indirectly connected and how important it is to the entire network. Hence, as shown in equation 5.3, we propose to use the TC, which is a combined value of two normalized BC and DC values.

For example, on the linear topology, the nodes' changes (outage or glitch) in the middle of the network may significantly impact the overall network than the nodes at the network's ends. However, as shown in Fig. 5.2, the DC alone cannot indicate the node's significance (centrality) correctly due to the identical number of neighbors. But the BC can present each node's importance on the network. Meanwhile, the BC does not offer a node significance on the grid networks in Fig. 5.3 and 5.4 due to the nature of the balanced node positioning. However, the DC can differentiate the importance of core group nodes to other nodes in the network. Hence, we combine DC and BC into TC to maintain a general significance on the various network topologies.

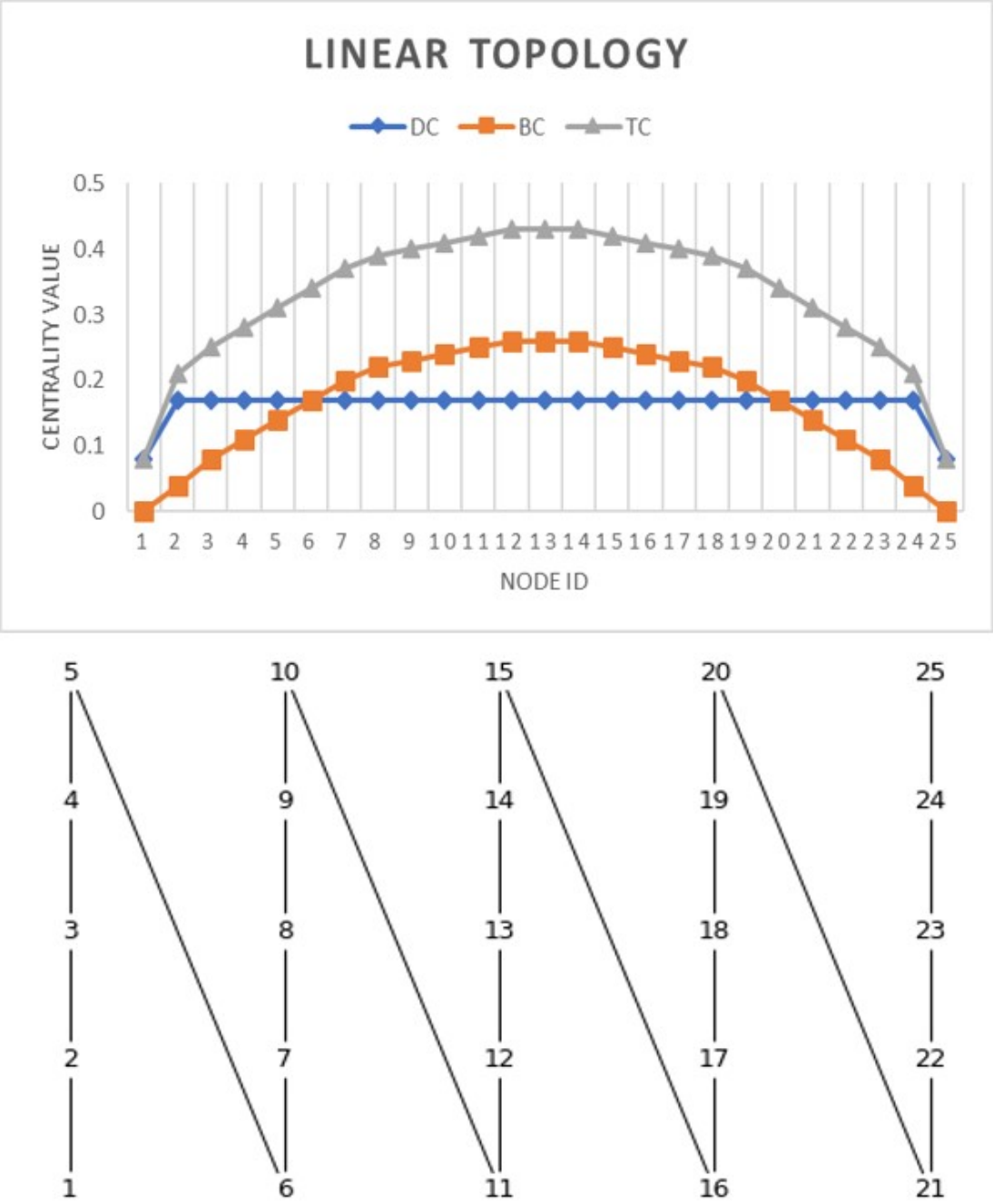


Figure 5.2: Centrality values for linear topology

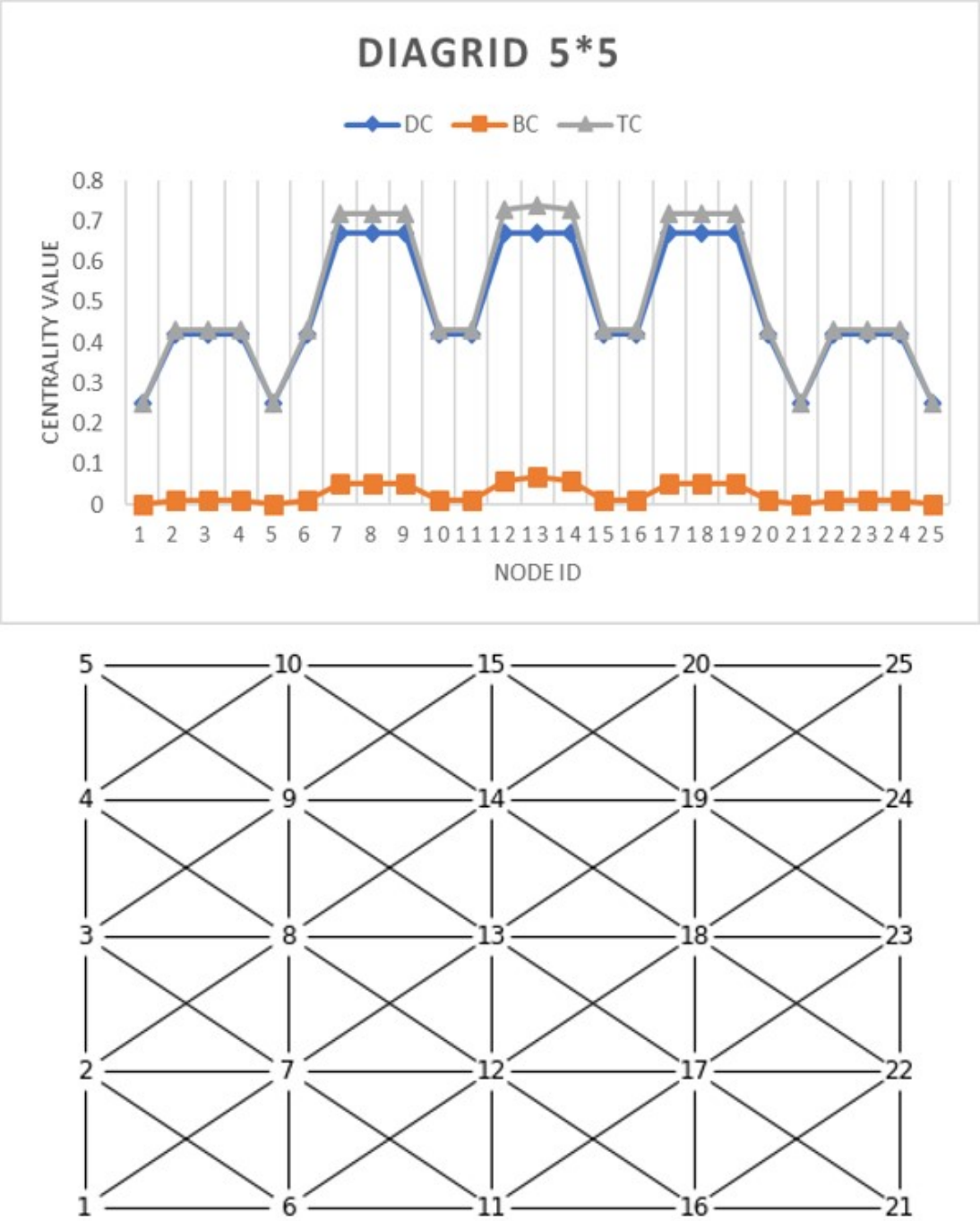


Figure 5.3: Centrality values for diagonal grid topology

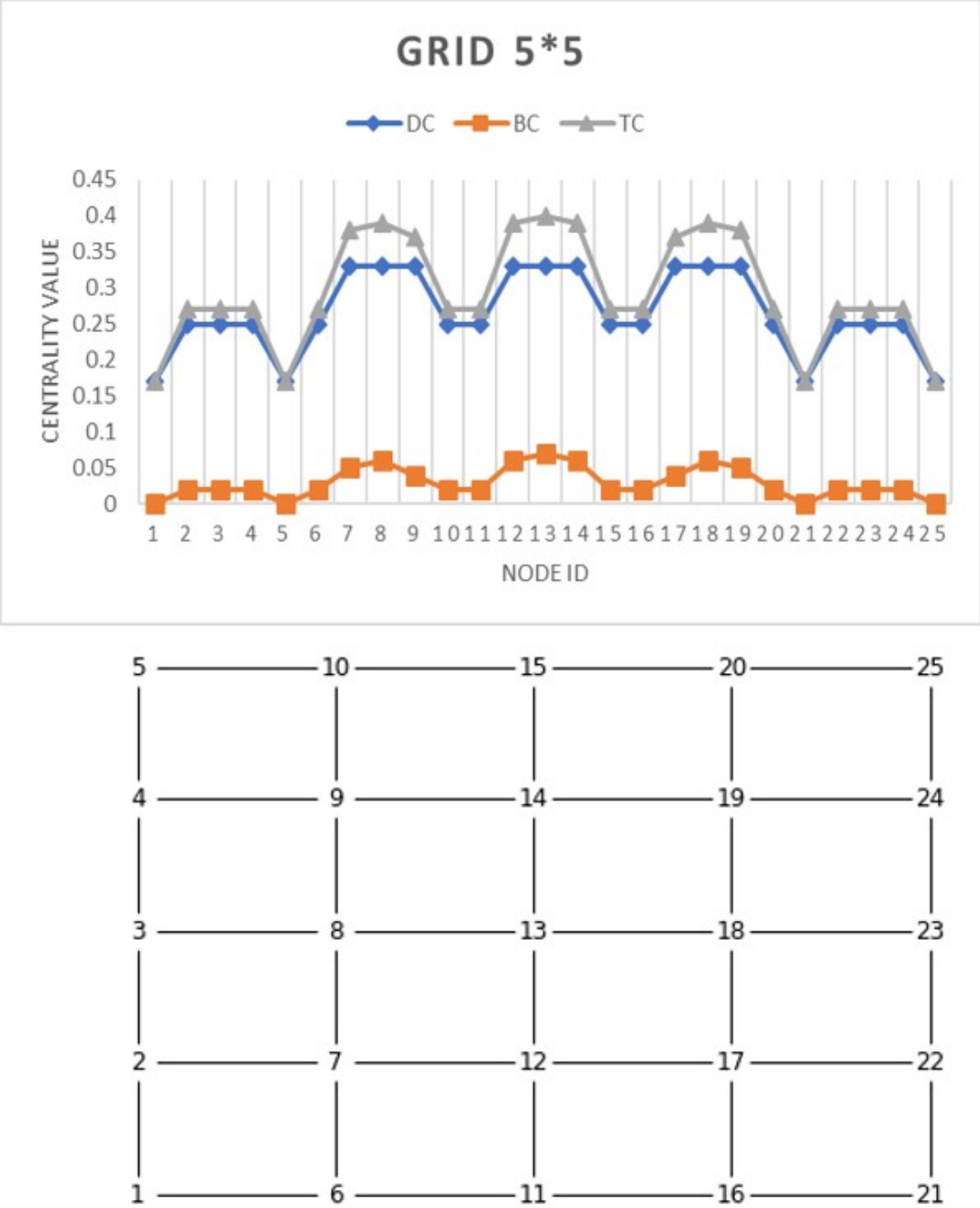


Figure 5.4: Centrality values for grid topology

5.2.1.2 Multi-Temporal Discovery Module

MTD modules facilitate multiple discovery timers for each target according to the significance instead of using a single timer for the entire network. MTD applies the identified NSI values to differentiate the discovery ratio from the significant nodes (the more discovery for the high centrality value nodes) to the less notable targets (the less discovery for the low centrality value nodes). The Zone_Classifier() in Algorithm 20 classifies nodes into different time zones using the K-means clustering algorithm. The classifier values consist of the number of zones n_zone and each zone's range. The register-timer() registers each node to the right timer frequency, and a Send_lldp_packet() thread schedules to send discovery packets according to each timer.

Figs. from 5.5 to 5.9 present the centrality values for different network topologies. For example, Fig. 5.5 shows the centrality values (DC, BC, and TC) for 15 nodes on tree topology. The K-means clustering with k = 3 in Zone_Classifier() classifies nodes into three zones (high, mid, and low), as shown in Fig. 5.6. This classification aligns with the typical data center network, which uses a three-tiered design comprising the core (i.e., nodes 1, 2, and 3), aggregate (i.e., nodes 4 - 7), and edge nodes (i.e., nodes 8 - 15). The impact of each target node or link failure on the network traffic (significance) is typically decreasing in order of core (high), aggregate (mid), and edge (low).

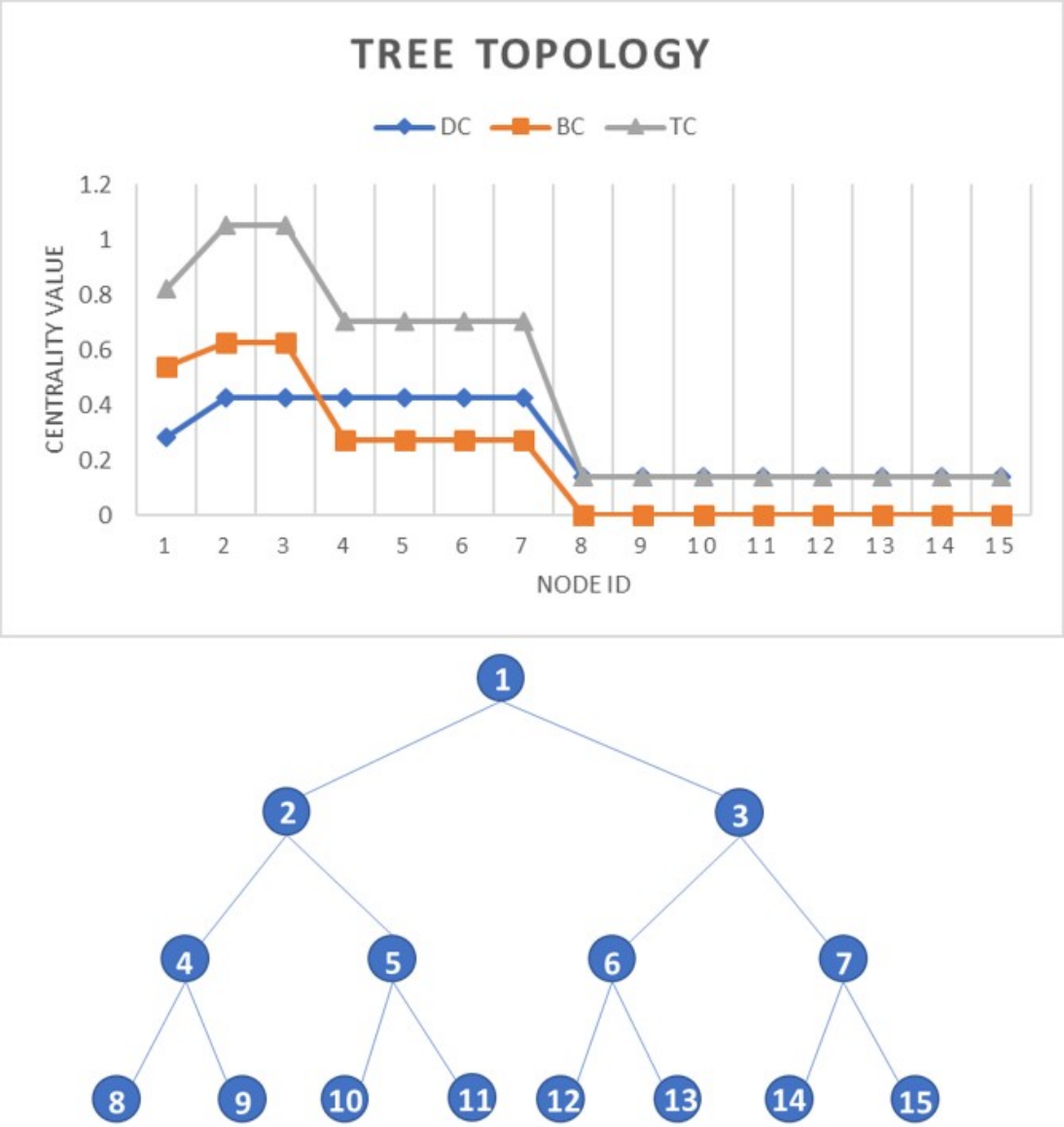


Figure 5.5: Centrality values for tree topology

Rank	DC	BC	TC
1	2,3,4,5,6,7	2,3	**2,3**
2	1	1	**1**
3	8,9	4	**4,5,6,7**
4	10,11,12,13,14,15	5	**8,9,10,11,12,13,14,15**
5		6	
6		8	
7		7,9,10,11,12,13,14,15	

