

FORENSIC INVESTIGATION OF LINK FABRICATION
ATTACK IN SOFTWARE DEFINED NETWORKS

SULEMAN KHAN

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

**FORENSIC INVESTIGATION OF LINK
FABRICATION ATTACK IN SOFTWARE DEFINED
NETWORKS**

SULEMAN KHAN

**THESIS SUBMITTED IN FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY**

**FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR**

2017

UNIVERSITY OF MALAYA
ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Suleman Khan

Registration/Matric No: WHA130036

Name of Degree: Doctor of Philosophy

Title of Dissertation: Forensic Investigation of Link Fabrication Attack in Software Defined Networks

Field of Study: Forensics in Mobile Cloud Computing (COMPUTER SCIENCE)

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This Work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature

Date:

Subscribed and solemnly declared before,

Witness's Signature

Date:

Name:

Designation:

ABSTRACT

Software Defined Networking (SDN) is an emergent network architecture with a unique feature of decoupling an infrastructure plane from the control plane. SDN enables network-wide visibility to the applications running on top of the controller by executing a topology discovery module. However, the adversaries try to exploit the controller visibility due to its centralized control management of the entire network. The SDN faces topology vulnerabilities due to lack of security concern in its initial development of the architecture. Thus, the existing vulnerabilities in the controller attract the adversary to exploit SDN for various illegitimate reasons. For instance, the controller lacks an authentication mechanism to differentiate between legitimate and spoofed Link Layer Discovery Protocol (LLDP) packets. The LLDP packets are used by the topology discovery module to determine a link between the switches which further assists the controller to build the network topology. The legitimate network topology is an utmost important in SDN because adversaries can inject fake links between the switches to fabricate the network topology. The fabrication of fake links in the network topology is called Link Fabrication Attack (LFA). The LFA occurs due to malicious switches and hosts that spoof the LLDP packets to generate fake links between the switches. The fake links are used for numerous illegal reasons including eavesdropping, diverting legitimate traffic, and packet drops. Currently, the available techniques are available to detect fake links, but they fail to identify the real source of the attack. Thus, SDN requires having a forensic method which not only detects fake links but determines the real source of the fake links. Therefore, we proposed a forensic-based investigation method (FoR-Guard) to detect fake links as well as determine the real source of the LFA. The FoR-Guard is composed of three main phases namely trigger, Detection and Source Identification (DeSI), and validation phase. The trigger phase triggers an alarm message to the DeSI phase by observing the fake link generated between the switches. The trigger phase used

Malicious Index Record (MIR) of the switches to trigger a message. The DeSI phase investigates to detect fake links between the switches by checking the Link Communication Direction (LCD) and MIR information of the respective link and switch respectively. Afterwards, a traceback mechanism is used to identify the cause of the attack by determining the malicious host connected to the switch. The validation phase verifies the true source of the attack by using probability and entropy measurements. Furthermore, the FoR-Guard is compared with state-of-the-art detection mechanism of LFA by controller processing time. It founds that by employing forensic-based investigation method (FoR-Guard) the processing time of the controller is reduced significantly. Results show that FoR-Guard reduces the controller processing time up to 30.03 microseconds as compared to 89.94 and 68.49 microseconds of TopoGuard and Sphinx for 10 switches, having 20 fake links out of 50 total numbers of links. Different experiments highlight that FoR-Guard utilize maximum 35 microseconds to detect up to 20 fake links in any network topology which is significant as compared to TopoGuard and Sphinx controller processing time. Hence, the FoR-Guard provides an efficient, comprehensive forensic-based solution for SDN.

ABSTRAK

Software Defined Networking (SDN) adalah satu rangkaian seni bina yang mempunyai ciri unik iaitu nyahgandingan satah data dari satah kawalan. SDN juga membolehkan darjah penglihatan rangkaian yang luas terhadap aplikasi yang beroperasi di atas pengawal dengan melaksanakan modul topologi penemuan. Walau bagaimanapun, darjah penglihatan pengawal dapat dieksploitasi kerana seluruh rangkaian dikawal secara berpusat. Selain itu, topologi SDN juga terdedah pada pelbagai serangan kerana kekurangan ciri-ciri keselamatan di dalam pengawal dan tidak dipertimbangkan di awal pembangunan SDN. Oleh itu, pengawal menjadi sasaran utama untuk mengeksploitasi SDN. Contohnya, pengawal tidak mempunyai satu mekanisme pengesanan untuk membezakan antara Link Layer Protocol Discovery (LLDP) paket yang sah atau tidak. Paket LLDP digunakan oleh modul topologi penemuan untuk menentukan hubungan antara suis yang membantu pengawal untuk membina topologi rangkaian. Rangkaian topologi yang sah adalah penting di dalam SDN untuk mengelakkan suntikan pautan palsu diantara suis bagi memalsukan topologi rangkaian. Fabrikasi pautan palsu dalam topologi rangkaian dipanggil Link Fabrication Attack (LFA). LFA berlaku kerana terdapat suis dan host hasad yang memalsukan paket LLDP untuk menjana pautan palsu di antara suis. Pautan palsu tersebut digunakan untuk melaksanakan aktiviti yang tidak sah seperti mencuri dengar, mengalihkan lalu lintas yang sah, dan menggugurkan paket. Pada masa ini, teknik yang sedia ada hanya dapat mengesan pautan palsu, tetapi gagal untuk mengenal pasti punca serangan tersebut. Oleh itu, SDN memerlukan kaedah forensik untuk mengesan pautan palsu dan terutama sekali menentukan sumber sebenar pautan palsu. Oleh itu, satu kaedah siasatan berdasarkan forensik (FoR-Guard) untuk mengesan pautan palsu serta menentukan sumber sebenar LFA dicadangkan. FoR-Guard terdiri dari tiga fasa utama iaitu pencetus, Pengesanan dan Mengenalpasti Sumber (DeSI), dan fasa pengesanan. Fasa pencetus mengeluarkan mesej penggera bagi fasa DeSI untuk

memerhatikan sekiranya terdapat rangkaian palsu dihasilkan antara suis. Fasa pencetus menggunakan Malicious Index Record (MIR) untuk mengeluarkan mesej. Fasa DeSI menyiasat sekiranya terdapat pautan palsu antara suis dengan memeriksa Direction Link Communication (LCD) dan maklumat MIR bagi pautan dan suis yang berkenaan. Selepas itu, satu mekanisme mengenal pasti digunakan untuk mencari punca serangan dengan menentukan host hasad yang bersambung dengan suis. Fasa pengesahan menentusahkan punca sebenar serangan dengan mengukur kebarangkalian dan entropi. FoR-Guard dibandingkan dengan mekanisme pengesanan terkini oleh LFA dari segi masa pemprosesan pengawal. Penggunaan kaedah siasatan berdasarkan forensik (FoR-Guard) didapati mengurangkan masa pemprosesan pengawal dengan ketara. Hasil kajian mendapati FoR-Guard mengurangkan masa pemprosesan pengawal sehingga 30.03 mikrosaat berbanding TopoGuard dan Sphinx iaitu 89.94 dan 68.49 mikrosaat bagi 10 suis yang mempunyai 20 pautan palsu dari jumlah 50 pautan. Dari kajian yang berbeza, FoR-Guard memerlukan maksimum 35 mikrosaat untuk mengesan 20 pautan palsu di dalam topologi rangkaian berbanding masa pemprosesan pengawal yang diperlukan oleh TopoGuard dan Sphinx. Oleh itu, FoR-Guard menyediakan penyelesaian forensik yang komprehensif dan cekap bagi SDN.

ACKNOWLEDGEMENTS

First of all, I am thankful to Almighty Allah for bestowing me the strength, wisdom, and endless blessings during my PhD study.

I am deeply indebted to my supervisors, Prof. Dr Abdullah Gani (Supervisor) and Dr Ainuddin Wahid Abdul Wahab (Co-Supervisor) for their invaluable guidance, supervision, and encouragement to me throughout this research. Their continuous guidance and support assisted me conducting a valuable piece of research reported in this thesis. They provided me with the opportunity to broaden my professional experience and prepare me for future challenges. The countless efforts of my supervisors encourage me to work hard to achieve my milestones in a defined time limit.

I would like to express my sincerest gratitude and appreciation to my family for their endless love and support during my life especially my parents and spouse. Without their moral support, this dissertation would never have been completed. No words can express my feelings how grateful I am to my parents and siblings for all of the sacrifices that they have made on my behalf so I dedicate the highest achievement of my student life to them. I would like to express my deep appreciation to my dear lab friends, who provided so much support and encouragement throughout this research and studies process. I wish them all the best in their future undertaking.

Finally, I would like to thank Bright Sparks Unit of the University of Malaya for offering me a full research scholarship throughout my doctoral study. I would also like to thank Faculty of Computer Science and Information Technology in helping me in all sorts of matters during my study.

TABLE OF CONTENTS

ABSTRACT	iii
ABSTRAK	v
ACKNOWLEDGEMENTS	vii
TABLE OF CONTENTS	viii
LIST OF FIGURES	xi
LIST OF TABLES	xiii
LIST OF ACRONYMS	xv
CHAPTER 1: INTRODUCTION	1
1.1 Introduction	1
1.2 Motivation	4
1.3 Statement of the Problem	7
1.4 Statement of Objectives	8
1.5 Proposed Methodology	9
1.6 Layout of Thesis	10
CHAPTER 2: LITERATURE REVIEW	14
2.1 Background	14
2.1.1 Software Defined Networks (SDN)	14
2.1.2 Threats to Software Defined Networks planes	17
2.1.2.1 Attacks on the SDN infrastructure plane	18
2.1.2.2 Attacks on the SDN control plane	19
2.1.2.3 Attacks on the SDN application plane	20
2.1.2.4 Attacks on the SDN interfaces	21
2.2 Topology Discovery	24
2.2.1 Importance	24
2.2.2 Topology Discovery in Traditional Networks	25
2.2.3 Topology Discovery in SDN	27
2.3 A Thematic Taxonomy: Topology Discovery	28
2.3.1 Discovery Entities	29
2.3.1.1 Host Discovery	30
2.3.1.2 Switch Discovery	31
2.3.1.3 Inter-connected link between switches	31
2.3.2 Controller Platform	35
2.3.2.1 Topology Discovery in Single Controller Platforms	35
2.3.2.2 Topology Discovery in Multiple Controller Platforms	36
2.3.3 Dependent Services	37
2.3.3.1 Routing	37
2.3.3.2 Mobility Tracking	38
2.3.3.3 Load Balancing	38
2.3.3.4 Topology-based Slicing	39
2.3.4 Objective	40
2.4 Topology Discovery Threats and Solutions	42
2.4.1 Attack Entity	43
2.4.1.1 Host-based Attacks	43
2.4.1.2 Switch-based Attacks	44
2.4.2 Controller Vulnerabilities	45
2.4.2.1 Host Tracking Systems (HTS)	45
2.4.2.2 Link Discovery Procedure	46
2.4.3 Current solutions	47
2.4.3.1 SPHINX	47
2.4.3.2 TopoGuard	48

2.4.3.3	Authentication of LLDP Packets.....	50
2.4.3.4	OFDPv2.....	50
2.4.4	Miscellaneous Threats.....	53
2.4.4.1	Man-in-the-middle.....	53
2.4.4.2	Denial of service.....	54
2.4.4.3	Identity spoofing.....	54
2.4.4.4	Repudiation.....	55
2.5	Future Challenges and Directions.....	56
2.5.1	Multiple SDN Domains.....	56
2.5.2	Topology Discovery through OF switches.....	57
2.5.3	Identification of fake links.....	58
2.5.4	Frequent migration.....	58
2.5.5	Topology discovery information safety.....	59
2.5.6	Controller upgrade.....	61
2.6	Conclusion.....	62
CHAPTER 3: PROBLEM ANALYSIS		64
3.1	Formal Methods.....	64
3.1.1	Formal Definition.....	64
3.1.2	HOL Syntax and semantics.....	65
3.2	Formal Representation of SDN.....	66
3.3	Formal Representation of SDN as a Service.....	70
3.4	Formal Representation of Network Vulnerability.....	72
3.5	Contextual Analysis of LFA Attack in SDN.....	74
3.5.1	Link Fabrication Attack in SDN.....	74
3.5.2	Formal Representation of Link Fabrication Attack.....	75
3.6	Link Fabrication Attack: An Illustration.....	77
3.6.1	Satisfiability Test.....	79
3.7	Performance metrics of Link Fabrication Attack in SDN.....	80
3.7.1	Effectiveness (Processing time).....	80
3.7.2	Sensitivity.....	81
3.7.3	Specificity.....	81
3.7.4	False Alarm Rate.....	82
3.8	Conclusion.....	83
CHAPTER 4: FORENSIC INVESTIGATION METHOD (FOR-GUARD)		84
4.1	Overview of FoR-Guard.....	84
4.2	Design principle of FoR-Guard.....	86
4.2.1	FoR-Guard states.....	87
4.2.2	FoR-Guard Events.....	88
4.3	Forensic Investigation Method (FoR-Guard).....	89
4.3.1	Trigger Phase.....	89
4.3.2	Detection and Source Identification (DeSI) Phase.....	96
4.3.3	Validation Phase.....	105
4.4	Conclusion.....	107
CHAPTER 5: EVALUATION.....		109
5.1	Experimental tools and SDN controller.....	109
5.1.1	Mininet.....	110
5.1.2	Floodlight Controller 1.0.....	111
5.1.3	Packet Tool.....	113
5.2	Performance Analysis.....	114
5.2.1	Confidence Interval of the Data Collection.....	114
5.2.2	Processing Time.....	116
5.2.2.1	Data collection of trigger phase processing time.....	117

5.2.2.2	Data collection of DeSI phase processing time.....	118
5.2.2.3	Data collection of validation phase processing time.....	119
5.2.3	Sensitivity.....	119
5.2.4	Specificity.....	123
5.2.5	False Alarm Rate.....	123
5.2.6	Data collection for sensitivity, specificity, and false alarm rate.....	123
5.2.6.1	Data collection of 10 switches and 50 total links.....	124
5.2.6.2	Data collection of 10 switches and 75 total links.....	127
5.2.6.3	Data collection of 20 switches and 50 total links.....	129
5.2.6.4	Data collection of 20 switches and 75 total links.....	135
5.3	Performance analysis of FoR-Guard with existing solutions.....	139
5.4	Conclusion.....	141
CHAPTER 6: RESULTS AND DISCUSSIONS.....		143
6.1	Performance Analysis of FoR-Guard.....	143
6.1.1	Trigger phase processing time.....	143
6.1.2	DeSI phase processing time.....	145
6.1.3	Validation phase processing time.....	147
6.2	Comparison of FoR-Guard phases in different scenarios.....	149
6.3	Performance analysis of FoR-Guard through ROC analysis.....	153
6.3.1	ROC graphs for 50 total number of links.....	154
6.3.2	ROC graphs for 75 total number of links.....	158
6.4	Comparison of FoR-Guard with existing solutions.....	162
6.4.1	Processing time.....	162
6.4.1.1	Empirical results of 10 switches and 50 total links.....	162
6.4.1.2	Empirical results of 10 switches and 75 total links.....	164
6.4.1.3	Empirical results of 20 switches and 50 total links.....	165
6.4.1.4	Empirical results of 20 switches and 75 total links.....	167
6.5	Conclusion.....	168
CHAPTER 7: CONCLUSION.....		170
7.1	Achievement of research objectives.....	170
7.2	Contribution of the research.....	172
7.2.1	Thematic taxonomy.....	172
7.2.2	Formal representation of SDN.....	173
7.2.3	Dynamic trigger mechanism for fake links.....	173
7.2.4	Algorithms for detection and source identification of fake links.....	173
7.2.5	Validation of source identification.....	174
7.2.6	FoR-Guard Evaluation.....	174
7.2.7	Accepted journal articles from thesis.....	174
7.2.8	Accepted journal articles in other research areas.....	175
7.2.9	Accepted journal articles in collaboration with group members.....	175
7.3	Research scopes and limitations.....	176
7.4	Future work.....	177
REFERENCES.....		179

LIST OF FIGURES

Figure 1.1: SDN Market Forecast	6
Figure 1.2: Outline of the Thesis.....	13
Figure 2.1: General SDN Layer Architecture	16
Figure 2.2: A Flow Chart for <i>Packet_In</i> Message	17
Figure 2.3: Classification of attacks on SDN Planes	22
Figure 2.4: A thematic taxonomy of topology discovery in SDN	29
Figure 2.5: The LLDP process in SDN environment.....	33
Figure 2.6: Single controller architecture.....	36
Figure 2.7: Multiple controllers	36
Figure 2.8: A diagram illustrating link fabrication attacks	40
Figure 2.9: Classification of topology discovery threats and solutions	44
Figure 3.1: A fake link between switches in SDN.....	75
Figure 4.1: The integration of proposed modules within SDN architecture	86
Figure 4.2: The state transition diagram of FoR-Guard.....	89
Figure 4.3: An illustration of a fake link between switches.....	98
Figure 5.1: Python code for a single switch and ‘k’ number of hosts.....	111
Figure 5.2: Floodlight controller supports communication with there is a single link between OF Island and non-OF Island	112
Figure 5.3: The connection of remote controller to the Mininet infrastructure	113
Figure 5.4: Floodlight controller does not support communication with there is a loop between OF Island and non-OF Island	113
Figure 6.1: Processing time of trigger phase for 10 switches and (50,75) total links... 144	144
Figure 6.2: Processing time of trigger phase for 20 switches and (50,75) total links... 145	145
Figure 6.3: Processing time of DeSI phase for 10 switches and (50,75) total links 146	146
Figure 6.4: Processing time of DeSI phase for 20 switches and (50,75) total links 147	147
Figure 6.5: Processing time of validation phase for 10 switches and (50,75) total links	148
Figure 6.6: Processing time of validation phase for 20 switches and (50,75) total links	149
Figure 6.7: Processing time comparison of FoR-Guard phases in 10 switches and 50 total links	150
Figure 6.8: Processing time comparison of FoR-Guard phases in 10 switches and 75 total links	151
Figure 6.9: Processing time comparison of FoR-Guard phases in 20 switches and 50 total links	152
Figure 6.10: Processing time comparison of FoR-Guard phases in 20 switches and 75 total links.....	153
Figure 6.11: ROC graph for single fake link in 50 total links.....	155
Figure 6.12: ROC graph for 05 fake links in 50 total links	156
Figure 6.13: ROC graph for 10 fake links in 50 total links	157
Figure 6.14: ROC graph for 15 fake links in 50 total links	158
Figure 6.15: ROC graph for 20 fake links in 50 total links	158
Figure 6.16: ROC graph for single fake link in 75 total links.....	159
Figure 6.17: ROC graph for 05 fake links in 75 total links	160
Figure 6.18: ROC graph for 10 fake links in 75 total links	161
Figure 6.19: ROC graph for 20 fake links in 75 total links	161
Figure 6.20: ROC graph for 20 fake links in 75 total links	162
Figure 6.21: Comparison of processing time of FoR-Guard with TopoGuard and Sphinx in 10 switches and 50 links	163

Figure 6.22: Comparison of processing time of FoR-Guard with TopoGuard and Sphinx in 10 switches and 75 links	165
Figure 6.23: Comparison of processing time of FoR-Guard with TopoGuard and Sphinx in 20 switches and 50 links	166
Figure 6.24: Comparison of processing time of FoR-Guard with TopoGuard and Sphinx in 20 switches and 75 links	167

University of Malaya

LIST OF TABLES

Table 2.1: SDN layers/Interfaces possible attacks and existing solutions	23
Table 2.2: Comparison between traditional network and SDN topology discovery.....	28
Table 2.3: Topology-dependent applications with its effected attacks	40
Table 2.4: A general description of state-of-the-art topology discovery	51
Table 2.5: A side effect of threats on topology discovery	55
Table 2.6: A description of future challenges and directions of topology discovery with its possible solutions	59
Table 3.1: The logical symbols used in HOL	66
Table 4.1: Symbol descriptions of FoR-Guard algorithms	92
Table 5.1: Python code for a single switch and ‘k’ number of hosts	111
Table 5.2: Network topology setup for our experiments	117
Table 5.3: Processing time of the controller in the trigger phase.....	119
Table 5.4: Processing time of the controller in the DeSI phase	120
Table 5.5: Processing time of the controller in the Validation phase	121
Table 5.6: Data collection for switches=10, total links=50, Legitimate links=49, fake links=01	124
Table 5.7: Data collection for switches=10, total links=50, Legitimate links=45, fake links=05	125
Table 5.8: Data collection for switches=10, total links=50, Legitimate links=40, fake links=10	1256
Table 5.9: Data collection for switches=10, total links=50, Legitimate links=35, fake links=15	1267
Table 5.10: Data collection for switches=10, total links=50, Legitimate links=30, fake links=20	1288
Table 5.11: Data collection for switches=10, total links=75, Legitimate links=74, fake links=01	128
Table 5.12: Data collection for switches=10, total links=75, Legitimate links=70, fake links=05	130
Table 5.13: Data collection for switches=10, total links=75, Legitimate links=65, fake links=10	130
Table 5.14: Data collection for switches=10, total links=75, Legitimate links=60, fake links=15	131
Table 5.15: Data collection for switches=10, total links=75, Legitimate links=55, fake links=20	131
Table 5.16: Data collection for switches=20, total links=50, Legitimate links=49, fake links=01	132
Table 5.17: Data collection for switches=20, total links=50, Legitimate links=45, fake links=05	133
Table 5.18: Data collection for switches=20, total links=50, Legitimate links=40, fake links=10	1334
Table 5.19: Data collection for switches=20, total links=50, Legitimate links=35, fake links=15	134
Table 5.20: Data collection for switches=20, total links=50, Legitimate links=30, fake links=20	1355
Table 5.21: Data collection for switches=20, total links=75, Legitimate links=74, fake links=01	136
Table 5.22: Data collection for switches=20, total links=75, Legitimate links=70, fake links=05	136

Table 5.23: Data collection for switches=20, total links=75, Legitimate links=65, fake links=10.....	1377
Table 5.24: Data collection for switches=20, total links=75, Legitimate links=60, fake links=15.....	137
Table 5.25: Data collection for switches=20, total links=75, Legitimate links=55, fake links=20.....	138
Table 5.26: Comparison of processing time in 10 switches and 50 total links.....	139
Table 5.27: Comparison of processing time in 10 switches and 75 total links.....	140
Table 5.28: Comparison of processing time in 20 switches and 50 total links.....	141
Table 5.29: Comparison of processing time in 20 switches and 75 total links.....	141
Table 6.1: Paired sample t-tests for 10 switches and 50 links.....	164
Table 6.2: Paired sample t-tests for 10 switches and 75 links.....	165
Table 6.3: Paired sample t-tests for 20 switches and 50 links.....	167
Table 6.4: Paired sample t-tests for 20 switches and 75 links.....	168

University of Malaya

LIST OF ACRONYMS

A-TrgM	:	Adaptive Trigger Manager
A ^T	:	Attack Target
AT	:	Attacker
AV	:	Attack Victim
BDDP	:	Broadcast Domain Discovery Protocol
BER	:	Bayesian error rate
BPDU	:	Bridge Protocol Data Units
CI	:	Confidence Interval
DeSI	:	Detection and Source Investigation
EM	:	Entropy Measurement
ES	:	Standard Error
FAR	:	False Alarm Rate
FN	:	False Negative
FP	:	False Positive
HMAC	:	Hash Message Authentication Code
HOL	:	High Order Logic
HTS	:	Host Tracking System
IoT	:	Internet of Things
LCD	:	Link Communication Direction
LFA	:	Link Fabrication Attack
LH	:	Link Handler
LLDP	:	Link Layer Discovery Protocol
LM	:	Link Manager
LSA	:	Link State Advertisements
MIB	:	Management Information Base
MLA	:	Malicious Link Arbitrator
OF	:	Open Flow
OLSR	:	Optimized Link State Routing
OSPF	:	Open Short Path First
SDN	:	Software Defined Networks
TC	:	Topology Controller
TCP	:	Transmission Control Protocol
TLS	:	Transport Layer Security
TLV	:	Type Length Value
UDP	:	User Datagram Protocol

CHAPTER 1: INTRODUCTION

This chapter provides a synopsis of the research work conducted in this thesis. The research work is focusing on the forensic investigation of Link Fabrication Attack (LFA) in Software Defined Networks (SDN). The overview of the work is explained in several sub-sections as follows. Section 1.1 provides the introduction of the chapter. Section 1.2 discusses the motivation of the research work and the statement of the problem is presented in Section 1.3 by emphasising on LFA in SDN. Section 1.4 describes the research objectives, and Section 1.5 presents a brief description of the research methodology to achieve the defined objectives. Finally, Section 1.6 provides a layout of the thesis.

1.1 Introduction

The SDN is a new emerging network architecture which decouples the network control and forwarding functions of the network devices (S. Khan, Gani, Wahab, Guizani, & Khan, 2016). SDN offer programmers an abstract network visibility and direct control over the underlying switches through logically-centralized SDN controller. The dynamic, adaptive, manageable, and cost-effective architecture of SDN provides an ideal situation for applications with dynamic and high bandwidth nature to execute efficiently. Currently, SDN is addressing the computing trends that need to be driven for a new network paradigm. The value proposition of SDN encompasses new capabilities, including rapid deployment of network functions at software speed rather than firmware product cycles or hardware and seamless integration of the network with IT processes through programmable APIs.

SDN enables an insight of the entire network as opposed to the traditional networks due to disaggregation of its architecture. The architecture of SDN is divided into three planes, namely infrastructure plane, control plane, and application plane (B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, & T. Turletti, 2014). An infrastructure plane

is a bottommost plane in SDN architecture which contains the network devices used to forward the data to the entire network. Usually, Open Flow (OF) switches are used in the infrastructure plane to forward the network flows through flow rules installed in the flow tables (McKeown et al., 2008). These flow rules in the OF switches apply to all OSI layers which forward the data to its destination.

The control plane provides an abstract view of the network infrastructure to the applications running on top of the controller (Kreutz, Ramos, & Verissimo, 2013). The controller is a logical entity of the control plane that acts as a brain in SDN. The role of the controller is to receive instructions from the application plane and relays them to the network devices down in the infrastructure plane. The controller used various programmable interfaces to interact with infrastructure and application planes. The well-known interfaces include south and north interfaces. The former interface is used to integrate infrastructure and control planes, whereas latter integrates control and application planes.

The application plane contains different services to manage network resources through specific requirements and used the controller as an intermediate entity (Haleplidis et al., 2015). These services include business applications, network management, and analytics mechanism to operate large data center networks usually in the cloud computing. For instance, network analytics application might be built to analyze the suspicious activity of the attacker for security purposes.

Although SDN layer architecture offers several advantages to the entire network (such as abstraction, flexibility, high asset usage, and low cost), they are susceptible to exploitation through different security breaches. It is because security was not the major concern in the initial development of SDN architecture. The attack on the infrastructure plane can be performed either from outside or inside the network. An attacker could gain unauthorized access to the network devices to exploit the victim connected to SDN by

destabilizing the network operation. The attacks can be in the form of Denial of Services (DoS), installation of malicious flows, eavesdropping flows, and disturbance of the routing paths (Kreutz et al., 2013). Moreover, the control plane is the most targeted location for attackers due to centralized control of the controller. Once the attacker gets an illegal control of the controller, the rest of the network can be easily affected. Moreover, the attacker might use an illegal access to perform numerous malicious behaviors such as transfer the spoof control messages between south or northbound interfaces, bypass the security policies, consume the controller resources, perform network diversion, and damage the controller network visibility. The attack on the application plane could be in the form of malicious services which bypass the security measures of the northbound-API. Once the malicious application bypass the northbound API's and reaches the controller, it easily exploits the core functionality of the controller such as topology discovery, routing services, and load balancing.

Topology discovery is an important function of the controller which provides a network visibility to the topology-dependent applications running on top of the controller (S. Khan et al., 2016). The topology discovery information help applications to execute their operation by having an abstract view of the underlying network infrastructure at the infrastructure plane of SDN. The topology information is managed by the controller, which is a key target for the attackers. Securing the topology information is important to prevent the attackers from exploiting the controller network visibility which further distracts the network applications. There are various ways to make the topology information malicious by modifying the network entities, including switches, hosts, and links between the switches. An illegal modification to the network entities will change the network topology by providing an incorrect view of the network to the controller. The illegal modification causes the topology to change falsely, and such a phenomenon is

known as topology poisoning. Onwards, our main focus will be on the link modification which causes the topological poisoning attacks in SDN.

One way to modify the network topology illegally is to inject fake links between the switches (Dhawan, Poddar, Mahajan, & Mann, 2015; Hong, Xu, Wang, & Gu, 2015). Such a malicious mechanism is known as LFA. The LFA is performed through spoofed Link Layer Discovery Protocol (LLDP) packets. The spoofed LLDP packets are generated from malicious hosts through malicious switches. It happens because there is no mechanism available in the controller to provide authentication for the LLDP packet. The controller receives LLDP packets from the malicious switches and updates the link information in the network topology by assuming the packets arrived from the trusted switches. Thus, LFA generates incorrect abstraction of the network to the controller that further propagates the wrong information to the applications executing on top of the controller. These applications include routing, load balancing, mobility tracking, and topology-based slicing. The execution of applications based on the spoofed topology information results with the disturbances in SDN.

Conclusively, SDN requires a forensic mechanism to detect and investigate the fake links in the network topology. However, due to large data centers containing numerous switches, frequent migration of the network instances, customized user system demands, and lack of trustworthiness, it is highly challenging to detect and investigate fake links among the switches in SDN. This research focuses on the forensic method to detect and investigate the fake links injected by the malicious host through malicious switches to poison the network topology in SDN.