High Mid low

Figure 5.6: Centrality rank after sort on tree topology

Fig. 5.7 shows the centrality values (DC, BC, and TC) for 16 nodes on star topology. The K-means clustering with k = 3 in Zone_Classifier() classifies nodes into three zones (high, mid, and low), as shown in Fig. 5.8. The classification shows that nodes 8 and 10 are in the high significance zone. They need to be checked more frequently than other nodes, as the impact of each target node or link failure on the network traffic (significance) is typically decreasing in order of core (high), aggregate (mid), and edge (low). Fig. 5.9 shows the centrality values (DC, BC, and TC) for 15 nodes on tree topology. The K-means clustering with k = 3 in Zone_Classifier() classifies nodes into three zones (high, mid, and low), as shown in Fig. 5.10. The classification shows that nodes 8 and A (10) are in the high significance zone. For example, as the node 8 is a single point of failure, a failure of node 8 will results network brain cuts for the half of the network nodes. They need to be checked more frequently than other nodes, as the impact of each target node or link failure on the network traffic (significance) is typically decreasing in order of core (high), aggregate (mid), and edge (low).

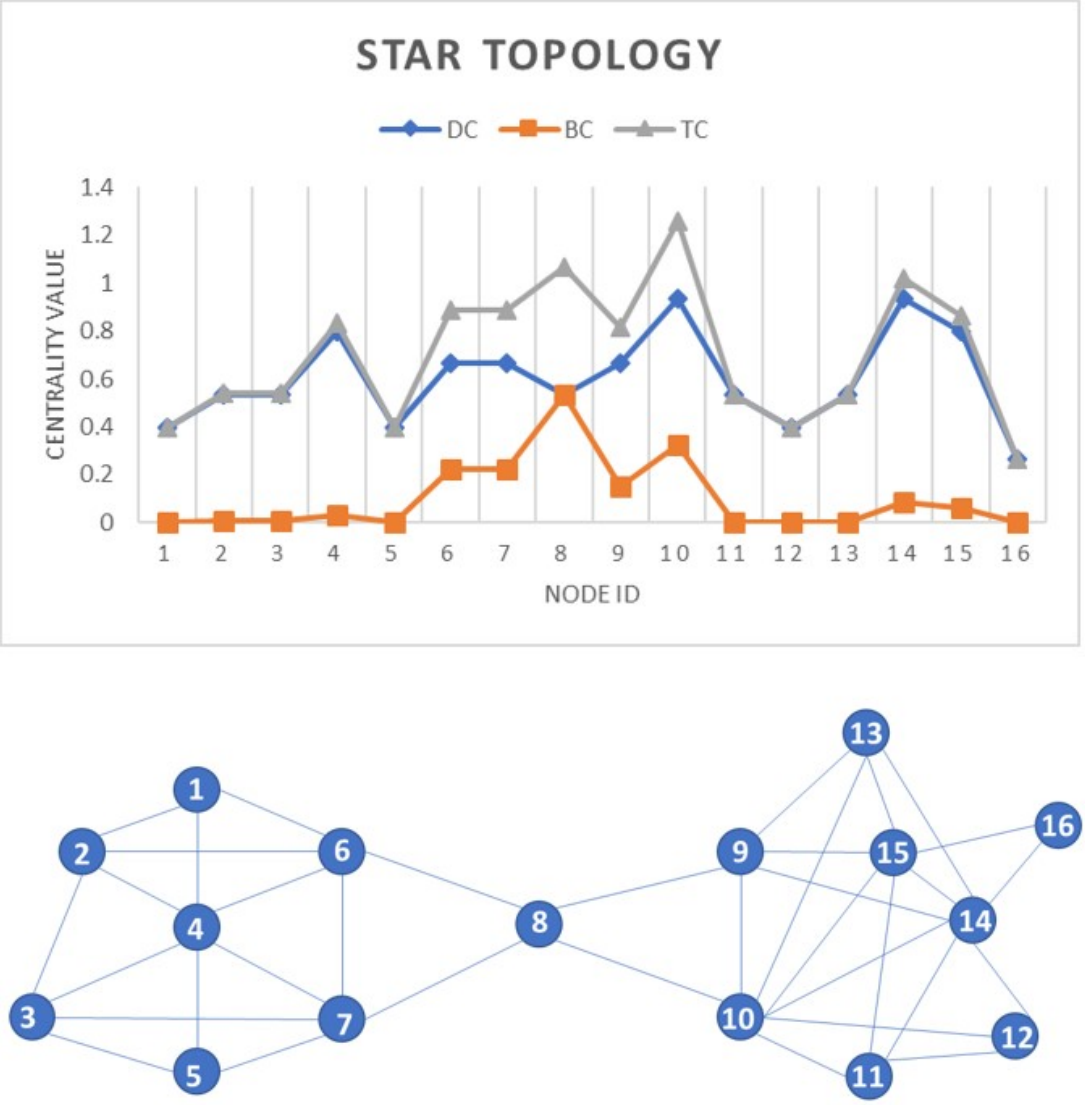
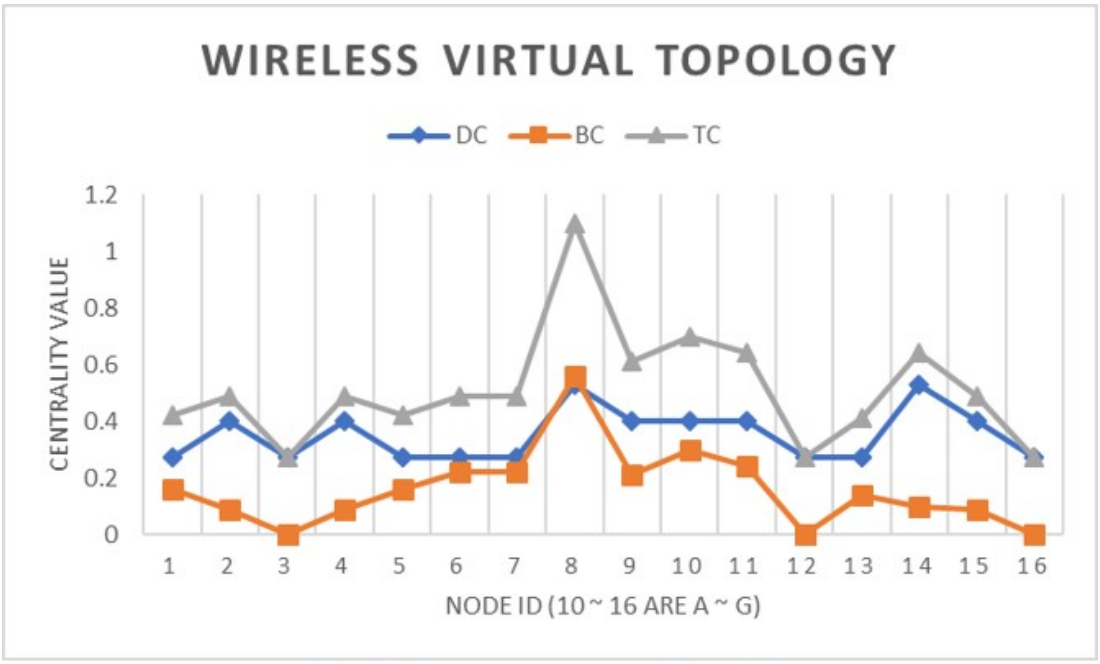


Figure 5.7: Centrality values for diagonal star topology

Rank	D	B	T
1	10,14	8	**10**
2	4,15	10	**8**
3	6,7,9	6,7	**14**
4	2,3,8,11,13	9	**6,7**
5	1,5,12	14	**15**
6	16	15	**4**
7		4	**9**
8		2,3	**2,3**
9		11	**11**
10		1,5,12,13,16	**13**
11			**1,5,12**
12			**16**

High Mid low

Figure 5.8: Centrality rank after sort on diagonal star topology



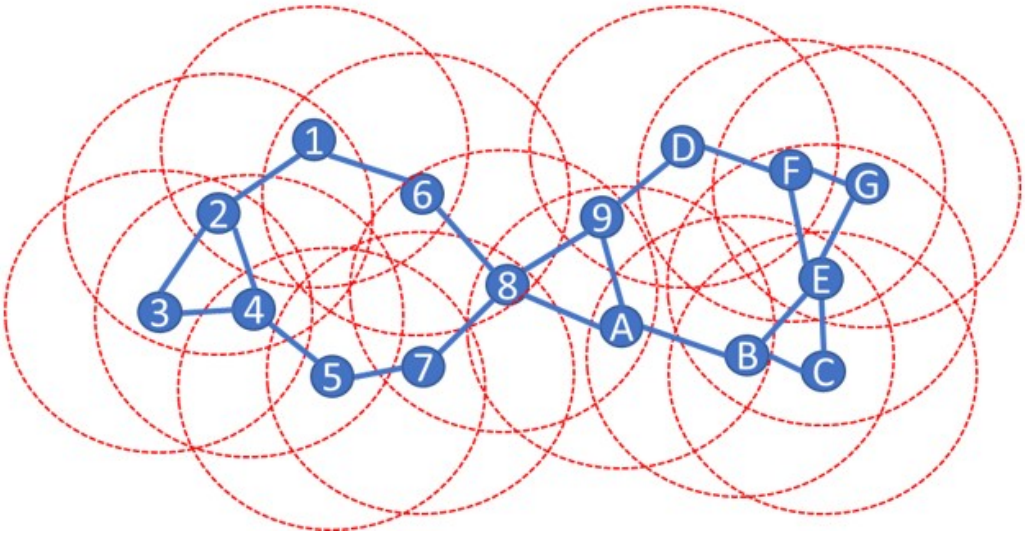


Figure 5.9: Centrality values for wireless virtual topology

Rank	D	B	T
1	8,E	8	**8**
2	2,4,9,A,B,F	A	**A**
3	1,3,5,6,7,C,D,G	B	**B,E**
4		6,7	**9**
5		9	**2,4,6,7**
6		1,5	**F**
7		D	**1,5**
8		E	**D**
9		2,4	**3,C,G**
10		F	
11		3,C,G	

High Mid low

Figure 5.10: Centrality rank after sort on wireless virtual topology

5.2.2 CAMEL Implementation

As illustrated in Fig. 5.11, we implemented the **NSI**, and **MTD** modules in the RYU controller's Switches(app_manager.RyuApp) class, which maintains the forwarding network status and supervises out-going event packets, including port and switch states. It also calls an LLDP_loop() function to send periodic discovery packets (packaged into Openflow PACKET_OUT messages), an Openflow module application for regular LLDP transmission. It maintains all the connected switch nodes and links (topology) for the Switches(app_manager.RyuApp) class by listening to the status-change events, such as adding and removing the network nodes and links.

We implement CAMEL modules (NSI and MTD) by intercepting the our-going packets from the LLDP_loop(). The $Check\_Centrality()$ function in Algorithm 20 (NSI) finds the degree (DC), betweenness (BC), and total (TC) centralities of the entire nodes to store into the n_list class. The $Zone\_Classifier()$ function in Algorithm 20 (MTD) classifies nodes into different timer zones by using a k-means clustering algorithm based on the NSI values (node's centralities). The zone assignment is reevaluated and dynamically reassigned according to the network changes. For the registered LLDP_frequency, multiple timer threads in register_timer() punt LLDP_packets to the OpenFlow Packet Processor module containing dpid (chassis ID), port_no (Port ID), dl_addr (source address), and TTL (Time-To-Live) parameters. It eventually calls the Send_lldp_packet() function to generate a packet-out message.

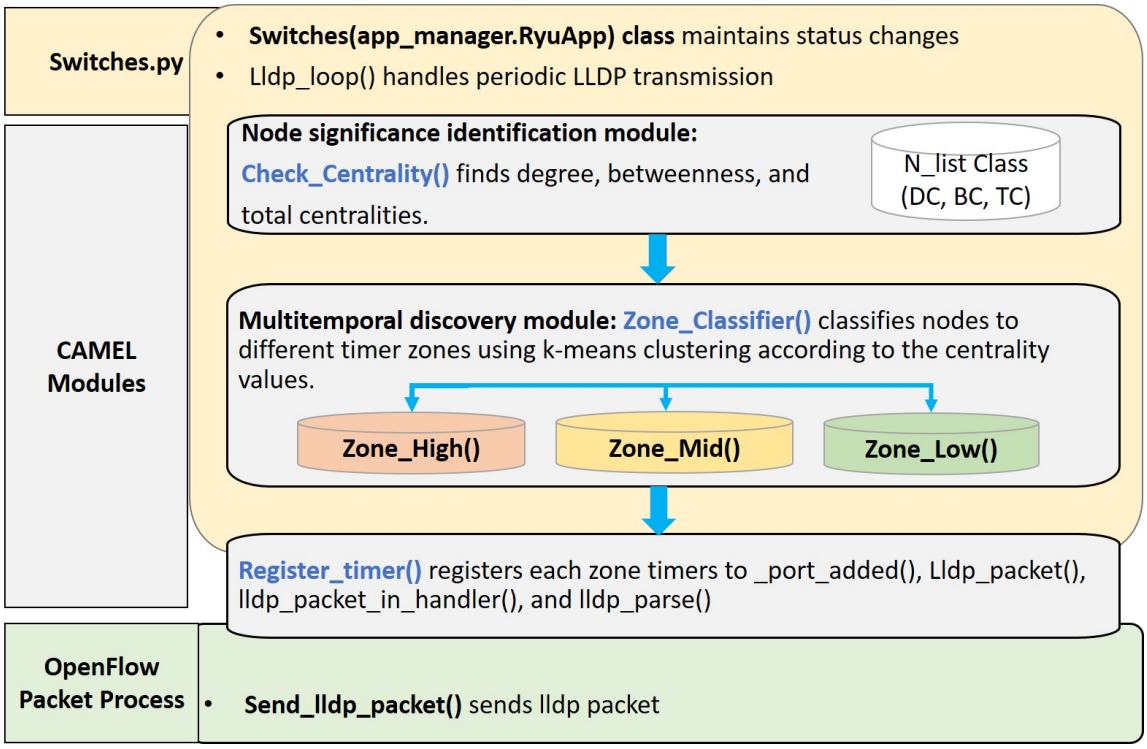


Figure 5.11: CAMEL implementation in RYU controller

5.3 Evaluations

We have investigated the performance of CAMEL using Mininet emulator [46] with real implementation on an RYU controller [43]. We have conducted experimental studies on control message overhead, control message burstiness, and node outage impact over the network.

5.3.1 Experimental Setup

For the emulation environment, we used iMac with 3.0GHz 6-core Intel Core i5 and Ubuntu 19.04 OS. We use an iperf tool to generate data traffic and Wireshark [51] to capture messages from the loop-back interface. We have conducted a couple of experiment sets: 1)

discovery protocol overheads and burstiness comparison between RYU and CAMEL, and 2) the accumulated impact of network service outage (delay). We use various topologies as shown in Fig. 5.12. We use different timer zones and timer values with both inband and out-of-band control connection scenarios. Moreover, we show some experimental results, including a 15 node tree topology and a ten node linear topology, respectively.

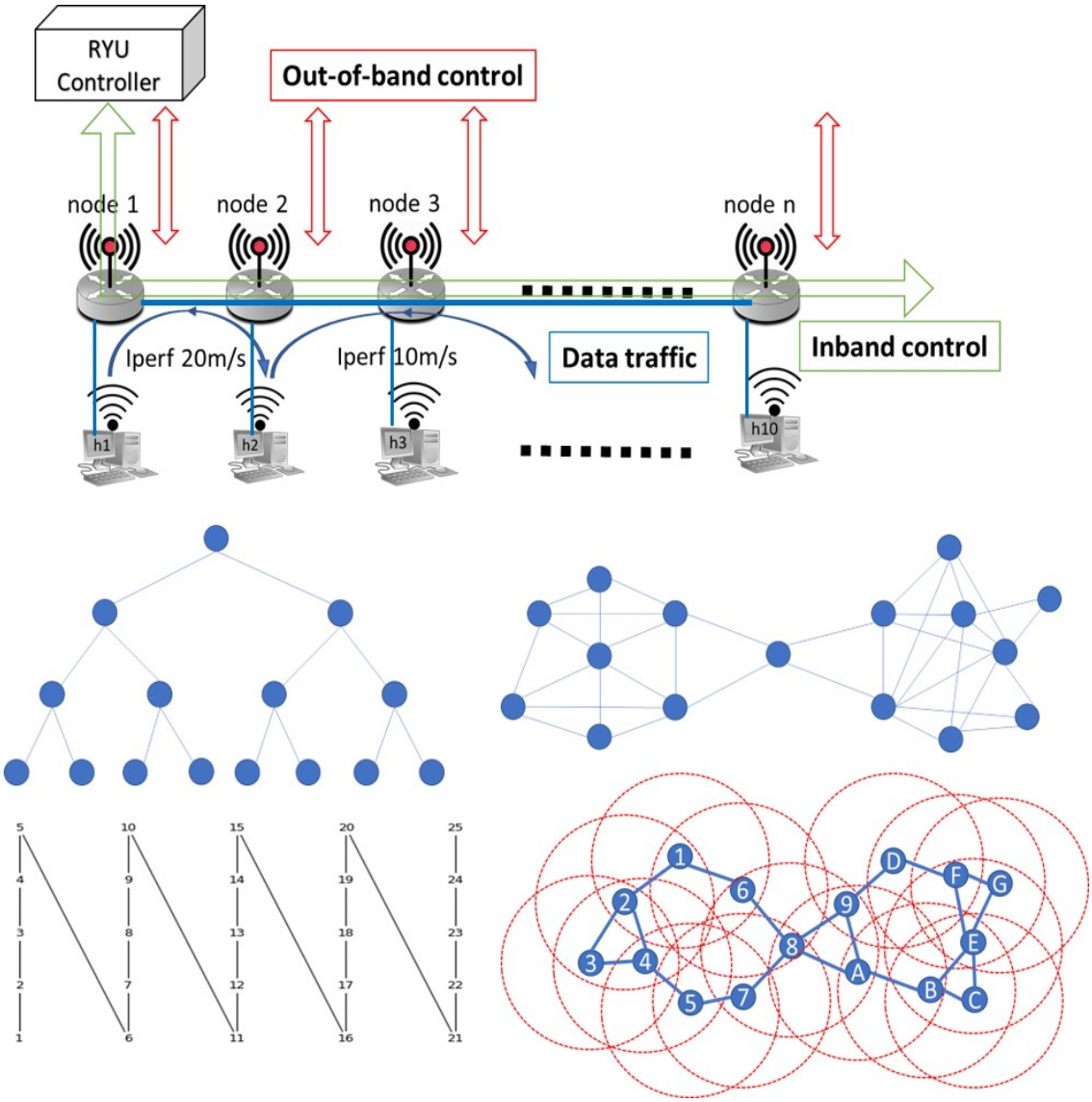


Figure 5.12: CAMEL experiment setup for wired (tree, star, linear) and wireless

According to the centrality value, we classify nodes into three different timer zones (High, Mid, Low) using k-means clustering (k=3). For example, we present RYU (1, 1, 1), as the RYU controller uses a default uniform interval (one second) in sending LLDP PACKET_OUT messages. CAMEL (0.25, 1, 2) is with timer intervals of 0.25, 1, and 2 seconds for high, middle, and low significant nodes, respectively. CAMEL (0.25, 1, 2) generates four times more frequent LLDP messages than RYU (1, 1, 1) to the nodes in the high zone but sends two times less LLDP messages than RYU (1, 1, 1) to the nodes in the low zone. It also indicates that CAMEL (0.25, 1, 2) can detect the status change four times faster than RYU (1, 1, 1) for the nodes in the high zone, but two times slower than RYU (1, 1, 1) for the nodes in the low zone. We use the default three consecutive LLDP message failures to change status.

5.3.2 Control Message Overheads

Fig. 5.13 presents an accumulative control message overhead for a 15 node out-of-band control tree topology network over 60 seconds. As shown in Fig. 5.6, out of 15 nodes, three nodes are in the high zone (nodes 1, 2, and 3), four nodes are in the middle zone (nodes 4 - 7), and eight nodes are in the low zone (nodes 8 - 15). CAMEL (0.25, 1, 2) generates four times more frequent LLDP messages than RYU (1, 1, 1) to the three nodes in the high zone and sends two times less LLDP messages than RYU (1, 1, 1) to the four nodes in the low zone. Hence, CAMEL (0.25, 1, 2) creates 33% more traffic than RYU (1, 1, 1) in total. Meanwhile, CAMEL (0.5, 1, 4) generates two times more frequent LLDP messages than RYU (1, 1, 1) to the three nodes in the high zone and sends four times less LLDP messages than RYU (1, 1, 1) to the four nodes in the low zone. Hence, CAMEL (0.5, 1, 4) creates 20% less traffic than RYU (1, 1, 1) in total.

5.3.3 Control Message Burstiness

Fig. 5.14 presents the control message patterns for ten nodes of the linear network topology with an in-band control connection for 3 seconds. RYU (1, 1, 1) presents bursty control traffic patterns, as the LLDP messages are concentrated in a single discovery period while idling most of the time. Traffic concentration may cause various collision scenarios. According to the standard deviation values, CAMEL (0.2, 0.5, 1) shows much less bursty than RYU (1, 1, 1). Using three different discovery frequencies, CAMEL (0.2, 0.5, 1) scatters the control traffic over different periods.

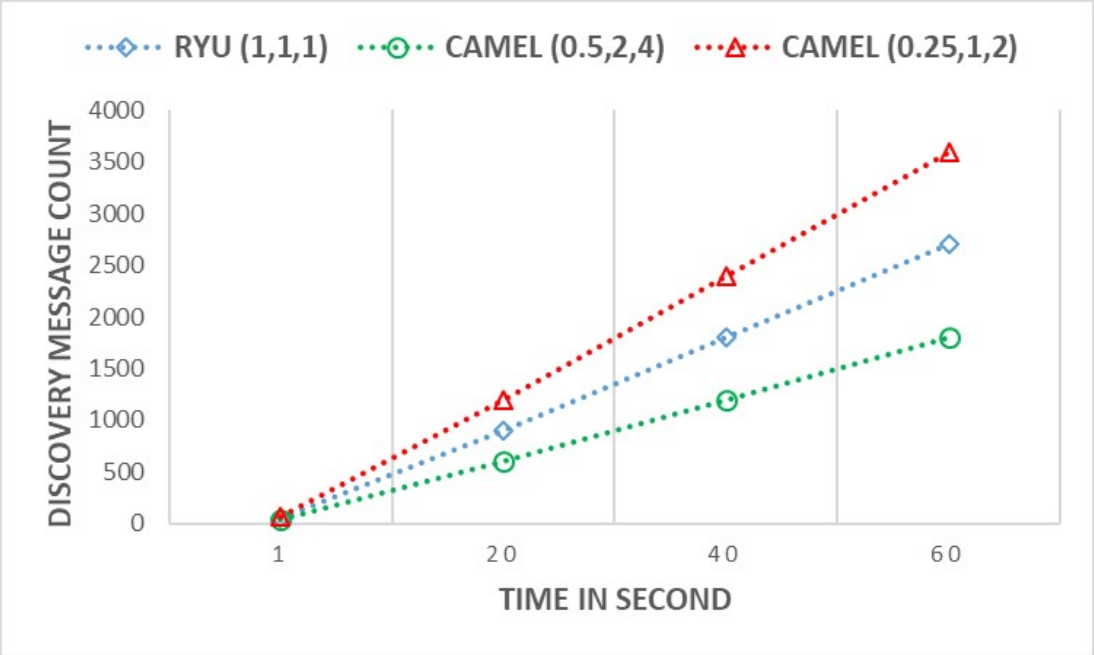


Figure 5.13: Control overhead

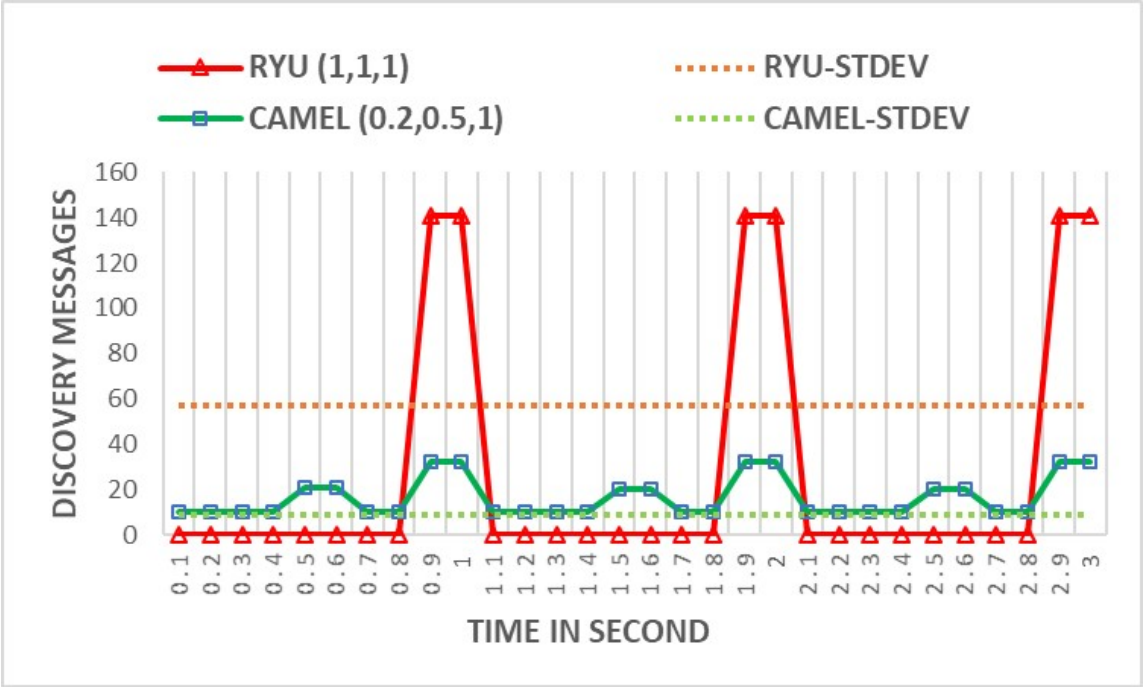


Figure 5.14: Burstiness

5.3.4 Accumulated Service Impact

The experiment quests the impact of a node outage on the network service and traffic. If the controller takes a long time to detect the outage, the outage impact is high. For example, as RYU (1, 1, 1) uses a single discovery timer, the impact of a node outage will be the same bounded by a single discovery timer regardless of its affected service and traffic. However, as CAMEL uses different discovery timers, the impact is diverse in each traffic zone.

We define an impact (I_o) as an accumulated service outage value, a product of the median TC value (%) in each zone (C_o), and the number of affected nodes (N_o). For example, using the TC values of 15 nodes on tree topology as shown in Fig. 5.5, C_o values are 105 (high), 70 (mid), and 14 (low) and I_o values become 315 (high), 280 (mid), and

112 (low). Fig. 5.15 presents the impact (I_o) of the node outages in the high, mid, and low zones of RYU (1, 1, 1), CAMEL (0.25, 1, 2), and CAMEL (0.5, 1, 4). It shows that node failures in the high zone cause far less impact to both CAMEL (0.25, 1, 2) and CAMEL (0.5, 1, 4) than RYU (1, 1, 1), as both CAMEL (0.25, 1, 2) and CAMEL (0.5, 1, 4) send more frequent discovery messages and could react much faster than RYU (1, 1, 1) for a node outage. CAMEL (0.25, 1, 2) has more impact on the low zones, but it affects less in the total impact as the overall affected service is 18% less than RYU (1, 1, 1). However, as CAMEL (0.25, 1, 4) has four times more impact on the low zones, its overall impact is 25% greater than RYU (1, 1, 1).

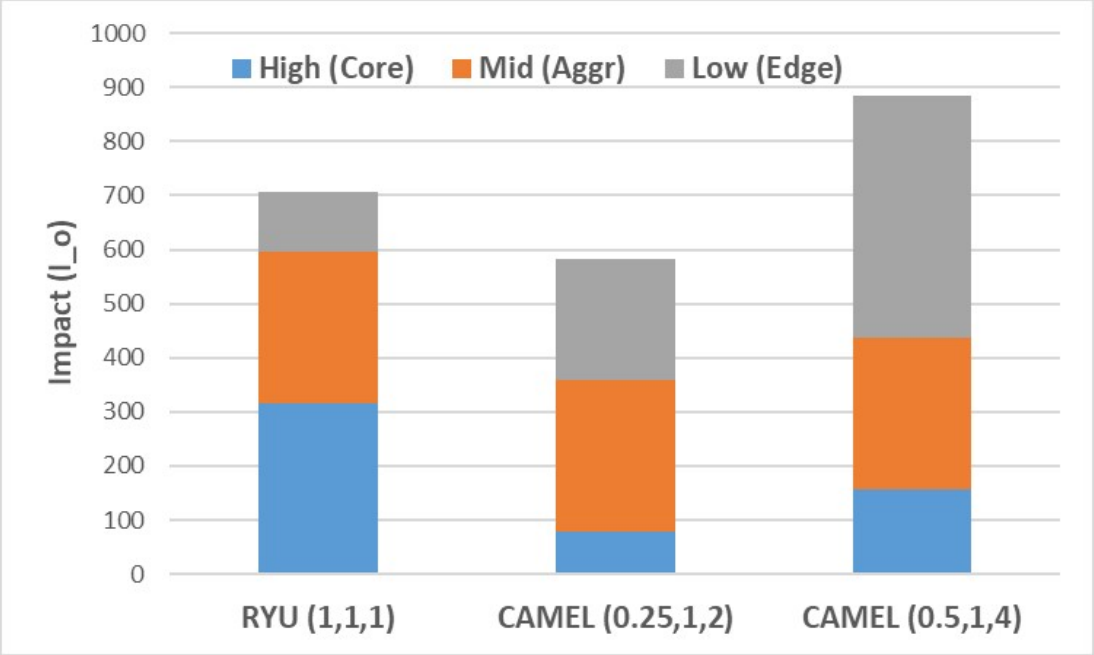


Figure 5.15: Network impact on tree topology (delay)

5.4 Conclusions

Discovery protocols in SDN have scalability and latency problems. We have proposed a novel Centrality-Aware Multitemporal (CAMEL) discovery protocol for software-defined networks to enhance the centralized discovery mechanism's scalability and latency issues. We facilitate multiple discovery timers for each target according to the significance instead of using a single timer for the entire network. CAMEL generalizes a node significance measurement for various network topologies by using the centrality models. We combine both degree and betweenness centralities to find an unbiased impact factor of each node. It normalizes the centrality value so that nodes with higher centrality value have more impact on the network. Applying the identified significance to a multitemporal discovery mechanism, CAMEL reduces network service impact by decreasing the discovery delay for the significant nodes without compromising the control message efficiency. We have implemented CAMEL on the RYU controller. The experiment results validate that CAMEL improves discovery message efficiency and makes the control traffic less bursty. It enhances the network service quality by reducing discovery delay to the significant nodes without increasing the overall control overhead.