1.2 Motivation

In today's world, SDN has transformed the fixed and predictable traditional network architecture to an agile and flexible architecture without increasing the cost significantly. The architectural approach of SDN delivers the output to the market with minimal

operating expenses and improves the time span of the network management. A survey conducted by Information Week regarding SDN in July 2012, interviewed 250 business technology professionals and reported that 29% of the data center cost will be reduced by adopting SDN (Jim, 2012). The report also highlights that IT organization can implement SDN for cost saving and further utilize its capabilities. Currently, IT organizations are frequently migrating towards SDN due to its cost-effective, flexible, and programmable features. Juniper which is the industry leader in the network innovation had announced in mid-2014 that 52.5 % of US businesses intend to adopt SDN-enabled technology and 74% of them are planning to adopt SDN by next year, i.e., 2015 (Hamel, 2014). The market report “*SDxCentral SDN and NFV Market Size*” published by SDxCentral in 2015 predicts that the combined revenue of SDN, network function virtualization, and next-generation networking paradigms will exceed US-\$105B per annum by 2020 as shown in Figure 1.1 (SDxCentral, 2015).

However, despite the increasing interest and progress of SDN, there are various challenges especially related to the security threats, organizations should be aware of it before adopting the SDN paradigm. The security was not the main characteristics in the early development of the SDN architecture. The security parameter should be widely adopted in SDN to provide integrity, privacy, and availability to all connected resources and information. The security features of SDN should not be limited to the upper level (services) but extended to the foundation (bottom level architecture) as well to have an error-prone environment. According to the *Juniper report* (Hamel, 2014), 34 % of 400 IT decision makers expressed that security is one of the biggest hurdles for them to adopt SDN. Another study reports that security in SDN is rising due to the increasing demand for cloud infrastructure, extended enterprise campuses, and IoT (T. SDxCentral, 2015). The report published by *Markets and Markets*; a forecast that cybersecurity spending will increase up to US-\$170 billion by 2020 (Rohan, 2016). It concludes that the security

threats are not only posed to SDN. Instead, but it is also a major challenge for all IT market today. Therefore, most organizations spend more time in the research phase to adopt security solution for their requirements (T. SDxCentral, 2015). Similarly, an organization requires more time to evaluate and test SDN security solutions before developing a proof-of-concepts. On the other hand, many organizations are expecting a great potential and benefits in a secure SDN. The report (Jim, 2012) indicates that SDN can benefit the network security in applying unified security policy (44%), data security (32%), granular access control (29%), security controls for application security (28%), packet inspection (22%), malware filtering (22%), application-aware security (22%), security appliance performance (17%), and DoS mitigation (17%).

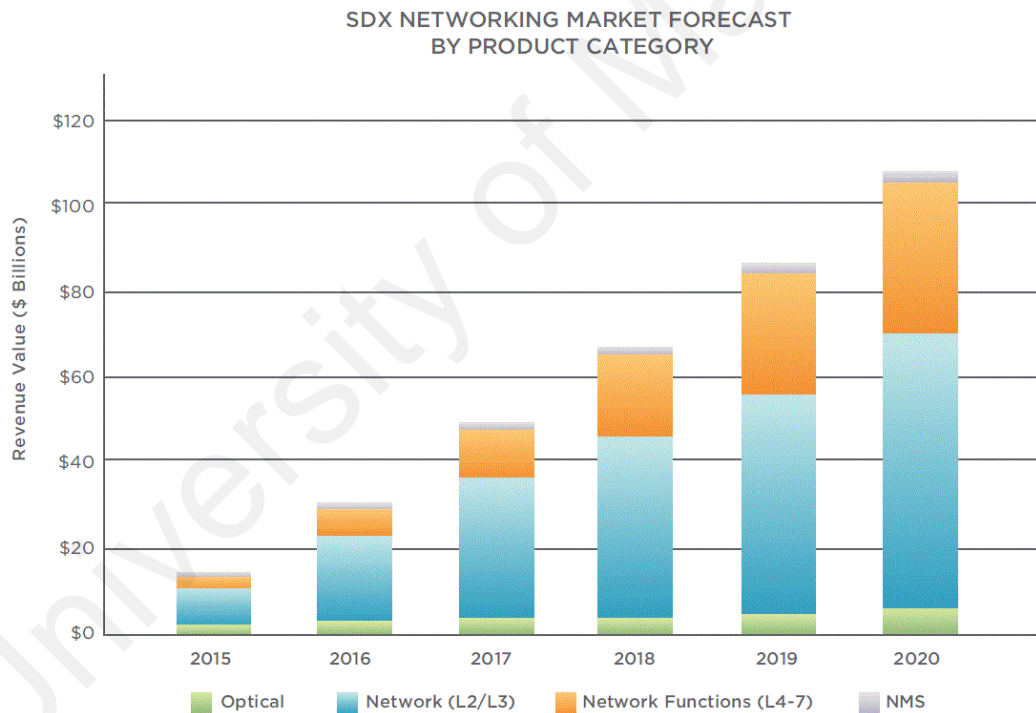


Figure 1.1: SDN Market Forecast

Although academia and industry have conducted much research in protecting SDN from threats, a solution to completely stop adversaries from their malicious behaviors has yet to be found. Among the threats, LFA is the most lethal attack in SDN as it inserts fake links between the switches to affect the visibility of the controller. The visibility of the controller is important to provide an abstraction to the topology-dependent applications for efficient execution. The controller visibility of the entire network is the unique feature

embedded in SDN. Therefore, traditional topology poisoning attack solutions failed to fit in SDN. Hence, there is a need for a solution to address the LFA in SDN. The solution should detect the attack at its early stage and determine the real source of the attack. The proposed research presented in this thesis is a step towards preventing the attackers from exploiting the network topology in an emerging network paradigm such as SDN and can become a part of the future SDN security technologies.

1.3 Statement of the Problem

The main function of the controller to work properly is to have correct information about the network topology. The controller discovers the network topology by collecting information about hosts, switches, and the interconnected link between the switches. However, any illegal insertions or deletions of information in the network topology can cause the controller to provide an erroneous abstraction of the network infrastructure to the applications running on top of the controller (Hong et al., 2015). The application produces incorrect outputs based on the falsified visibility of the network provided by the controller. One way to poison the network topology and falsified the network visibility is to insert fake links between the switches (Dhawan et al., 2015). The fake links are different from the legitimate links in SDN which are generated based on the spoofed information crafted inside the LLDP packets. The crafted LLDP packet is embedded in the *Packet_In* message and forwards it to the controller by the switches. The controller receives the message assuming the information in the *Packet_In* message is correct. The fake links are generated in SDN due to lack of packet authentication schemes in the controller and verification of the OF switches. Thus, the controller fails to detect *Packet_In* message that contains either legitimate or fake LLDP packet information.

The situation becomes sophisticated for the controller to have dynamic changes in the network. Today, a dynamic network topology is inevitable due to customer demands, virtualization, and a large choice of resources available to the users in cloud computing.

The network topology changes in seconds and the controller have to cope-up with such a dynamic environment to update its topology accordingly. To update and verify the legitimacy of the network topology in a short time span is a challenging task. The insertion of the topology verification models in the controller will create extra computation and time overhead, which will affect the performance of SDN.

Based on the discussion, we conclude that detecting and investigating the LFA is utmost important in SDN as it has been neglected to date. Therefore, to overcome the LFA, we identified a research problem and addressed it in this thesis.

The network topology enables the controller to assist topology-dependent applications to execute accurately. However, the incorrect network topology affects the controller network visibility by injecting spoofed information during the topology discovery mechanism. One way to indulge the visibility of the controller is to craft an LLDP packet, which generates fake links among the switches. Under such circumstances, applications view the incorrect network topology which leads them to produce falsified outputs.

1.4 Statement of Objectives

This research aims to address the problem of fake links generated between the OF switches to indulge the visibility of the SDN controller. We define the following objectives to be achieved to attain the aim of this research.

- To study and review the recent topology poisoning methods in SDN to gain insight about the LFA and identify their shortfalls.
- To investigate and analyze the impact of fake links in the network topology through high order logic formal methods.
- To propose a forensic investigation method for LFA that is capable of detecting and investigating the attack with less processing overhead of the controller and high detection rate.

- To evaluate and validate the proposed method in the emulation environment and compare the performance with the state-of-the-art LFA proposed detection methods.

1.5 Proposed Methodology

This research study is divided into four main phases to accomplish the defined objectives shown in Section 1.4. Each phase is further divided into several steps to achieve the objectives.

A comprehensive review and synthesis of the current topology discovery methods are undertaken to have an insight about the visibility of the SDN controller. The thematic taxonomy of the topology discovery is devised to have enough knowledge about its research area. The investigation reveals that visibility of the controller can distract through topology poisoning attacks. Through our literature review, we have identified two main entities in SDN which plays a vital role in launching the LFA such as (a) hosts, and (b) switches. In our research, we have focused on the fake links that distract visibility of the SDN controller through the generation of the spoofed LLDP packets from the malicious host through malicious switches.

The research problem is investigated using High Order Logic (HOL). The HOL is a group of formal information representation of a specific domain which helps to establish LFA in SDN environment. To investigate the LFA, we have defined the formal representation of SDN regarding systems liability, service information, dependencies, and switch & host vulnerabilities. Finally, we have represented LFA formally and performed its satisfiability analysis test.

To alleviate the LFA, we propose and implement a forensic investigation method (FoR-Guard). The aim of FoR-Guard method is to detect and investigate the LFA in SDN with high detection accuracy rate and less processing overhead on the controller. The FoR-Guard is divided into three main phases: a) Trigger phase, b) Detection and Source

Identification (DeSI) phase, and c) Validation phase. The FoR-Guard uses the switching behavior to trigger the detection mechanism, Link Communication Direction (LCD) along the switching behavior to perform detection, and Entropy Measurement (EM) to calculate the uncertainty presented in the network topology which further helps the controller to verify the real source of the attack.

The performance of the proposed method is implemented and evaluated in an emulated SDN environment. Different modules are designed and developed for each phase of the proposed method that executes in the Floodlight controller. The experimental data are collected by running the proposed method in different scenarios to evaluate the detection accuracy and processing time of the Floodlight controller.

1.6 Layout of Thesis

This thesis is composed of seven chapters, and each chapter contains a part of our research work conducted to address the research problem. The thesis layout is shown in Figure 1.2 and organized as follows.

Chapter 2: Literature Review

This chapter reviews the research undertaken in the field of topology discovery and state-of-the-art network topology threats in SDN. Also, a comprehensive knowledge about SDN, SDN forensics, and different threats related to SDN layer architecture are described. The thematic taxonomy of topology discovery is devised by considering several aspects to explore the domain knowledge of topology discovery management in SDN. A classification of topology discovery threats is presented to understand threats that affect the network topology. The LFA is comprehensively explained to know about its working steps and exploitation to the network topology. Moreover, we listed the potential threats that can be generated through topology poisoning attack in SDN. Finally, we have identified potential challenges that need important consideration in the future to have effective network discovery in SDN.

Chapter 3: Problem Analysis

This chapter presents a formal analysis of the LFA in SDN using HOL formal methods. The HOL assists in representing the LFA formally. The formal representation of LFA in HOL helps to understand better steps of the attack performed in SDN. We investigate the attack by building the HOL for the entire systems liabilities, service information, dependencies, and switches as well as hosts vulnerabilities. Moreover, we perform satisfiability analysis to verify the formal representation of LFA. Finally, various parameters are selected to analyze the effect of the LFA.

Chapter 4: Forensic investigation method (FoR-Guard)

This chapter proposed a forensic investigation method to determine LFA in SDN. It explains the phases in the proposed method (FoR-Guard) along with the algorithms presented in each phase. The distinct features of the proposed method are also highlighted.

Chapter 5: Evaluation

This chapter describes the techniques used for the experimental setup and data collection method for the evaluation of the proposed method. It also explains the tools used to evaluate the proposed method. Moreover, confidence interval, a statistical technique is used to provide a range of values obtains from the sample statistics that tend to represent the value of an unknown population size of the data collection.

Chapter 6: Results and Discussions

This chapter reports the results obtained from different experiments and analyzed the effectiveness of the proposed method. We compare and contrast our experimental results of the proposed method with the benchmark results of the state-of-the-art methods. The statistical validation is performed to know the significant differences between the FoR-Guard and state-of-the-art detection methods.

Chapter 7: Conclusion

This chapter concludes the thesis by explaining how the research objectives have been achieved. The main contributions of this research work are summarized, and research significance is highlighted. Finally, limitations and future research directions of the proposed method are discussed.

University of Malaya

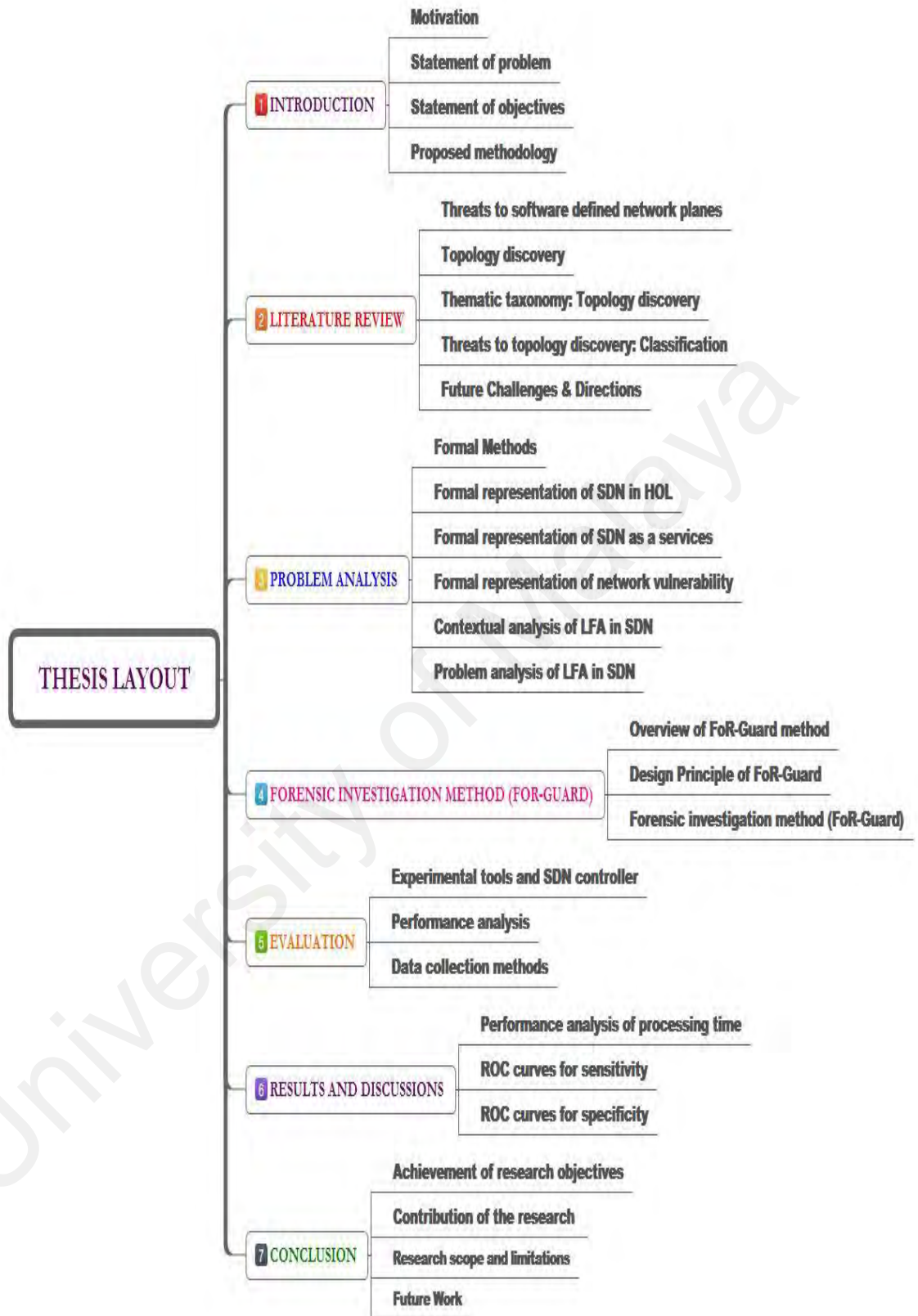


Figure 1.2: Outline of the Thesis

CHAPTER 2: LITERATURE REVIEW

This chapter presents an overview of the security aspect of SDN to elaborate topological vulnerabilities found in the controller. The objective of this chapter is to highlight the significance of source identification in LFA which has been neglected up to date. The topology discovery domain of SDN is reviewed to gain an insight into the problems faced by the controller during the topological poisoning attacks. A thematic taxonomy of topology discovery in SDN is devised based on significant parameters such as discovery entities, controller platform, topology-dependent services, and objectives. Moreover, threats faced by the topology discovery are classified into four categories such as attack entity, controller vulnerabilities, current solutions, and miscellaneous threats.

The remainder of this chapter is as follows. Section 2.1 introduces brief description about SDN, and possible threats occurred in SDN layers architecture. Section 2.2 presents an importance of topology discovery regarding traditional networks and SDN. Section 2.3 devises a thematic taxonomy presenting insight of the topology discovery in SDN. Section 2.4 classifies topology discovery threats and solutions. Section 2.5 highlights future challenges and directions. Finally, Section 2.6 concludes with summarizing the chapter.

2.1 Background

This section presents comprehensive information about the SDN and threats faced to its layered architecture.

2.1.1 Software Defined Networks (SDN)

The widely known separation of the control plane and the forwarding infrastructure plane in SDN is shown in Figure 2.1. This architecture results in numerous benefits, including easy insertion of applications and services, streamlined processes, improved efficiency, reduced complexity, and better user experiences (B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, & T. Turetletti, 2014). The control plane is controlled by

logically centralized controller instead of the conventional control mechanisms present in the border gateway protocol (Stewart III, 1998) and open shortest path first (J. T. Moy, 1998). The centralized control assists network administrator to change the network traffic without re-configuring the network devices dynamically. For instance, the controller can dynamically change the network flow towards high bandwidth channels while observing high delays on low bandwidth network channels without affecting the network operation (Saha, Sambyo, & Bhunia, 2016).

In Figure 2.1, SDN architecture is divided into three main layers/planes, i.e., infrastructure, control, and application plane (Kreutz et al., 2015; Suleman, Abdullah, Ainuddin, Ahmed, & Mustapha, 2016). The infrastructure plane consists of all the network devices that communicate and share information with each other (B. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, & T. Turlitti, 2014). For instance, OF switches forward the packets towards the destination using rules specified by the controller.

The controller (in the control plane) acts as a brain in SDN, which manages the entire network through the logically centralized controller (Civanlar, Lokman, Kaytaz, & Murat Tekalp, 2015),(Shu et al.). The controller has an abstract view of the network topology that assists different applications running on top of the controller in the application plane (Cui, Yu, & Yan, 2016). The application plane is responsible for implementing essential network services (application, algorithms, protocols, and others) through the controller (Zou, Xie, & Yin, 2013). With the given abstract network, the application plane deploys various network applications. These applications include load balancing (Zou et al., 2013), intrusion detection systems (Chung, Khatkar, Xing, Lee, & Huang, 2013), network monitors (Chowdhury, Bari, Ahmed, & Boutaba, 2014), firewalls (H. Hu, Han, Ahn, & Zhao, 2014), and scheduling (L. E. Li, Mao, & Rexford, 2012).

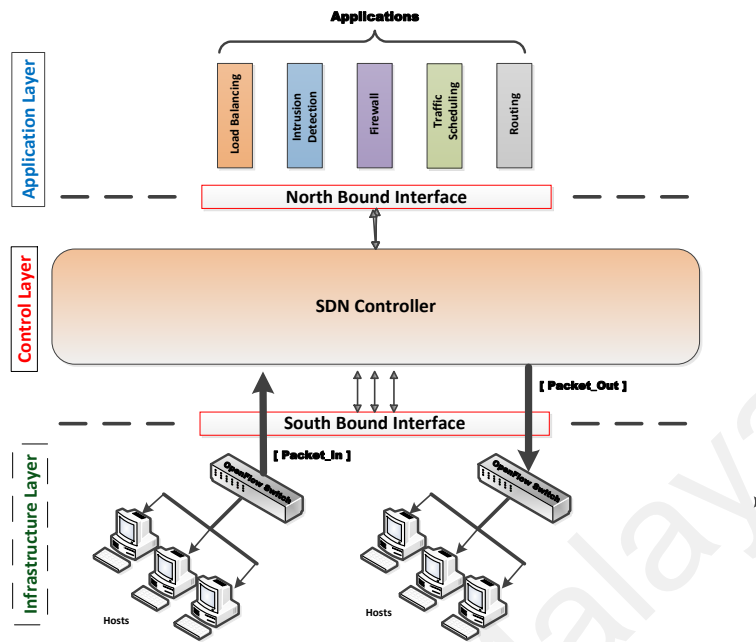


Figure 2.1: General SDN Layer Architecture

The controller has different interfaces/API's to communicate with other planes and network devices such as south, north, east, and westbound API's (Jarraya, Madi, & Debbabi, 2014). The most common API's used in SDN are the southbound and the northbound. The southbound API enables communication between the infrastructure plane network devices and the controller. Initially, when a new packet is received by the OF switch from the host, it checks the matching field of the packet header that match with the flow rules installed in the flow table (Banikazemi, Olshefski, Shaikh, Tracey, & Wang, 2013; Moshref, Bhargava, Gupta, Yu, & Govindan, 2014). If the match is not found, a *Packet_In* message is generated by the OF switch and it is sent to the controller on the southbound API. The controller checks the packet header for the necessary information and replies back to the OF switch through a *Packet_Out* message. The *Packet_Out* message contains the specific rules for the respective network flows which are inserted in the flow table of the OF switch. When a similar type of flow (i.e., the same source and destination) arrives at the OF switch, it is forwarded based on the previously

inserted flow rule in the flow table (Akyildiz, Lee, Wang, Luo, & Chou, 2014). The flow chart of the *Packet_In* message is shown in Figure 2.2.

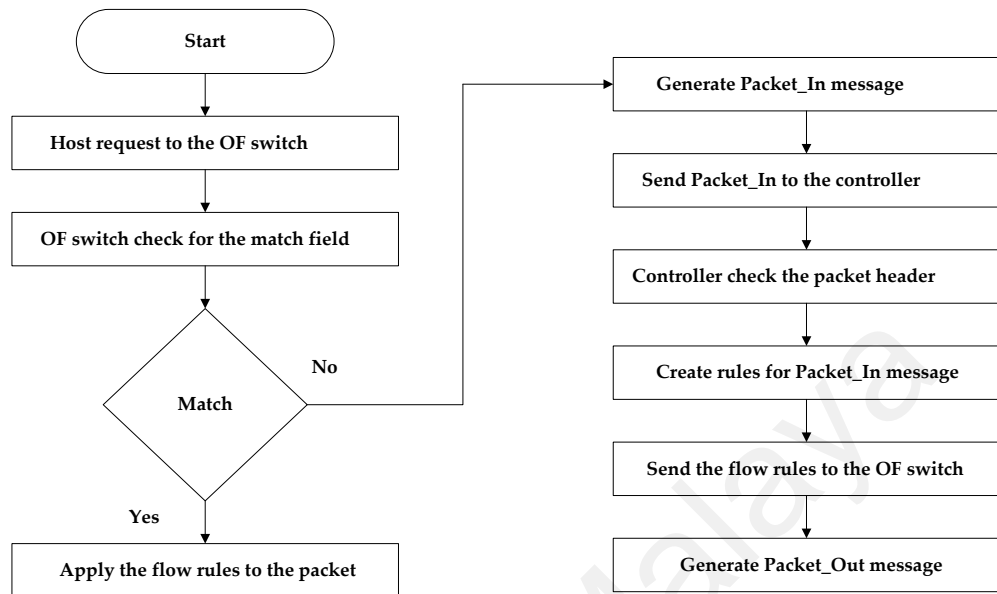


Figure 2.2: A Flow Chart for *Packet_In* Message

Moreover, the controller modifies the packet header information in real-time by modifying source/destination addresses and ports (Feamster, Rexford, & Zegura, 2014). This characteristic of the controller provides flexibility and reliability to the network. Similarly, the northbound API connects the controller to various network applications that deploy algorithms and protocols to operate an SDN (Zhou, Li, Luo, & Chou, 2014). Unlike, the southbound API, the standard northbound API is not available yet, which presents several security threats (Hakiri, Gokhale, Berthou, Schmidt, & Gayraud, 2014; Sharma et al., 2013). The eastbound and westbound API's manage the distributed controllers in SDN (Sezer et al., 2013). That is, multiple controllers can be deployed in SDN to manage different parts of the network (Dixit, Hao, Mukherjee, Lakshman, & Kompella, 2013) due to different assigned functions such as load balancing, monitoring and task allocation.

2.1.2 Threats to Software Defined Networks planes

The centralized control, network abstraction, and software-based network changes attract malicious users to perform attacks in SDN. Attacks can be on the 1) network

devices in the infrastructure plane, 2) control modules in the control plane, 3) applications in the application plane, or 4) different API's in the SDN (Kreutz et al., 2013). In this section, we discuss and classify different attacks as illustrated in Figure 2.3. Moreover, we explain attacks performed on various interfaces of SDN. Table 2.1 illustrates the existing available solutions for each attack in the SDN planes and interfaces.

2.1.2.1 Attacks on the SDN infrastructure plane

There are different ways to exploit the network devices in the infrastructure plane wherein some of these attacks are specific to the SDN while others are inherited from the traditional networks. Each of these attacks is discussed as follows.

(a) Malicious OF Switches:

Forwarding network flows through a malicious OF switch allows it to alter the network packets. In this case, the network flow diverts and the legitimate traffic is dropped, which interrupts the communication between the SDN devices. It slows down the network traffic and may prevent the legitimate switch from receiving the traffic due to an excessive idle time specified for the flow entries in the flow tables. It causes the network packets to be dropped (Kamisiński & Fung, 2015) or generate numerous *Packet_In* messages to the controller due to mismatch at the OF switch.

(b) Malicious hosts in the infrastructure plane:

Malicious hosts can attack any switch and controller in the SDN by generating forged network packets (Antikainen, Aura, & Särelä, 2014). In forged network packets, various fields (such as the IP field, the MAC field or other fields), can be modified to hide the identity of the attacker. Also, a malicious host can generate millions of packets in the form of a Denial-of-Service (DoS) attack to overload the memory of the OF switches (Shin, Yegneswaran, Porras, & Gu, 2013). Similarly, for every new forged packet (i.e., unique source IP address), OF switches generates the *Packet_In* message to the controller

which can result in decreasing the performance of the controller (Tootoonchian, Gorbunov, Ganjali, Casado, & Sherwood, 2012).

2.1.2.2 Attacks on the SDN control plane

The attacker is more interested in the control plane due to its significant function such as the network control, network abstraction, and support to various network applications. There are various types of attacks which can be performed by the controller as follows.

(a) Malicious modules inside the controller:

The integration of core controller functions creates an initial setup of the SDN. For instance, the topology manager stores information regarding devices such as switches and hosts in the network (Syrivelis et al., 2012) and uses the LLDP to discover the interconnected links between the OF switches (Kempf et al., 2012). An attacker can exploit vulnerabilities within these building blocks. As an example, a recent topological poisoning attack (Hong et al., 2015) exploits the link discovery module running in the controller by generating fake links between the switches. As a result, fake links affect the functionalities of the entire network (Klaedtke, Karame, Bifulco, & Cui, 2015).

(b) Compromised controllers:

The controllers can also be distributed which exchanges information from time-to-time to update their states (Phemius, Bouet, & Leguay, 2014). The attacker can exploit the communication among the distributed controllers. For instance, Open Daylight controller uses ODL-SDNi app for distributed communication among multiple controllers. The problem arises when one of the controllers is performing maliciously and shares wrong information among the controllers. To identify the malicious controller among a pool of the controllers is a challenging task due to the isolated functionality of each controller. The malicious controller disseminates incorrect topological updates to another controller to make the network malfunction (H. Li, Li, Guo, & Nayak, 2014).

(c) Attack on management consoles:

The management console allows authorized individuals to access the SDN. An attacker can get unauthorized access to the management console through a brute-force password attack or leaking the password from different sources. Once the attacker gets access to the SDN, attacks can be generated on the controller as well as on different resources of the network. Usually, access to the management console is defined in the policy agent module of the controller. The compromised management console empowers the attacker to create a gateway in launching various other attacks in SDN.

2.1.2.3 Attacks on the SDN application plane

The SDN application plane consists of different applications/software (Braun & Menth, 2014) such as load balancing, routing, firewall, and intrusion detection. These applications/software may be used to monitor the traffic, extract statistical traffic features, apply authentication mechanism to different user domains, and diverts the traffic based on the network load. The application development in the application plane is considered as a dramatic change to an SDN architecture (Qazi et al., 2013). A single network infrastructure can be used by multiple applications at the same time to fulfil their requirements. However, this is not possible in the traditional network where the configuration of network device needs update upon using different network applications. A user can easily develop an application module and embed it in the application plane (Jarschel, Wamser, Hohn, Zinner, & Tran-Gia, 2013). It allows malicious users to affect the entire network. There are various possible attacks in the SDN application plane which are briefly discussed as follows.

(a) Unauthorized access to applications:

Unauthorized access to the applications, help the attackers to bypass the security level of the controller (Akyildiz, Lee, Wang, Luo, & Chou, 2016; Sandra Scott-Hayward, O'Callaghan, & Sezer, 2013). The controller treats all applications as normal network

services because of the absence of a trust mechanism between the application and control layers. The unauthorized access to various applications can inform attackers about the operation of the network which further creates a chance to exploit various parts of the network.

(b) Disclosure of information through the application server:

Once an attacker gains access to the application server, the information of any application that is currently or previously executed in the RAM can be accessed and disclosed. In a traditional network, such kind of attack is called a RAM scraper attack (Saini, Rao, & Panda, 2012). In SDN, an attacker can scan the RAM processes in the application server to gain access to the application information through the northbound API. An attacker can further identify the rules of the controller for various network flows.