CHAPTER 6

SDN-BASED WORMHOLE ANALYSIS USING THE NEIGHBOR SIMILARITY FOR A MOBILE AD HOC NETWORK (MANET)

6.1 Introduction

As a MANET uses an open-architecture based broadcast medium to support mission-critical applications in challenging environments, it is susceptible to various security attacks.

Challenge 1: Network security management is frequently limited in its ability to understand and diagnose problems, manage end device and user expectations, and optimize resource allocations. It is exacerbated in a mobile ad hoc network (MANET). As the number and types of devices increases, the network membership control becomes highly dynamic, radio access resources are managed across multiple aggregated carriers, and network problem detection, isolation, and root cause analysis become increasingly expensive.

Challenge 2: Software-defined Networking (SDN) increases attack surface due to increased complexity and a separate multi-layer control plane. SDN needs to verify that the flow commands come from authorized sources. Still, the attack surfaces include forwarding switches, controllers (logically a single logical point of failure), the connection between the switch and the controller, and various applications. Also, a centralized controller take-over will have disastrous consequences due to the concentration of control logic in one logical block. Attack on the controller can be amplified by exploiting the Link Layer Discovery Protocol (LLDP) vulnerability. SDN creates additional security challenges, in-

cluding unauthorized access to data and control planes of networks (i.e., wormhole attacks) and data leakage by timing analysis. Moreover, attackers can modify the rules at network devices, exploit compromised applications to alter system configuration, and perform a denial of service (DoS) (i.e., flooding control and data packets).

A wormhole attack [52, 53] is a particularly challenging security problem because it can silently deploy the attack without compromising other security means as long as the communication channel is known. Wormhole attackers allure data packets into one point and replay them at distant locations by tunneling through the attacker's implicit direct communication links. The attackers can also perform various malicious data and control traffic manipulations by selectively dropping, flooding, recording, or modifying packets without revealing their identity. Most of the existing countermeasures against wormhole attacks are for static SDNs by using the network topology information. Furthermore, to achieve a certain accuracy level, they use sophisticated devices, including directional antennas [54], GPS [55], ultrasound [56], and accurate timing devices [57]. However, those devices are too expensive to harness the resource-limited MANET nodes. It is crucial yet challenging to design wormhole attack countermeasures in a software-defined MANET because they depend upon the network topology, distance, direction, and location among the pivotal neighbors [54, 23, 58].

This project designs and develops a novel wormhole countermeasure in a Software-defined MANET, namely an SDN-based Wormhole Analysis using the Neighbor Similarity (SWANS) approach. SWANS apprehends wormhole attacks using a light-weight neighbor counting algorithm at a centralized SDN controller. We developed an online outlier detection algorithm to detect any neighbor counts abnormality (the lack of similarity). We also utilize node mobility itself to identify the anomaly caused by wormholes. For example, when a node moves inside a wormhole attack area, the node would experience the rapid change of its neighbors' characteristics due to the virtual tunnel created by wormhole at-

tackers. As the neighbor discovery protocol (i.e., Link Layer Discovery Protocol (LLDP) in SDN) is one of the essential functionalities in a software-defined MANET, SWANS does not require any additional communication overhead. SWANS also countermeasures various false-positive and false-negative attack scenarios generated by the intelligent wormhole attacker in assessing Link Layer Discovery Protocol (LLDP) vulnerability (attacker models defined in 6.2). We performed extensive studies via both analysis and simulations. We completed comprehensive studies on false-positive and false-negative rates via analysis and simulation. Our simulation results show that SWANS can detect various intelligent wormhole attacks efficiently with low false-positive and false-negative rates. SWANS is the first wormhole countermeasure in a software-defined MANET that does not require global topology information or special hardware to the best of our knowledge.

Our contributions include:

1. Proposing a Wormhole Analysis using the Neighbor Similarity (SWANS) method for SDNs, which is a novel wormhole countermeasure for Software-defined MANETs.

2. Designing a real-time neighbor similarity analysis algorithm on a centralized SDN controller, which apprehends wormhole attackers not only without requiring any detailed topology information but also without causing significant communication and coordination overhead.

3. Countermeasuring various intelligent wormhole attacker models (generate false-positive and false-negative scenarios), which assesses the Link Layer Discovery Protocol (LLDP) vulnerability.

The remainder of this chapter is organized as follows. We present the wormhole attacker types, models, and countermeasure designs in Section 6.2. We discuss the proposed SWANS algorithm and the implementation details in Section 6.3. Section 6.4 provides the

experimental setup, assumptions, and detecting results for each attacker model, and Section 6.5 concludes the chapter.

6.2 Wormhole Attacker Models and Analysis Methods

6.2.1 Wormhole Attacker Types and Models

As described in Table 6.1, a wormhole attacker x, which consists of more than one end nodes (W_1 & W_2). W_x can be three different types, including Full_Stealthy, Partial_Stealthy, and No_Stealthy types.

- Full_Stealthy wormhole attacker type can be secretly allocated in an SDN without registering to the SDN controller. Hence, it does not receive any specific request from the SDN controller. However, it eavesdrops all LLDP messages in a promiscuous mode.
- Partial_Stealthy wormhole attacker type registers only one wormhole endpoint node to the SDN controller. Hence, it receives LLDP requests from the SDN controller. However, the controller does not know if it is a wormhole node or not.
- No_Stealthy wormhole attacker type registers only both wormhole endpoint nodes to the SDN controller. Hence, both endpoints receive LLDP requests from the SDN controller. However, the controller does not know the relationship between the wormhole attacker nodes. Both Partial_Stealthy and No_Stealthy wormhole attackers can be deployed by compromising the existing SDN node instead of installing new physical wormhole attacker nodes.

A wormhole attacker tries to increase compromised neighbors to attract more network traffic to the attacker nodes. Hence, the number of neighbors increases when a node gets compromised by the attacker. However, unlike the distributed algorithm, a centralized

Table 6.1: Notations

Notation	Explanation
S_x	Wireless & mobile node x
H_x	Wireless & mobile host x
W_x	Wormhole attacker x, which consists of more than one end nodes (W_1 & W_2). W_x can be 3 different modes (Full_Stealthy, Partial_Stealthy, and No_Stealthy)
X_{com}	Node X's communication range area with radius r
F	Entire network field area
N	Total number of the nodes in the field
$nNum_x$	The number of neighbors of node x
NSI	The Neighbor Similarity Index (NSI) is the distance of two K means clusters
ACI	The Augmented Concentration Index (ACI) is the value of the similar cluster nodes in the surrounding neighbors
T_sh	Threshold class for both NSI and ACI

SDN controller can efficiently detect the attack by checking the neighbor count abnormality from the neighbor table. Hence, a wormhole attacker tries to manipulate their neighbor counts intelligently using the LLDP vulnerabilities in SDN. We identify and tackle the following attacker models used in this chapter:

- The attacker can observe and manipulate (eavesdrop, spoof, alter, etc.) LLDP control messages within its range. The attacker can use data traffic observation such as the packet transmission time and frequency to perform traffic analysis and infer target objects' locations.

- The attacker can adjust neighbor counts by manipulating its reply to LLDP requests. The attacker tries to hide its location by reducing the neighbor counts. It can shrink its

communication range smaller than the existing node's range (RRWA in Figure 6.1 (b)). It also can send LLDP request to only one of the two endpoints instead of both (ROWA in Figure 6.1 (c)).

- The attacker can randomly spoof LLDP Packet-In messages on behalf of other nodes (RSWA in Figure 6.1 (d)). The attacker tries to hide its location by increasing other node neighbors' reporting of fake neighbors (randomly choosing both source and destination node IDs and sending LLDP response).
- The attacker cannot spoof neighbor nodes with the location information for the out of range nodes (random locations only).

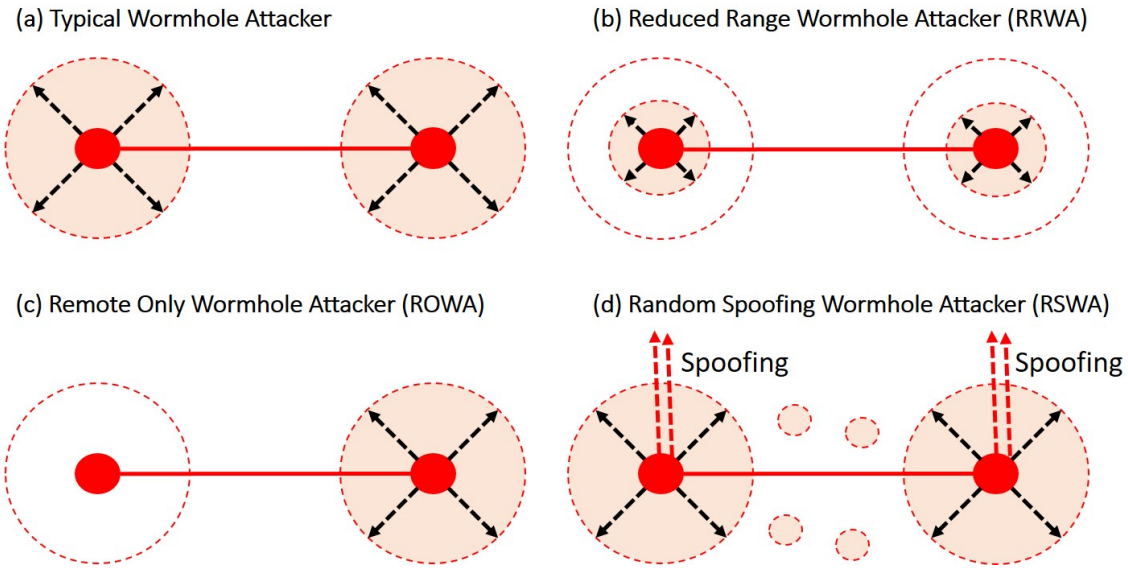


Figure 6.1: Wormhole attacker models

6.2.2 Wormhole Attacker Analysis Methods

When there is a wormhole attacker, the number of neighbors of the nodes within the wormhole range increases beyond a range of statistical fluctuation. A few nodes with a high

number of neighbors are relatively concentrated around the attack areas. Hence, SWANS uses a K-means clustering algorithm to determine the Neighbor Similarity Index (NSI). We define the NSI by the neighbor count difference (distance) between the two clusters' centroids created by k means clustering.

The equation 6.1 is a typical k-means clustering algorithm. For the number of neighbors on each node x, $x_i = (x_{i1}, ...x_{ip})$, if centroids are $m_1, m_2, ...m_k$, and partitions are $c_1, c_2, ...c_k$, then one can show that K-means converges to a *local* minimum of (6.1), which is within-cluster sum of squares.

$$\sum_{k=1}^{K} \sum_{i \in c_k} ||x_i - m_k||^2 \qquad \text{Euclidean distance} \tag{6.1}$$

The process steps are as follows:

0. Start with initial guesses for cluster centers (centroids)
1. For each data point, find the closest cluster center (partitioning step)
2. Replace each centroid by average of data points in its partition
3. Iterate 1+2 until convergence

NSI, which classifies the area of high neighbor count nodes from normal neighbor count nodes, alone may not identify the wormhole attackers due to potential noises. Some nodes may temporarily have a high number of neighbors by chance. Also, a spoofing attack can intentionally increase the number of neighbors of a target node. Hence, in addition to the NSI, SWANS proposes an Augmented Concentration Index (ACI) algorithm. It finds the core of the wormhole attacker, which is surrounded by the nodes with relatively high neighbors. As illustrated in Figure 6.2, for a high neighbor cluster (cluster 1), ACI

augments the highly concentrated area by incentivizing the nodes surrounded by the same cluster nodes and penalizing the nodes surrounded by the different cluster nodes. Using k means clustering with the ACI, it can denoise the less concentrated areas.

As nodes move around in the network, the number of neighbors on a node changes over time. The analysis approach is to discover if any neighbor-counts exhibit abnormal increments than the nodes, out of the wormhole range. There are several approaches to outlier detection. SWANS conducts a general framework for estimating the underlying distribution of the neighborhood counts. The problem of finding outliers can be solved efficiently if the data distribution is calculated accurately. There are several model estimation techniques proposed in the literature including wavelets [59], histograms [60], and kernel density estimators [61]. For mobile nodes, we use a scheme to quantify the difference between the previous data set P of neighborhood counts (i.e., training set) and the new or recent data set Q (i.e., test set). As defined in the equation 6.2, the *Kullback-Liebler (KL) divergence*, $D_{KL}(P||Q)$ is widely used, where $H(P,Q)$ is called the cross-entropy of distributions P and Q, and $H(P)$ is the entropy of P.

$$\begin{aligned} D_{KL}(P||Q) &= -\sum_{x} p(x)\log q(x) + \sum_{x} p(x)\log p(x) \\ &= H(P,Q) - H(P) \end{aligned} \tag{6.2}$$

A practical issue is that $q(x)$ may be zero when $p(x) > 0$, especially when the test set is from a small size. To avoid the issue, we use a variation of the KL divergence, called the *Jensen-Shannon (JS) divergence*, as defined in the equation 6.3, where M is the average of the two distributions, i.e., $M = 1/2(P + Q)$.

$$D_{JS}(P||Q) = (D_{KL}(P||M) + D_{KL}(Q||M))/2 \tag{6.3}$$

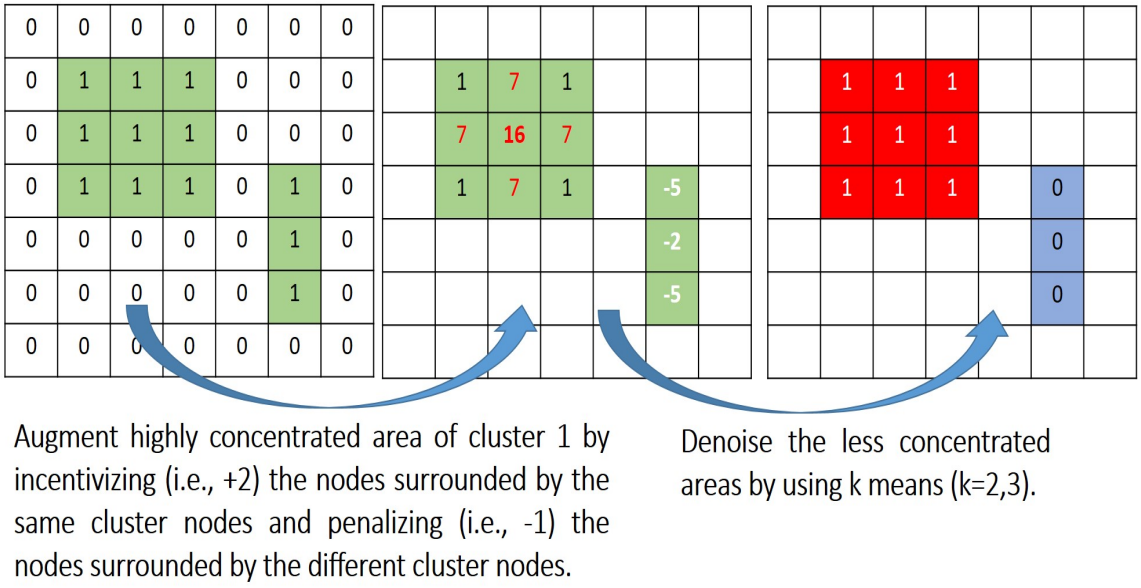


Figure 6.2: ACI Analysis Algorithm

The outlier decision is to check if the JS-divergence is greater than a threshold as defined in the equation 6.4.

$$D_{JS}(P||Q) > T\_sh \tag{6.4}$$

6.3 SWANS: Algorithm and Implementation

The SWANS Algorithm 23 is a centralized approach to detect wormhole attackers in the network. First, it calls a Check_Wormhole () using the neighbor table (n-table) and T_sh parameters. T_sh is a class with four different threshold values, including neighbor-distance ($T\_sh.ND$), neighbor-balance ($T\_sh.NB$), annotated-density-distance ($T\_sh.AD$), and annotated-density-balance ($T\_sh.ND$). $T\_sh.ND$ is a threshold of the NSI. As illustrated in Figure 6.3, a node B would have $(X_{com}/F) * N$ neighbors on average. We assume the node's transmission ranges of both W_x are the same as X_{com}, and the wormhole tunnel replays all the beacon messages (Figure 6.1 (a)). According to (6.5), the number of neighbors will increase approximately from two times (when a node's X_{com}

almost overlaps with one of the wormhole nodes' X_{com}) to three times on average (when a node gets into the range of a wormhole attacker).

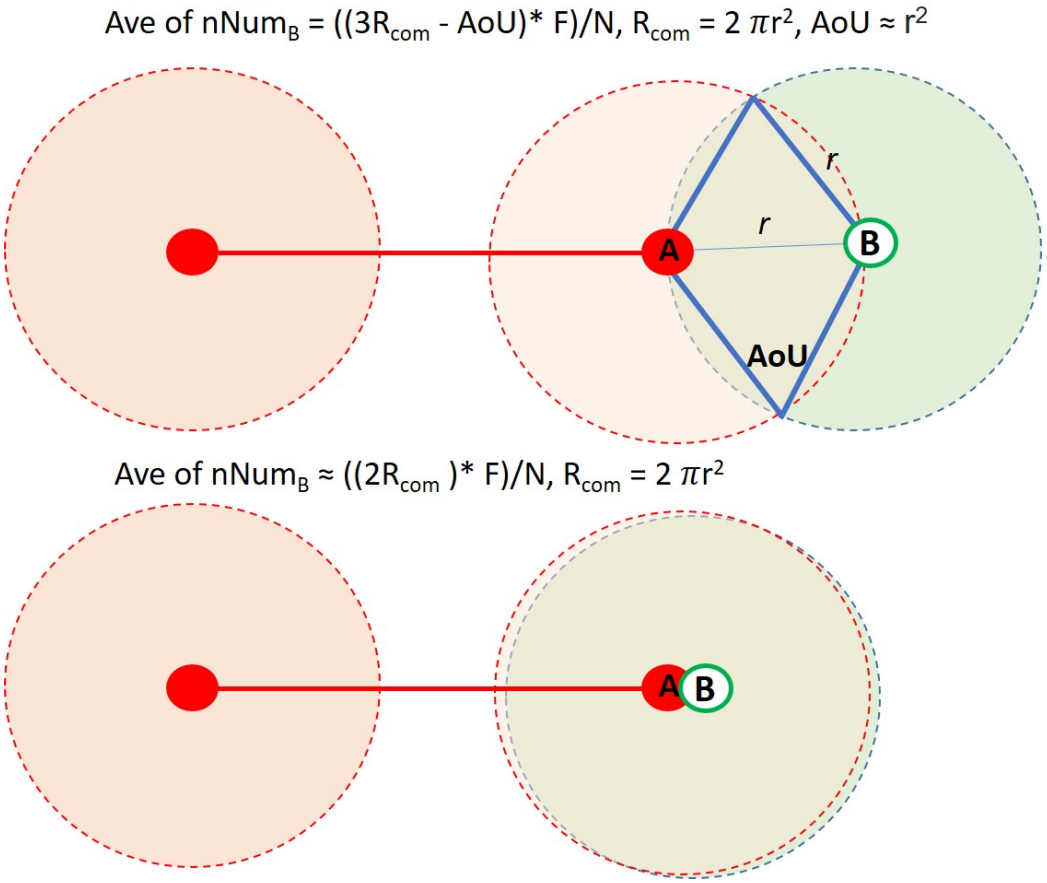


Figure 6.3: NSI Distance Analysis

$$nNum_B = ((3X_{com} - AoU)/F)N, AoU \approx r^2,$$
$$X_{com} \approx 2\pi r^2 \tag{6.5}$$

When there is no wormhole attacker, the NSI is bounded by the average number of neighbors $(X_{com}/F) * N + \alpha$. However, the average number of neighbors increases

Algorithm 4 SWANS()

Input: n_list, T_sh // a list of all nodes in the network, Thresholds
Output: wormhole_list, wormhole

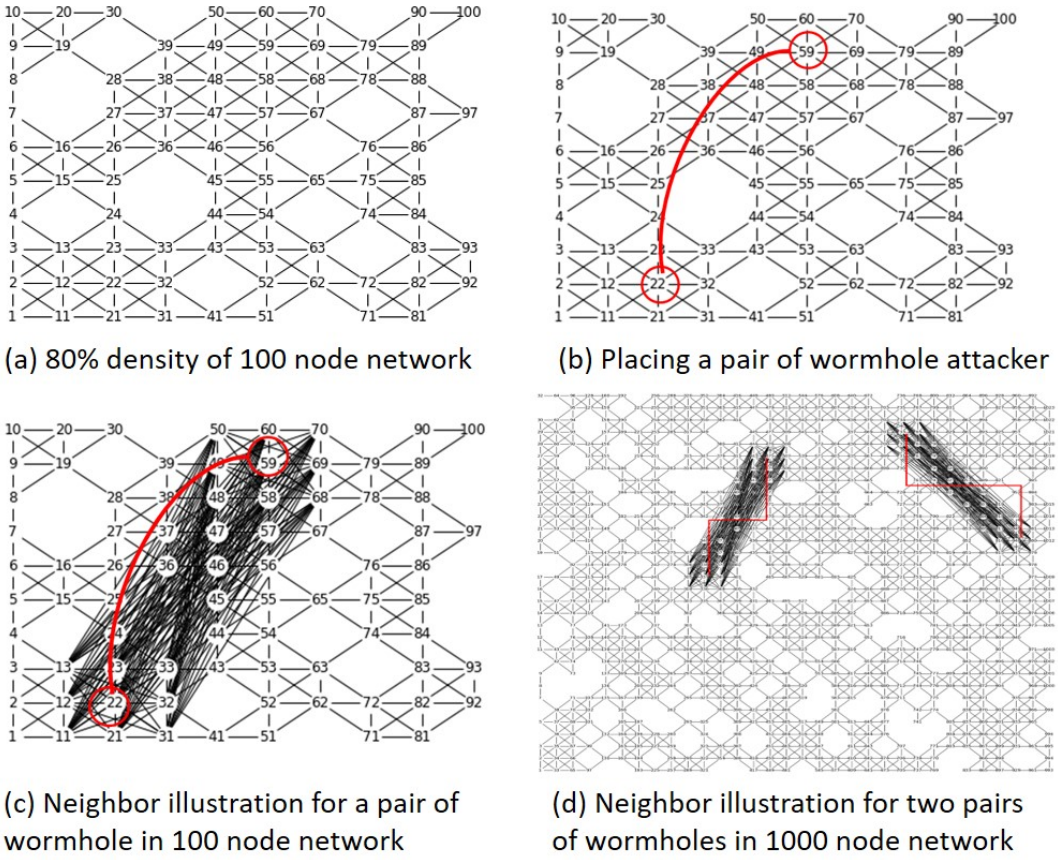
```
function MAIN()(n_list, T_sh)
    for i ← 0 to n − 1 do
        n-input[i]= read-neighbor-count(**n-table)
    end for
    /*check neighbor counts (NSI) to find abnormal
clusters*/
    if wormhole = check-wormhole(n-input, T_sh.ND, T_sh.NB) == True then
     return
    else
       /*annotate high neighbor cluster to find the ACI of
the clusters*/
       for i ← 0 to kmeans(k − 1) do
            a-input[i] = annotate-density(**kmeans(k-1));
       end for
       if wormhole = check-wormhole(a-input, T_sh.AD, T_sh.AB) == True then
     return
       end if
    end if
     return
end function
/*check NSI and ACI patterns*/
function CHECK_WORMHOLE()(input, Dtsh, Btsh)
    for k ← 2 to 3 do
        kmeans = KMeans(n_clusters=k, **input)
       if cluster-distance(kmeans) > Dtsh & count-balance(kmeans) < Btsh then
     return True
       end if
    end for
     return False
end function
```

approximately three times when there is a wormhole attacker. Hence the difference of the wormhole affected and not affected clusters (NSI) roughly increases by two times $(X_{com}/F) * N + \alpha$. Therefore, if the NSI value is greater than $(X_{com}/F) * N + \alpha$, it indicates a potential wormhole attacker in the network ($T\_sh.ND$). Besides, neighbor-balance ($T\_sh.NB$) is used to verify the wormhole attacker. Usually, the number of wormhole attacker affected nodes is far less than other nodes in the network. Typically, $T\_sh.NB$ should be less than $(2 * X_{com}/F) * 100 + \alpha$. However, due to various intelligent wormhole attacker models, it may not satisfy the value. Check_Wormhole () runs the k-means clustering algorithm with k=2 to find an initial NSI. However, Check_Wormhole () runs k-means clustering with k=3, if it cannot satisfy the $T\_sh.NB$. When the NSI tests with both k=2 and 3 k-means clustering fail, SWANS also checks ACI parameter. The NSI usually fails if a spoofing wormhole attacker model (RSWA in Figure 6.1 (d)) is used, which increases the number of high-neighbor nodes over the $T\_sh.NB$. As illustrated in Figure 6.2, there are two clustered areas of cluster 1. We annotate cluster 1 with different weights $+cluster1 * 2and - cluster0$. As the same cluster nodes usually surround the wormhole attacker range, they have higher density values (ACI). When a k-means clustering with k=2 or 3 is applied, it separates the less concentrated cluster from the more concentrated cluster.

6.4 Evaluations

We have evaluated the SWANS with all four wormhole attacker model scenarios, as illustrated in Figure 6.1. Our simulator is based on a discrete event simulation implemented by using Python. In the simulation system setup, we use both 100 and 1000 network nodes, uniformly distributed in the square field of a 100 by 1000 meters network area. We set a couple of discrete radius as a communication range (short and extended range) and vary the network density by configuring % of network nodes from 10% to 80%, randomly allocating

nodes in the network. One or two pairs of fixed wormhole nodes are randomly placed in the network field as an attack scenario. The mobility model used in the simulation is the Random Waypoint Model [62], which randomly changes the neighbor node IDs and empty slots. The total simulation time is set to 2000 seconds.



(a) 80% density of 100 node network

(b) Placing a pair of wormhole attacker

(c) Neighbor illustration for a pair of wormhole in 100 node network

(d) Neighbor illustration for two pairs of wormholes in 1000 node network

Figure 6.4: Evaluation network setup scenarios

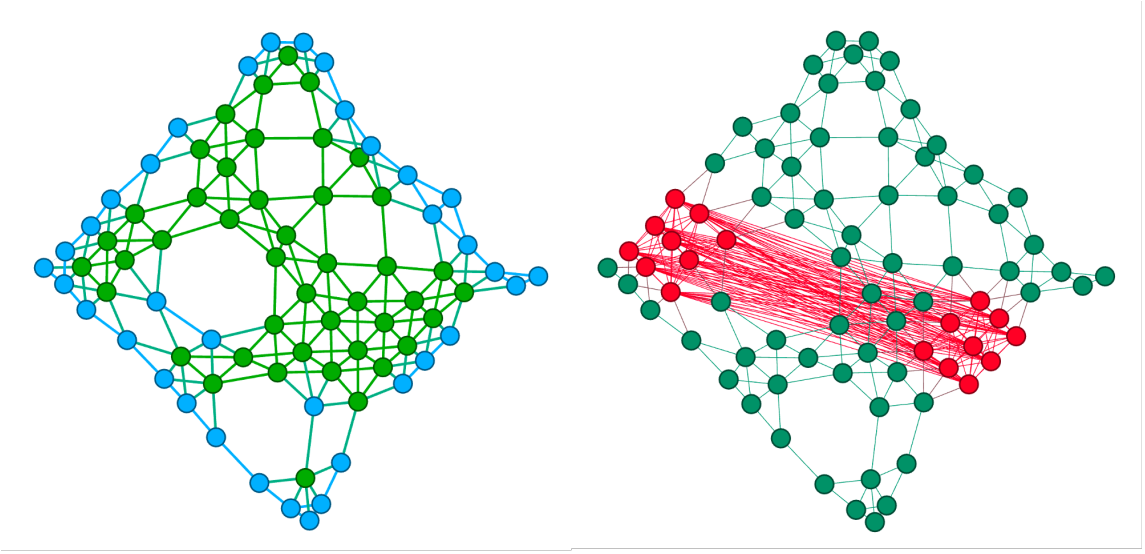
Figure 6.4 illustrates a few network setup and wormhole attack scenarios. For example, in Figure 6.4 (a) to (c), 80 initial nodes are randomly allocated in the 100 node network with 80% configuration. We keep on collecting neighbor counts during the simulation. We randomly arranged a pair of wormhole attackers (node id 59 and 22 selected as the endpoints of wormhole attackers) on the network. Also, in Figure 6.4 (d), we randomly

placed a couple of wormhole attacker pairs on the 1000 node network.

6.4.1 Typical Wormhole Attack

In this type of attack, the wormhole nodes Send LLDP request to nodes in its range and to the other wormhole endpoint. We tested the following cases for this type of attack: 1) 100 nodes network with one pair of wormhole attackers for both short and extended ranges 2) 1000 nodes network with one and two pairs of wormhole attack for both short and extended ranges.