(c) Modification of user privileges for application execution

Due to network virtualization, each user can treat the network with its requirement provided with isolation (Blenk, Basta, Reisslein, & Kellerer, 2016; Drutskoy, Keller, & Rexford, 2013). Each user is provided with specific rights to execute different applications according to its requirement. However, if the attacker accesses the application server, the user privileges can be changed to produce the malfunctioning results (Z. Hu, Wang, Yan, Yin, & Luo, 2015).

2.1.2.4 Attacks on the SDN interfaces

The southbound and northbound interfaces are used for the centralized controller environment while the eastbound and westbound interfaces are used in a distributed controller environment respectively. These interfaces are used to send and receive network information which attracts the attackers to eavesdrop the information (Antikainen et al., 2014).

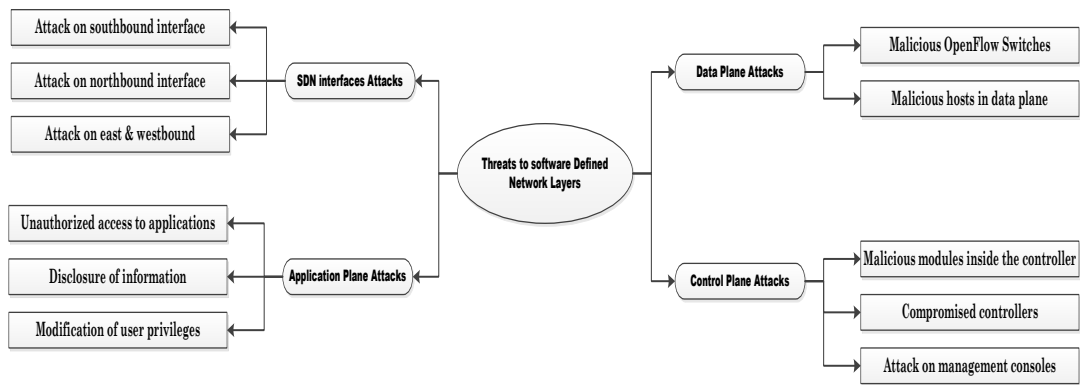


Figure 2.3: Classification of attacks in SDN Planes

(a) Attacks on the southbound interface

Mostly, the southbound interface in SDN uses the standard OF protocol (Nadeau & Gray, 2013). The OF protocol allows communication between the OF switch and the controller. Each OF switch has to communicate with the controller through *Packet_In* message upon reception of the new packets (Monaco, Michel, & Keller, 2013). It makes the southbound interface more suitable for information extraction from the *Packet_In* messages. The attacker can exploit the Transport Layer Security (TLS) vulnerabilities and to take control over the southbound interface (Dover, 2013). Subsequently, the attacker can create, modify, and delete the flow rules. It produces malfunctioning results in the network due to malicious flow entries in the flow table. Moreover, an attacker can generate forged packets to the OF switches with a unique identity to force OF switch to generate a large number of *Packet_In* messages to overload the bandwidth channel used between the OF switches and the controller (Wang, Xu, & Gu, 2014).

(b) Attacks on the northbound interface

The northbound interface is used for communication among the applications of the application plane with the controller (Cho et al., 2015). Unlike the southbound interface, the northbound interface does not use a standard protocol because of its initial stage of development (Hakiri et al., 2014; Sharma et al., 2013). The attacker can use the northbound interface to interfere the communication between the application and the

controller. An attacker can get unauthorized access to the northbound interface and may delete some information which can lead to the falsified output of the application.

Table 2.1: SDN layers/interfaces possible attacks and existing solutions

SDN Layers/Interfaces	Possible Attacks	Existing Solutions	Attack Nature
Infrastructure Plane	Malicious switches	FortNOX, SDNsec	SDN-based attack
	Malicious hosts	VAVE, OFGUARD, FlowVisor	TN-based attack
Control Plane	Malicious modules	VeriCon, FRESCO, SPIRIT	SDN-based attack
	Compromised controllers	Fleet , DISCO, HyperFlow	SDN-based attack
	Management consoles	Sandbox-based system	SDN-based attack
Application Plane	Unauthorized access	PermOF, SDN Rootkits	TN-based attack
	Disclosure of information	Proactive strategies and Randomization	TN-based attack
	Modification of user privileges	OFX	TN-based attack
SDN Interfaces	Southbound interface threats	VeriFlow, HSCS architecture	SDN-based attack
	Northbound interface threats	Dynamic Filtering	SDN-based attack
	Eastbound & westbound threats	DRS	SDN-based attack

Similarly, the attacker can use a malicious application to inform the controller to disconnect other applications leading to a flow rule modification problem. Moreover, the malicious application can send numerous requests to overload the CPU of the controller as well as to occupy the available bandwidth of the northbound interface. Note that, proper authentication and encryption mechanism is not standardized for the northbound

interface. Various APIs for the northbound interface can increase the security threats because of the built-in vulnerabilities. It then decreases the trustworthiness between the controller and various applications.

(c) Attacks on the eastbound & westbound interfaces

The eastbound and westbound interfaces are also prone to various attacks. The information updates through these interfaces can be exploited by the attacker with unauthorized access to the management console. The attacker can take advantage of unencrypted communication of data between the controllers for sharing the network information updates (Dixon et al., 2014). An attacker can also compromise the network by tapping the application to eavesdrop on clear text communications between two controllers.

2.2 Topology Discovery

2.2.1 Importance

The topology management is a unique feature of SDN which allows the controller to facilitate the applications in the application plane (Shenker, Casado, Koponen, & McKeown, 2011). For instance, a routing application uses the network topology to route the network traffic to its destination (Kotronis, Dimitropoulos, & Ager, 2012; Mahdi et al.; Staff, 2016). The controller discovers a topology through (Hong et al., 2015) a) Host discovery, b) Switch discovery, and c) Inter-connected links between the switches. The controller discovers the host by receiving a *Packet-In* message from the switch. The switches are discovered during the initial handshake process with the controller, and inter-connected links between switches are discovered through the Open Flow Discovery Protocol (OFDP). However, there are vulnerabilities found in the core applications of the controller which are exploited to initiate LFA (Zaalouk, Khondoker, Marx, & Bayarou, 2014).

If an attacker poisons the network topology information, its effect will immediately be visible to all its dependent applications (Dhawan et al., 2015). Therefore, it is important to detect a topology poisoning attack at its early stage. Note that, detecting a fake link between the OF switches created by the topology poisoning attack is relatively easy than identifying the source of the attack. Mostly, attackers hide their identity information after they perform the attack (Hwang & Kim). Similarly, in a topology poisoning attack, the attacker creates a fake link between the OF switches by spoofing the LLDP packet to hide its identity (Alharbi, Portmann, & Pakzad, 2015). The controller should be aware of fake links upon their insertion in the network so that the attack can be prevented at its early stage.

2.2.2 Topology Discovery in Traditional Networks

Topology poisoning attacks are not new to traditional networks. The main aim of a topology poisoning attack is to fabricate the network topology and disturb normal network operations regarding control and management (Jajodia, Noel, & O'Berry, 2005). If a malicious router advertises its routing information to its neighbors, it will result in a falsified network traffic distribution based on the malicious routing information (Padmanabhan & Simon, 2003). For instance, a network using the Routing Information Protocol (RIP) protocol allows each router to send its link with an update information of their topological view to its neighbors (Pei, Massey, & Zhang, 2003). The information includes a destination identifier and a cost metric to the destination. However, the information sent by a malicious router can update the neighbor routers link database with the incorrect information and can affect the entire routing process (Mizrak, Cheng, Marzullo, & Savage, 2006). Also, the malicious router may advertise the least cost path to a specific destination, thus causing the traffic to be diverted from other sources to a malicious destination (Suleman Khan, Gani, Wahab, Shiraz, & Ahmad, 2016; Suleman Khan, Shiraz, et al., 2014).

A similar type of attacks can also be performed in the link-state protocol, i.e., OSPF, where every router is bound to send its link update to its neighbors to calculate the optimal path depending on the metrics (Katz, Kompella, & Yeung, 2003). In OSPF, the update link information sent by the router is called Link State Advertisements (LSA). A malicious router may send a false LSA to its neighbors defining other routers by forging their original information (J. Moy, Pillay-Esnault, & Lindem, 2003). It diverts the network traffic towards the malicious router which may forward the packet to the longer path, perform eavesdropping, modify the packet information, and drop some/all the packets in the network flow. Besides the wired networks, topological information can be exploited in the wireless networks as well. For instance, the Optimized Link State Routing (OLSR) is used in mobile ad-hoc networks to discover and disseminate the link-state information throughout the network (Clausen et al., 2003). This information helps nodes to compute the optimal path to the next node in the network to reach the destination.

The OLSR determine and forward the link state information to the neighbor nodes by using hello and topology control messages. These messages can be falsified to disseminate the wrong information and results in a false topological development. Moreover, Bridge Protocol Data Units (BPDU's) in the Spanning Tree Protocol (STP) (Choudhary, 2010) can be forged to exploit the information. Such exploitation can be performed by the attacker to make the malicious switch as a root bridge in the network and therefore gain access to the network traffic. Such type of attack is also called STP mangling (Ornaghi & Valleri, 2003). The STP mangling affects the topology of the network by selecting the incorrect switch as a root bridge. The root bridge has easy access to the network traffic that is costly when a malicious switch is selected as a root bridge in the selection process.

2.2.3 Topology Discovery in SDN

The topology management is a unique characteristic of SDN as compared to traditional networks. Table 2.2 provides a comparison between the traditional network and SDN topology discovery. The decoupling of the control plane from the infrastructure plane enables an SDN to have a logically centralized control of the network (Akyildiz et al., 2016; Kloti, Kotronis, & Smith, 2013). To achieve the centralized control, a controller (responsible to control the network centrally) should have a global visibility of the entire network (Staff, 2016). The controller incorporates various core modules that assist various SDN applications (Mogul et al., 2013). Among the core modules, a topology manager is a module that builds the network topology of the entire SDN infrastructure (Pakzad, Portmann, Tan, & Indulska, 2015). The topology not only facilitates the controller but also assists the application plane services to perform its operation using the network programmability (Aslan & Matrawy, 2016). The network topology is significant to both control plane and application plane because it provides an abstract visibility of the entire network devices.

The OF protocol is a standard approach used for communication between the controller and the OF switches on the southbound interface in SDN (McKeown et al., 2008). The southbound interface carries requests and replies to both the controller and the OF switches. The updated network topology information is significant to the controller in providing efficient control and management of the network. As a result, the efficient topology discovery is considered to be an important characteristic for the controller. Developing a topology of the network requires switch discovery, host discovery, and interconnected switches' discovery (Hong et al., 2015). Each of these discovery mechanisms is briefly explained in Section 2.3.1.

In the work (Pakzad et al., 2015), an efficient topology discovery mechanism is proposed which reduces the topology discovery overhead up to 40 % by minimizing

Packet_Out messages generated from the controller. A single LLDP packet is sent to each of the OF switches rather than the de-facto standard of sending each LLDP packet to each port of the OF switch. The switch broadcasts the LLDP packet to all its active ports which further discovers links between the switches. The work in (Saha et al., 2016) proposes to represent network topology, find loops, and determine alternative paths at the time of link failure in SDN. An adjacency matrix is used to represent the LLDP packets corresponding to the switches in the network. It finds the loops and alternative paths at the time of link failure in SDN. Moreover, the work in (Alharbi et al., 2015) presents the security of topology discovery in SDN and shows that how information can be spoofed to generate fake links in the network topology. Finally, it also presents a countermeasure by using the Keyed-Hash Message Authentication Code (HMAC) authentication.

Table 2.2: Comparison between traditional network and SDN topology discovery

Features	Topology Discovery in Traditional Networks	Topology Discovery in SDN
Host Discovery	NMAP	<i>Packet_In</i> message
Switch Discovery	SNMP	Initial Handshaking process
Link Discovery	Various updates (RIP, OSPF, LSA, OLSR)	LLDP
Control Management	Independent	Controller
Scalability	No. of switches	No. of OF switches
Communication updates	Switch- Switch	Controller-Switch-Controller

2.3 A Thematic Taxonomy: Topology Discovery

In this section, we provide in-depth information about the topology discovery in SDN. We devise a thematic taxonomy of the topology discovery in SDN as illustrated in Figure 2.4. The thematic taxonomy can be used to establish a conceptual knowledge of the topology discovery (Thomas et al., 2016). The taxonomy consists of four main categories including (1) Discovery Entities, (2) Controller Platform, (3) Topology-Dependent

Services, and (4) Objective. These categories provide a clear understanding of the topology discovery in SDN.

2.3.1 Discovery Entities

The controller has a visibility of the entire network topology. To create a topology of the entire network, the controller has to discover network entities and inter-connected links among them. In particular, the controller has to discover three entities for a complete view of the network topology, i.e., a) Hosts, b) Switches and c) Inter-connected links between the switches. The hosts are the physical or virtual systems (virtual machines) connected to the switches that are used by users to execute their services.

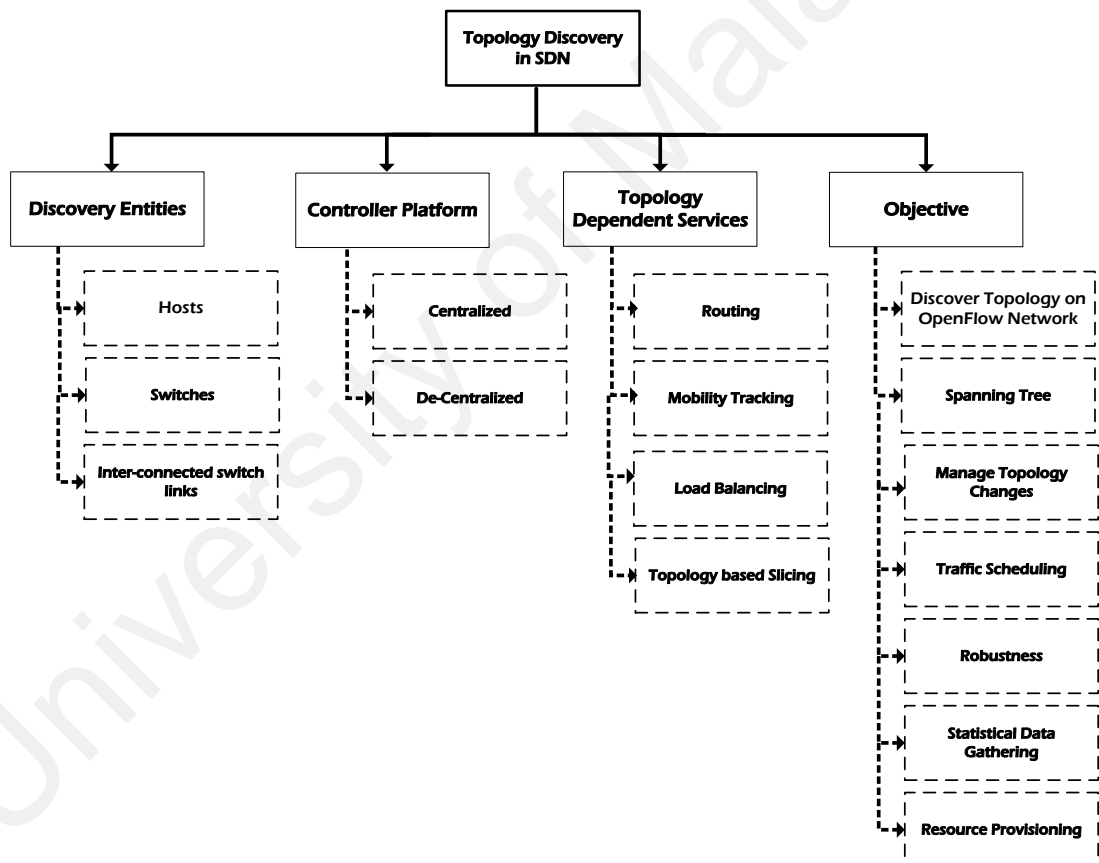


Figure 2.4: A thematic taxonomy of topology discovery in SDN

The switches are known as the OF switches that forward the packets from source to the destination upon receiving flow rules from the controller. The inter-connected links are the physical or the virtual links between the OF switches which are used to transfer

the network packets. Discovery of these entities is significant for the topology management in updating a network topology view of the controller.

2.3.1.1 Host Discovery

The discovery of the host helps the controller in identifying the exact location of the host in the network which allows for traffic monitoring, assisting in traffic routes, and determining the source of the packets (Scott et al., 2014). A host tracking function is available in most of the controllers, which determines the host and its respective port with the switch (Goncalves, Martins, Corujo, & Aguiar, 2014). The host tracking can trace the virtual machine migrations in the data centers, which are difficult if done manually due to their frequent migrations. The controller maintains a host profile table for each of the hosts that join the network.

Similarly, the controller deletes the host profile table when a host leaves the network. To populate the empty host table, the controller uses the *Packet_In* message to generate a host profile table for each host sent by the OF switch. For example, a host attached to a port of the OF switch generates a request message. This request message is encapsulated by the OF switch in the form of *Packet_In* message and it is then sent to the controller. Based on the *Packet-In* message, a controller identifies the identity of the host.

The host profile is built by *Packet-In* message which contains information such as a) IP address, b) MAC address and c) Meta information (DPID, port number, and last timestamp). When a host migrates from one switch to another, its port and switch IDs are changed due to its new location. The controller updates the record for the migrated host based on the *Packet-In* messages received from another OF switch. The payload information in the *Packet-In* message helps the controller to track the location of the host. Different controllers have different host tracking applications to discover hosts in the network (Hong et al., 2015). For instance, the *'hosttracker.cc'* is used in NOX controller,

the *'host_tracker.py'* is used in Ryu controller, the *'DeviceManagerImpl.java'* is used in Floodlight controller, and the *'OFMDeviceManager.java'* is used in OpenIRIS controller.

2.3.1.2 Switch Discovery

Typically, OF switches communicate with the controller on the arrival of new packets, i.e., *Packet_In* messages. The controller replies with a *Packet_Out* message to insert flow entries in the OF switches. The location of the OF switches is vital to the controller due to its frequent two-way communication. The controller discovers the location of the OF switches in its initial handshake process.

The OF switches are discovered in the initial handshaking process by the controller. Once the OF switch is added to the network, the controller gets the existence and key properties of the OF switch. The controller records the MAC address, the number of ports, and other information about the OF switch. There is no requirement for a separate protocol to discover the location of the OF switch in SDN.

2.3.1.3 Inter-connected link between switches

The discovery of interconnected links between the switches is significant to generate a topology by the controller in SDN. The inter-connected links determine the connectivity between the OF switches that helps the controller and the application plane services to utilize the network according to their requirements. In most of the times (if not all), the OFDP is used to discover the inter-connected links between the OF switches. The OFDP uses LLDP to advertise the capabilities and neighbor information of the nodes in the network (Hollander, 2007). The LLDP is usually used in the Ethernet switch, which actively sends and receives LLDP packets to each of its active ports. The extracted information from the LLDP packet is stored in a Management Information Base (MIB) in the switch.

The collected information from different MIB's of the switches via SNMP helps to determine the network topology. When the switch sends the LLDP packet through its

active ports, the Ethernet frame encapsulates the payload of LLDP and set the Ether Type field to *0x88cc*. The Ethernet frame contains the LLDP Data Unit (LLDPDU) that consists of a Type Length Value (TLV) structure. The TLV contains a switch identifier (chassis ID), Port ID, Time to live value, and other optional values. The OFDP uses a similar format of LLDP packet, however; it operates differently due to its limited API's match-action functionality. Moreover, in SDN the OF switches does not send, receive, and process the LLDP messages itself but the controller initiates the process. The operation of the LLDP packet in an SDN is briefly explained below.

(a) *Inter-connection between OF switches:*

The link discovery using LLDP does not require any other discovery approach because both ends of the link consist the OF switches which support the topology discovery mechanism. The topology discovery determines the initial IP address and the TCP port of the controller which helps the OF switch to establish a connection soon after it is connected to the controller. The OF switch also has a pre-configured rule, which generates the *Packet_In* message to the controller when it is received on the ports other than the controller. Initially, when an OF switch establishes a connection with a controller, the controller passes a request message to the OF switch such as *FEATURE_REQUEST_MESSAGE*, wherein the switch response with an *FEATURE_REPLY_MESSAGE*. The response includes the switch ID and active ports along with their respective MAC addresses.

The controller encapsulates the LLDP packet in a *Packet_Out* message and sends it to each active port of all OF switches in the network. The destination address in the LLDP packet is the multicast MAC address defined in the IEEE 802.1AB standard. The total number of *Packet_Out* messages sent by the controller is equal to the number of active ports in the network, i.e., (Total *Packet_Out* message = Number of active ports of all the switches). The *Packet_Out* message installs the flow entries in the OF switch to route

each LLDP packet to its destination port as indicated in the TLV field. The OF switches forward the received LLDP packet to the corresponding port which connects to another OF switch. When the neighbor of the OF switch receives the LLDP packet on the port other than the connected controller port, the switch encapsulates the LLDP packet in a *Packet_In* message and forwards it to the controller. The fields in the *Packet_In* message include the switch ID and Port ID on which the LLDP packet is received. The controller updates its network topology based on LLDP messages and by default; this process is repeated every 5 seconds. The illustration of this process is shown in Figure 2.5.

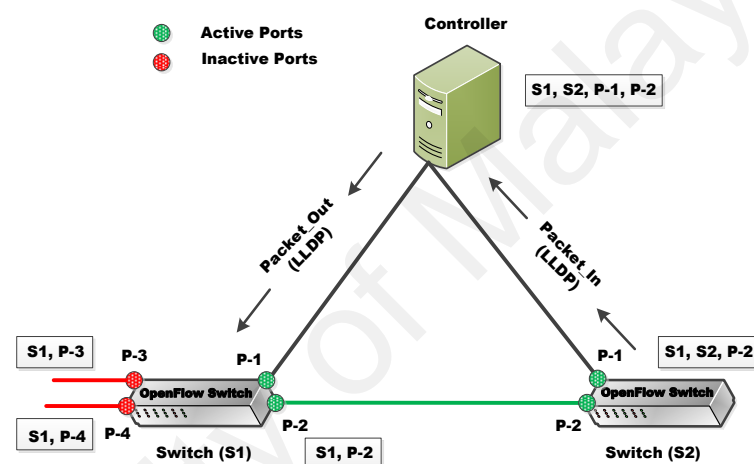


Figure 2.5: The LLDP process in SDN environment

(b) *Inter-connection between the OF switch and the traditional switch*

Currently, the adoption of SDN architecture in the current emerging networks integrates OF protocol with the existing traditional network technologies. It requires a new mechanism to operate in a new network infrastructure without affecting the performance. Similarly, using both traditional and OF switches in data centers will create problems to identify the inter-connected link between these switches. The approach in finding the inter-connected link between the OF switches is not implemented in a hybrid switch infrastructure. The controller needs a mechanism for the handshake with an OF switch to identify its information and capabilities. However, a handshake is not performed for the traditional switches.

A controller in a hybrid switch infrastructure can identify the inter-connected links between the OF switch and traditional switch which must be connected to another OF switch. This scenario can be considered as a non-direct connection between two OF switches. Simply, a controller can find the multi-hop connections between the OF switches. The LLDP is a single-hop discovery mechanism, and it is not applicable to a multi-hop connection. It requires a new approach for finding non-directed connections among the OF switches. To identify the inter-connection between two OF switches, the OF switches should be in the same broadcast domain or the controller will not be able to associate addresses to the multi-hops among the OF switches. The current open source controller such as Floodlight and Open Daylight controller have integrated layer 2 topology discovery protocols such as the LLDP and the Broadcast Domain Discovery Protocol (BDDP) to discover multi-hop links between OF switches and traditional switches within a broadcast domain (Ochoa Aday, Cervelló Pastor, & Fernández Fernández, 2015).

The BDDP message and the LLDP messages are identical but have different destination MAC address fields. The BDDP message has a broadcast address in its destination field while the LLDP message has a multicast address in its destination field. This characteristic allows the traditional switch to forward a BDDP message to find multi-hop links between the OF switches within a broadcast domain. The controller sends each BDDP message to each active port of the switch by encapsulating it in the *Packet_Out* message. When the *Packet_Out* message is sent to the OF switch, it installs a flow entry in the flow table indicating that the OF switch has received the message. Then, the OF switch forwards the message to the neighbor switches via a port indicated in the TLV field. If the neighbor switch is the traditional switch, it examines the destination MAC address and further floods the packet to all its active ports. The port connecting the controller receives the *Packet_In* message that incorporates the metadata required to

identify multi-hop links. The *Packet_In* message contains a BDDP packet which helps the controller to know indirect links between two OF switches such as through multi-hop links.

2.3.2 Controller Platform

The SDN has an architecture which consists of a single or multiple controllers to control the entire network as illustrated in Figure 2.6 and 2.7 respectively. Usually, a small data center network incorporates a single controller while large data centers are distributed and have multiple controllers. This section explains the significance of a topology discovery in single and multiple controller platforms in SDN.

2.3.2.1 Topology Discovery in Single Controller Platforms

The single controller platform is used for a homogeneous network, which is a network of devices connected within a single physical location. The controller is responsible for discovering the network topology by querying the switches through the LLDP packets as described in Section 2.3.1.3.

The controller communicates with the switches through LLDP packet after a specified time interval (i.e., after every 5 seconds) to identify links between the OF switches in the network. In discovering the network topology, the position of the controller is crucial. That is, the controller that is closer to the switches will result in a faster transmission of the LLDP packets to the OF switches as well as receiving a quick response from the OF switches.

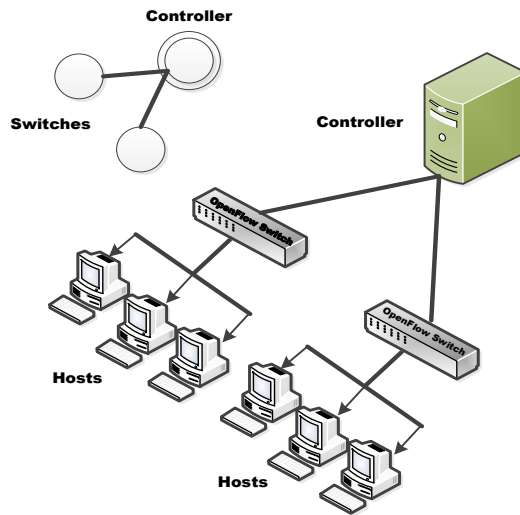


Figure 2.6: Single SDN controller architecture

2.3.2.2 Topology Discovery in Multiple Controller Platforms

Large networks are employed using the heterogeneous setting that includes multiple controllers responsible for different portions of the network. All these controllers coordinate through a logically centralized controller. Each controller requires discovery of the network topology of the assigned SDN domain. The topology discovery information is sent to the centralized controller and also to the neighboring controllers for the latest updates. However, sharing topology information among controllers requires a standard procedure which is not available till date.

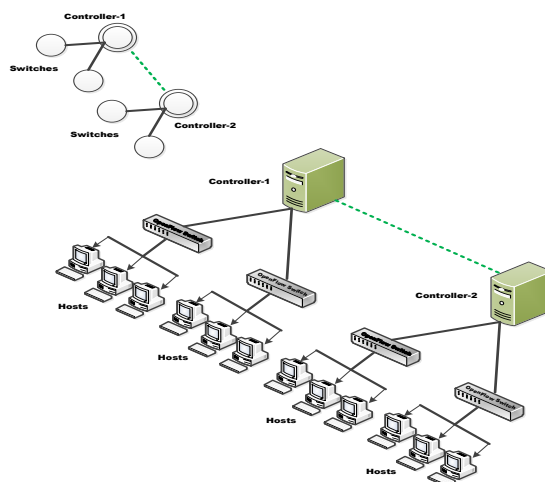


Figure 2.7: Multiple SDN controllers' architectures

The shared topology information is not verified during the sharing process and might be shared by the malicious controller. It may affect the performance of the other controllers regarding routing, load balancing, scheduling and various other services. The importance of topology discovery in multiple controller platforms increases due to the distributed controllers, sharing of topology updates, and instance (virtual machine) migration (Aslan & Matrawy, 2016) among different SDN domains.

2.3.3 Dependent Services

This section discusses the topology-dependent services used in SDN. The logically centralized visibility of the network supports various network applications to perform tasks and control the network devices efficiently. We have explained some of the topology-dependent services to highlight the significance of the topology discovery in SDN as shown in Table 2.3.

2.3.3.1 Routing

The routing application depends on the controller's abstract view of the topology to have the entire visibility of the network (Karakus & Durresi, 2015). For instance, a routing application will require information about the network topology to route the network traffic to its destination on the shortest path (Lee, Yoon, & Shin, 2016). However, falsified topological information may lead the routing application to route its network traffic on to a malicious route.

In the case of an LFA, an attacker can spoof the LLDP packet with a malicious switch DPID and Port ID to inject a fake link in the network topology. It may affect the existing legitimate shortest path towards the destination. For example, as shown in Figure 2.8, host 1 requires four hops (switches) to reach host 4. However, during an LFA, host 1 will send the LLDP packet with DPID-3 and Port ID-1 to switch 1, which further inserts DPID-1 and ingress Port ID-1 in the metadata of the *Packet_In* message and informs the controller that there is a direct link between switch-1 and switch-3. Subsequently, the

controller may incorrectly update its topology information by assuming a direct link between switch 1 and switch 3. It affects the legitimate shortest path, as the traffic from host 1 can be sent to host 4 through the newly added fake link. As a result, the malicious switch 3 can eavesdrop or modify the traffic before it reaches the destination.

2.3.3.2 Mobility Tracking

The mobility tracking refers to a mechanism for tracking a mobile node in the network. Mobility tracking is associated with the cellular networks (Pakzad et al., 2015; Yu & Leung, 2002). The mobility tracking in SDN is achieved through a mobility management function running on top of the controller (Karimzadeh, Valtulina, & Karagiannis, 2014). The mobile management function is responsible for monitoring nodes' movements. Mobility tracking depends on the network topology information of the current and future location of the network nodes. Usually, when a network node (host) changes its position from one switch to another, it changes its IP address and results in a connection break down. However, in SDN, with the help of a mobility management function, the forwarding function informs the controller about the nodes' movement which then recalculates the forwarding rules and forwards it to the forwarding function to route the IP packets accordingly.