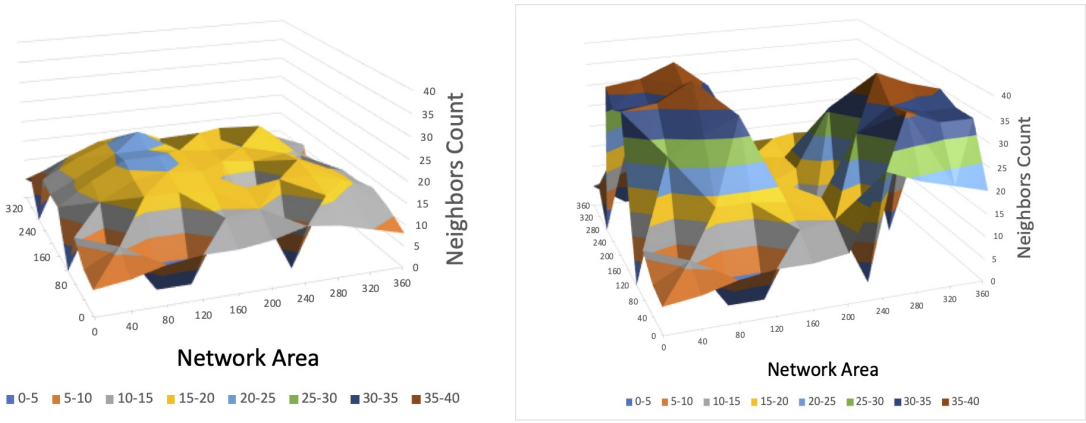
Figure 6.5 presents k means clustering results when k is 2. The neighboring count-based clustering results reveal the clusters of nodes with an abnormally high number of neighbors after a wormhole attack. It also shows the potential area of the attacker. We calculated the neighbor similarity index (NSI) using the distance between the centroid of each cluster. For example, in 100 node network with a short-range, cluster 0's centroid is 4.08, and cluster 1's centroid is 6.66. Hence, the NSI before any wormhole attack is 2.58, and each cluster has a similar size of nodes (balanced). It suggests almost all the nodes have similar neighbor counts. However, NSI after a wormhole attack becomes 13.63 as cluster 0's centroid is 5.26, and cluster 1's centroid is 18.89. Also, the number of nodes in cluster 1 is only 22.5% (unbalanced), and NSI is abnormally high, according to the average neighbor counts (4 to 9 is the normal NSI range).



a) K-means cluster (k=2) before attack

b) K-means cluster (k=2) after attack

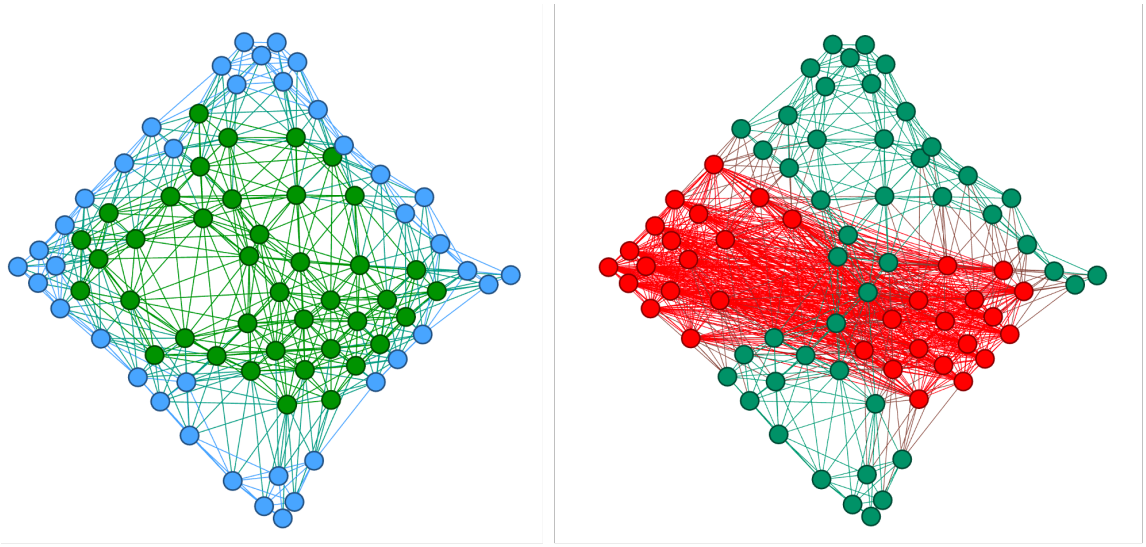
Figure 6.5: Short-range over 100 node network K-means results



(a) Extended-range network before attack

(b)Extended-range network after attack

Figure 6.6: Extended-range over 100 node network



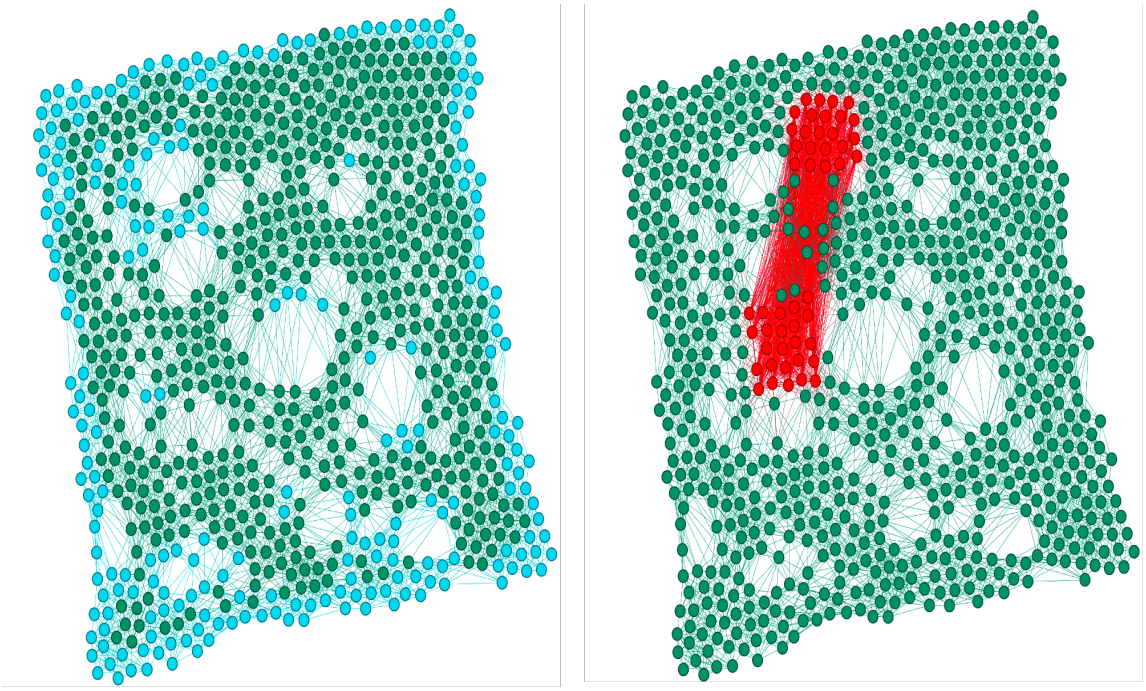
a) K-means cluster (k=2) before attack

b) K-means cluster (k=2) after attack

Figure 6.7: Extended-range over 100 node network k-means results

Figure 6.6 presents the neighbor counts in a 100 node network with an extended communication range before and after a wormhole attack. Before any wormhole attack, the neighbor counts are similar over the entire system. cluster 0's centroid is 10.80 with 41 nodes, and cluster 1's centroid is 17.97 with 39 nodes. The NSI before any wormhole attack is 7.17, and each cluster has a similar size of nodes (balanced). However, NSI after a wormhole attack becomes 19.54 as cluster 0's centroid is 13.46, and cluster 1's centroid is 33. The NSI is abnormally high comparing to the normal average neighbor counts which is around 13 nodes. As shown in Figure 6.7, which presents k means clustering results before and after an attack when k is 2, the algorithm accurately identifies the nodes with abnormal neighbor counts after the wormhole attack.

Figure 6.8 presents k-means clustering results when k is 2 in a 1000 node network (around 80%) with an extended communication range before and after a wormhole attack. After a wormhole attack, the results clearly show the potential wormhole attack areas. The NSI before any wormhole attack is 5.92, and the number of nodes in each cluster



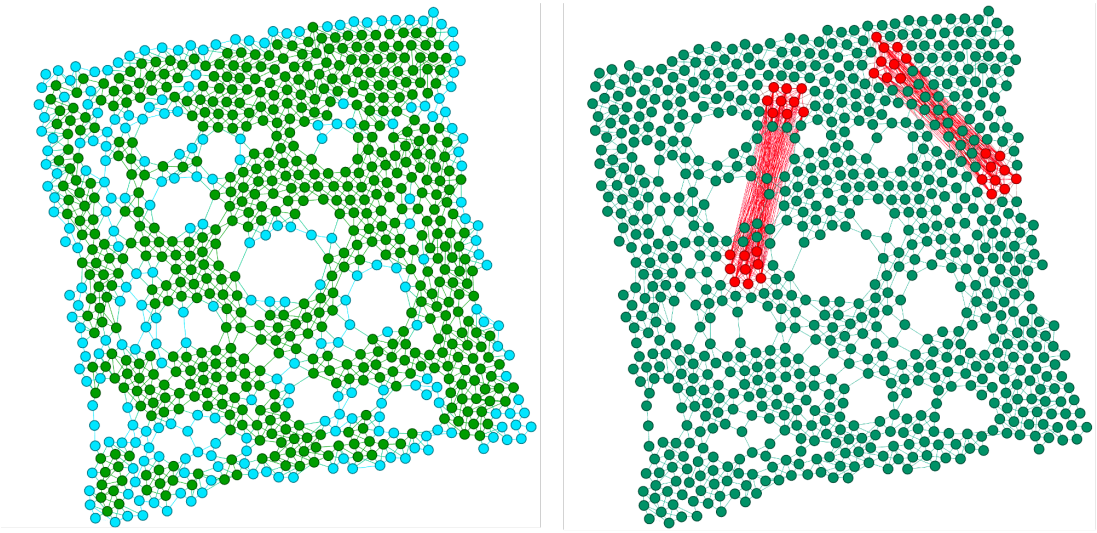
a) K-means cluster (k=2) before attack

b) K-means cluster (k=2) after attack

Figure 6.8: Extended range over 1000 node network against a pair of wormhole attackers

was balanced (cluster 0 is 13.31 with 337 nodes, and cluster 1 is 19.23 with 482 nodes). However, after a wormhole attack, the NSI becomes very high, 36.58, and cluster1 has only 5.8% of the nodes (cluster 0 is 16.61 with 772 nodes, and cluster 1 is 53.19 with 47 nodes).

Figure 6.9 presents the neighbor counts in a 1000 node network with a short communication range before and after a couple of concurrent wormhole attacks. Before any wormhole attack, the neighbor counts are similar over the entire system. However, after a wormhole attack, the results show the potential four wormhole attack areas. The NSI before any wormhole attack is 2.41 (cluster 0's centroid is 4.42, and cluster 1's centroid is 6.83 using a k means clustering (k is 2)), and the number of nodes in each cluster was balanced. However, after a couple of concurrent wormhole attacks, the NSI is very high, 14.18 (cluster 0's centroid is 6.01, and cluster 1's centroid is 20.19 using a k means clustering (k is 2)), and cluster1 has only 4.5% of the nodes.



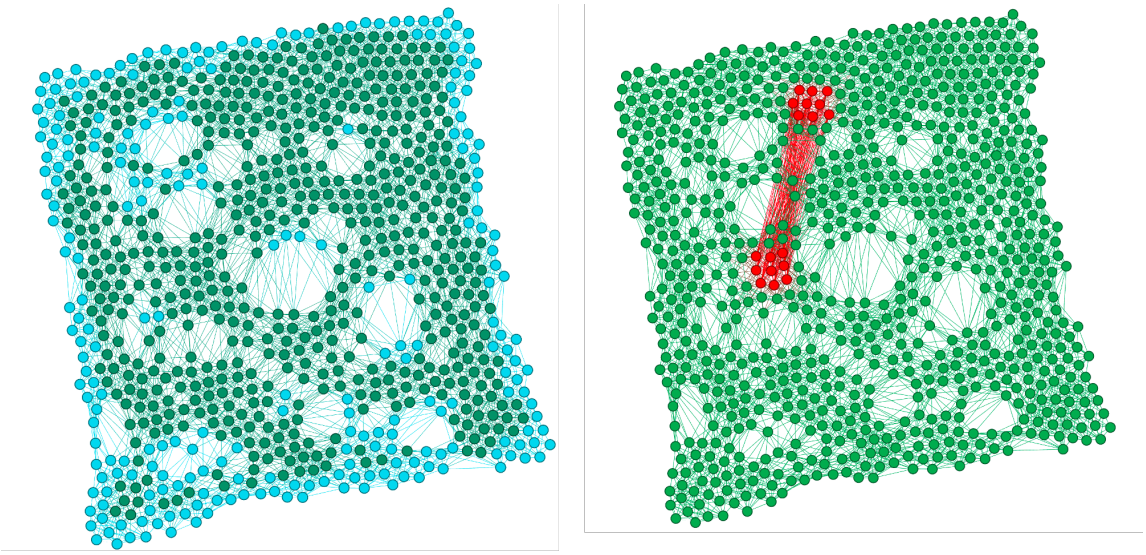
a) K-means cluster (k=2) before attack

b) K-means cluster (k=2) after attack

Figure 6.9: Short range on 1000 node network with two pairs of wormhole attackers

6.4.2 Reduced Range Wormhole Attack

In this type of attack, the attacker can adjust neighbor counts by manipulating its reply to LLDP requests.The attacker tries to hide its location by reducing the neighbor counts. It can shrink its communication range smaller than the existing node's range. We tested this type of attack by using 1000 nodes network with an extended-range for normal nodes, and a short-range for wormhole nodes. Figure 6.10 presents the k-means clustering results in a 1000 node network (around 80%) with an extended communication range before and after a reduced range wormhole attack (RRWA). After an RRWA, the results clearly show the potential wormhole attack areas. After a wormhole attack, the NSI with k=2 is 6.26, and the number of nodes in each cluster was balanced (cluster 0 is 13.31 with 337 nodes, and cluster 1 is 19.27 with 482 nodes), which cannot detect any wormhole attacker. However, running with k=3, the NSI of cluster 2 becomes very high, 17.49, and cluster2 has only 2.25% of the nodes (cluster 0 is 12.68 with 273 nodes, cluster 1 is 18.77 with 528 nodes, and cluster 2 is 30.17 with 18 nodes).



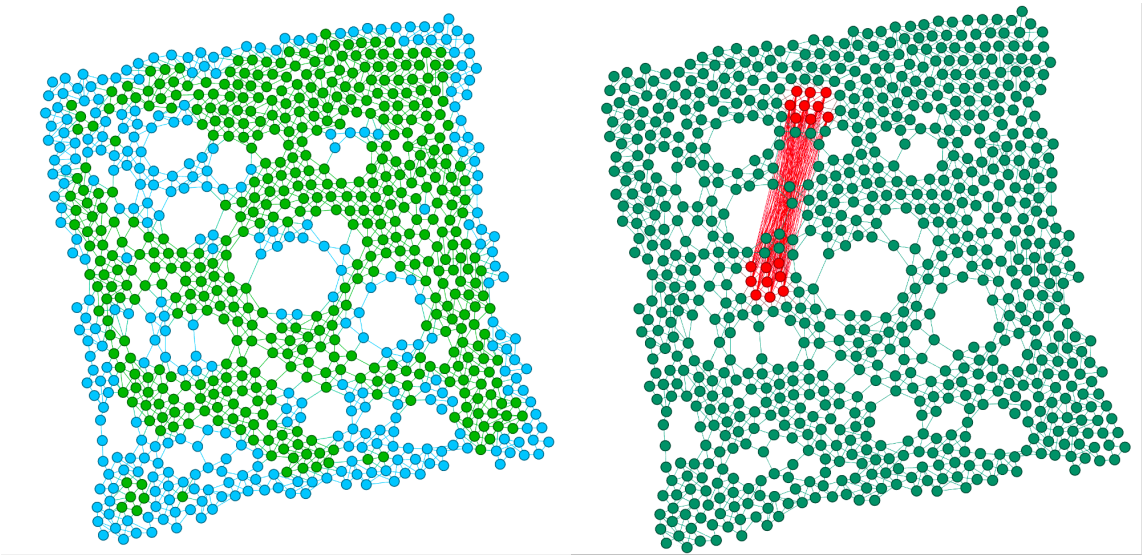
a) K-means cluster (k=2) before attack

b) K-means cluster (k=3) after attack

Figure 6.10: Extended range over 1000 node network against RRWA

6.4.3 Remote Only Wormhole Attack

In this type of attack, the attacker can send LLDP request to only one of the two endpoints without sending to nodes in its range. The second endpoint then broadcasts the LLDP requests to its neighbors.We tested this type of attack by using a 1000 nodes short-range network. Figure 6.11 presents the k-means clustering results in a 1000 node network (around 80%) with a short communication range before and after a remote only wormhole attack (ROWA). After a wormhole attack, the results clearly show the potential wormhole attack areas. The NSI before any wormhole attack is 2.41, and the number of nodes in each cluster was balanced (cluster 0 is 4.42 with 254 nodes, and cluster 1 is 6.83 with 565 nodes). However, after a wormhole attack, the NSI becomes very high, 10.62, and clusters are not balanced (cluster 0 is 6.05 with 801 nodes, and cluster 1 is 16.67 with 18 nodes).



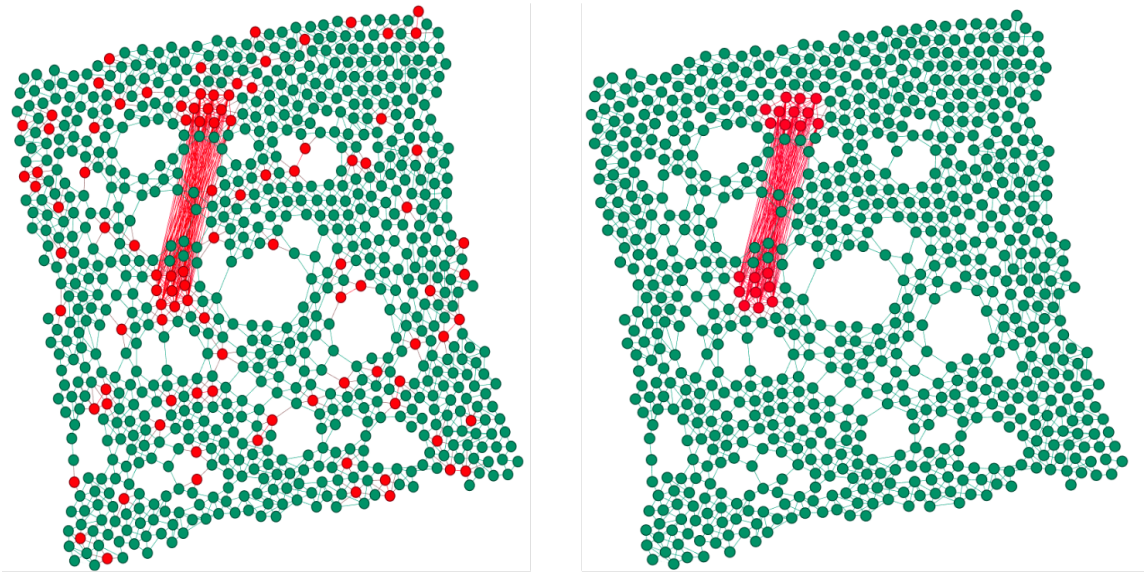
a) K-means cluster (k=2) before attack

b) K-means cluster (k=2) after attack

Figure 6.11: Short-range over 1000 nodes network against ROWA

6.4.4 Spoofing Wormhole Attack

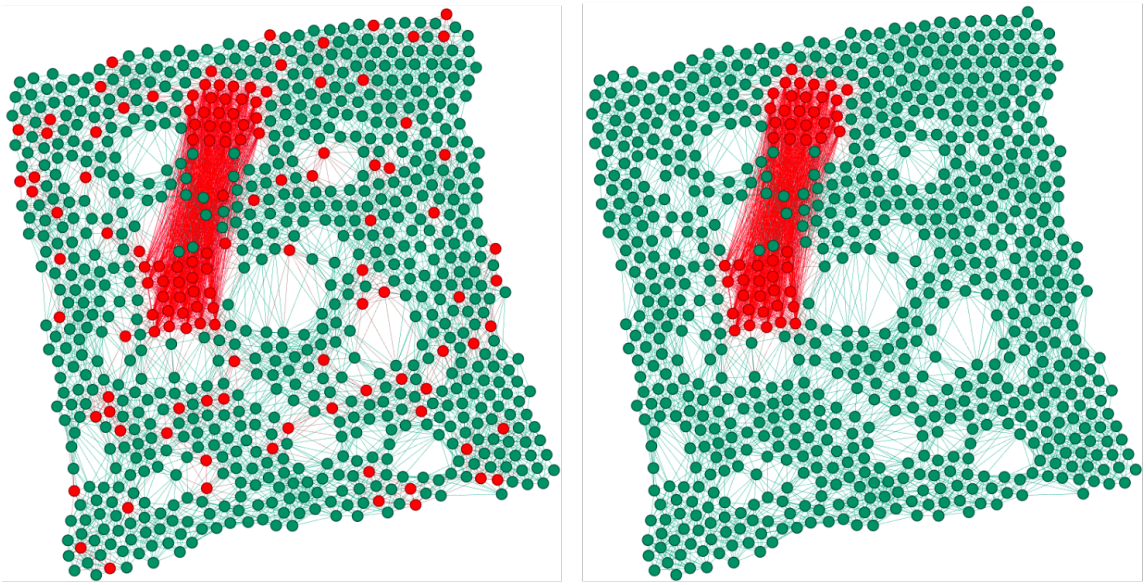
In this experiment, we evaluate the fourth type of attack, spoofing wormhole attack. We configure the network with different percentages of nodes under spoofing for both short and extended network ranges. We evaluate different networks against 10%, 20%, 30%, 40%, and 50% of nodes under spoofing attack and show how our approach reacts to them. We recognized that as the number of nodes under spoofing attack increases, the number of high neighbors' nodes in cluster 1 increases, making it challenging to identify the wormhole attackers' locations due to noise around wormhole areas or false-positive areas. The Augmented Concentration Index (ACI) algorithm clearly recognizes the location of the wormhole attack in all attack cases.



a) K-means cluster (k=2) before ACI

b) K-means cluster (k=2) After ACI

Figure 6.12: Augmented Concentration Index (ACI) Results against 10% Spoofing (Short-range)



a) K-means cluster (k=2) before ACI b) K-means cluster (k=2) After ACI

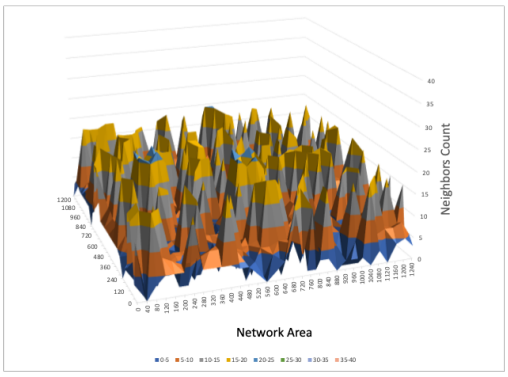
Figure 6.13: Augmented Concentration Index (ACI) Results against 10% Spoofing (Extended-range)

Figures 6.12, 6.13 presents the k-means clustering results after 10% of random spoofing wormhole attack (RSWA) in both short and extended range networks. After the spoofing attack (82 node's neighbor counts become as high as the wormhole attacker's neighbor nodes), the NSI alone cannot identify any wormhole attacker. For example, NSI becomes high, 12.35 (cluster 0 is 6.05, and cluster 1 is 18.4) in a short-range and 32.44 (cluster 0 is 16.6, and cluster 1 is 49.04) in an extended range. However, the cluster nodes are not balanced (cluster 1 has 100 nodes in a short-range and 129 nodes in an extended range). It can be observed from the graphs that there are many scattered nodes (noises) with high neighbors in Figure 6.12 and Figure 6.13 (a). However, after augmenting cluster 1 by incentivizing and penalizing each node, the ACI with k=2 k means indicates the wormhole attacker areas in Figure 6.12 and Figure 6.13 (b).

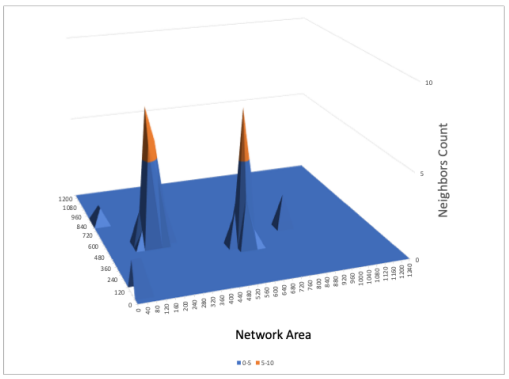
Figures from 6.14 to 6.16 exhibits the results before and after applying the Aug-

mented Concentration Index (ACI) algorithm to 20%, 30%, and 40% spoofing wormhole attacks for both short and extended range networks. For the number of nodes, 164 nodes are under a spoofing attack for 20% attack, 246 nodes for 30% attack, and 328 nodes for 40% attack. As shown in (a) for short-range networks and (c) for extended-range Networks, the location of wormhole attackers is hidden between the scattered nodes with high neighbors. However, by implementing the k-means algorithm on the ACI results, the algorithm accurately locates the wormhole attackers in (b) and (d), even with many nodes under a spoofing attack.

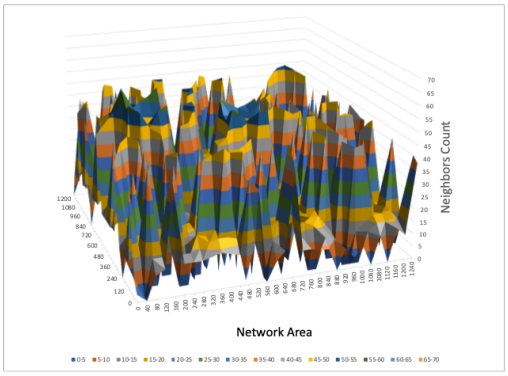
Figure 6.17 presents the k-means clustering results after 50% of random spoofing wormhole attack (RSWA) in an extended range network. After the spoofing attack (409 node's neighbor counts become as high as the wormhole attacker's neighbor nodes), the NSI alone cannot identify any wormhole attacker. For example, NSI becomes high, 30.56 (cluster 0 is 16.68, and cluster 1 is 47.24). However, the cluster nodes are not balanced (cluster 1 has 456 nodes). It can be observed from the graph that there are many scattered nodes (noises) with high neighbors in Figure 6.17 (a). After augmenting cluster 1 by incentivizing and penalizing each node, the ACI with k=2 Kmeans still presents many noises. However, the ACI with k=3 Kmeans indicates the wormhole attacker areas clearly in cluster 2 (23.29 with 42 nodes), as shown in Figure 6.17 (b).



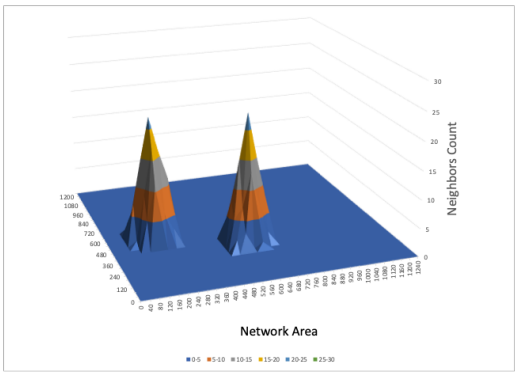
a) Short-range network 20% spoofing before ACI



b) Short-range network 20% spoofing after applying ACI

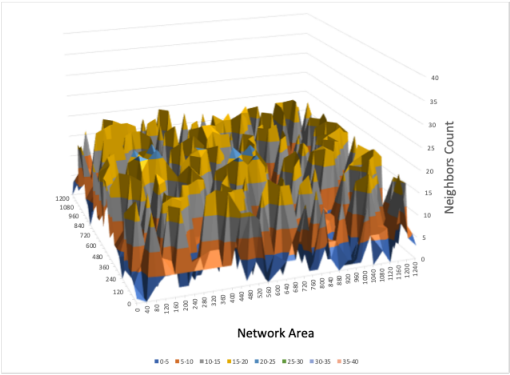


c) Extended-range network 20% spoofing before ACI

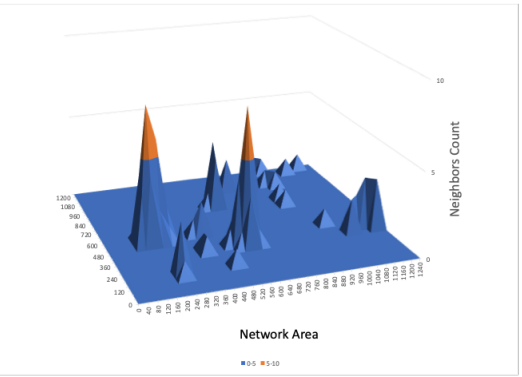


d) Extended-range network 20% spoofing after applying ACI

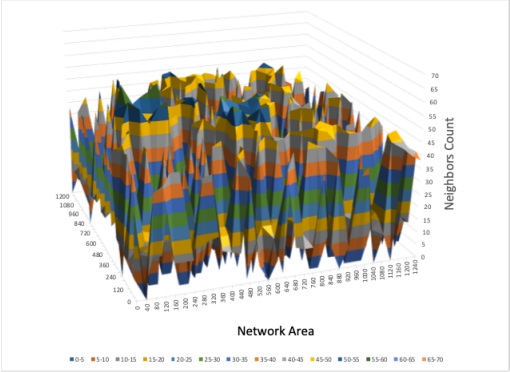
Figure 6.14: Augmented Concentration Index (ACI) Results against 20% Spoofing



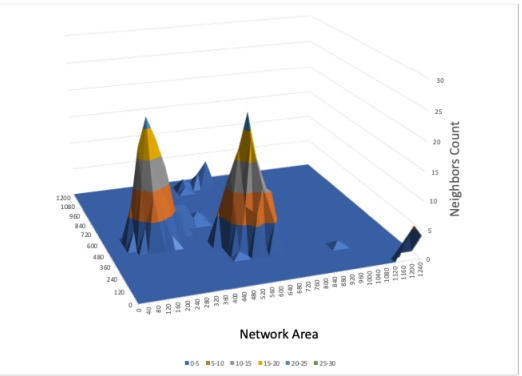
a) Short-range network 30% spoofing before ACI



b) Short-range network 30% spoofing after applying ACI

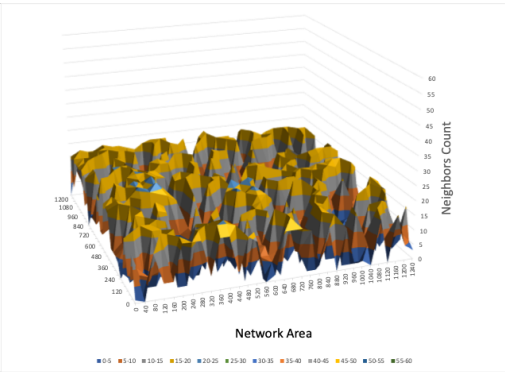


c) Extended-range network 30% spoofing before ACI

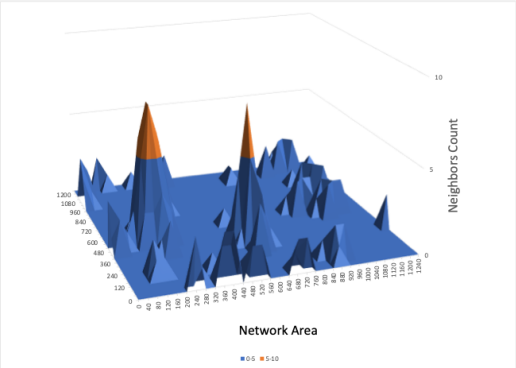


d) Extended-range network 30% spoofing after applying ACI

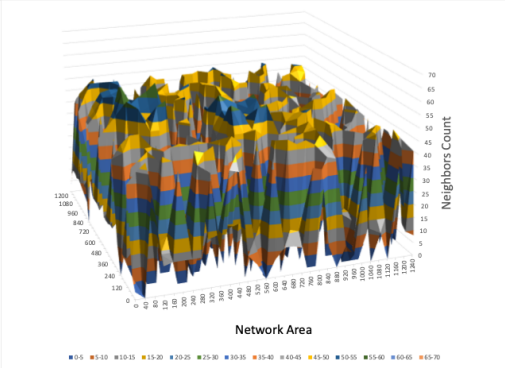
Figure 6.15: Augmented Concentration Index (ACI) Results against 30% Spoofing