As a result, it continues with the application session and makes the movement of the node without changing the IP address. Node mobility changes the network topology which is updated by the controller based on the information received from the forwarding function. Thus, the node movement should be sent to the controller and mobility tracking function on a timely basis to keep the network topology up to date in SDN.

2.3.3.3 Load Balancing

The load balancing is used to improve the utilization of resources and power by distributing the traffic more simply and more efficiently. The load balancer uses a logically centralized control of the SDN to perform traffic load balancing (Namal,

Ahmad, Gurtov, & Ylianttila, 2013b). The dependency of the load balancing on the network topology is significant, which is its selection of the optimal server for the traffic execution. For instance, the load balancing application that is installed on top of the controller requires the location of the servers and optimal path to access them in the network.

The optimal path is selected by computing augmented bandwidth of links between the switches. Any modification in the network topology causes the load balancer to recalculate the bandwidth for the optimal path. Thus, topology discovery plays a vital role in executing the load balancing properly in SDN.

2.3.3.4 Topology-based Slicing

The topology-based slicing is a mechanism of the Flow-visor in SDN that divide the network topology into different parts/slices (Sherwood et al., 2009). The aim of slicing is to provide a dedicated link to each of the tenants in the multi-tenant environment. Topology-based slicing, also known as port-based slicing, creates different slices based on the switch ports. Each switch port has a subset of full network topology controlled by the Flow-visor (You, Qian, He, Qian, & Tao, 2014).

The Flow-visor handles the network traffic on each of the connected links by adding a flow space. The slicing phenomenon reduces the controller load by focusing on specific OF switches of the topology. Therefore, the slicing depends on locations and ports of the OF switches which are key entities of the network topology discovery. However, any modification in the topology will cause Flow-visor to re-compute the specific slice that is affected by the change in the specific domain.

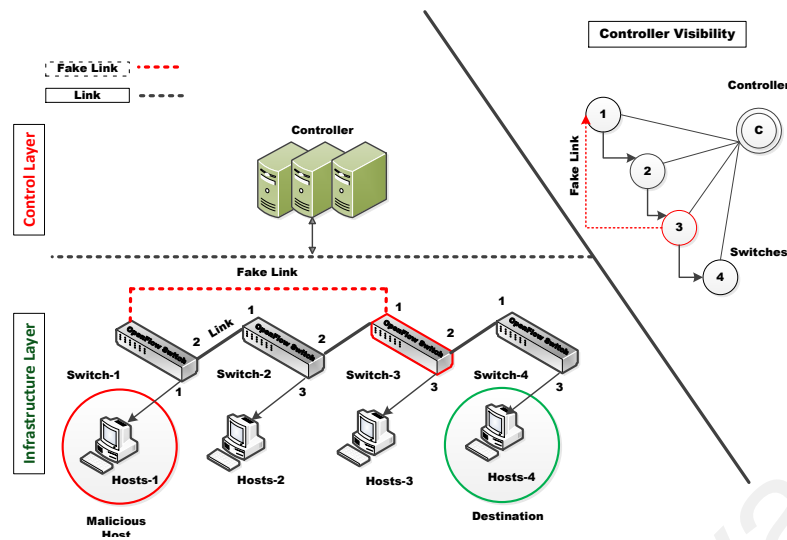


Figure 2.8: A diagram illustrating link fabrication attacks

2.3.4 Objective

The key objectives which are achieved through an efficient topology discovery in SDN are listed as follows.

(a) Multiple switches:

The topology discovery provides an easy way to identify OF switches in SDN. The OF switches could be in single or multiple management domains, controlled by single or multiple distributed controllers (Heller, Sherwood, & McKeown, 2012). The identification of OF switches in the network assists the topology discovery in updating its topology information respectively.

Table 2.3: Topology-dependent applications with its effected attacks

Topology-Dependent Applications	Description	Effected attacks
Routing	Route network traffic from source to the destination	Link Fabrication
Mobility Tracking	Determine the location of the host in the network	Host Location Hijacking
Load Balancing	Distribute network traffic among different servers	Link Fabrication
Topology-based Slicing	Divide single network topology into sub-topologies	Link Fabrication

(b) *Spanning Tree:*

The spanning tree protocol in SDN (Nelson, Ferguson, Scheer, & Krishnamurthi, 2014) provides a loop-free topology. It utilizes discovery services to identify neighbor link detection between the OF switches. The spanning tree installs flow entries in the OF switches. However, without having an efficient topology discovery, the spanning tree is unable to select an appropriate path to eliminate a loop from the network.

(c) *Managing Topology changes:*

The host migration, isolation of working domains, and insertion/deletion of the network devices in SDN can cause modifications in the network topology. The function of topology discovery includes coping up with the change detected in the network. For instance, in a data center environment, virtual machines (hosts) often migrate from one resource to another, which results in topological changes such as the appearance of new switches and ports ID (Mayoral, Vilalta, Muñoz, Casellas, & Martinez, 2015). Consequently, the changes in the network should be sent to the controller to update the topology based on its discovery mechanism.

(d) *Traffic Scheduling:*

Often, the optimal path, i.e., the shortest path is selected to route the network traffic from source to the destination. The topology discovery assists the traffic scheduler in finding the optimal path with fewer propagation delays between a different number of hops (switches) (Akyildiz et al., 2014; Berde et al., 2014). However, incomplete information regarding the network topology may lead to improper traffic scheduling that causes high bandwidth delays and time overhead.

(e) *Robustness:*

The ability to tolerate the packet loss depends on the topology. If the controller timely updates the topology, the network application runs smoothly without causing any packet

loss. The correct topology of the network reduces the overhead of the controller that cause to increase the robustness of the SDN without affecting its normal operation.

(f) Statistical Data Gathering:

The OF switches provide different levels of statistical information to the controller including port statistics, flow statistics, and other counter measurements (Rotsos, Sarrar, Uhlig, Sherwood, & Moore, 2012). The statistical information helps the controller to have an in-depth observation of the flows, network devices and overall behavior of the network. However, changes in the network topology (due to the insertion of new hosts, flow entries, and inter-connected links) between the OF switches can cause changes in the statistical data previously gathered by the controller. The controller has to update its database information based on the new topology of the network.

(g) Resource Provisioning:

To operate an elastic data center infrastructure through the SDN architecture, a resource provisioning mechanism is required to enable on-demand resources for the applications (Bakshi, 2013). The resource provision depends on the network topology to understand the allocation and processing of available resources to different applications. The topology discovery information assists the resource provisioning module in selecting the right resources for the right application.

2.4 Topology Discovery Threats and Solutions

In this section, we provide a comprehensive description of the potential threats to the topological discovery. The threats exploit the vulnerabilities in the controller by performing attacks on the network. We devise a classification of the topology discovery threats as illustrated in Figure 2.9. The classification comprises of four categories, such as a) Attack entity, b) Controller vulnerabilities, c) Current solutions, and d) Miscellaneous threats. Each of the categories is explained as follows.

2.4.1 Attack Entity

Several security threats from different parts of the SDN architecture can be recognized through literature. In this section, we focus on the topology poisoning attacks. The topology poisoning attack is generated by two entities in SDN architecture, i.e., hosts and OF switches. These attacks are explained based on their working operations as follows.

2.4.1.1 Host-based Attacks

The topology poisoning attack generated from the host (system connected to the switch) is called host location hijacking attack. In this attack, the attacker impersonates the target host location in the network and starts receiving traffic intended for the target host. The attacker exploits Host Tracking System (HTS) of the controller which lacks an authentication mechanism especially for the host mobility in SDN. The controller uses HTS to record parameters such as joining and mobility of the host in SDN by maintaining a host profile table. The controller uses *Packet-In* message to update the host profile table by monitoring the DPID, Ingress Port ID, and other metadata information. However, lack of security consideration in HTS provides an opportunity for the attacker to temper the target host location information by diverting the host target traffic towards itself.

The controller may assume that the target host has moved towards the new location but the host remains at its location. The attacker easily hijacks the traffic of the target host by generating a spoofed IP address of the target host using the *Packet_In* message. Upon receiving the *Packet-In* message, the controller updates the host profile table of the target host using its new location. As a result, it affects the topology-dependent applications including routing, load balancing, and various others.

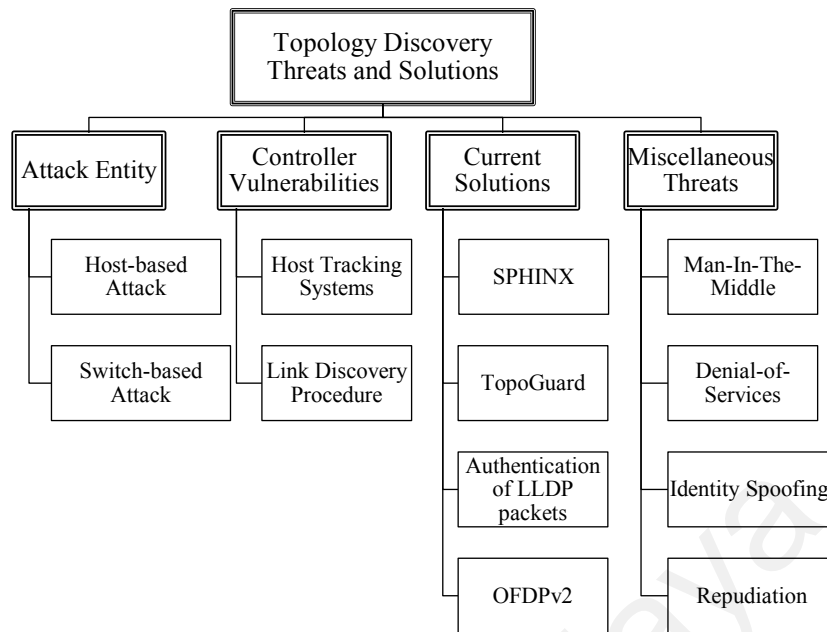


Figure 2.9: Classification of topology discovery threats and solutions

Moreover, the malicious host in SDN can spoof the legitimate LLDP packets and forward it to the OF switch as well. That is, the OF switch can forward the spoofed LLDP packet to all of its active ports, which may reach the controller and can update its link record between the OF switches. The malicious host can also send a legitimate LLDP packet to another OF switch, which may create a fake link between the OF switches. Therefore, the controller may route the traffic on fake links that get forward to the malicious host.

2.4.1.2 Switch-based Attacks

The topology poisoning attack can also be performed through the malicious OF switches. The malicious OF switches forward the spoofed LLDP packets by creating fake links in the network. This type of attack is called an LFA in SDN. The malicious switches can affect a large scale of the network due to fake connections with many network devices. The topology poisoned through malicious switches is difficult to detect due to the minimal clues about the fake link creation in the network. For instance, it does not require any host to create a fake link between the OF switches during a topological poisoning attack.

After receiving the LLDP packet from a single OF switch, the malicious OF switch relays the packets to another OF switch instead of forwarding them to the controller. Subsequently, upon reception of LLDP packets by the new OF switch, the LLDP packets are sent to the controller in the form of *Packet-In* message. It betrays the controller into believing that there exists a link between the malicious OF switch and the legitimate OF switch that forward the LLDP packet to the controller. Such fake link injection attracts possible future attacks such as the DoS attack, man-in-the-middle attack and many others.

2.4.2 Controller Vulnerabilities

This section describes the vulnerabilities in the controller that are used by the attacker to launch an attack.

2.4.2.1 Host Tracking Systems (HTS)

The vulnerability discovered in the HTS attracts attackers to hijack the location of the hosts. As stated earlier that the host profile in the controller contains the DPID, Ingress Port ID, and other metadata information which exhibits the controller with the location of the host and the connected OF switch. The key exploited vulnerability includes the lack of authentication mechanism that can be used to verify the host updates received by the controller through *Packet-In* message. All information received by the controller is considered as genuine (even if received from a malicious OF switch) and the host profile is updated accordingly.

In previous versions of Floodlight and Open Daylight controllers, an empty shell API *'isEntityAllowed'* is provided, which accepts all updates related to the host locations. The attacker simply spoofs the packet with target host identity and forwards it to the connected OF switch which further sends it to the controller in the form of *Packet_In* message. The controller assumes a shift of the position of the host and updates the host profile for the target host. The lack of authentication mechanism in HTS makes the controller update

the topology with falsified host information and this affects numerous services, especially the routing.

2.4.2.2 Link Discovery Procedure

The vulnerability in the link discovery procedure can also be exploited by fabricating the false link between the OF switches. Firstly, there is no authentication mechanism for the controller to ensure the origin of the LLDP packet. Secondly, the controller is unable to verify the traversed path used by the LLDP. Addressing these issues is critical in preventing the OF switches from inserting a fake link. Note that; the OF switches receive LLDP packets from each of its ports. It allows the attacker to spoof the LLDP packets to create a fake link between the OF switches which is known as LFA.

This attack can be performed in two ways 1) modification of LLDP packets, and 2) through the LLDP relay. In the case of the LLDP modification, a fake link is established between the OF switches by spoofing the DPID and the Port ID of legitimate OF switch. It causes the controller to update a new link between legitimate and malicious OF switch. The LFA generated through the modified LLDP packet is explained with an example shown in Figure 2.8.

The host attached to switch-1 learns about the LLDP syntax from receiving the LLDP packet from the controller. The switch-1 then forwards it to all its ports except the controller port. The host-1 sends LLDP packet to the switch-1 with spoofed information including DPID=3 of switch-3 and Port ID=1 of switch 3. The switch-1 inserts the DPID=1, and Port ID=1 in the metadata and forwards the *Packet-In* message to the controller. The controller checks the *Packet_In* message and perceives a link between switch 3 and switch 1. The controller takes the LLDP source information from the TLV such as (DPID=3, Port ID=1) and the link information from the metadata such as (DPID=1, Port ID=1). Thus, the controller is updated with the incorrect information related to a fake link.

In another type of LFA, the attacker simply forwards one of the legitimate LLDP packets to another OF switch and resulting in falsified link information received by the controller. The malicious OF switch requires a relay OF switch to forward the LLDP packet to the target OF switch. The relay OF switch is identified through a connection test. Also, some controllers such as POX and Floodlight disable the HTS on the internal link switch ports. However, an attacker can still launch the attack by using a tunnel-based LLDP relay attack. The tunnel-based LLDP relay attack is used to launch fake links between multi-hop link ports having OF switches connected to the traditional switches. It is difficult to disable these ports in SDN due to the availability of the hybrid switches in the network. Thus, the LFA also opens doors to numerous attacks including DoS and man-in-the-middle attack.

2.4.3 Current solutions

We explain the state-of-the-art topological poisoning solutions in this section. However, the literature has very few solutions for LFA. We briefly explain the state-of-the-art solutions concerning their proposed methodology. Table 2.4 presents the comparison between the proposed solutions using parameters such as techniques, SDN features, attack entity, the problem addressed, and future work. The parameter techniques highlight the key module/application developed by the proposed solution. The SDN features parameter points out which features have been used to model the solution in SDN. The attack entity shows which type of attacks can be detected through the current solution. The parameter problem addressed points out an objective function which has been addressed to detect the attack. Finally, the parameter future work explains future research directions of the current solution.

2.4.3.1 SPHINX

In (Dhawan et al., 2015), the work has presented several attacks which target the network topology and forwarding devices in SDN. It has been shown that an attack can

be launched from malicious hosts and OF switches. A proposed solution as a SPHINX is presented to detect an unknown attack on network topology and the forwarding devices in SDN. The SPHINX provides a real-time and accurate verification solution of the network behavior by a) monitoring all OF messages; b) analyzing features set of the messages, and c) providing a fast validation of the network updates. The SPHINX focuses on four messages, i.e., *Packet_In*, *stats_reply*, *features_reply*, and *flow_mod* to get the metadata about network topology and forwarding device attacks. First, the SPHINX intercepts the OF messages transferred between the switch and the controller. Then, it builds the incremental flow graphs with new updates and validates the network behavior. These intercepts are important to identify the malicious behavior of the attacker. After getting the latest update, SPHINX increments and updates its network topology flow graphs and detects malicious behavior based on the tangible changes observed in the network topology and the infrastructure plane forwarding.

Specifically, IP/MAC address binding, MAC/port binding, and flow statistics of the host are used to provide metadata for assisting SPHINX to detect malicious behaviors in the network topology and the infrastructure plane forwarding. The network behavior is validated through the SPHINX policy engine. The policy engine enables administrators to validate the incremental flow graphs. The constraints specified by the administrators are written in the policy language. However, validating the policy itself is not considered in the SPHINX and is left for future work. In (Yan, Yu, Gong, & Li, 2016), a policy-based security is provided in SDN.

2.4.3.2 TopoGuard

Hong et al. (Hong et al., 2015) have first time proposed an attack in SDN architecture that affects the visibility of the controller by providing poisoned network topology view. The attack illegally modifies the network visibility by hijacking the host location and inserting fake links between the OF switches. These attacks disturb the operation of

different network applications that run on top of the controller such as packet routing, network virtualization, and mobility tracking. A TopoGuard application is proposed to overcome the problem of the poisoned network topology in SDN.

The TopoGuard application is executed in the OF controller that is composed of three main modules namely, port manager, host prober, and topology update checker. Each of these modules in the TopoGuard application depends on the *Packet_In* message to investigate and detect the illegal modifications in the network topology. The port manager provides information related to the device type connected to the switch. The device type contains values of host, switch, or any (another device rather than hosts and switches). The value any is the default value for the device type and will change to host or switch once the packet has been forwarded. The port manager detects the attack and generates an alert to the topology update checker by receiving LLDP packets from the host. The alerts are generated when LLDP packets traverse between internal link ports of the switches rather than the hosts.

Upon migration to a new location, the host probe module is responsible for checking whether the host's previous location is unreachable or not. The location is checked by sending the probe packets (i.e., ICMP echo packets) to the host's previous location. If the host probe receives the echo replies, it will inform the topology update checker to hold the update of a new host location due to the host location hijacking attack. Similarly, topology update checker is also responsible for checking and verifying the information on the host migration and new link discovery in the network topology. Once the host migration is detected, the topology update checker collects the host's previous location from the host probe and then updates its topology discovery of the network.

For the link discovery, topology update checker checks cryptographic hash value for the integrity of the LLDP packet. After the integrity check, the device type is checked from which the LLDP packet is generated. If the device found has a host entity, the

topology update checker considers it as an attack and holds the update of the discovery link in the network topology. Therefore, TopoGuard enables a real-time detection of the topology poisoning attacks in SDN.

2.4.3.3 Authentication of LLDP Packets

In (Alharbi et al., 2015), a countermeasure based mechanism is proposed to overcome the security problem presented in the OFDP. The OFDP lacks an authentication of LLDP packets that may create a risk for the packets to be forged. The proposed method uses a cryptographic Message Authentication Code (MAC) in each of the LLDP packets to authenticate the packet's integrity. The HMAC is used to compute the MAC code. The uniqueness of the HMAC in authenticating the LLDP packet is the use of a dynamic key instead of the static key. In each round of the topology discovery, a dynamic key is used for each LLDP packet which makes difficult for the adversaries to speculate the key. Guessing the key is critical to computing the MAC value and launches a successful LFA.

The key is selected randomly for security and it is difficult for the attacker to guess the key especially when the key is generated with an entropy measurement. Moreover, the controller can detect any attempt made by the attacker for guessing the key. The controller keeps track of each key generated for each packet and verifies the authenticity of the received LLDP packets. The Chassis ID and Port ID are combined to provide a necessary identifier while hashing is performed through an MD5 hashing function. The HMAC value is inserted in the optional TLV field of the LLDP packet which shows that OFDP having HMAC can detect any fabrication of the LLDP packets generated by the attacker. The proposed method using HMAC values in the LLDP packet creates 8% of the CPU overhead which is lower than identifying fabricated links in the network.

2.4.3.4 OFDPv2

In (Pakzad, Portmann, Tan, & Indulska, 2014), a simple and practical modification is performed on the existing topology discovery approach for reducing the control load and

to increase the efficiency of the controller. The proposed approach modifies the de-facto standard of the topology discovery by introducing OFDPv2-A and OFDPv2-B. The two new proposed versions have the same functionality of OFDP with significantly less number of control messages used for link identification between the OF switches. The reduction of control messages significantly decreases the controller load.

In OFDPv2-A, a specified rule is inserted in the flow table of every switch. The rule is inserted to direct the OF switch to create a copy of the received LLDP packet and forward it to all of its active ports. The forwarded message has a modified MAC address for each port. The LLDP packets are limited to the number of the available OF switches. However, the unique LLDP packet in OFDP is sent to the active ports of the switches that cause to increase the workload on the controller due to handling a large amount of the LLDP packets. Also, *Packet-In* message event handler in the controller is changed to know the source MAC address of the Ethernet frame in place of Port ID TLV field of the LLDP payload. The OFDPv2-B operates similar to OFDPv2-A, but it has no rules to handle the LLDP packets send from the controller. An action list is added to each of the *Packet_Out* messages to inform OF switches about forwarding the packets.

Table 2.4: A general description of state-of-the-art topology discovery

Classification	Techniques	SDN Features	Attack Entity	Problem Addressed	Future Work
SPHINX	SPHINX controller application	Incremental flow graphs with metadata information	Host-based attack	To detect suspicious changes observe in network topology and infrastructure plane	Sphinx will consider flow rule aggregation, Proactive OF environment, and Mixed networks in the future.

			Classification
			Techniques
			SDN Features
			Attack Entity
			Problem Addressed
			Future Work
OFDPv2	Authentication of LLDP packets	TopoGuard	
OpenFlow discovery protocol	Hash Message Authentication Code	TopoGuard application	
Modify de-facto standard of topology discovery, i.e. OFDP	Using HMAC inside LLDP packet in every topology discovery round	Extension of OF controller by designing topology update checker	
Controller-based attack	Host-based attack	Host-based attack, Switch-based attack, Controller-based attack	
Reduce the control messages used to identify the links between switches	Provides authentication and packet integrity for LLDP packets	Detection of network topology poisoning attacks	
The discovery of hosts in SDN network	Check the impact of the proposed solution in another area rather than routing.	Design a new security framework which detects more vulnerabilities in SDN	

The action list contains the forwarding logic similar to the OFDPv2-A. The key advantage of the OFDPv2-B is the minimum use of the OF switch memory. However, OFDPv2-B has a disadvantage of increased size the of *Packet_Out* messages due to the insertion of an action list for each switch. The experiment results prove that OFDPv2-A and OFDPv2-B reduce 40% of the CPU overload and control traffic overhead as compared to the de-facto standard such as OFDP.

2.4.4 Miscellaneous Threats

With a successful topology poisoning attack, the chance for other security threats also increases. The threats including man-in-the-middle, denial of service, identity spoofing, and repudiation are explained in the subsequent sections, and their side effects on the topology discovery are presented in Table 2.5.

2.4.4.1 Man-in-the-middle

The man-in-the-middle attack is performed in SDN in various ways (Namal, Ahmad, Gurtov, & Ylianttila, 2013a). One of way is to inject a fake link in the network topology. The attacker eavesdrops the traffic from source to destination by using a false update by the controller. The fake link may force the controller to divert traffic to the attacker location due to shortest path. It might affect the confidentiality as well as the integrity of the network traffic passing through the fake link created between the OF switches.

To mitigate the man-in-the-middle attack in SDN, a proposed solution (Hong et al., 2015) is presented. The proposed solution used device types such as (switch or host) to detect the spoofed LLDP packet, i.e., generated from the host rather than switches. Usually, no host participates in the legitimate LLDP propagation process. The LLDP packets traverse between the switches to determine a link between each other. The proposed solution determines whether the LLDP packet is generated from the host. In that case, the packet is considered as spoofed, and further propagation of the packet in the network is stopped.

The host devices are easily detected through the normal network traffic such as the TCP and UDP. Once the device type is detected as a host, then any information regarding its topology update will not be considered as legitimate. Thus, the solution effectively prevents the man-in-the-middle attack at its early stage by finding the malicious host in SDN.

2.4.4.2 Denial of service

The controller uses the spanning tree algorithm to remove redundant ports after each topology update. However, an attacker can use the same feature to shut down the normal OF switch ports after injecting the fake links in the topology. It causes a burden on the other links connected to the target OF switch and results in a DoS attack (Yan et al., 2016). A legitimate link can be removed by sending a fake LLDP packet by the attacker to the OF switch having lower DPID. The attacker announces a link with a target switch as well. However, if the DPID of the selected OF switch by the attacker is lower than the target switch connected to another switch, then the port of the target switch connected to another switch is removed. It causes overhead on the selected OF switch which causes DoS attack due to an increase in the network traffic overload. The availability of the OF switch is affected due to numerous flow rules in the flow table generated through as a result of a DoS attack.

The literature has various solutions to mitigate the DoS attacks in SDN. In (Shin et al., 2013), a connection migration tool is proposed to reduce infrastructure control plane interaction that detects dynamic flow changes in the network traffic. In (Fonseca, Bennesby, Mota, & Passito, 2012), a seamless primary controller backup is proposed to defend the centralized network operating system from failure. Another DoS prevention mechanism is proposed in (Wang, Xu, & Gu, 2015), which uses proactive flow rules to preserve network policy enforcement. It uses packet migration mechanism to defend the controller from overloading its memory due to numerous *Packet_In* messages. However, the solution fails to find the real source of the attack.

2.4.4.3 Identity spoofing

When a malicious host injects spoofed LLDP packets, the controller updates the false information in the host profile system. The controller thinks the host has changed its position, and the information is sent to its new position from where the LLDP is received,

i.e., malicious host. The target host remains at its position, however; the malicious host pretends to be a legitimate (target) host. The controller passes the information to the malicious host hence, affecting the confidentiality and integrity of the data by modifying it and forwarding it to the further destinations.

To overwhelm the spoofed identity problem in SDN, work in (Hong et al., 2015) proposes a solution based on the pre-condition and post-condition of the host migration. In the pre-condition, once the host migrates from its position, it has to inform SDN controller about its previous port_shutdown. In the post-condition, the controller confirms that host is not reachable by sending ping messages to its previous location. Thus, the controller can effectively track the real location of the host and can determine the spoofed identity of the malicious host.

2.4.4.4 Repudiation

The lack of a proper authentication mechanism of LLDP packet in the controller may cause repudiation attacks. The repudiation attacker creates a fake link through injected spoofed LLDP packet and then denies it to generate by him. The attacker inserts the spoofed DPID and Port ID of the victim OF switch and forwards the packet to the controller by showing it has come from another OF switch. The spoofed LLDP packet loses its confidentiality and integrity by modifying its original value in the packet field.

The work presented in (Dhawan et al., 2015) builds an updated flow graph based on metadata of the *Packet_In* and *FEATURES_REPLY* messages to detect the fake links generated through spoofed LLDP packets. The MAC-IP address binding mechanism of the proposed solution is built by using the language policy engine that assists the controller to detect the fake links upon observing the deviation from such bindings.

Table 2.5: A side effect of threats on topology discovery

Threats	Possible Reason	Affect	Confidentiality	Integrity	Availability
Man-in-the-middle	Injected fake link	Change the shortest path	Yes	Yes	No
Denial of services	Removing legitimate link	Increase a workload	No	No	Yes
Identity spoofing	Spoofing host identity	Illegal information exploitation	Yes	Yes	No
Repudiation	Spoofed LLDP packet	Hiding the attacker identity	Yes	Yes	No

2.5 Future Challenges and Directions

In this section, future research challenges and directions of topology discovery in SDN are presented. The research on topology discovery is still in its early stages. Therefore, ample opportunities exist for future work to mitigate the challenges in topology discovery. The following future directions will help academicians, industrialists, SDN vendors, and network specialists to explore novel solutions in making the topology discovery secure and sustainable in SDN. The descriptions of possible solutions for each future direction are given in Table 2.6.

2.5.1 Multiple SDN Domains

Practically, the SDNs are created by the network operators in the enterprise according to their network requirements. Mostly, the enterprise has different domains which are controlled by each controller resulting in a multiple SDN domains environments. However, small-scale data center network may require a single SDN domain while a large-scale data center network (carrier networks) may require several SDN domains that are controlled by the logically centralized controller. The division of SDN domains varies on the requirements that include physical locations, traffic monitoring, load balancing, and various others.

However, interconnecting multiple SDN domains and sharing the network topology updates in the topology discovery can be a very challenging task. These interconnections require a standard protocol to efficiently share and secure the control information between the SDN devices. Moreover, the standard protocol must be able to consider various important aspects of the topology discovery including for instance a) how the network topologies in various SDN domains are connected, b) how one controller communicates with its neighbor controller, c) what will be the form of information format to share among the controllers, d) how to get the controller addresses, and e) which policies and procedures have to be adopted for the communication. The work presented in (S. Scott-Hayward, Natarajan, & Sezer, 2016) proposed Auto Slice virtualize layer for the SDN architecture to separate multiple SDN domains on the shared network physical resources. It enables efficient sharing of topology discovery information among the controllers placed in multiple SDN domains.

2.5.2 Topology Discovery through OF switches

Currently, the topology discovery function is executed by the controller, and the SDN controller is a single point of failure. The applications in SDN can be affected by the malicious attacks on the controller. One way to overwhelm this issue is to shift the responsibility of a topology discovery to the OF switches. It will reduce the liability on the controller and protects the topology discovery function at the time the controller is attacked. The OF switches can send the LLDP packets to their ports after a specified time interval to determine the links between their neighbor OF switches. The LLDP packet should contain the switch ID and an output port number to identify the origin of the LLDP packet. The OF switch should update the controller on every new link detected for the latest updates. Therefore, the topology discovery cost will be independent of the number of controllers in SDN. However, discovering a network topology for each controller

domain is costly regarding the LLDP messages used in communication between controllers and the OF switches, network bandwidth consumption, and the time overhead.

The above complications can be minimized through the dependable and simpler topology discovery mechanisms in SDN. For instance, the use of the OF switch-based topology discovery can decrease the controller cost linearly because a single discovery mechanism will work for all the controllers in the network. Moreover, the OF switches can increase the priority values for the LLDP packets in the flow tables to transfer the packets on time even in the heavily loaded network links. It assists the controller to have more consideration towards the other core functionalities.

2.5.3 Identification of fake links

The injection of a fake link in the network topology will critically damage the controller visibility and affect the network services to produce the false results. To determine whether the link is fake or legitimate, the controller has to be intelligent enough to decide the legitimacy of the link in network topology within a specified time. However, currently, a proposed mechanism is lacking essential features to distinguish between legitimate and fake links.

A potential solution to this issue is to access the historical information of the OF switch to identify its involvement of malicious activities. Another solution is to check the traffic flow on the newly inserted link as most of the fake links are created to overload the resources, i.e., OF switches by flooding the link with packets. Also, selecting the optimal feature of the network traffic plays a vital part in the detection of the fake link. Therefore, utilizing a machine learning techniques can make the topology discovery management more secure with identifying various features.

2.5.4 Frequent migration

Mostly, in medium and large data center networks, SDN architecture is implemented for different purposes. The topology discovery is more sophisticated in these data center

networks due to the frequent migration of the virtual instances in a virtualized network environment (Gani et al., 2014; Qi, Shiraz, Gani, Whaiduzzaman, & Khan, 2014). It overloads the controller which requires to frequently updating the network instances to have a clear and fair network visibility of the network. It opens the opportunity for the malicious node to connect to other network nodes and create fake links in a different part of the network.

An intelligent mechanism based on statistical probability is required to track the network nodes behavior to assist in determining the malicious activities that affect the topology discovery mechanism. One way is to use the entropy measurement technique to determine the uncertainty in SDN after the attack (Yan et al., 2016). For instance, the attacker injects the fake links in the network which create uncertainty in the network due to incorrect network topology. The entropy can be used to determine the locations where the fake links are inserted by calculating the uncertainty in the network. It can support forensics mechanism in reaching the real source of the attack (Suleman Khan, Ahmad, et al., 2014; Suleman Khan, Gani, Wahab, & Bagiwa, 2015). As a result, the topology discovery will be performed efficiently by focusing on the visibility rather than the security parameters.

2.5.5 Topology discovery information safety

The internal state information of the controller is recorded in the Network Information Base (NIB). The NIB is a separate module in the controller that stores the critical states of the controller. These states can be used to regenerate the events at a specific time as required. Similarly, the topology discovery information is saved in the NIB module of the controller. Nowadays, the controller has become a key focus of attacks due to its core management functions and logically centralized control.

Table 2.6: A description of future challenges and directions of topology discovery with its possible solutions

Future Directions	Description	Possible Solutions
Multiple SDN domain	The SDN controllers controlling different domains create complication in sharing topology discovery information	— Standard protocol
Topology discovery through OF switches	Reduce less burden on the controller due to topology discovery	— Use OF switches for Topology discovery
Identification of fake links	To know about the status of the link	— Check OF switch history record — Verify traffic flow on the link
Frequent migration	Topology discovery mechanism becomes sophisticated due to frequent migration of instances	— Statistical probability — Entropy measurement
Safety of topology discovery information	The attackers can exploit the topology discovery states	— Strong authentication mechanism — Redundant Topology discovery states
Upgradation of the controller	The topology discovery should be consistent at the time of the controller up gradation	— Redundant Topology discovery states

The controller can be attacked through various channels to produce a false output. The decision of the malicious controller cannot be trusted and can lead to an incorrect decision. Similarly, during the attack on the controller, the NIB states can be affected, which might destroy the topology states stored in the NIB. As a result, the controller in the next iteration of the topology discovery updates its record without having the information from the previous topology discovery iteration. It may cause the controller to update the malicious information injected by the attacker after exploiting the records of topology discovery state in the NIB.

To circumvent the issue above, a controller should forward a copy of its topology discovery states to its neighbor controller. The neighbor controller can re-generate the

topology discovery states whenever the topology discovery states are affected by an attack. Alternatively, having a strong authentication mechanism will prevent the attackers from exploiting the core management modules of the controller. The work presented in (Phemius et al., 2014) proposed an extendable control plane, i.e., DISCO to deliver end-to-end network services using a distributed controller environment. It enables highly manageable control channels for sharing aggregated network information among the controllers. Thus, it can traverse topology discovery information among the controllers in a controlled environment.

2.5.6 Controller upgrade

The controller must be periodically upgraded by adding features, fixing bugs to improve its performance. It is important due to frequent change in the network infrastructure in the dynamic virtualized environment. Currently, the SDN lacks the effective techniques to assist the controller in upgrading without affecting the current operation of the network. In existing controller up gradation techniques, the controller is restarted, or the old states of the controller are recorded and then replayed in the upgraded controller to recover its previous states. Similarly, the situation is same for the topology discovery states. Upon upgrading the controller for its new assignments, the previous topology discovery states are lost. It incurs the overhead of re-executing the topology discovery right from its initial stage to acquire the network visibility of the network.

One of the possible solutions is to save the topology discovery state in the neighbor controller (Blenk et al., 2016). After the controller is upgraded, the operation of the topology discovery is resumed from the last recorded status. However, when the network topology changes during the upgrades, the records will be inadequate in the respective topology. To minimize the inadequate records, the controller should be upgraded at the time when the chance for the topological change is less in the network.

2.6 Conclusion

This chapter explains the concept of SDN with forensic aspect by focusing on topology poisoning attacks. It analyzes topology discovery mechanism of SDN controller by devising a thematic taxonomy that presents various parameters. It provides insight into various threats that affect the network visibility of the controller. It discusses state-of-the-art topology discovery techniques used to detect topological poisoning attacks. Moreover, a brief discussion is presented to highlight possible solutions for efficient topology discovery in SDN. Finally, various challenges are highlighted regarding topology discovery of SDN.

Source identification is an emerging research area of SDN forensic which has been neglected for various reasons. Firstly, security was not a part of the initial development of SDN architecture. When SDN starts receiving various attacks after its implementation, people realize that security should be a major part of SDN architecture. Secondly, identification of source attacks is not a simple and easy task. It is considered one of a difficult task in the security field, i.e., forensic science. Mostly, SDN researchers have focused on the detection of attacks rather than focusing on the cause of the attack. Thirdly, multiple controllers in SDN have made the situation more complicated for source identification of attacks regarding control management of SDN instances. For instance, a malicious host controlled by a single controller at the time of attack may be a part of other controlled network during the time of the investigation. Fourthly, virtual machines migrations have to make source identification more sophisticated due to their frequent mobility, especially in cloud computing.

The literature advocates that topological poisoning attack especially LFA damages network visibility of the controller which is an important requirement for control and application plane in SDN architecture. Several techniques have been proposed to overwhelm such a problem by detecting the attack. However, it is utmost important to

find the real source of the LFA to assist SDN controller in preventing similar attacks in the future. Thus, trace back of LFA is a challenging problem of SDN security domain that is addressed in this research.

University of Malaya

CHAPTER 3: PROBLEM ANALYSIS

The complex nature of SDN coupled with the huge number of services they provide; make the system as a whole prone to some malicious attacks. An attack such as LFA is becoming more and more common with a high degree of sophistication. Different techniques exist which examines the occurrence of LFA in SDN. To ensure that SDN does not suffer from the same attack from the same origin, we have to consider in general the entire system liability, service information, dependencies, and switch as well as host vulnerabilities. However, due to the complex nature and scalability property of SDN, determining the origin of the LFA is not a straight forward. Although other solutions have been proposed to address this problem in traditional networks, a formal method to security in SDN is lacking. In this chapter, we discuss a formal method for SDN using Higher Order Logic (HOL) and as a case study use to examine the LFA in SDN.

This chapter is organized into eight sections. Section 3.1 explains formal method along semantics of HOL. Section 3.2 presents formal representation of SDN whereas Section 3.3 describes formal representation of SDN as services. Section 3.4 describes formal representation of network vulnerabilities. Section 3.5 provides contextual analysis of LFA in SDN. Section 3.6 presents formal analysis of LFA by providing satisfiability test. Section 3.7 presents performance metrics used to evaluate proposed method with state-of-the-art solutions. Finally, Section 3.8 concludes the chapter.

3.1 Formal Methods

In this section, we have explained briefly the formal methods used in our problem analysis phase of the research.

3.1.1 Formal Definition

In this section, we provide a brief discussion on Higher Order Logic (HOL) (Van Benthem & Doets, 2001). We also argument our discussion with the reasons why we think HOL can be suitable in describing SDN framework, liabilities, and attack scenarios.

HOL is a group of formal information representation of a specific domain of an application. HOL has more expressive power than the first order and second order predicate logic. The centre behind the brain of HOL is (typically) the decidable and proficient methods.

HOL are atomic formulas that are generated from a given set (L) of non-logical constants, among which we can distinguish individual constants, function symbols, and relation signs. The HOL is obtained from turning these atomic formulas into an inductive definition allowing the formation of more complicated types and considering quantifiers for all such types.

3.1.2 HOL Syntax and semantics

There are some choices for higher logic that might be selected for a research study of this nature. However, we adopt an approach that extends the approach of HOL programming by describing an analogue of Horn clause within a rich HOL that was proposed in church's simple theory of types. There are two types of symbols in HOL. These symbols are the logical and non-logical symbols. The logical symbols used in HOL and their interpretations are at this moment summarized in Table 3.1.

Using the HOL symbols in Table 3.1, we now define a formal representation of SDN. This involves the formal representation of network devices and how they are connected in SDN. We will also define access level on network devices (switches) and host to have clear understandability to differentiate between authorized and unauthorized users. The services running on the host is also be defined. It aims at obtaining the entire network configuration at each point in time. In general, we will examine and represent the formal framework of SDN using HOL as a formal language.

Table 3.1: The logical symbols used in HOL

Symbols	Usage	Interpretations
\forall	Quantifier	\forall_x For all possible values of x
\exists	Quantifier	\exists_x There exist a value x
\wedge	Logical connective	<i>Logical conjunction:</i> The statement: $(A \wedge B)$ is true if and only if A is true and B is true
\vee	Logical connective	<i>Logical disjunction:</i> The statement: $(A \vee B)$ is true if either one of A is true or B is true
\rightarrow	Logical connective	<i>Implication:</i> The statement: $A \rightarrow (B \wedge C)$ asserts the truth of A if and only if B and C are true. Similarly, the statement $A \rightarrow (B \vee C)$ asserts the truth of A if either B or C are true
\leftrightarrow	Logical connective	<i>Logical Bi-conditional:</i> The statement $(A \leftrightarrow B)$ asserts A if and only if B
\neg	Logical connective	<i>Logical Negation:</i> The statement $(\neg A)$ asserts to does not yield A
$\stackrel{\text{def}}{=}$	Equality	<i>The statement:</i> $(A \stackrel{\text{def}}{=} B \wedge C)$ asserts that the truth of B and the truth of C equals A by definition

3.2 Formal Representation of SDN

The unique feature of SDN facilitates the network operators to develop a simple program that controls the network in a programmable way. It is important to verify the security properties of these programs before it is deployed in SDN for its execution. Thus, an effective approach to check and ensure the security of these programs or systems is to use formal methods.

Formal methods assist network operators in protecting the systems or network from unexpected errors that might not be detected at the time of development and deployment. Therefore, we have used HOL as a formal specification to represent the SDN infrastructure.

Definition 1: SDN (S) is a four-tuple system that is made up of $(\Gamma, \Phi, \lambda, \Delta)$ where (Γ) represent a non-empty set of controllers, (Φ) represent a non-empty set of switches, (λ) represent a non-empty set of hosts and (Δ) represent non-empty links that connect switches to each other. The links are between switches, switches to hosts, and switches

to a controller. Thus, we now model the SDN which is represented by the term (S) in HOL as follows in

$$\forall S \stackrel{\text{def}}{=} \exists \Gamma \wedge \exists \Phi \wedge \exists \lambda \wedge \exists \Delta \quad (3.1)$$

Theorem 1: If we assume the definition in equation 3.1, (S) is non-biased for all occurrences of SDN, and then we can imply the truth of the following statement in equation 3.2 for all occurrences of SDN.

$$\forall S \rightarrow \exists \Gamma \wedge \exists \Phi \wedge \exists \lambda \wedge \exists \Delta \quad (3.2)$$

Proof:

The statement in Theorem 1 for SDN is non-trivial since it will always evaluate to be a universal truth for all occurrences of SDN when

$$\begin{aligned} (\infty < \Gamma > 0) \\ (\infty < \Phi > 0) \\ (\infty < \lambda > 0) \\ (\infty < \Delta > 0) \end{aligned}$$

Since we have now defined the SDN and have proven that SDN is a non-empty set of a four-tuple containing $(\Gamma, \Phi, \lambda, \Delta)$. Therefore, we move further to define (Γ) which we use to represent the non-empty set of controllers. The controller is the brain behind the operation in SDN; we now modelled the controller in HOL as follows.

Definition 2: A controller (Γ) in well-defined SDN system consist of three main entities which are defined in a couple as (M, I, R)

$$\forall \Gamma \stackrel{\text{def}}{=} \exists M \wedge \exists I \wedge \exists R \quad (3.3)$$

Where (M) are the core modules within the controller that involve the routing information, topology monitoring, load balancing, and others. The (I) is the interfaces of the controller and (R) are different services that are provided by the controller. Note that the controller is made up of four interfaces which includes south, north, and east & westbound. Since our focus is on southbound interface that links controller to the switches, then we define its specification in formal representation using HOL as follows.

Definition 3: A southbound interface of a control plane which we represent as (si) is a component of the controller that connect to a switch (Φ) in SDN

$$\forall \Gamma \exists si \leftrightarrow \exists_{\geq 1} \Phi \quad (3.4)$$

Since switches have information associated with them and are made up of one or more interfaces and an operating system, then we define every switch as follows.

Definition 4: A switch is a two-tuple component that is made up of the switch information and switch interfaces. We represent these two components as $\Phi(\chi, \psi)$. Thus we define a switch in HOL as follows.

$$\forall \Phi (\exists \chi \wedge \exists \psi) \quad (3.5)$$

Where (χ) represent the switch information and (ψ) represent the switch interfaces. Thus,

$$\forall \Phi (\forall \chi (\exists ip(1) \wedge \exists pno \wedge \exists sn)) \quad (3.6)$$

Where (ip) represent the IP address of the switch whereas its value (1) indicates that only one IP address is allowed for each switch. The (pno) represent the switch ports and (sn) represent the switch name.

Definition 5: A link (Δ) in SDN connects host through switches using interfaces. Thus we define a link as

$$\forall \Delta (\exists_{>1} \psi) \quad (3.7)$$

Theorem 2: For every link to be successful there has to be more than one functional interface from the participating devices. Such as, we define a switch interface as follows

$$\forall (\Phi. interface) \rightarrow ip(1) \wedge \exists_{\geq 1} pno \wedge name \quad (3.8)$$

Proof

This is a straight forward prove because the expression in equation 3.8 will always evaluate to be true, given that all host computers within an SDN will have a unique IP

address, name, and port no > 0. Hence, the theorem is a universal truth if the host computer is connected to a switch in the network.

Definition 6: A host (λ) is a representation of a computer system that is defined over three entities in SDN. These entities include the host information, interface, and operating system.

$$Host \stackrel{\text{def}}{=} \lambda. information \wedge \lambda. interface \wedge \lambda. operating\ system \quad (3.9)$$

Since host is primarily for providing services, we define the host information as

$$\begin{aligned} \lambda. information \stackrel{\text{def}}{=} & \exists \text{ named host} \vee \\ & \exists \text{ unnamed host} \end{aligned} \quad (3.10)$$

The host information includes the proper identification of the computer system in the network which is known as *named host* or the host which joins the network through illegal circumstances is known as the *unnamed host*. These both types of hosts are important to be differentiated to know about the legitimate and malicious hosts in SDN.

$$\begin{aligned} \forall \text{ named host} \stackrel{\text{def}}{=} & (\exists(\lambda. name) \wedge \exists(\lambda. interface) \wedge \\ & \exists(\lambda. operating\ system) \wedge \exists(\lambda. service)) \end{aligned} \quad (3.11)$$

The named host consists of proper name, its interface details, an operating system which it is using and the services it is going to be provided. However, such information is missing for the unnamed host.

$$\lambda. interface \stackrel{\text{def}}{=} (\exists(ipaddress(1)) \cap \exists(pno(mac\ address(1)))) \quad (3.12)$$

The host interface consists of IP and mac address along with its port number. The value (1) of IP and mac address shows that it will have only one value each.

$$\begin{aligned} \lambda. operating\ system \stackrel{\text{def}}{=} & \exists brand \wedge \exists version \wedge \exists component \wedge \\ & \exists configuration \end{aligned} \quad (3.13)$$

The operating system of host consists of brand name, version, core components, and its configuration.

$$\forall components (\exists name \wedge \exists Version) \quad (3.14)$$

Each host component consists of its name and version to differentiate it from others.

$$\forall configuration (\exists para_name \wedge \exists para_value) \quad (3.15)$$

Each configuration file contains various parameters which are composed of parameter name and values.

$$\forall Application (\exists Name \wedge \exists version) \quad (3.16)$$

The application which runs on the host provides a name along its version to have services for the users in the network.

3.3 Formal Representation of SDN as a Service

In this section, we will now model important information in SDN as a service. However, to successfully do that, we have described services with unique identification protocols such as service name, service version, service protocol, the port used for service communication, and access policy. We, therefore describe services as identification protocol as follows.

Definition 7: We define SDN services (Z) through its identification protocol as $Z(\mathbb{N}, \wp)$ where (\mathbb{N}) represent the service information and (\wp) represent the access policy.

$$Z \stackrel{\text{def}}{=} \exists \mathbb{N} \wedge \exists \wp \quad (3.17)$$

$$\mathbb{N} (\exists name \wedge \exists protocol \wedge \exists version) \quad (3.18)$$

The service information (\mathbb{N}) is elaborated with information of name, protocol and its version.

$$\wp (\exists controlled \vee \exists controlled \text{ with trust} \vee \exists Anonymous) \quad (3.19)$$

The access policy is defined regarding three aspects including controlled environment, uncontrolled environment, and anonymous environment. The controlled environment is

defined where the user has given legal access to the services in the network. However, in an uncontrolled environment, the user has illegal access to the services in the network. In anonymous environment users from outside the network access the services in the network through different illegal channels.

$$\textit{Controlled} (\exists \textit{global account} \wedge \exists \textit{local account}) \quad (3.20)$$

To ensure the controlled environment, user access to the network through two accounts such as global and local accounts. The global account users can access the SDN services from outside the network through legal authorization, however; local account users can only access the SDN services from inside the network.

$$\textit{global account} (\exists \textit{username} \wedge \exists \textit{password} \wedge \exists \textit{access level}) \quad (3.21)$$

The global account contains the username and password along with its access level. The access level specifies the services a user can access in SDN. It depends on the pre-defined control access level in SDN.

$$\textit{local account} (\exists \textit{host} \wedge \exists \textit{username} \wedge \exists \textit{password} \wedge \exists \textit{access level}) \quad (3.22)$$

The local account is comprised of host identity, username, password, and access level of user to access the services in SDN. The host identity indicates that from which part of the SDN the services are accessed by the host.

Theorem 3: A host will be trusted if the access policy is controlled and verified through a trusted list by the true evaluation of the global and local account access as shown in equation 3.20. Thus, we define the theorem in equation 3.23 for controlled with trusted host in the controller.

$$\textit{Controlled with trust} \rightarrow (\exists \textit{controlled} \wedge \exists \textit{trust_list}) \quad (3.23)$$

Proof

The proof of Theorem 3 is non-trivial because the expression in equation 3.23 may not always evaluate to be true, given that provided services originated from anonymous sources. Hence, the theorem is only true if services originate from a secured host.

Theorem 4: A host will not be trusted if the access policy is controlled but not verified through a trust list by the evaluation of the global and local account access from equation 3.20. Thus, we define the theorem in equation 3.24 for an uncontrolled host with no trust in the controller.

$$\vdash \textit{Controlled with trust} \rightarrow (\exists \textit{controlled} \vee \exists \textit{trust_list}) \quad (3.24)$$

Proof

The proof of Theorem 4 is trivial as the expression defined in equation 3.24 may not always evaluate to be true. This because the certain host may be controlled but not trusted within the SDN infrastructure. Hence the theorem is considered as a non-universal truth and only evaluates to be true if and only if a host is controlled and trusted.

3.4 Formal Representation of Network Vulnerability

We have successfully modelled SDN as formal logics up to now. Further, we will go to represent the network vulnerability regarding formal representation. Thus, we define vulnerability by the vulnerability name, a precondition and a postcondition respectively.

Definition 8: The equation 3.25 presents a definition of vulnerability present in the network system.

$$\textit{Vulnerability} \stackrel{\text{def}}{=} \exists \textit{name} \wedge \exists \textit{precondition} \wedge \exists \textit{postcondition} \quad (3.25)$$

Theorem 5: An attacker needs certain access level before he or she can access the vulnerabilities of the network or system. Therefore, we define vulnerability precondition and post conditions as follows.

$$\textit{precondition} \stackrel{\text{def}}{=} \exists \textit{preconditionElement} \vee (\textit{complex precondition} \wedge \textit{test restriction})$$

$$\exists \textit{preconditionElement} \vee (\textit{complex precondition} \wedge \textit{test restriction})$$

$postcondition \stackrel{\text{def}}{=}$

$postconditionElement \vee (complex\ postcondition \wedge test\ restriction)$

However, we note that a precondition that defines vulnerability may be made up of two or more preconditions which are referred to as a complex precondition. However, to determine whether a complex precondition is true, we need a series of logical AND operations to be true which we require the network to be passed. As such, we redefine vulnerability as follows.

$Vulnerability \rightarrow \exists precondition.rule \vee (\exists complex\ precondition \wedge \exists test\ restriction)$ (3.26)

Using this definition, we can now abstractly define a simple attack model and complex attack model as follows.

$\forall Simple\ Attack$

$\stackrel{\text{def}}{=} (\exists liability \vee (\exists attack \wedge \exists test\ restriction))$ (3.27)

The simple attack model is true when the attacker exploits the liabilities in the network by attacking a network entity through by passing some test restrictions. Thus, we define test restrictions regarding post and preconditions as follows.

$PrecondTestRes$

$\stackrel{\text{def}}{=} (Pcond1 \vee Pcond2 \vee \dots \vee Pcondn) \wedge \exists non$

$- encrypted\ value$

$PostcondTestRest$

$\stackrel{\text{def}}{=} (Tcond1 \vee Tcond2 \vee \dots \vee Tcondn)$

$\wedge \exists encrypted\ value$

$\forall Complex\ attack \stackrel{\text{def}}{=} \exists liability \vee (\exists_{>1} attack \wedge \exists test\ restriction)$ (3.28)

The complex attack is similar to the simple attack model, but it is the combination of more than one attack. It happens when an attacker generates more than one attack or generates some attack attempts on the single network entity.

3.5 Contextual Analysis of LFA Attack in SDN

In this section, we utilize the proposed model in representing an example of SDN and perform a contextual analysis against an LFA. Initially, we briefly explain the LFA; we then define the attack using formal representation. Finally, a practical example of LFA is analyzed using the attack model.

3.5.1 Link Fabrication Attack in SDN

In this section, we have explained the way an LFA is performed in SDN. A complete description about the LFA is provided in section 2.4.2.2 of chapter 2 and is illustrated in Figure 3.1. The LFA generates a fake link between the switches. The attack can be generated through malicious hosts and compromised switches in SDN. The attacker used a LLDP packet to generate fake links in SDN.

The attacker spoofs the LLDP packets which are sent by the controller to the switches. Once the switches receive the LLDP packet from the controller, it forwards to all the active ports to determine connected links between the switches. The malicious host also receives the LLDP packets as it is connected to a port of the switch. The malicious host finds out the format of the LLDP packets and generates a false LLDP packet by spoofing the switch identity in the packet. The spoofed LLDP packet is forwarded to the connected switch which further forwards the LLDP packet to its neighbor switches connected with its active ports.

The neighbor switch gets the incorrect information from the LLDP packet and generates a *Packet_In* message by encapsulating the LLDP packet and forwards it to the controller. The controller receives the LLDP packets and assumes that there is a link between the two switches. The controller updates a fake link between the switches in its

topology database which may results in falsified outputs during the execution of network applications such as load balancing and routing.

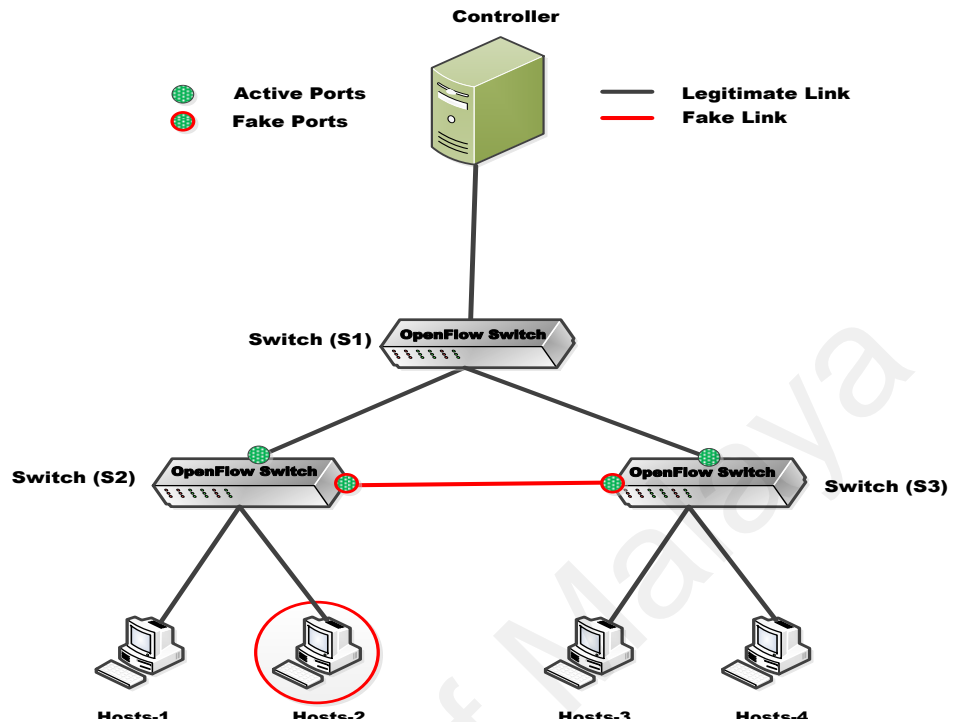


Figure 3.1: A fake link between switches in SDN

3.5.2 Formal Representation of Link Fabrication Attack

To model an LFA, we need to represent it as to our SDN proposed HOL representation. The fundamental definitions of the entity involved in the SDN LFA are as follows.

$$AV, AT, A^T, \text{Host } [i]$$

The AV is the attack victim, A^T is the attack target, AT is the attacker, and Host [i] is the number of host in the network. The AV is the network node used by an attacker AT as a relay node during the attack. The relay node does not take part in the process of an attack but is used as a support to launch an attack. The A^T is the targeted node to which an attack is launched by the attacker AT in the network.

$$\forall \lambda (\exists S_v)$$

The S_v is the services provided by the hosts regarding the application running on it.

$$\forall A^T (\forall_{\geq 1} \Delta)$$

The attacker AT can generate one or more than one links with the switches in the network.

As per our initial definition of the attack, the action of the attacker is considered to be an attack if the following condition is true such as

$$\forall Attack (\exists_{(True)} vulnerability)$$

This shows that the attacker has generated an attack by exploiting the vulnerabilities found in the network.

$$\forall AV \stackrel{\text{def}}{=} ((\exists Attack \wedge \exists vulnerability) \vee (\exists Complex\ attack \wedge \exists test\ restriction)) \quad (3.29)$$

The victim attack node can be formally defined as it is a part of the attack when the attacker exploits the network vulnerabilities through generating complex attacks by bypassing the test restriction of the network security.

$$\forall A^T \rightarrow ((\exists Host[i] \wedge \exists Vulnerability) \vee (\exists Complex\ attack \wedge \exists test\ restriction)) \quad (3.30)$$

Similarly, the attack target node is any switch that has been targeted by the attacker (Host[i]) through a complex attack by bypassing the test restriction of the network security to exploit the vulnerabilities found in the network.

Theorem 6: A host is expected to provide application services to other hosts through switches. Thus, we define a host that is either an attack victim or an attack target in equation 3.31.

$$Host[i] \rightarrow \exists Sv \wedge (\exists AV \vee \exists AT) \quad (3.31)$$

Proof

The statement in equation 3.31 of Theorem 6 is also non-trivial, although it is a known fact that hosts in SDN provide services. However, it is not that certain that a host will be a victim of an attack or a target of an attack or both at the same time. Thus, we infer this

theorem as a non-universal truth since it only evaluates to be true when certain conditions are fulfilled.

Theorem 7: The LFA occurs when the adversary host creates a fake link by accessing the system vulnerabilities and spoofing the LLDP packets. Thus, we extend Theorem 6 to LFA in equation 3.32.

$$LF_{Attack} \rightarrow \exists Host[i] \wedge \exists Vulnerability \wedge (\exists A V \vee \exists AT) \wedge LLDP_{Spoofed} \quad (3.32)$$

Proof

The equation 3.32 in Theorem 7 is straight forward since it will evaluate to be true for all LFA. The malicious host will use vulnerabilities by using attack victim to attack the attack target node. The LLDP packets are spoofed by inserting incorrect information in the fields.

Thus, we have demonstrated the LFA situation. In the remaining parts, we portray an example of SDN which we model in HOL; we now move on to describe an attack we need to analyze its problem against LFA in SDN.

3.6 Link Fabrication Attack: An Illustration

Assume an attack situation in which some hosts and switches are situated on the same network. A host in the network is also assumed to be connected to one or more switch. We note that for an attacker to successfully launch an attack on a target, it is imperative that the attacker be on the same network as the intended victim. In this manner, the attacker can fabricate a fake link to its intended victim using an unencrypted session. Now, to develop the system model we utilize a few assumptions which cover the knowledge of the network.