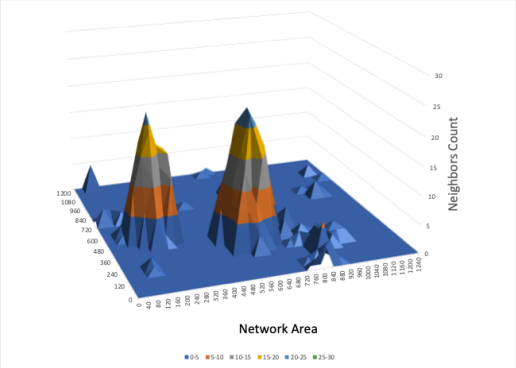
a) Short-range network 40% spoofing before ACI



b) Short-range network 40% spoofing after applying ACI

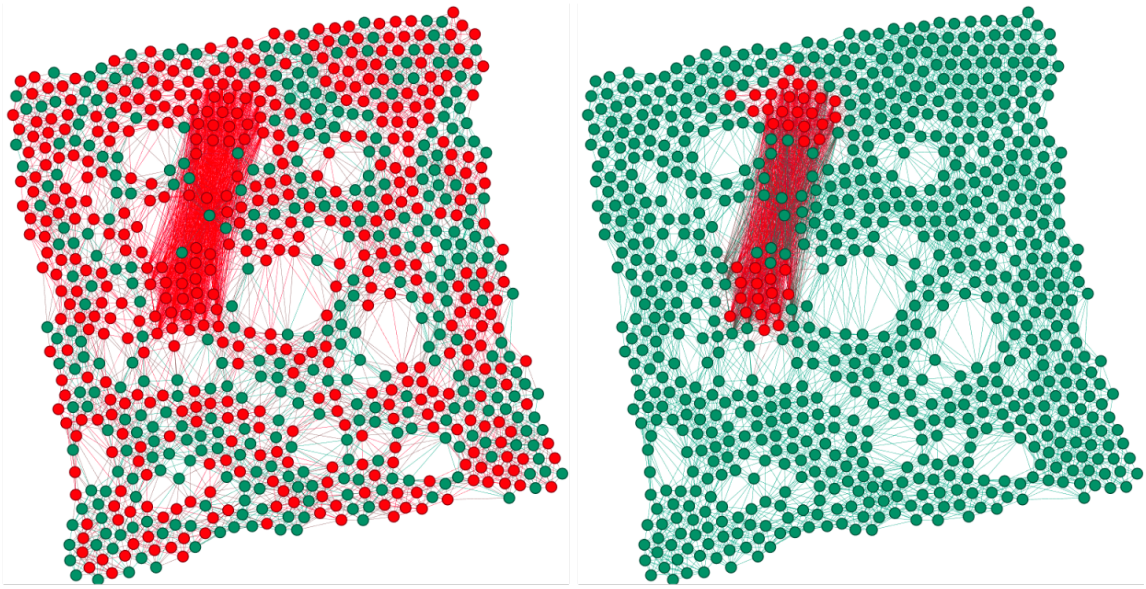


c) Extended-range network 40% spoofing before ACI



d) Extended-range network 40% spoofing after applying ACI

Figure 6.16: Augmented Concentration Index (ACI) Results against 40% Spoofing



a) K-means cluster (k=2) before ACI b) K-means cluster (k=3) After ACI

Figure 6.17: Augmented Concentration Index (ACI) Results against 50% Spoofing

6.5 Conclusions

An SDN-based Wormhole Analysis using the Neighbor Similarity (SWANS) approach is presented as a novel wormhole countermeasure in a Software-defined MANET. As SWANS analyses the similarity of neighbor counts at a centralized SDN controller, it apprehends wormholes not only without requiring any particular location information but also without causing significant communication and coordination overhead. SWANS algorithm also countermeasures various intelligent wormhole attacker model scenarios applying NSI and ACI values. The extensive experimental results show that SWANS can detect wormhole attacks efficiently against very sophisticated attack scenarios.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary

This work presents a management framework for software-defined networks to build a reliable and secure discovery protocol for wired, wireless, and mobile networks. Specifically, we provide a security, reliability, and scalability analysis to the current OpenFlow Discovery Protocol (OFDP) in SDN. Our work consists of three major components, performance and reliability enhancement for wireless and mobile SDN (TaDPole), performance and reliability enhancement for wired and Static Wireless SDN (CAMEL), and Wormhole attack security analysis for wireless and mobile SDN (SWANS).

In TaDPole, we first introduce a multiple timer module, which uses various discovery frequency timers for each target instead of using a uniform period for the entire network. Mainly, Tadpole sends more frequent control messages to the significant nodes to expedite the detection of any status updates. However, unlike wired networks, we cannot rely on the network topology and structure to identify the significance of each node because nodes frequently churn in a mobile SDN environment. For that reason, we propose using the data traffic usage in each node in the assumption that the busier node has more importance. TaDPole also redesigns the OpenFlow discovery protocol (OFDP) to support the port-neutral broadcast-based discovery for wireless and mobile SDN. We modified the OFDP protocol to send a single packet for both Packet-Out and LLDP neighbor discovery to the node instead of each port. The node then broadcasts the message to neighboring

nodes, which reduces the amount of control traffic. The work results show that TaDPole can smooth the control traffic burstiness, decrease the discovery protocol delay, and reduce the control traffic overhead.

In CAMEL, we proposed a Centrality-Aware Multitemporal discovery protocol for software-defined networks. CAMEL promotes multiple discovery timers for each target according to the significance instead of using a single timer for the entire network. CAMEL generalizes a node significance measurement for various network topologies by using a combination of degree and betweenness centralities models to find an unbiased impact factor of each node. It assumes that nodes with higher centrality value have more impact on the network. CAMEL reduces network service impact by decreasing the discovery delay for the significant nodes without compromising the control message efficiency. The experiment results prove that CAMEL enhances discovery message performance and makes the control traffic less bursty. It improves the network service quality by lessening the discovery delay to the significant nodes without increasing the overall control traffic overhead.

In SWANS, we provide an comprehensive analysis and countermeasure to one of the most challenging security problems in wireless networks wormhole attack. SWANS apprehends wormhole attacks using a lightweight neighbor counting algorithm at a centralized SDN controller. After doing many experiments to analyze the behavior of the wormhole attack, we found that the number of neighbors of a node increases when a node gets in the communication range of a wormhole node. SWANS uses a K-means clustering algorithm to determine the neighbor count abnormality and detect the Wormhole Attackers' areas. The simulation results show that SWANS can efficiently recognize various intelligent wormhole attacks with low false-positive and false-negative rates. SWANS also does not require global topology information or special hardware to identify the attack.

In general, this dissertation provides practical approaches to tackle security, reliability, and scalability issues in software-defined networks for different types of computer

networks, including wired, wireless and mobile networks. The proposed methods help the SDN controller to offload many control traffic overhead. Secure the system from wormhole attacks and provide a fast detection for node and link failure while maintaining overall low control traffic.

7.2 Future Work

Network reliability management and security are pretty challenging fields of research that need to be addressed by future studies. Our future work could provide an analysis of optimal discovery frequency timers for TaDPole and CAMEL that ensure they detect significant node failure faster while having less overall control traffic. Regarding identifying node importance in the network, we would like to explore more centrality measures such as Closeness and EigenVector and find a shared relationship between them to identify significant nodes in the network accurately. For SWANS, we would like to discuss other unsupervised clustering algorithms such as hierarchical clustering and compare the results to our results in SWANS in terms of performance, accuracy, and detection rate.

REFERENCE LIST

[1] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: An Authoritative Review of network Programmability Technologies*. " O'Reilly Media, Inc.", 2013.

[2] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[3] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Network*, vol. 31, no. 5, pp. 21–27, 2017.

[4] A. K. Arahunashi, S. Neethu, and H. V. Ravish Aradhya, "Performance analysis of various sdn controllers in mininet emulator," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT)*, 2019, pp. 752–756.

[5] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, pp. 1–1, 01 2016.

[6] M. Singh and R. Das, "A survey of different techniques for detection of wormhole at-

tack in wireless sensor network," *International Journal of Scientific and Engineering Research*, vol. 3, no. 10, 2012.

[7] S. Kumar, "A survey of wormhole attacks in mobile ad hoc networks," *International Journal of Recent Research Aspects*, vol. 4, no. 4, pp. 93–99, 2017.

[8] G. Kaur, V. Jain, and Y. Chaba, "Wormhole attacks: Performance evaluation of on demand routing protocols in mobile adhoc networks," in *2011 World Congress on Information and Communication Technologies*, 2011, pp. 1155–1158.

[9] S. Khan, A. Gani, A. W. Abdul Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 303–324, 2017.

[10] L. Ochoa Aday, C. Cervelló Pastor, and A. Fernández Fernández, "Current trends of topology discovery in openflow-based software defined networks," Research Report: https://upcommons.upc.edu/bitstream/handle/2117/77672/Current%20Trends%20of%20Discovery%20Topology%20in%20SDN.pdf, 09 2015, accessed: 2019-12-20.

[11] A. Nicolae, L. Gheorghe, M. Carabas, N. Tapus, and C.-L. Duta, "Lldp packet generator," in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 2015, pp. 7–11.

[12] L.-D. Chou, C.-C. Liu, M.-S. Lai, K.-C. Chiu, H.-H. Tu, S. Su, C.-L. Lai, C.-K. Yen, and W.-H. Tsai, "Behavior anomaly detection in sdn control plane: A case study of topology discovery attacks," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 357–362.

[13] T.-H. Nguyen and M. Yoo, "Analysis of link discovery service attacks in sdn con-

troller," in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 259–261.

[14] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into sdn topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1101–1114.

[15] A. Azzouni, N. T. M. Trang, R. Boutaba, and G. Pujolle, "Limitations of openflow topology discovery protocol," in *2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. IEEE, 2017, pp. 1–3.

[16] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in openflow-based software defined networks," *Computer Communications*, vol. 77, pp. 52–61, 2016.

[17] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 109–114. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491187

[18] G. Tarnaras, E. Haleplidis, and S. Denazis, "Sdn and forces based optimal network topology discovery," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[19] Y. Gu, D. Li, and J. Yu, "Im-ofdp: An improved openflow-based topology discovery protocol for software defined network," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 628–630.

[20] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández *et al.*, "Discovering the network topology: an efficient approach for sdn," *Advances in Distributed Computing andArtificial Intelligence Journal*, vol. 5, no. 2, pp. 101–108, 2016.

[21] Y. Jia, L. Xu, Y. Yang, and X. Zhang, "Lightweight automatic discovery protocol for openflow-based software defined networking," *IEEE Communications Letters*, vol. 24, no. 2, pp. 312–315, 2020.

[22] H. Gebre-Amlak, G. Banala, S. Song, B.-Y. Choi, T. Choi, and H. Zhu, "Tarman: Topology-aware reliability management for softwarized network systems," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2017, pp. 1–6.

[23] Y. . Hu, A. Perrig, and D. B. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 3, 2003, pp. 1976–1986.

[24] P. Shukla, "Ml-ids: A machine learning approach to detect wormhole attacks in internet of things," in *2017 Intelligent Systems Conference (IntelliSys)*, 2017, pp. 234–240.

[25] S. Jamali and R. Fotohi, "Dawa: Defending against wormhole attack in manets by using fuzzy logic and artificial immune system," *The Journal of Supercomputing*, vol. 73, no. 12, pp. 5173–5196, 2017.

[26] S. Song, H. Wu, and B.-Y. Choi, "Statistical wormhole detection for mobile sensor networks," in *2012 Fourth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2012, pp. 322–327.

[27] M. A. Patel and M. M. Patel, "Wormhole attack detection in wireless sensor network,"

in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, pp. 269–274.

[28] R. Singh, J. Singh, and R. Singh, "Wrht: a hybrid technique for detection of wormhole attack in wireless sensor networks," *Mobile Information Systems*, vol. 2016, pp. 1–13, 12 2016.

[29] A. Shaon, Mohammad Nurul and K. Ferens, "Wireless sensor network wormhole detection using an artificial neural network," in *Proceedings of the International Conference on Wireless Networks (ICWN)*. The Steering Committee of The World Congress in Computer Science, Computer, 2015, p. 115.

[30] K. Singh, G. Singh, and A. Agrawal, "A trust based approach for detecting and preventing wormhole attack in manet," *International Journal of Computer Applications*, vol. 94, no. 20, pp. 1–5, 2014.

[31] F. Rezaei and A. Zahedi, "Dealing with wormhole attacks in wireless sensor networks through discovering separate routes between nodes," *Engineering, Technology & Applied Science Research*, vol. 7, no. 4, pp. 1771–1774, 2017.

[32] M. Patel, A. Aggarwal, and N. K. Chaubey, "Detection of wormhole attacks in mobility-based wireless sensor networks," *International Journal of Communication Networks and Distributed Systems*, vol. 21, no. 2, pp. 147–156, 2018.

[33] R. A. Prakash, W. S. Jeyaseelan, and T. Jayasankar, "Detection, prevention and mitigation of wormhole attack in wireless adhoc network by coordinator," *Applied Mathematics and Information Sciences*, vol. 12, pp. 233–237, 2018.

[34] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai, "Tesseract: A 4d network control plane," in *Proceedings of the 4th USENIX*

Conference on Networked Systems Design and Implementation, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 27–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973430.1973457

[35] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 37, no. 4, pp. 1–12, Aug. 2007. [Online]. Available: http://doi.acm.org/10.1145/1282427.1282382

[36] O. N. Foundation, "Openflow switch specification," https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, accessed: 2018-09-21.

[37] Cisco, "Link layer discovery protocol," https://learningnetwork.cisco.com/s/article/link-layer-discovery-protocol-lldp-x, accessed: 2018-09-21.

[38] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD)," in *RFC 5880 (Proposed Standard), Internet Engineering Task Force*, 2010.

[39] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band openflow networks," in *Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the*, March 2013, pp. 52–59.

[40] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2559899.2560327

[41] R. Bifulco, J. Boite, M. Bouet, and F. Schneider, "Improving SDN with inspired switches," in *Proceedings of the Symposium on SDN Research*, ser. SOSR

'16. New York, NY, USA: ACM, 2016, pp. 11:1–11:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890962

[42] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[43] "RYU controller," https://ryu-sdn.org, accessed: 2019-12-20.

[44] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.

[45] "ODL opendaylight," https://www.opendaylight.org/, accessed: 2018-09-22.

[46] "Mininet mininet," http://mininet.org/, accessed: 2019-12-20.

[47] "Lldp," https://iperf.fr, Sep 2018, accessed: 2018-09-21.

[48] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Special Interest Group on Data Communication Computer Communication Review*, vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[49] Z. Wan, Y. Mahajan, B. W. Kang, T. J. Moore, and J.-H. Cho, "A survey on centrality metrics and their implications in network resilience," *arXiv preprint arXiv:2011.14575*, 2020.

[50] J. Zhang and Y. Luo, "Degree centrality, betweenness centrality, and closeness centrality in social network," in *Proceedings of the 2017 2nd International Conference*

on Modelling, Simulation and Applied Mathematics (MSAM2017), vol. 132, 2017, pp. 300–303.

[51] "Wireshark," https://www.wireshark.org, Sep 2018, accessed: 2018-09-21.

[52] Y.-C. Hu, A. Perrig, and D. Johnson, "Wormhole attacks in wireless networks," *IEEE Transactions on Selected Areas in Communications*, vol. 24, no. 2, pp. 370 – 380, Feb. 2006.

[53] A. Shrivastava and R. Dubey, "Wormhole attack in mobile ad-hoc network: a survey," *International Journal of Security and its Applications*, vol. 9, no. 7, pp. 293–298, 2015.

[54] L. Hu and D. Evans, "Using directional antennas to prevent wormhole attacks," in *Network and Distributed System Security Symposium*, vol. 4, 2004, pp. 241–245.

[55] S. Khurana and N. Gupta, "Feepvr: First end-to-end protocol to secure ad hoc networks with variable ranges against wormhole attacks," in *2008 Second International Conference on Emerging Security Information, Systems and Technologies*, 2008, pp. 74–79.

[56] S. Capkun, K. Rasmussen, M. Cagalj, and M. Srivastava, "Secure location verification with hidden and mobile base stations," *IEEE Transactions on Mobile Computing*, vol. 7, no. 4, pp. 470–483, 2008.

[57] M. Khabbazian, H. Mercier, and V. K. Bhargava, "Severity analysis and countermeasure for the wormhole attack in wireless ad hoc networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 2, pp. 736–745, 2009.

[58] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in

ACM Workshop on Wireless security, ser. WiSe '03, New York, NY, USA, 2003, pp. 1–10.

[59] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, p. 79–88.

[60] S. Guha and N. Koudas, "Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation," in *International Conference on Data Engineering*, 2002, pp. 567–576.

[61] D. Scott, *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley & Sons, 1992.

[62] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '98, 1998, pp. 85–97.

VITA

Faheed A.F. Alenezi earned his bachelor's degree in Information Technology from Northern Border University (NBU), Saudi Arabia, in 2012. He worked as a teaching assistant at (NBU). He obtained his M.S. degree in Computer Science and Electrical Engineering at the University of Missouri-Kansas City. Currently, he is pursuing his Ph.D. degree in Telecommunications and computer networking at the University of Missouri-Kansas City. His research interest lies in Software-defined networks, Network analysis, and Network systems security.