We expect to extract this knowledge from the SDN controller using some network scanning tools. Once the knowledge is extracted successfully, we can use formal derivative rules for the model network verification and the network satisfiability check

for LFA. It is noted that we are assuming the host as an attacker and switch as an attack target in the below-mentioned scenario. The following assumptions are as follows.

Assumption-1: $\forall AV(\exists Host)$

The attack victim (AV) nodes in SDN are hosts.

Assumption-2: $\forall A^T(\exists (\Phi))$

The attack targets (A^T) nodes in SDN are switches.

Assumption-3: $\forall Attack(\exists Sv)$

The attack is a malicious activity or services used to spoof LLDP packets to launch the LFA.

Assumption-4:

$\forall Remote\ access(\exists Unencrypted\ service \vee \exists encrypted\ service)$

The attacker AT can have remote access to the network through encrypted or un-encrypted services.

Assumption-5: $\forall User(\exists Local\ account)$

The host in the network can access the network services through its local account.

Based on the assumptions mentioned above, we are going to represent the LFA in SDN formally.

Theorem 8: Lack of authentication mechanism causes an attack victim to accept the attacker as a trusted authenticated host, and as such continue to create a session with the attacker through the attack victim. Thus, LFA involves three agents namely attack victim, attack target and exploited vulnerabilities. Therefore, we define LFA in equation 3.33.

$$\begin{aligned} \langle LF_{Attack} \rightarrow (AV) \wedge \exists (AT) \wedge vulnerabilities \wedge \\ \exists (Unencrypted\ service \vee (Complex\ attack \wedge test\ restriction)) \wedge \\ LLDP_{Spoofed} \rangle \end{aligned} \quad (3.33)$$

The LFA is formally defined in the equation 3.33, as the attack is launched when the attacker AT use the victim attack node to attack the attack target node through un-encrypted services by launching a complex attack by bypassing the test restriction of the network security.

3.6.1 Satisfiability Test

After representing the LFA in its formal description, now we should attempt to check the satisfiability of term (LF_{Attack}). To do this, we have to fabricate the term from existing presumptions so that we could demonstrate that this class is not equivalent to an empty set. Initially, the attacker is in the same network as follows.

Assumption-6: $\forall AV, \forall A^T \exists (theSameNetwork)$

This assumption would yield to a simple precondition. Then again four other assumptions that may come to existence would include the following preconditions:

Assumption-7: $\forall AT, \forall Remote\ access \exists (Sv)$

This assumption indicates that an attack target node can be accessed remotely through its services. The remote access could be legally through a global legal account, or it would be through illegal account bypassing the network security.

Assumption-8: $\forall Remote\ access, \forall User \exists (Sv)$

The remote access is performed by the users to access the services in the network. The users may be legitimate or malicious users outside from the network.

Assumption-9: $\forall User, \forall AV \exists (Sv)$

The users can use the victim attack node as a relay node to access the attack target node to access its services.

The existence of the above precondition would result in the occurrence of an unencrypted service. Thus, it is expected that the reasoner that is embedded in the SDN controller would catch the presence of LFA which will suggest that the investigated SDN

is vulnerable. This analysis would help SDN security overseers to discover framework vulnerabilities and shortcomings which may permit security infringement.

3.7 Performance metrics of Link Fabrication Attack in SDN

Various solutions have been proposed by different researchers for the detection of LFA in SDN. It is intended to create a high end secured SDN structure. If there should be an occurrence of LFA in SDN, the controller should be capable enough to trace back the origin of the attack. To keep the consideration of source identification of the attack in SDN, we identify performance metric for evaluating the strength of these solutions. These metrics includes

- Effectiveness (Processing time)
- Sensitivity
- Specificity
- False Alarm

3.7.1 Effectiveness (Processing time)

This is the response of the SDN controller against an occurrence of the LFA. It consists of the total application execution time $I\left(\frac{s}{t}\right)$ which comprises of Time to Detect an Attacker (TTDA), controller Response Time (RT), and Link Test Transfer Time (LTTT). Thus the total effectiveness can be mathematically represented as in equation 3.34.

$$effectiveness = TTDA + RT + LTTT \quad (3.34)$$

Thus, the total effectiveness can be formally defined using HOL as:

$$effectiveness = \left(\forall \left(I \left(\frac{s}{t} \right) TTDA \right) \wedge \left(\forall \left(I \left(\frac{s}{t} \right) RT \right) \wedge \right. \right. \\ \left. \left. \forall \left(I \left(\frac{s}{t} \right) LTTT \right) + ERR \right) \quad (3.35)$$

3.7.2 Sensitivity

This is a statistical measurement of the performance of a binary classification test. Sensitivity is sometimes also referred to as a classification function in statistics. In most fields, it is termed as the true positive rates or the recall. It measures the rate of positives that are correctly identified as positives. For instance, a percentage of the attack hosts in an SDN that are correctly identified as the host attackers. Mathematically, sensitivity can be expressed as follows.

$$\text{Sensitivity} = \frac{TP}{TP+FN} \quad (3.36)$$

Note that; a perfect predictor would be described as 100% sensitive if all elements in the data are identified as true (e.g., all attack host in SDN are identified as attackers), that is no host in SDN is identified as genuine. However, theoretically, any predictor will possess a minimum error bound known as the Bayes Error Rate (BER). Thus, sensitivity can be further mathematically expressed as follows.

$$\text{Sensitivity} = \frac{TP}{TP+FN} + BER \quad (3.37)$$

Where (BER) is the Bayesian error rate and can be mathematically expressed as

$$BER = \sum_{c_1 \neq c_z} \forall x \in H_i \int p(x|c_i)p(c_i) dx \quad (3.38)$$

Where (x) is an instance, (c_i) is a class into which an instance is classified, (H_i) is the area/region that a classifier function (h) classifies as (c_i).

Thus, we define sensitivity using HOL as:

$$\text{sensitivity} \stackrel{\text{def}}{=} \forall \left(\frac{TP}{TP+TN} \right) + \exists \left(\sum_{c_1 \neq c_z} \forall x \in H_i \int p(x|c_i)p(c_i) dx \right) \quad (3.39)$$

3.7.3 Specificity

This is also a statistical measurement of the performance of a binary classification test. Specificity, in most fields, is termed as the true negative rate. It measures the rate of negatives that are correctly identified as negatives. For instance, the percentage of the non-attacker hosts in an SDN that are correctly identified as the non-attacker hosts.

Mathematically, specificity can be expressed as:

$$Specificity = \frac{TN}{TN+FP} \quad (3.40)$$

We also note that a perfect predictor would be described as 100% specific just as in the case of specificity. Therefore, theoretically, any predictor will possess a minimum error bound known as the (BER). Thus, specificity can be further mathematically expressed as follows.

$$Specificity = \frac{TN}{TN+FP} + BER \quad (3.41)$$

Where (BER) is the Bayesian error rate which is mathematically expressed in equation 3.38 such as follows.

$$BER = \sum_{c_1 \neq c_z} \int_{\forall x \in H_i} p(x|c_i)p(c_i) dx$$

Where (x) is an instance, (c_i) is a class into which an instance is classified, (H_i) is the area/region that a classifier function (h) classifies as (c_i).

Thus, we define specificity using HOL as:

$$specificity \stackrel{\text{def}}{=} \forall \left(\frac{TN}{TN+FP} \right) + \exists \left(\sum_{c_1 \neq c_z} \int_{\forall x \in H_i} p(x|c_i)p(c_i) dx \right) \quad (3.42)$$

3.7.4 False Alarm Rate

The False Alarm Rate (FAR) which is also referred to as the false positive rate is the expectancy of the occurrence of false positives during a statistical inference. The FAR usually obtained when a result indicates that a certain condition has been fulfilled, which is not true, i.e., a presumed erroneous positive effect has occurred. The FAR can be mathematically expressed as

$$FAR = \frac{FP}{FP+TN} \quad (3.43)$$

Thus, we define FAR using HOL as follows.

$$FAR \stackrel{\text{def}}{=} \frac{\forall FP}{\forall FP \wedge \forall TN} \quad (3.44)$$

3.8 Conclusion

In this chapter, we used HOL as formal methods to representation the LFA in SDN. To better understand the LFA attack, we represent the key components of SDN formally by focusing the entire system liability, service information, dependencies, and switch as well as host vulnerabilities. Formal methods assist network operators in protecting the systems or network from unexpected errors that might not be detected at the time of development and deployment. Therefore, we have used HOL as a formal specification to represent the SDN infrastructure. Moreover, we formally represent of network devices and how they are connected in SDN.

After formally representing SDN infrastructure, we have represented LFA through HOL and illustrated with an example. Moreover, we have identified important performance metrics of LFA in SDN and represent them in HOL formal methods. These performance metrics will help us to evaluate our proposed method while comparing with state-of-the-art LFA detection solutions.

Therefore, we proposed forensic investigation method that aims to minimize the processing time of the controller and increases the accuracy of detecting fake links. The proposed method triggers the alarm messages dynamically upon observing the suspicious links and start investigating the behavior of fake links while identifying the real source of the attack.

CHAPTER 4: FORENSIC INVESTIGATION METHOD (FOR-GUARD)

This chapter presents a forensic investigation method (FoR-Guard) to identify and investigate fake links in SDN. The proposed method detects the fake links before it is updated in the controller topology database. The FoR-Guard uses various mathematical techniques to detect the fake links and verifies its sources in SDN. The FoR-Guard is divided into three sequential phases. Each phase of the FoR-Guard is briefly explained and provided with algorithms to elaborate its working steps. The first phase is used to trigger the detection phase after observing the malicious behavior of the switch at the time of link insertion. The second phase deals with finding the status of the link either it is legitimate or fake link. It also identifies the source of the LFA by using a traceback mechanism. The third phase verifies the source of a fake link by using entropy measurement concept adopted from information theory.

This chapter is organized into four sections. Section 4.1 starts with an overview of the proposed method regarding SDN. It followed the design principle of the proposed method in Section 4.2, where we provide a description of the state transitions along its event generation for each phase. Section 4.3 elaborates the proposed method with detail description of each of its phase. Also, we explain each phase along with its algorithms to have an insight of its working steps. Finally, this chapter is concluded in Section 4.4.

4.1 Overview of FoR-Guard

The SDN controller is responsible for discovering a network topology of the entire network. However, the attacker aims to generate fake links in the network to disturb the network visibility of the controller (Dhawan et al., 2015). As a result, topology-dependent applications in the application plane that runs on top of the controller results with falsified outputs in SDN. To overcome this problem, we have proposed a forensic investigation method (FoR-Guard) to identify and investigate fake links in the network topology of SDN. The FoR-Guard uses various methods of mathematical sciences that include graph

theory, adjacency matrix, and entropy measurement to detect and investigate fake links. The FoR-Guard assists the controller in finding a source of fake links by tracing back the port of the switch in SDN. The FoR-Guard is composed of three main phases namely, trigger phase, Detection and Source Identification (DeSI) phase, and validation phase as depicted in Figure 4.1. Each of the phases contains different application modules running on the controller. The FoR-Guard assumes that all the proposed modules for each phase are safe from all types of security threats.

The trigger phase of FoR-Guard has two goals to achieve that include (a) discovery of the network topology after every pre-defined time interval and (b) trigger of the DeSI phase. To discover the network topology, FoR-Guard used a de-facto standard of the SDN topology discovery mechanism while to trigger the DeSI phase, the historical record of the switch is checked that have generated fake links. The trigger phase consists of two modules known as Adaptive Trigger Manager (A-TrgM) and Malicious Link Arbitrator (MLA). Both of these modules integrate together to generate a trigger message to the DeSI phase for investigating the link.

The DeSI phase investigates the link upon receiving the trigger message from the trigger phase. The main objective of this phase is to investigate and detect the fake link generated through the malicious switch in SDN. Moreover, it traces back the malicious switch by finding the cause of the attack. The proposed traceback mechanism assists the controller in identifying fake links generated from the malicious switches. This phase consists of four main modules which integrate together to perform detection and source identification of the fake link. These modules include Topology Converter (TC), Link Handler (LH), Link Manager (LM), and trace back. The TC module is responsible for converting network graph to the adjacent matrix and storing it in a database for future investigation purposes. The LH module checks the type of the link either it is symmetric or asymmetric. The type of link information is required to identify the link behavior either

it is a legitimate or fake link. The LM module is considered to be the main module of this phase which detects and declares a fake link upon receiving the information from the LH and MLA modules. The traceback module is responsible for finding the real source of the attack by using the LLDP record stored by MLA module during the topology discovery process in SDN.

The validation phase assists the controller in verifying the real source of the fake link using entropy measurement concept adopted from the information theory. The entropy helps to measure the randomness created through fake links in the network. This phase has a module called Source Verifier (SV) which extracts the information from the TC module and performs Shannon entropy on it to verify the real source of the fake link identified at the DeSI phase.

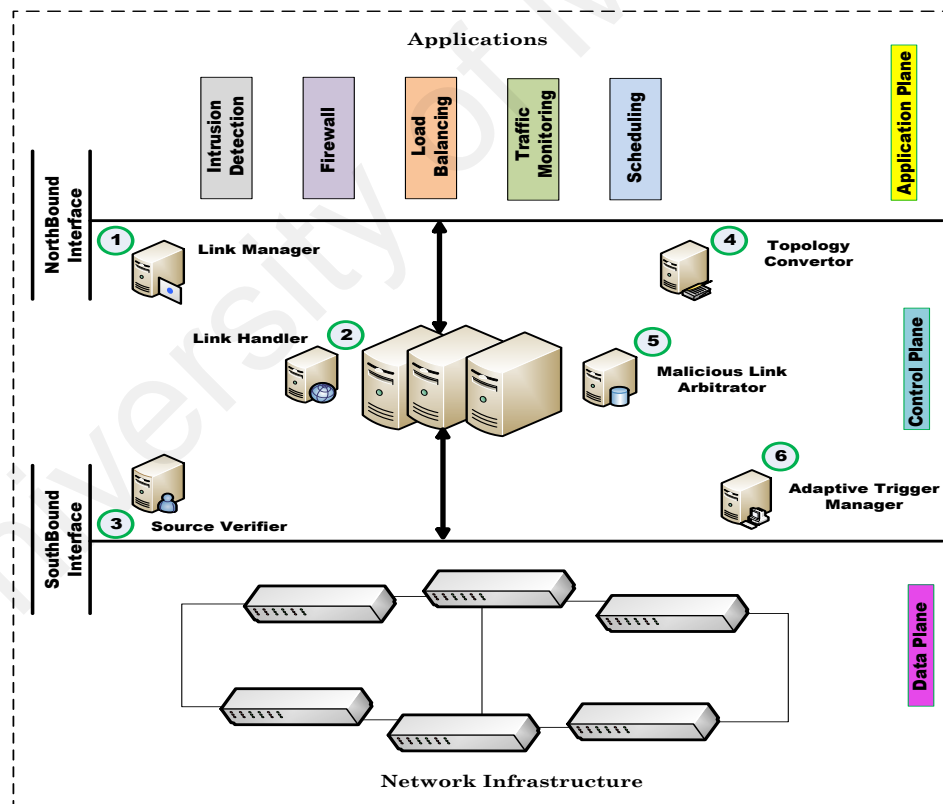


Figure 4.1: The integration of proposed modules in SDN architecture

4.2 Design principle of FoR-Guard

In this section, we present a state diagram to explain the design principle of FoR-Guard. There are three states and five events which describe the working steps of our

system as depicted in Figure 4.2. The states represent the main phases of our system whereas; events represent the transition between different states. The three main states used in our system are a trigger, DeSI, and validation. The description of each state is as follows.

4.2.1 FoR-Guard states

In this section, we present the states of FoR-Guard by describing its individual role and transition from one state to another state. Each state along its event is illustrated in Figure 4.2.

Trigger State: When the new link is generated between the switches, the system is said to be in a trigger state. This state is responsible for providing correct information about the new link to the topology manager module executing in the controller. The topology manager is responsible for discovering the network topology and providing an abstract view of the controller that further assists topology-dependent applications running on top of the controller. The trigger state remains in its state unless and until the legitimate switch generates the link. However, the trigger state changes to DeSI state when the new link is generated by the malicious switch in SDN. The event generated upon the state transition is explained in Section 4.2.2.

DeSI State: Upon receiving the trigger message from the trigger state, the investigation process of the DeSI state begins to identify the status of a new link such as a legitimate or fake link. The investigation process of a new link is performed because the correct topology information needs to be modified by the topology manager in the controller during every topology discovery round. The system remains in DeSI state until it does not find either the link is legitimate or fake. It integrates the historical fake link generation information of the switch and the communication direction of a new link to detect the link status. After the investigation and detection of the link, the source of the fake link is identified. It helps the controller to prevent the fake link generated again from

the same port of the switch. The state is changed to the validation state when the source of the fake link is identified. The events generated by the transition of DeSI state is explained in Section 4.2.2.

Validation State: It is the final state of our system that computes the uncertainty presented in the network topology that is generated through fake links. The computation of uncertainty and the declaration of a threshold value assist the controller in validating the real source of the fake link that has been previously identified in the DeSI state. However, if the source is not validated, then the validation state performs a transition to the trigger state mentioning that the source has not been found. The event generated upon the transition from the validation state to DeSI state is explained in Section 4.2.2.

4.2.2 FoR-Guard Events

In this section, we explain the events generated upon the transitions from one state to another. In our proposed method, there are five events which are generated based on the transition between the states. Each of the events is described as follows.

(a) Event-1 (Trigger): The trigger event occurs when a fake link is created in the network topology. The trigger event informs the DeSI state that the link has been generated by the switches in the network, and it needs to be investigated.

(b) Event-2 (Detection failure): The detection failure event is generated from the DeSI state to the trigger state when the fake link is not detected. It confirms that the new link generated between the switches is a legitimate link.

(c) Event-3 (Detection success): The detection success event occurs when the link is detected as a fake link, and its source is identified. This event is generated between DeSI and validation states. The information is forward to the validation state to confirm the source of a fake link.

(d) **Event-4 (Validation failure):** The validation failure event is generated from the validation state to the DeSI state. The validation state informs the DeSI state that the source identified is not the real source of the LFA.

(e) **Event-5 (Validation success):** The validation success event occurs when there is a transition between validation and trigger states. The event is generated when the source identified by a state validation tally with the source identified in the DeSI state.

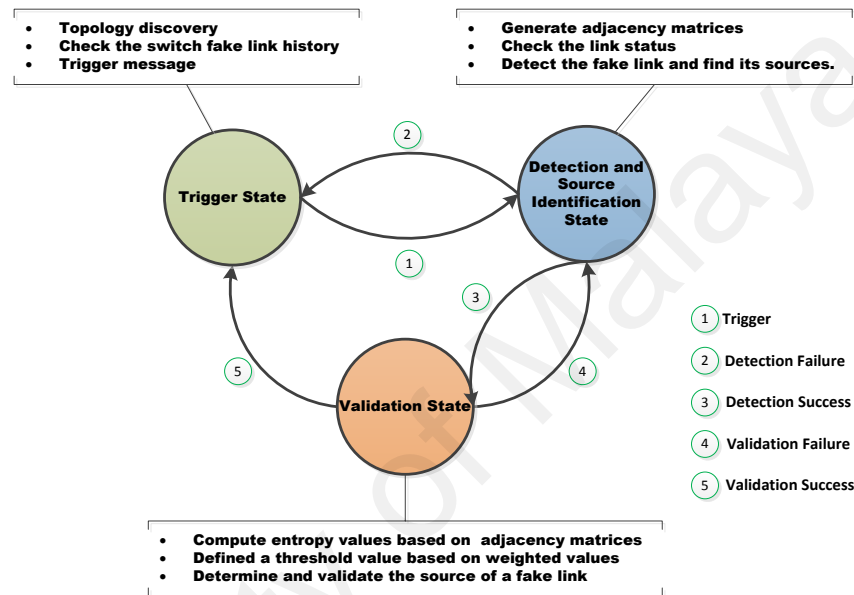


Figure 4.2: The state transition diagram of FoR-Guard

4.3 Forensic Investigation Method (FoR-Guard)

We proposed a forensic investigation method (FoR-Guard) to detect and determine fake links in the network topology which may cause a LFA. The FoR-Guard has three phases which execute in a sequence. Each phase has a main contribution in detecting and determining the fake links in the network topology along with finding the cause of the attack. Each of the FoR-Guard phases is comprehensively described and explained through algorithms. The symbol descriptions used in various FoR-Guard algorithms are shown in Table 4.1.

4.3.1 Trigger Phase

The trigger phase is an initial and important phase of FoR-Guard that discovers the network topology along with triggering the detection mechanism in the DeSI phase of

FoR-Guard. The trigger phase integrates a triggering mechanism with a de-facto standard topology discovery mechanism of SDN. This integration assists the SDN controller to have an abstract view of the entire network. Furthermore, the trigger phase triggers the next phase (DeSI) of FoR-Guard based on the observation of the malicious behavior of the switches.

The aim of the trigger phase is of twofold such as a) discover the network topology and b) trigger the detection mechanism in the next phase upon observing the malicious behavior of the switch regarding fake link injection in the network topology. The discovery of the network topology is performed through LLDP packets. The controller sends the LLDP packet to the switch with its DPID and port ID. The switch receives the LLDP packet from the controller and inserts its information including DPID and forwarding port ID and forwards it to the neighbor connected switches. The neighbor switches upon receiving the LLDP packet forward the same packet to the controller by inserting its identities such as DPID and incoming switch port ID. The LLDP packet is encapsulated in the *Packet_In* message before forwarding it to the controller. The controller checks the LLDP packet and observes that the packet arrives from another switch. The controller extracts information from the received *Packet_In* message and assumes that there exists a link between both the switches. The controller updates its network visibility accordingly.

The current SDN controllers do not know about the status of the link either it is legitimate or fake links due to lack of an authentication mechanism to verify the links. The controller normally updates the newly generated link in the topology upon receiving the information through LLDP packets. However, the link can be legitimate link generated for the purpose of determining the shortest routing path, minimizing latency, and maximizing throughput. Moreover, the link can be a fake link used to exploit switches

and hosts in the network to extract information, forge network traffic, and divert network traffic to the longest routing paths before reaching the destination.

The trigger message in the trigger phase has a great impact on the performance of the DeSI phase of the FoR-Guard. The early trigger of the DeSI phase could prevent the attack from exploiting the network at its initial stage of its occurrence. However, if the DeSI phase is triggered late to inform about the malicious switch found in the network topology, it may generate numerous fake links which might affect the topology-dependent applications using network topology for their execution. Therefore, it is utmost important to have real-time triggering mechanism in the network that triggers the DeSI phase suddenly after observing the malicious behavior of the switches in the network. To overcome this problem, we have proposed a novel adaptive trigger mechanism that contains A-TrgM module to trigger the DeSI phase based on the switching behavior.

We assumed that the switch has two states such as legitimate and malicious states. With a passage of a time the switch can change from one state to another state. The transition of the switch states depends on the generation of the links in the network. The switch is considered to be in a legitimate state or approaching to a legitimate state when it starts to create legitimate links whereas, the switch is in a malicious state or approaching towards the malicious state when it starts to generate fake links in the network.

Based on the switch states information, A-TrgM decides to trigger the DeSI phase. To determine states of the switches, Malicious Link Arbitrator (MLA) which is the main module executing in the controller is proposed to maintain a Malicious Index Record (MIR) for each of the switches as well as LLDP packets in the network. The MLA records MIR value for each switch which joins the SDN. The MIR value is maintained by taking into consideration the historical information of the links generated by the switches. The switch that generates more fake links will have higher MIR value and vice versa. When

the switch generates a new link, MLA checks the MIR value of the switch before informing to the topology manager module in the controller.

Table 4.1: Symbol descriptions of FoR-Guard algorithms

Symbols	Descriptions
$S(i)$	Number of switch/es in SDN
$L(i)_{0,1}$	Link between the switches, i.e., $S_1 \leftarrow S_0$
F_i	Malicious index record of switch $S(i)$.
C_{LM}	Link manager module in DeSI phase
MLA	Malicious link arbitrator module in trigger phase
$L_F(i)_{0,1}$	Link frequency of the link between the switches, i.e., $S_1 \leftarrow S_0$
TC	Topology controller module in DeSI Phase
P.adjM	Previous adjacency matrix
C.adjM	Current adjacency matrix
S.sw	Source switch
D.sw	Destination switch
Ctr	Controller
W.adjM	Weighted adjacency matrix
LH	Link handler
$\bar{S}(i)$	Opponent switch
src.idtf	Source identification
Sy	Symmetric
Asy	Asymmetric
$E(n)$	Entropy of value (n)
$P(n)$	Probability of value (n)
θ_m	Minimum threshold limit
θ_n	Maximum threshold limit

Higher the MIR value of the switch; more malicious is the switch due to more generation of fake links and lower is the MIR value; less malicious is the switch due to a high number of legitimate links. At the initial deployment of the network topology, MIR value for each of the switch is equal to zero (0), as no links are generated between different switches. The MIR value increases every time by one (1) when a fake link is generated between two or more switches in the network. However, the legitimate link can also be generated dynamically in SDN due to the frequent migration of network instances. The MIR value decreases every time by one (1) when the malicious switch starts reacting normally and generates legitimate links. Once the link is declared legitimate by the LM

in the DeSI phase of FoR-Guard, LM sends the information back regarding the link to MLA for updating the MIR value of the switch. The MIR value decreases maximum to the value equal to zero (0). The value zero shows that the switch is capable and trustworthy enough to generate a legitimate link in the network. The MLA keeps a record of each LLDP packet generated from the source switch towards the destination switch. Such information is necessary for the traceback algorithm (algorithm 5) to trace back the attack source. The working step of traceback algorithm is explained in DeSI phase of FoR-Guard. Moreover, the working steps of the MLA module are presented in the algorithm 1.

<p>Algorithm 1: - MLA: Compute the switch status Input: The switch and its link generation record. Output: To maintain and find status of the switch</p>
<ol style="list-style-type: none"> 1. Select switch $S(i)$ 2. If $links\ S(i) = (L(i)_{0,1} > 0)$ then 3. <i>check status</i> F_i 4. <i>record</i> $F_i \leftarrow value$ 5. If $L(i)_{0,1} \leftarrow new\ link$ then 6. request info $C_{LM} \leftarrow L(i)_{0,1}$ 7. receive $C_{LM} \leftarrow info$ 8. If $L(i)_{0,1} \leftarrow fake\ link$ then 9. increment $F_i = F_i + 1$ 10. <i>set status</i> $S(i) = malicious\ switch$ 11. else If $L(i)_{0,1} \leftarrow legitimate\ link$ then 12. decrement $F_i = F_i - 1$ 13. else if $F_i > 0$ then 14. <i>set status</i> $S(i) = malicious\ switch$ 15. else $F_i = 0$ then 16. <i>set status</i> $S(i) = legitimate\ switch$ 17. else $L(i)_{0,1} \leftarrow no\ new\ link$ then 18. unchanged $F_i \leftarrow value$ 19. else $links\ S(i) = (L(i)_{0,1} = 0)$ then 20. <i>set status</i> $S(i) = legitimate\ switch$

The MLA algorithm explains that how the MIR value of the switch is increased or decreased determining states of the switch. The MIR values depend on the link

information provided by the LM module running in the DeSI phase of FoR-Guard proposed method.

The trigger phase builds and updates the network topology accurately without inserting fake links. To determine fake links, one has to investigate the switch status. To do this, MLA computes the MIR value for each of the switch in the network as shown in the algorithm 1. The algorithm 1 presents that if a switch has some links, i.e., greater than 0 than the MIR value is recorded as shown with a symbol F_i (lines 1-4). Moreover, the switch status is computed by requesting information about the link status from an LM module running in the controller. (lines 5-7). Whenever the link is declared as a fake link than MLA increments the counter F_i by 1 such as $F_i = F_i + 1$ and the switch status is set as a malicious (lines 8-10). However, if the link is declared as a legitimate link than the counter of F_i value is decremented by 1 such as $F_i = F_i - 1$ (lines 11-12). Also, after decrementing the value of F_i , still if the value of F_i is greater than 0 than the switch status is set as a malicious (lines 13-14) otherwise if the value of F_i is equal to 0 than the switch status is set as a legitimate switch (lines 15-16). Furthermore, initially, if the switch has no links than it is considered to be a new switch in the network and its status set to a legitimate switch (lines 17-20).

The module A-TrgM triggers a message for a new link to the DeSI phase based on MIR value such as F_i . The working steps of the A-TrgM module are shown in the algorithm 2 as follows.

The A-TrgM algorithm checks when to trigger a message to the DeSI phase for investigating a new link inserted in the network. Initially, a switch which generates a link is investigated based on its previous link record (lines 1-3). For investigating the switch, F_i value of the switch is checked which is obtained from the MLA module (lines 4-6). The F_i value is checked if it is greater than 0 then a trigger message is send to the DeSI phase to investigate the link before updating it in the network topology database of the

controller (lines 7-9). However, if the F_i value is equal to 0, then the link frequencies are checked for a switch in the last 4 seconds. The 4 seconds have been selected based on the default topology discovery process in SDN. Mostly, controllers update its topology after 5 seconds. Thus, we keep it less than the default value and higher than its half value, i.e., $[4 > (5 / 2)]$.

<p>Algorithm 2: - A-TrgM: Trigger a message based on the switch status Input: The new link between the switches. Output: To trigger an alarm message to the DeSI phase</p>
<ol style="list-style-type: none"> 1. for switch $S(i)$ 2. If $link\ S(i) = (L(i)_{0,1} \leftarrow new\ link)$ 3. <i>investigate status</i> $S(i)$ 4. <i>check</i> $F_i \leftarrow value$ 5. request info $MLA \leftarrow F_i$ 6. receive $MLA \leftarrow info$ 7. If $value\ F_i > 0$ then 8. trigger <i>send.message</i> ($$) 9. <i>message</i> (<i>investigate</i>($$).<i>link</i>) 10. else if $value\ F_i = 0$ then 11. set threshold $value=4$ 12. <i>check frequency</i> $S(i) \leftarrow L_F(i)_{0,1}$ 13. else If $frequency\ S(i) \leftarrow (L_F(i)_{0,1} \geq 3)$ then 14. trigger <i>send.message</i> ($$) 15. <i>message</i> (<i>investigate</i>($$).<i>link</i>) 16. else $frequency\ S(i) \leftarrow (L_F(i)_{0,1} < 3)$ then 17. send.info (<i>controller</i>) 18. update <i>set.updatetopo</i> ($$) 19. else $link\ S(i) = (L(i)_{0,1} \leftarrow no\ new\ link)$ 20. send.info (<i>controller</i>) 21. <i>set.topo</i> (<i>same</i>)

We have defined a threshold value for link frequencies of the switch equal to 3. As mentioned above, if the F_i value is equal to 0, then the switch link frequency (L_F) is checked to decide that a trigger message is sent to the DeSI phase or not. The link frequency of a switch assists the controller to know how frequent a switch is generating a link with other switches in the network. Usually, a switch in a malicious state generates a frequent number of links within a short period to exploit the network. If the link frequency of a switch is greater or equal to 3, then a switch is considered to be in a

malicious state and a trigger message is generated to the DeSI phase to investigate the link (lines 10-15). However, if the link frequency is less than 3 then the switch is considered to be in a legitimate state which does not require to trigger a message and the network topology is updated in the controller database accordingly (lines 16-18). Also, if there is no link generated between the switches within topology discovery round, i.e., 5 seconds, the controller keeps the same current topology with itself (lines 19-21).

Therefore, A-TrgM dynamically triggers the DeSI phase so that the controller investigates a fake link at its early stage of creation with a less computational overhead. However, if the trigger mechanism is not adaptive than it may require more computation and time overhead to detect and investigate the fake links due to instance migration of adversary, anti-forensics mechanisms, and falsified impact on the network.

4.3.2 Detection and Source Identification (DeSI) Phase

The A-TrgM module in the trigger phase triggers a message to the DeSI phase to investigate the new link generated in the network. The DeSI is the second phase of FoR-Guard that aims to investigate the status of a new link either it is a legitimate or fake link. In FoR-Guard, the network topology is represented by a directed graph. Each vertex represents a switch, and an edge represents a link between the switches in the directed graph. The two vertices V_i and V_j of a directed graph are adjacent if there exist an edge from V_i to V_j or from V_j to V_i . For a link from V_i to V_j , then V_i and V_j are the endpoints and V_i is a tail and V_j is ahead of the edge. The directed graph can be represented using an adjacent matrix to show a direct connectivity between the switches.

The adjacent matrix is one of the best choices to represent the network topology (directed or undirected graph). It provides a way to understand the connectivity between the switches. The adjacent matrix is a square matrix which represents a directed link between the vertices in the graph. The adjacency matrix rows and columns are labelled by graph vertices, with a value of 1 or 0 in a position (V_i, V_j) according to whether V_i and

V_j are adjacent or not. The value 0 indicates that there is no link between the switches while 1 indicates a link between the switches.

Suppose we have an adjacency matrix A with its values $a_{i,j}$ as follows.

$$a_{i,j} = \begin{cases} 1 & V_i \text{ and } V_j \text{ are adjacent} \\ 0 & \text{Otherwise} \end{cases}$$

The value 1 for an element $a_{i,j}$ of the adjacency matrix A shows that there exist a link between the switches whereas the value 0 for an element $a_{i,j}$ represents no link between the switches. The adjacency matrix in current shape cannot represent a link communication direction (LCD) status, i.e., either it is symmetric or asymmetric. To find the LCD status, we have proposed a weighted adjacency matrix called W -adjacency matrix. The W -adjacency matrix contains values $[0, 0.5, 1]$. The value in the W -adjacency matrix represents the LCD status along with its adjacency between the switches. The LCD is important in the investigation of the link behavior in SDN, i.e., either it is a legitimate or fake link.

Suppose we have a W -adjacency matrix B with its values $b_{i,j}$ as follows.

$$b_{i,j} = \begin{cases} 1 & V_i \text{ and } V_j \text{ are adjacent, and the link is symmetric.} \\ 0.5 & V_i \text{ and } V_j \text{ are adjacent, and the link is asymmetric.} \\ 0 & \text{otherwise} \end{cases}$$

The value 1 for an element $b_{i,j}$ of the adjacency matrix B shows a symmetric link whereas the value 0.5 shows an asymmetric link, and the value 0 shows no link between the switches.

As discussed above, the LCD status is an important factor to know about a link, i.e., fake or legitimate. When the switch generates a fake link with another switch for an illegal purpose in SDN, a link is usually generated as an asymmetric link, i.e., a direct link from the malicious switch to the legitimate switch. For instance, in a network topology shown in Figure 4.3, the malicious switch $S-1$ wants to divert the network traffic from switch $S-$

2 to switch S-3 by using a fake link generated by switch S-1 with switch S-3. Figure 4.3 illustrates that a fake link between switch S-1 and switch S-3 is asymmetric. Therefore, a host-1 attached with the switch S-1 can ping the host-3 attached with the switch S-3 but cannot receive a response to the ping messages accordingly.

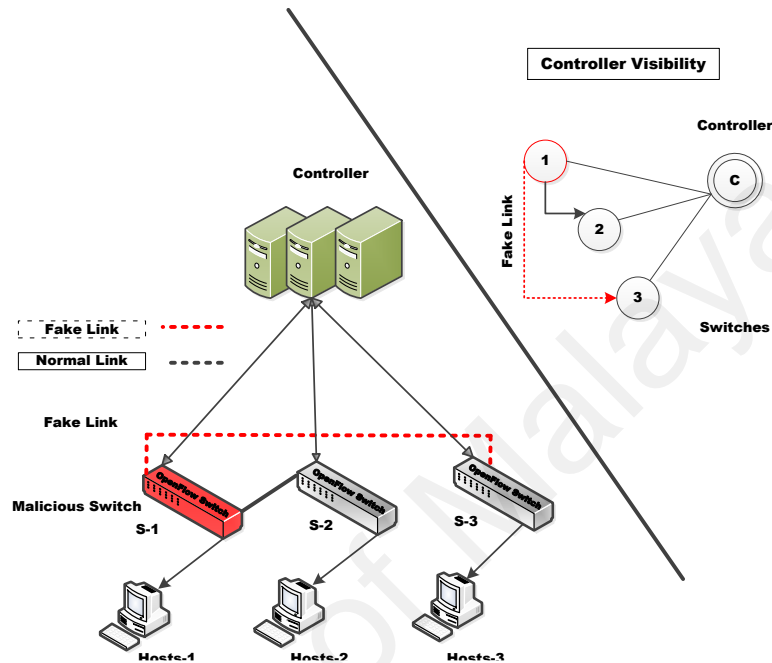


Figure 4.3: An illustration of a fake link between switches

Initially, when the DeSI phase receives a trigger message from the A-TrgM module of the trigger phase for a topological change (new link); the module called Topology Controller (TC) received the trigger message and informed the Link Handler (LH) module in the controller. The role of a TC module is to generate an adjacency matrix for the current network topology and store it for a future reference. Based on the network information collected from the trigger phase, TC generates an adjacency matrix for the network graph to have better understandability of the links. The TC generates an adjacency matrix for the every update of the network topology. The LH module collects the current and previous adjacency matrix from the TC to confirm which switch has created a link in which location of the network.

Algorithm 3: - LH: Determine the link communication direction (LCD)

Input: The new link between the switches.

Output: Direction of the communication on the new link

```
1. If link S(i) = ( L(i)0,1 ← new link ) then
2.   request info TC ← [P.adjM, C.adjM]
3.   receive TC ← info
4.   compare matrix [P.adjM, C.adjM]
5.   record change [P.adjM, C.adjM] = success
6.   select link S.sw && D.sw
7.   request install Ctlr ← flow rules
8.   Ctlr ← install.flow rules ( )
9.   generate test.traffic ( )
10.  if (response ( ) = success) then
11.    L(i)0,1 ← symmetric link
12.    record status ← L(i)0,1
13.    update W.adjM (status, value=1)
14.  else (response ( ) = fail) then
15.    L(i)0,1 ← asymmetric link
16.    record status ← L(i)0,1
17.    update W.adjM (status, value=0.5)
18. else link S(i) = ( L(i)0,1 ← new link ) then
19.  request info TC ← [P.adjM, C.adjM]
20.  receive TC ← info
21.  compare matrix [P.adjM, C.adjM]
22.  record change [P.adjM, C.adjM] = fail
23.  update info Ctlr ← link.info = wrong
24.  set.investigation(stop)
```

The role of an LH module is to find and record LCD status of the new link. The switch that has generated a link is identified by comparing the previous and current adjacency matrices. Once the switch is identified, the LH instructs the controller to install the temporary flow rules in the flow tables of both the switches connecting the new link. The flow rules are installed in the flow tables of both the switches to generate test traffic on the new link. The idle time for each flow rules is set to 1 second which is kept small due to temporary rules installed in the flow tables of the switches. The test is conducted to know about the LCD status of the new link generated in the network. The test packets are transferred on the link from a source switch to the destination switch of the link and waits for a response to the packets.

In the case of a fake link, we have observed that no response from the destination is received which shows that the link status is asymmetric. However, in the normal situation, the response is received from the destination switch which indicates that a new link is symmetric in nature. After finding the LCD status, a W-adjacency matrix is generated by the LH for the specified switch with its LCD values. Till this moment, the LH finds either the link is symmetric with its value 1 or asymmetric with its value 0.5. The complete working steps of the LH module are shown in the algorithm 3 as follows.

The algorithm 3 explains that upon a new link generation; LH module is informed to investigate its behavior; it first requests TC to send the current and previous adjacency matrices of the network topology to confirm the switch which has generated a link (lines 1-4). The LH compares both the adjacency matrices for a change value. The change of a value determines the switch which has a connection with a link. The LH informs the controller to install flow rules between the switch connecting the link to send a test message to find the LCD status (lines 5-9). If a response is received on the link for test messages than LH updates such information, i.e., value 1 in the W-adjacency matrix and declare the link as a symmetric link (lines 10-13). Otherwise, it declares as an asymmetric link and with an updated value of 0.5 in the W-adjacency matrix (lines 14-17). However, if LH compares the two adjacency matrices and there is no difference between the values, then the controller is informed that the trigger message generated by the A-TrgM has been wrong. The controller stops investigating the link furthermore (lines 18-24).

Algorithm 4:- LM: Investigate the new link status along its source identification

Input: The new link along with its LCD status.

Output: Determine the link status (fake or legitimate).

1. **receive.** *Info* (investigate)
2. link. *investigate* ()
3. **request** *info* MLA && LH
4. **receive** MLA \leftarrow *info*
5. **receive** LH \leftarrow *info*

Algorithm 4:- LM: Investigate the new link status along its source identification

Input: The new link along with its LCD status.

Output: Determine the link status (fake or legitimate).

```
6. record info  $F_i$  &&  $L(i)_{0,1} \leftarrow status$ 
7. If [ $S(i) \leftarrow (F_i = 0)$ ] && [ $L(i)_{0,1} \leftarrow status.Sy()$ ] then
8.   set link status (legitimate)
9.   update net.topo (link)
10. If [ $S(i) \leftarrow (F_i = 0)$ ] && [ $L(i)_{0,1} \leftarrow status.Asy()$ ] then
11.  update net.topo (null)
12.  check status  $\bar{S}(i)$ 
13. else if  $\bar{S}(i) \leftarrow (F_i > 0)$  then
14.   set link status (legitimate)
15.   update net.topo (link)
16.   else  $\bar{S}(i) \leftarrow (F_i = 0)$  then
17.     set link status (fake)
18.     update net.topo (null)
19.     traceback src.idtf [ $S(i), L(i)_{0,1}$ ]
20. If [ $S(i) \leftarrow (F_i > 0)$ ] && [ $L(i)_{0,1} \leftarrow status.Asy()$ ] then
21.  set link status (fake)
22.  update net.topo (null)
23.  traceback src.idtf [ $S(i), L(i)_{0,1}$ ]
24. If [ $S(i) \leftarrow (F_i > 0)$ ] && [ $L(i)_{0,1} \leftarrow status.Sy()$ ] then
25.  update net.topo (null)
26.  check status  $\bar{S}(i)$ 
27. else if  $\bar{S}(i) \leftarrow (F_i > 0)$  then
28.   set link status (fake)
29.   update net.topo (null)
30.   traceback src.idtf [ $S(i), L(i)_{0,1}$ ]
31.   else  $\bar{S}(i) \leftarrow (F_i = 0)$  then
32.     set link status (legitimate)
33.     update net.topo (link)
```

The algorithm 4 aims to investigate the behavior of a link as well as to find the source of a fake link. When LM receives a message to investigate a link, it requests MLA and LH to provide information regarding link status of the switches (source and destination switches) and LCD of the link respectively (lines 1-6). The LM investigates the switch based on information sent by the MLA and LH such as F_i and link status LCD. If $F_i = 0$ and LCD is symmetric, then a link is declared as a legitimate link because the switch has no previous fake link record and packets can be sent and receive on the link. The LM informs the controller to update a link in the network topology (lines 7-9). However, if F_i

= 0 and LCD status is asymmetric, then the LM informs the controller not to update a link in the network topology. The link status of the opponent switch $\bar{S}(i)$ is checked to investigate the behavior of the link further. If the opponent switches $F_i > 0$, then the link is declared as a legitimate link because a normal switch can make a link with a malicious opponent switch without knowing its status. It can happen due to the frequent migration of the virtual instances in the data centers. The LM informs the controller to update a link in the network topology (lines 10-15).

However, if the opponent switch $F_i = 0$, then the link is considered to be a fake link. The reason is that when a normal switch creates a link with a normal switch it has to be symmetric in nature but this case both the switches have $F_i = 0$, and the link is asymmetric. The opponent switch has not recognized the source switch. Therefore, the controller does not update this link in the network topology and start to trace back its source to prevent the switch from regenerating further fake links in the network (lines 16-19).

Similarly, the LM checks a condition for switches which have F_i value greater than 0, i.e., $F_i > 0$. If the value of $F_i > 0$ and a link status is asymmetric than LM, treat this link as a fake link. It is due to the switch has previously created more fake links as well as its link status is asymmetric in nature. So the controller not updates a link in the network topology and start to trace back the source of the link (lines 20-23). However, if the value of $F_i > 0$ and a link status is symmetric than the LM informs the controller not to update the network topology with the current link. A link is investigated based on its opponent switch status (lines 24-26). The status of the opponent switch is checked. If $F_i > 0$, then the link is declared as a fake link because both the switches are malicious, and the link is symmetric in nature. The controller starts to trace back the link to find its source (lines 27-30). However, if $F_i = 0$ than a link is declared as a legitimate link because the opponent switch has no previous fake link records and both the switches has recognized each other

by having a symmetric link between them. Thus, the source switch which has $F_i > 0$ is moving towards its legitimate state by generating a legitimate link (lines 31-33).

The algorithm 5 explains the traceback mechanism used to find the source of fake links after detecting them. The working steps of traceback algorithm are explained in algorithm 5. Once LM identifies the fake link, a traceback function is called to identify the source of fake links. The traceback algorithm starts executing by requesting LLDP record (TKR_{LLDP}) for the switches in SDN. The TKR_{LLDP} is maintained by MLA module running in the controller (lines 1-3). Once the TKR_{LLDP} information has received, traceback function starts traversing back the switches from the switch where the fake link is attached. The traversing start checking the similarity between current and previous LLDP information which is used by the controller to update the links between the switches (lines 4-6). If the switch has same LLDP information for its ports based on its previous and current status than traversing the source of the attack is continued. (lines 7-9). However, if the LLDP information is different for the port than the source switch (S_{src}) of the fake link is selected to investigate the number of hosts attached to the switch (lines 10-12). Afterwards, the traceback enters into the second level to start investigating the host to find the attacker (host).

There is the probability that the attacker (host) generates a fake link and start migrated to another location to hide its identity. If any host among the host list has changed its position, then its new location is tracked based on the host profile table maintained by the HTS (lines 13-16).

<p>Algorithm 5: - traceback: Traceback the real source of the attack Input: The fake link along with its direct connected switch. Output: Malicious host responsible for fake link generation.</p>
<ol style="list-style-type: none"> 1. info link status (<i>fake</i>) 2. request <i>info</i> TKR_{LLDP} 3. receive $TKR_{LLDP} \leftarrow info$ 4. traverse back (P_{rec}, C_{rec})

Algorithm 5: - traceback: Traceback the real source of the attack

Input: The fake link along with its direct connected switch.

Output: Malicious host responsible for fake link generation.

```
5. for (i=Sdes, i>=S0, i--)  
6.   check diff LLDP ← (Sdes, Ssrc)  
7.   If diff LLDP ==false then  
8.     Decrement by 1, i.e., (i --)  
9.     Continue;  
10.  else If diff LLDP ==true then  
11.    record diff (Sdes, Ssrc)  
12.    check Ssrc ← Host list  
13.    for (j=0, j<=n, j++)  
14.      If H [j] == position.change then  
15.        record H [j] ← new location  
16.        check H [j] ← LLDPrecord  
17.        If LLDPrecord == spoofed then  
18.          H [j] ← attacksrc  
19.        else LLDPrecord != spoofed then  
20.          H [j] ← Incorrect attacksrc  
21.      else H [j] != position.change then  
22.        check H [j] ← LLDPrecord  
23.        If LLDPrecord == spoofed then  
24.          H [j] ← attacksrc  
25.        else LLDPrecord != spoofed then  
26.          H [j] ← Incorrect attacksrc  
27. record attacksrc  
28. inform LM ← Info
```

Once the location is found, the LLDP information is checked either there exist any spoofed information or not. If the LLDP generated by the host is spoofed than the host is considered to be an attacker else it fails to find the attacker (lines 17-20). Similarly, if the hosts attached to the selected source switch (S_{src}) have not changed their positions than its LLDP record is checked. Any spoofed information found in the LLDP will lead to declare the host as an attacker otherwise the attacker is not found when there is any spoofed information in the LLDP packets (lines 21-26). Thus, upon finding the attacker by founding spoofed information in the LLDP packet, the attacker information is recorded and is forwarded to LM module for further necessary actions (lines 27-28).

4.3.3 Validation Phase

The link which is either legitimate or fake is detected in the second phase, i.e., DeSI phase of FoR-Guard. The LM in the DeSI phase assists the controller only in updating a link in the network topology which is declared as a legitimate link. The link which is affirmed to be fake link is prevented to be updated in the network topology during the topology discovery updates. It enables the controller not only to detect but also investigate the fake link to identifying the real source of a fake link, i.e., the malicious switch. The identification of the switch which helps to generate a fake link is important due to preventing the malicious switch from regenerating a fake link again. In DeSI phase, we have used a traceback algorithm to find the source of the link. However, in this phase, the concept of entropy is used to verify the source switch either it is a real source of the attack or not. For this, we use the information from the LH module which records the W-adjacency matrix for every link updated (insertion and deletion) in the network. The LH compares previous and current adjacency matrices to determine a new link location in the network.

The module known as Source Verifier (SV) is used in the validation phase to verify the source of a fake link (malicious switch). The SV first find the probability of the each switch based on their corresponding links obtain from previous and current adjacency matrices. It helps the SV module to know about the modification occurred in the network due to a new link between the switches after comparing the two probabilistic matrices. Based on the probabilistic values of both matrices, entropy is used to measure the average weight of the switches to detect a new link generation. The entropy is a concept used to determine a randomness of the network.

Algorithm 6: - SV: Validation of a source of the attack

Input: The previous and current adjacency matrices of the network.

Output: Verification of the attack source

1. info **link** status (*fake*)
2. **request** *info* LH

Algorithm 6: - SV: Validation of a source of the attack**Input:** The previous and current adjacency matrices of the network.**Output:** Verification of the attack source

3. **receive** LH \leftarrow *info*
4. **record** *matrix* [P.adjM, C.adjM]
5. **compute** *probability* [P.adjM, C.adjM]
6. **calculate** *entropy weight matrix* [P.adjM, C.adjM]
7. $E(n) = - \sum p(n) \log(p(n))$
8. **record** E(n)
9. set **threshold** *value* (θ_m, θ_n)
10. **If** *value* between (θ_m, θ_n) **such as**
11. $\theta_m \leq E(n) \leq \theta_n$ **then**
12. **exclude** switches
13. Investigate == null
14. **else** *value* not between (θ_m, θ_n) **such as**
15. $\theta_m \leq E(n) \leq \theta_n$ **then**
16. Investigate == yes
17. **select** switch
18. **compare** switch S(i) with a selected switch S(j) from LM
19. **if** [S(i) == S(j)] **then**
20. **source.validate** (*yes*)
21. **else** [S(i) \neq S(j)] **then**
22. **source.validate** (*no*)

We used entropy as a tool to verify the source of the fake link. The fake links created randomness in the network due to illegal connection with the various switches causing to disturb the network operation. Entropy is one of the best tools to measure the randomness occurred due to fake links in the network topology. Also, we have used a threshold value to discard normal links and focus on the fake links generated in the network. The complete working steps of the SV module in our validation phase are explained in the algorithm 6. The SV module uses Shannon entropy to measure the average weight of the switches which helps to determine the real source (malicious host) of fake links.

The algorithm 6 aims to validate the source of a fake link which has been identified in the DeSI phase of the FoR-Guard method. Once the fake link is detected and the source switch is identified, SV module starts executing its process by requesting LH module for information regarding the previous and current W-adjacency matrices of the network

topology (lines 1-2). The SV records the previous and current W-adjacency matrices values after receiving it from the LH module (lines 3-4). The probability for each switch for the previous and current W-adjacency matrices is computed based on the link values present in the W-adjacency matrices. Furthermore, a probability value of each switch is used to calculate the entropy value for previous and current W-adjacency matrices. (lines 5-8).

The entropy value assists the controller to know about the randomness presented after the switch has generated the fake link. Moreover, the min and max threshold value are defined within a range of entropy weighted value of the matrices to exclude the switches from the investigation list. The switches are excluded as they are considered to be legitimate switches. It reduces the number of switches to be investigated to reach the real source of the fake link, i.e., malicious host. If the weighted value of the switches lies between the min and max threshold value, then the switch is not considered to be the source of a fake link (lines 9-12). However, if a weighted value of the switches does not lie between the min and max threshold value than these switches are considered to be potential switches that may generate a fake link and can be investigated.

To find the exact switch among the number of switches which are highlighted through a threshold value, previous historical fake links of the switches are checked to verify a real source of the fake link, i.e., malicious host (lines 13-15). Once the source of a fake link is identified, it is compared with the source determine at the DeSI phase of FoR-Guard. If the source matches than it is verified that SV has identified the real source of the attack and if it does not match then the real source of the attack is not found (lines 16-20).

4.4 Conclusion

We propose a forensic investigation method to identify the fake links along determining its real source of the attack in SDN. The proposed method (FoR-Guard) used

three phases to investigate LFA in SDN. The trigger phase aims to trigger an alarm message to the DeSI phase upon observing the malicious behavior of the switches during the time of generating links in SDN. It reduces the computational overhead cost of the controller by enabling the controller to investigate only the links which are generated by the malicious switches. However, state-of-the-art techniques investigate all newly generated links to find out their behavior.

The DeSI phase detects the fake links by investigating the switching behavior and checking the communication direction of the links. Four modules are developed which runs in the controller to determine the fake links. Moreover, the DeSI phase identifies the real source of the attack by using a traceback algorithm. The validation phase verifies the source of the attack by measuring uncertainty in the network generated through fake links by using entropy concept adopted from the information theory. The FoR-Guard provides less controller processing time and high detection accuracy in determining the fake links in SDN as compared to the state-of-the-art techniques available for LFA.

CHAPTER 5: EVALUATION

This chapter discusses the data collection method for the effectiveness and evaluation of the proposed FoR-Guard method in SDN environment. The objective of this chapter is to explain the experimental setup and data collection techniques to evaluate our proposed method. The experimental setup discusses the Mininet emulation tool used to emulate the SDN environment; Floodlight controller used to have centralized control of the entire SDN environment, and Packit network auditing tool used to spoof LLDP packets for generating the fake links in SDN. The data is collected from different experiments to evaluate the processing time of the proposed FoR-Guard method in distinctive phases such as a trigger, DeSI, and validation. Also, the data is collected to measure the sensitivity, specificity, and false alarm rate of FoR-Guard method. Moreover, data collection for comparison of FoR-Guard with existing solutions regarding processing time is presented.

The chapter is organized into four main sections. Section 5.1 explains the benchmarking tools and controller used to run our experimental setup. Section 5.2 presents data collection methods to obtain data from different experimental scenarios to evaluate the performance of FoR-Guard method. The data is evaluated for processing time, sensitivity, specificity, and false alarm rate metrics. Section 5.3 presents the comparison data of FoR-Guard, TopoGuard, and Sphinx regarding their processing time in detecting a different number of fake links in SDN. Finally, Section 5.4 concludes the chapter.

5.1 Experimental tools and SDN controller

In this section, we explained the tools used to evaluate FoR-Guard method along with SDN controller used to provide centralized control of the entire network.

5.1.1 Mininet

The Mininet is an open-source network emulator that designs to support research in SDN (de Oliveira, Schweitzer, Shinoda, & Prete, 2014). Mininet enables realistic virtual network environment that runs real switches, kernel, and user code on a single machine. The single kernel runs hosts, switches, routers, and links in a lightweight virtualized environment to make a single machine look like a single entire network.

The host created in a Mininet is a real host where it can do what it can be done on the physical machine. A program can be installed that support the Linux kernel and can remotely connect the host system through *ssh* and various other operations. The only difference between Mininet network components and real network components lies at a difference of software and hardware. The Mininet network components are created through software while real network components are built through hardware. The functionality is same for both the network components.

Mininet is used to emulate real-world SDN scenario which helps to emulate our experimental setup to evaluate the proposed method, i.e., FoR-Guard. We have used the latest version of Mininet 2.2.1 to emulate our SDN experimental setup. Mininet allows us to create a customize network topologies from a single switch to a large data center networks by using few line of Python code. Figure 5.1 illustrates the Python code to create a single switch with ‘k’ number of hosts in the network. Moreover, Mininet assists in providing flexible network topologies which are configured based on different parameters and can be reused in different experiments. Table 5.1 depicts main functions used to create a network topology in Mininet emulation environment.

In our experimental setup, we have selected OF 2.3.1 switches instead of using Mininet switches to make the real OF switches. An OF 1.0 is selected for the control channel management as many controllers support it. We have not enabled TLS option in the

communication between OF switches and the controller to avoid the unnecessary complexity as it is not effected by the attacks.

```

class SingleSwitchTopo( Topo ):
    "Single switch connected to k hosts."
    def build( self, k=2, **_opts ):
        "k: number of hosts"
        self.k = k
        switch = self.addSwitch( 's1' )
        for h in range( 1, k ):
            host = self.addHost( 'host' % h )
            self.addLink( host, switch )

```

Figure 5.1: Python code for a single switch and ‘k’ number of hosts

Table 5.1: Basic topology functions with its descriptions

Topology Functions	Description
build()	This method is overridden in topo class by passing parameter through Topo.__init__()
addSwitch()	Insert the switch in the topology and return its name
addHost()	Insert the host in the topology and return its name
addLink()	Create bi-directional link in the topology
start()	It starts the network
stop()	It stops the network
pingAll()	It testifies the connectivity between all host in the topology
dumpNodeConnections()	Dumps connection between specified nodes

5.1.2 Floodlight Controller 1.0

Floodlight is a Java-based open-source SDN controller supported by a community of developers and network engineers from Big Switch Networks (Wallner & Cannistra, 2013). It is extensible Java development environment that provides an easy way for its used. Floodlight controller supports a broad range of physical and virtual OF switches which helps to run large experimental setups. We select Floodlight controller for our experiments as it is mostly used in the SDN security literature and due to its popularity in the market. The Floodlight controller is adopted by different research universities, network vendors, users, and developers. The complete solution of Floodlight controller

regarding the modular system, easy Web UI, and active research and developer community has attracted many researchers to use Floodlight controller for their developments and experiments.

The Floodlight controller uses a single shortest path between any source and destination OF switches in the OF Island. The OF Island is also known as OF cluster; that connects different OF switches to form a topology. Floodlight supports OF Island communication with a non-OF Island when it is connected only with a single link. There should be no loop between OF and non-OF Island. Figures 5.2 and 5.4 illustrate the situation of a connected OF Island with a non-OF Island through a single and double link respectively.

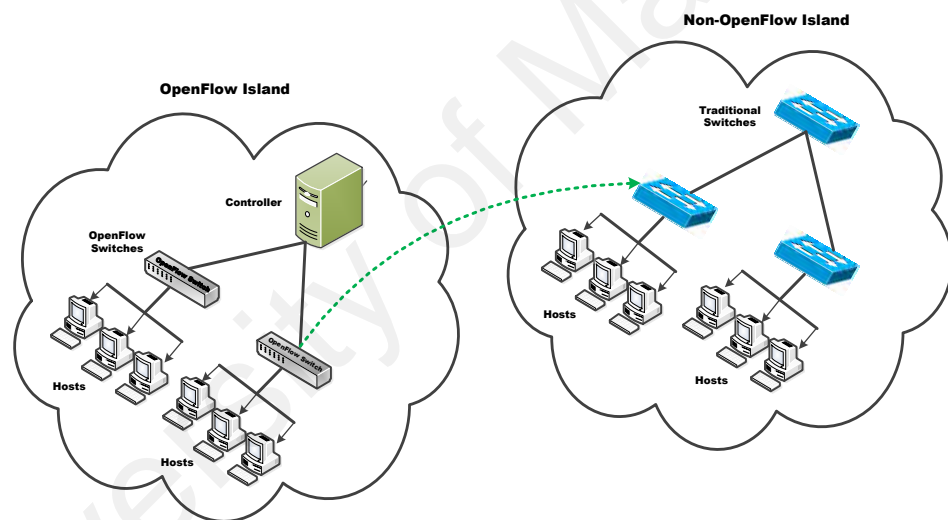


Figure 5.2: Floodlight controller supports communication with single link between OF Island and non-OF Island

In our experiments, we have used Eclipse to set up Floodlight controller with the OF network. It is easy to use Eclipse rather than manually installing the Floodlight controller in the virtualized network such as Mininet. We used 'ant' to import Floodlight package in the Eclipse other than Mininet environment. It provides an easy way to stop and run the controller during our experiments. Afterwards, running the Floodlight controller in the Eclipse, we have to connect the controller with the Mininet infrastructure. For this, we used a command in the Mininet as shown in Figure 5.3.


```
sudo mn --controller=remote, ip=<controller ip>,port=6653 --switch
ovsk,Protocols=OF10
```

Figure 5.3: The connection of remote controller to the Mininet infrastructure

The Mininet network infrastructure connects with the Floodlight controller through its remote IP address. Moreover, we can easily import our proposed modules in the Floodlight controller. After developing modules, we have to upload it in the starting modules stack of the Floodlight controller such as *'net.floodlight.core.module.IFloodlightModule'*. The starting module stack allows executing our modules in the Floodlight controller.

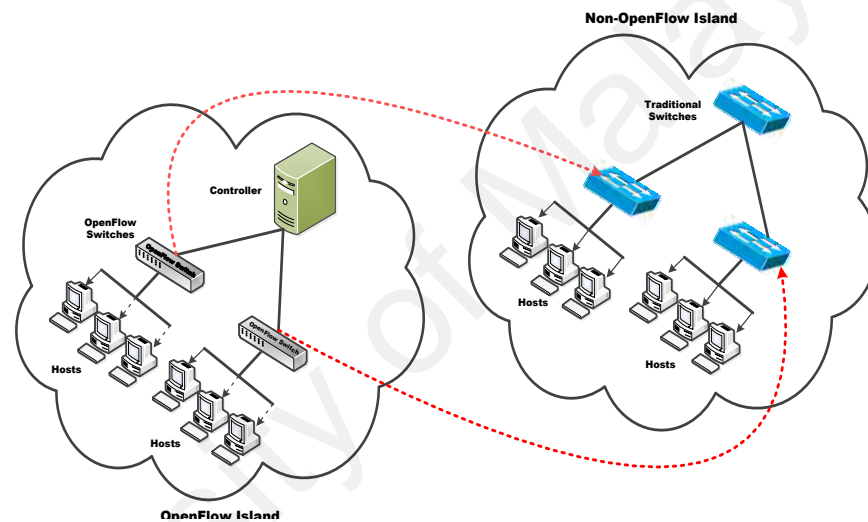


Figure 5.4: Floodlight controller does not support communication with a loop between OF Island and non-OF Island

5.1.3 Packit Tool

The Packit is a network analysis and auditing toolkit. It is famous for spoofing the packet header of the packets in the network traffic (Darren, 2016). Packit used to monitor, inject, and manipulate the network traffic. Packit toolkit is mostly used to testify intrusion detection systems, firewalls, proxies, security tests, port scanners, and IP auditing systems. We used Packit toolkit to spoof the LLDP packet. The spoofed LLDP packet provides falsified information to the controller which further creates a fake link between the switches.

In the process of creating a fake link between the switches, we used a malicious host to spoof LLDP packet by using Packit toolkit. We modify the fields in the LLDP packet such as DPID value of the switch. The Port ID TLV field can also be used to spoof the LLDP packet to create a fake link between the switches. For instance, instead of port # 2, value port # 3 is inserted in the field of Port ID TLV field of the LLDP packet.

5.2 Performance Analysis

This section explains the metrics that are applied to examine the efficiency of our proposed solution (FoR-Guard), such as processing time, sensitivity, specificity, and false alarm rate.

5.2.1 Confidence Interval of the Data Collection

There are different intervals used in statistics to characterize the results. However, we have used a confidence interval based on our experimental data. A confidence interval provides a range of values obtain from the sample statistics that tend to represent the value of an unknown population size (Nakagawa & Cuthill, 2007). The confidence interval shows the uncertainty and precision of a selective sample. In our experimental evaluation, we have conducted each of our experiment 10 times to estimate each parameter performance based on our sample statistics.

We have selected 95% level of confidence for our sample data used to evaluate performance parameters. The 95% level of confidence means 9.5 out of 10 samples will represent the same confidence interval that contains the population parameter. In other words, one can be confident that 95% of the entire population mean will fall in the range of the sample intervals.

To calculate the confidence interval, we have computed the processing time of our proposed solution at its each phase such as a trigger, DeSI, and validation. The processing time is calculated for our different experiments depending on our topologies. Each topology has different network entities such as the number of switches, the total number

of links, and the number of fake links which results with different processing time at each phase of FoR-Guard.

The experimental topologies upon which we have collected the data have been explained in the next subsequent sections. The confidence interval formula is shown in equation 5.1 as follows.

$$CI = MS * ES \quad (5.1)$$

Where (CI) is the confidence interval, (MS) is the sample mean value, and (ES) is the standard error. The steps used to calculate the confidence interval for our sample data is as follows.

Step 1: We have run each experiment 10 times and recorded the processing time for each of the phase of FoR-Guard.

Step 2: A mean is calculated for the 10 values recorded from each of the experiment.

Step 3: Standard deviation is calculated based on our sample size and the mean value which is calculated in step 2.

Step 4: The level of confidence, i.e., 95% is selected to compute the confidence interval.

Step 5: The confidence interval (CI) is calculated based on the sample mean (MS) and standard error (ES) as shown in equation 5.1. The standard error is the standard deviation divided by the square root of the sample size multiples the critical value. The ES can be calculated as follows.

$$ES = z * \sigma / \sqrt{n} \quad (5.2)$$

The critical value (z) is used to measure the margin of the error. The critical value (z) for the 95% confidence interval is 1.96 as per z-table. Thus, the standard error can be calculated as follows.

$$ES = 1.96 * \sigma / \sqrt{n} \quad (5.3)$$

The next section describes the confidence interval of the data collected for the processing time during the each phase of FoR-Guard.

5.2.2 Processing Time

The processing time of the controller is considered an important metric to examine the overall SDN efficiency. The controller performs the maximum task executions due to its centralized control over the entire network. The integration of various functionalities in the control plane will increase the processing time of the controller. We have integrated our proposed forensic investigation modules (FoR-Guard) in the control plane to trigger, detect, and determine the source of LFA in SDN.

The FoR-Guard method involves three phases; trigger, DeSI, and validation phase. As explained in Chapter 4, each phase is responsible for fulfilling a specific task which incurs a processing time of the controller. As a result, proposed method such as triggering, detection, and source identification of LFA incurs following processing times of the controller.

The processing time of trigger phase: This phase utilizes the processing time of the controller when the switch is examined for its MIR value to identify about its malicious or legitimate states.

The processing time of DeSI phase: This phase used the controller processing time to detect and identify the source of the fake link in SDN.

The processing time of validation phase: This phase utilizes the processing time of the controller when entropy measurement is performed to validate the real source of the attack.

In our emulation, we have conducted our experiments based on the number of switches and links in SDN. We have first selected 10 switches with varying number of links to perform our experiment. We first select 50 and then 75 total links with having a varying number of fake links generated from a single malicious switch. The number of fake links,

i.e., 01, 05, 10, 15, 20 is generated by 10 switches within 50 and 75 links. Each experiment is performed 10 times and an average value for the processing time is selected. The complete elaboration of our experimental setup is shown in Table 5.2.

Table 5.2: Network topology setups for our experiments

Experiments	Total number of switches	Total number of links	Legitimate links	Fake links	No. of Experiments
1	10	50	49	01	10
2			45	05	10
3			40	10	10
4			35	15	10
5			30	20	10
6		75	74	01	10
7			70	05	10
8			65	10	10
9			60	15	10
10			55	20	10
11	20	50	49	01	10
12			45	05	10
13			40	10	10
14			35	15	10
15			30	20	10
16		75	74	01	10
17			70	05	10
18			65	10	10
19			60	15	10
20			55	20	10

5.2.2.1 Data collection of trigger phase processing time

In this first phase of FoR-Guard, the controller is responsible for building a topology upon changes observes in the network. The changes can be through insertion, deletion, and migration of hosts, switches, and links in SDN. The controller is responsible for updating its topological view to have centralized abstract view of the entire network. In our case, we are focusing on the insertion of a link in the network which can be a

legitimate or fake link. The controller has to update only the legitimate link in the topology whereas; to trigger an alarm message to the detection phase upon observing the fake link. In our proposed method, the controller processing time in trigger phase is used when the attacker generates a link through a malicious switch.

The module MLA running in the controller checks the MIR value of the switch to decide either a trigger message will be triggered or not. If the MIR value of the switch is greater than zero then MLA triggers an alarm message. However, if MIR value is equal to zero then it considers has a legitimate switch which allows the controller to update its topology by considering the new links as a legitimate link. Table 5.3 depicts the processing time of the controller to determine the MIR value of the switch and triggers an alarm message to the detection phase of our proposed solution, i.e., DeSI phase.

5.2.2.2 Data collection of DeSI phase processing time

The DeSI is the second phase of our proposed method which detects and identifies the source of the fake links in SDN. This phase utilizes the controller processing time by investigating to detect and identify the source of the fake links. The modules LH and LM running in the controller are responsible for detecting and identifying the source of the fake links.

The controller processing time is utilized when the module LH find the link direction of the newly generated link such as symmetric or asymmetric. The fake links generated in SDN are asymmetric in nature as malicious switch builds a connection with the legitimate switch for various illegal purposes. Moreover, the controller processing time is utilized when the module LM what to declare a newly generated link as a fake link or legitimate link upon MIR value of the switch and LCD value of the link. The processing time of the controller used in the DeSI phase of FoR-Guard is shown in Table 5.4.

5.2.2.3 Data collection of validation phase processing time

The validation phase is a third and last phase of our proposed method which validates the source of the fake link identified in the DeSI phase. This phase uses entropy measurement to find the randomness in the network caused by fake links. The processing time of the controller is less in this phase as compared to the previous phases due to single module SV running in the controller which is responsible for computing the entropy value for the W-adjacency matrices before and after the insertion of the fake link in SDN.

The probability values for both the matrices are used to calculate the Shannon entropy to identify the randomness presented in the network. The SV modules find the average weight of the entropy value for each matrix which is further used to determine the source switch of the fake link by using threshold values. Table 5.5 depicts the processing time of the controller used to validate the source of the fake links.

5.2.3 Sensitivity

The sensitivity is another parameter to evaluate our proposed FoR-Guard. As discussed in Chapter 3, sensitivity measures the rate of positives that are correctly identified as positives. For instance, the occurrence of an attack in SDN is correctly identified as an attack. In our evaluation case, we measure the sensitivity value regarding LFA, i.e., generation of fake links. The sensitivity can be measured by the equation 5.4.

$$\text{Sensitivity} = \frac{\text{Number of True positives}}{\text{Number of True positives} + \text{Number of False Negatives}} \quad (5.4)$$

In our evaluation criteria, the sensitivity metric shows the strength of FoR-Guard to detect the fake links generated in the network topology. Higher the sensitivity value, better FoR-Guard is in detecting the fake links. The data collected to evaluate the sensitivity metric is shown in Tables (5.6- 5.25) due to our different experimental setups.

Table 5.3: Controller processing time in the trigger phase

No. of Switches	Total Links	No. of Legitimate Links	No. of Fake links	Controller Processing Time Mean (10)	Standard Deviation	Confidence Interval (95%)	
10	50	49	01	0.0118765	3.71	0.0118765 (+/-) 1.33	
	50	45	05	0.0183279	1.95	0.0183279 (+/-) 0.70	
	50	40	10	0.0235505	1.63	0.0235505 (+/-) 0.58	
	50	35	15	0.0257810	2.08	0.0257810 (+/-) 0.74	
	50	30	20	0.0291104	2.55	0.0291104 (+/-) 0.91	
	75	74	01	0.0123484	1.54	0.0123484 (+/-) 0.55	
	75	70	05	0.0197853	1.44	0.0197853 (+/-) 0.52	
	75	65	10	0.0291327	1.39	0.0291327 (+/-) 0.50	
	75	60	15	0.0304424	2.43	0.0304424 (+/-) 0.87	
	75	55	20	0.0321456	2.50	0.0321456 (+/-) 0.89	
20	50	49	01	0.0123555	1.54	0.0123555 (+/-) 0.55	
	50	45	05	0.0189068	2.42	0.0189068 (+/-) 0.87	
	50	40	10	0.0245121	9.12	0.0245121 (+/-) 3.26	
	50	35	15	0.0260081	1.54	0.0260081 (+/-) 0.55	
	50	30	20	0.0294520	1.55	0.0294520 (+/-) 0.55	
	75	74	01	0.0147828	2.26	0.0147828 (+/-) 0.81	
	75	70	05	0.0207653	1.82	0.0207653 (+/-) 0.65	
	75	65	10	0.0307911	1.67	0.0307911 (+/-) 0.60	
	75	60	15	0.0320034	2.27	0.0320034 (+/-) 0.81	
	75	55	20	0.0337770	1.28	0.0337770 (+/-) 0.46	

Table 5.4: Controller processing time in the DeSI phase

No. of Switches	Total Links	No. of Legitimate Links	No. of Fake links	Controller Processing Time Mean (10)	Standard Deviation	Confidence Interval (95%)	
10	50	49	01	0.0006213	1.29	0.0006213 (+/-) 0.46	
	50	45	05	0.0006829	1.81	0.0006829 (+/-) 0.65	
	50	40	10	0.0007134	1.90	0.0007134 (+/-) 0.68	
	50	35	15	0.0007844	2.47	0.0007844 (+/-) 0.88	
	50	30	20	0.0009219	1.34	0.0009219 (+/-) 0.48	
	75	74	01	0.0006663	1.98	0.0006663 (+/-) 0.48	
	75	70	05	0.0006767	1.67	0.0006767 (+/-) 0.60	
	75	65	10	0.0007429	1.36	0.0007429 (+/-) 0.49	
	75	60	15	0.0008047	8.58	0.0008047 (+/-) 3.07	
	75	55	20	0.0008178	1.82	0.0008178 (+/-) 0.65	
	20	50	49	01	0.0006179	9.59	0.0006179 (+/-) 3.43
		50	45	05	0.0007025	2.09	0.0007025 (+/-) 0.75
50		40	10	0.0007936	1.30	0.0007936 (+/-) 0.47	
50		35	15	0.0009176	1.37	0.0009176 (+/-) 0.49	
50		30	20	0.0010831	1.61	0.0010831 (+/-) 0.58	
75		74	01	0.0007344	1.67	0.0007344 (+/-) 0.60	
75		70	05	0.0007860	2.57	0.0007860 (+/-) 0.92	
75		65	10	0.0008435	1.79	0.0008435 (+/-) 0.64	
75		60	15	0.0008625	7.76	0.0008625 (+/-) 2.78	
75		55	20	0.0009467	1.77	0.0009467 (+/-) 0.63	

Table 5.5: Controller processing time in the Validation phase

No. of Switches	Total Links	No. of Legitimate Links	No. of Fake links	Controller Processing Time Mean (10)	Standard Deviation	Confidence Interval (95%)	
10	50	49	01	0.0001257	1.63	0.0001257 (+/-) 0.58	
	50	45	05	0.0001463	1.90	0.0001463 (+/-) 0.68	
	50	40	10	0.0001540	2.36	0.0001540 (+/-) 0.84	
	50	35	15	0.0001871	8.19	0.0001871 (+/-) 2.93	
	50	30	20	0.0001923	9.22	0.0001923 (+/-) 3.30	
	75	74	01	0.0002757	6.26	0.0002757 (+/-) 2.2	
	75	70	05	0.0002943	1.12	0.0002943 (+/-) 0.40	
	75	65	10	0.0003115	8.17	0.0003115 (+/-) 2.92	
	75	60	15	0.0003034	7.73	0.0003034 (+/-) 2.77	
	75	55	20	0.0003581	1.23	0.0003581 (+/-) 0.44	
20	50	49	01	0.0005062	1.36	0.0005062 (+/-) 0.49	
	50	45	05	0.0005416	8.89	0.0005416 (+/-) 3.18	
	50	40	10	0.0005583	8.84	0.0005583 (+/-) 3.16	
	50	35	15	0.0006415	7.31	0.0006415 (+/-) 2.62	
	50	30	20	0.0006483	8.44	0.0006483 (+/-) 3.02	
	75	74	01	0.0005617	7.94	0.0005617 (+/-) 2.84	
	75	70	05	0.0005359	6.68	0.0005359 (+/-) 2.39	
	75	65	10	0.0005346	8.84	0.0005346 (+/-) 3.16	
	75	60	15	0.0006135	7.30	0.0006135 (+/-) 2.61	
	75	55	20	0.0006324	7.30	0.0006324 (+/-) 2.61	

5.2.4 Specificity

The specificity metric of FoR-Guard is calculated by looking towards true negatives and false positives. The equation of specificity is shown in equation 5.5. The specificity measures the rate of negatives that are correctly identified as negatives. For instance, correctly identified legitimate links as legitimate links regarding FoR-Guard.

$$\text{Specificity} = \frac{\text{Number of True negatives}}{\text{Number of True negatives} + \text{Number of False positive}} \quad (5.5)$$

The specificity metric evaluates the FoR-Guard that it should identify the legitimate link as a legitimate link during its forensic investigation in the DeSI phase. Higher the specificity value, better FoR-Guard has identified the legitimate links. The data collected to evaluate the specificity metric is shown in Tables (5.6- 5.25) due to our different experimental setups.

5.2.5 False Alarm Rate

The false alarm rate is the expectancy of the occurrence of false alarm during a statistical inference. It can be calculated as the ratio between negatives identified wrongly as positives and the total number of negatives. The equation to calculate false positive rate is shown in equation 5.6.

$$\text{False Alarm Rate} = \frac{\text{Number of False positive}}{\text{Number of False positive} + \text{Number of True negatives}} \quad (5.6)$$

The data collected to evaluate the false alarm rate metric is shown in Tables (5.6- 5.25) due to our experiment different setups.

5.2.6 Data collection for sensitivity, specificity, and false alarm rate

In this section, we have explained the method of collecting the data for sensitivity, specificity, and false alarm rate metrics based on our experiments. The experimental setup for collecting data for different metrics is shown in Table 5.2. Each experiment is performed 10 times, and their results are shown in Tables (5.6-5.25) respectively.

5.2.6.1 Data collection of 10 switches and 50 total links

In our current experiment, we have selected 10 switches with having 50 total numbers of links in our network topology. For conducting different experiments to evaluate FoR-Guard performance, we have selected a different number of fake links such as 1, 5, 10, 15, and 20 in the network topology.

Table 5.6 shows 10 different experiment values for calculating sensitivity, specificity, and false alarm rate. In our current experiment, we selected 10 switches with having 50 total numbers of links. Among 50 total links, there are 49 legitimate links and 01 fake links. The correct identified (true positive) attribute shows the value for correctly identifying the number of fake links. The incorrect identified (false positive) attribute shows the value for not identifying the number of fake links.

Table 5.6: Data collection for switches=10, total links=50, Legitimate links=49, fake links=01

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	49	01	01	00	100%	100%	00%
2	50	49	01	01	00	100%	100%	00%
3	50	49	01	01	00	100%	100%	00%
4	50	49	01	01	00	100%	100%	00%
5	50	49	01	01	00	100%	100%	00%
6	50	49	01	01	00	100%	100%	00%
7	50	49	01	01	00	100%	100%	00%
8	50	49	01	01	00	100%	100%	00%
9	50	49	01	01	00	100%	100%	00%
10	50	49	01	01	00	100%	100%	00%

The sensitivity attribute calculates the value through $TP/(TP+FP)*100$, specificity attribute calculates through $(TN/TN+FP)*100$, and false alarm rate calculates through $(FP/FP+TN)*100$ respectively. For instance, experiment # 6 in Table 5.6 depicts that

there are 50 total links, 49 is legitimate links, and 01 is a fake link. The FoR-Guard identifies 01 fake links as a fake link so the value for incorrect identified remains zero. The value for sensitivity and specificity will be 100% while false alarm rate remains 00%.

Table 5.7 presents 10 switches with having 50 total numbers of links. Among 50 total links, there are 45 legitimate links and 05 fake links. We have observed that in experiments # 5 and 7, values for sensitivity is 80% and 60 % respectively. It is due to the low-frequency rate (LF) of the link generation of the switch in the specified time interval as explained in Chapter 4. The switch is in the malicious state, but its frequency rate for its link generation is less than the threshold value. The FoR-Guard treated such links as a legitimate link. However, it happens very rarely because the malicious switches have a high number of frequency rates of link generation in the network.

Table 5.7: Data collection for switches=10, total links=50, Legitimate links=45, fake links=05

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	45	05	05	00	100%	100%	00%
2	50	45	05	05	00	100%	100%	00%
3	50	45	05	05	00	100%	100%	00%
4	50	45	05	05	00	100%	100%	00%
5	50	45	05	04	01	80%	97.8%	2.17%
6	50	45	05	05	00	100%	100%	00%
7	50	45	05	03	02	60%	95.7%	4.25%
8	50	45	05	05	00	100%	100%	00%
9	50	45	05	05	00	100%	100%	00%
10	50	45	05	05	00	100%	100%	00%

Table 5.8 depicts 10 switches with having 50 total numbers of links. Among 50 total links, there are 40 legitimate links and 10 fake links. The sensitivity value is 70% with

the specificity rate of 93.1% in case of experiment # 7. It occurs due to the switch have less MIR value, but it starts to behave maliciously by generating fake links.

Table 5.8: Data collection for switches=10, total links=50, Legitimate links=40, fake links=10

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity TP/(TP+FP)*100	Specificity (TN/TN+FP)*100	False Alarm Rate (FP/FP+TN)*100
1	50	40	10	09	01	90%	97.5%	2.43%
2	50	40	10	10	00	100%	100%	00%
3	50	40	10	10	00	100%	100%	00%
4	50	40	10	10	00	100%	100%	00%
5	50	40	10	10	00	100%	100%	00%
6	50	40	10	10	00	100%	100%	00%
7	50	40	10	07	03	70%	93.1%	6.97%
8	50	40	10	10	00	100%	100%	00%
9	50	40	10	10	00	100%	100%	00%
10	50	40	10	10	00	100%	100%	00%

Table 5.9 summarizes the values for 10 switches with having 50 total numbers of links. Among 50 total links, there are 35 legitimate links and 15 fake links. In experiment # 2, the false alarm rate is 5.40 % due to not identifying the 02 out of 13 fake links. The 02 fake links are generated from the switch which has MIR value equals to zero, but they have first time involved in generating fake links.

Table 5.10 shows the values for 10 switches with having 50 total numbers of links. Among 50 total links, there are 30 legitimate links and 20 fake links. The sensitivity value is 95% and 90% for experiments # 5 and 9 respectively. It is due to a high number of malicious switches in the network which generates more fake links. The malicious switches hide their illegitimate identity by starting to generate legitimate links in the network.

5.2.6.2 Data collection of 10 switches and 75 total links

In this current experimental setup, we have selected 10 switches with having 75 total numbers of links in our network topology. For conducting different experiments to evaluate FoR-Guard performance we have selected a different number of fake links such as 1, 5, 10, 15, and 20 in the network topology.

Table 5.9: Data collection for switches=10, total links=50, Legitimate links=35, fake links=15

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity TP/(TP+FP)*100	Specificity (TN/TN+FP)*100	False Alarm Rate (FP/FP+TN)*100
1	50	35	15	15	00	100%	100%	00%
2	50	35	15	13	02	86.6%	94.5%	5.40%
3	50	35	15	15	00	100%	100%	00%
4	50	35	15	15	00	100%	100%	00%
5	50	35	15	15	00	100%	100%	00%
6	50	35	15	15	00	100%	100%	00%
7	50	35	15	15	00	100%	100%	00%
8	50	35	15	15	00	100%	100%	00%
9	50	35	15	15	00	100%	100%	00%
10	50	35	15	15	00	100%	100%	00%

Table 5.11 shows the values for 10 switches with having 75 total numbers of links. Among 75 total links, there are 74 legitimate links and 01 fake links. The experiment # 5 indicates that FoR-Guard does not find the fake link due to MIR value of the switch, i.e., ($F_i = 0$). The switch has created more legitimate links with other switches in the network to hide its illegitimate identity in the network.

Table 5.10: Data collection for switches=10, total links=50, Legitimate links=30, fake links=20

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	30	20	20	00	100%	100%	00%
2	50	30	20	20	00	100%	100%	00%
3	50	30	20	20	00	100%	100%	00%
4	50	30	20	20	00	100%	100%	00%
5	50	30	20	19	01	95%	96.7%	3.22%
6	50	30	20	20	00	100%	100%	00%
7	50	30	20	20	00	100%	100%	00%
8	50	30	20	20	00	100%	100%	00%
9	50	30	20	18	02	90%	93.7%	6.25%
10	50	30	20	20	00	100%	100%	00%

Table 5.11: Data collection for switches=10, total links=75, Legitimate links=74, fake links=01

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	74	01	01	00	100%	100%	00%
2	75	74	01	01	00	100%	100%	00%
3	75	74	01	01	00	100%	100%	00%
4	75	74	01	01	00	100%	100%	00%
5	75	74	01	00	01	00%	98.6%	1.33%
6	75	74	01	01	00	100%	100%	00%
7	75	74	01	01	00	100%	100%	00%
8	75	74	01	01	00	100%	100%	00%
9	75	74	01	01	00	100%	100%	00%
10	75	74	01	01	00	100%	100%	00%

Table 5.12 shows the values for 10 switches with having 75 total numbers of links. Among 75 total links, there are 70 legitimate links and 05 fake links. The specificity value is 98.5% for experiment # 6 due to less frequency rate of link generation for the switch. It indicates that the switch which generates the fake link is passively acting in the network due to generating fake links time to time.

Table 5.13 shows the values for 10 switches with having 75 total numbers of links. Among 75 total links, there are 65 legitimate links and 10 fake links. The sensitivity values are 80%, 90%, and 70% for experiments # 1, 7, and 10 respectively. It is due to increase the number of fake links in the network, the value of MIR frequently changes due to the generation of links in the network topology.

Table 5.14 shows the values for 10 switches with having 75 total numbers of links. Among 75 total links, there are 60 legitimate links and 15 fake links. The sensitivity value is 93.3% for experiment # 1 due to the switch has generated the fake link at the time the MIR value was equal to zero.

Table 5.15 summarizes the values for 10 switches with having 75 total numbers of links. Among 75 total links, there are 55 legitimate links and 20 fake links. The specificity value is 98.2% and 96.4% for experiments # 2 and 9 respectively. The specificity value is less than 100% due to switches are in legitimate states, and they are shifting towards malicious states due to the generation of fake links.

5.2.6.3 Data collection of 20 switches and 50 total links

In this current experimental setup, we have selected 20 switches with having 50 total numbers of links in our network topology. For conducting different experiments to evaluate FoR-Guard performance we have selected a different number of fake links such as 1, 5, 10, 15, and 20 in the network topology.

Table 5.12: Data collection for switches=10, total links=75, Legitimate links=70, fake links=05

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	70	05	05	00	100%	100%	00%
2	75	70	05	05	00	100%	100%	00%
3	75	70	05	05	00	100%	100%	00%
4	75	70	05	05	00	100%	100%	00%
5	75	70	05	05	00	100%	100%	00%
6	75	70	05	04	01	80%	98.5%	1.40%
7	75	70	05	05	00	100%	100%	00%
8	75	70	05	05	00	100%	100%	00%
9	75	70	05	05	00	100%	100%	00%
10	75	70	05	05	00	100%	100%	00%

Table 5.13: Data collection for switches=10, total links=75, Legitimate links=65, fake links=10

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	65	10	08	02	80%	97.1%	2.98%
2	75	65	10	10	00	100%	100%	00%
3	75	65	10	10	00	100%	100%	00%
4	75	65	10	10	00	100%	100%	00%
5	75	65	10	10	00	100%	100%	00%
6	75	65	10	10	00	100%	100%	00%
7	75	65	10	09	01	90%	98.4%	1.51%
8	75	65	10	10	00	100%	100%	00%
9	75	65	10	10	00	100%	100%	00%
10	75	65	10	07	03	70%	95.5%	4.41%

Table 5.14: Data collection for switches=10, total links=75, Legitimate links=60, fake links=15

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	60	15	14	01	93.3%	98.3%	1.63%
2	75	60	15	15	00	100%	100%	00%
3	75	60	15	15	00	100%	100%	00%
4	75	60	15	15	00	100%	100%	00%
5	75	60	15	15	00	100%	100%	00%
6	75	60	15	15	00	100%	100%	00%
7	75	60	15	15	00	100%	100%	00%
8	75	60	15	15	00	100%	100%	00%
9	75	60	15	15	00	100%	100%	00%
10	75	60	15	15	00	100%	100%	00%

Table 5.15: Data collection for switches=10, total links=75, Legitimate links=55, fake links=20

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	55	20	20	00	100%	100%	00%
2	75	55	20	19	01	95%	98.2%	1.78%
3	75	55	20	20	00	100%	100%	00%
4	75	55	20	20	00	100%	100%	00%
5	75	55	20	20	00	100%	100%	00%
6	75	55	20	20	00	100%	100%	00%
7	75	55	20	20	00	100%	100%	00%
8	75	55	20	20	00	100%	100%	00%
9	75	55	20	18	02	90%	96.4%	3.50%
10	75	55	20	20	00	100%	100%	00%

Table 5.16 summarizes the values for 20 switches with having 50 total numbers of links. Among 50 total links, there are 49 legitimate links and 01 fake link. It has been observed that the sensitivity is 98% for experiment # 9 due to MIR value equal to zero for the switch at the time of investigating the link.

Table 5.16: Data collection for switches=20, total links=50, Legitimate links=49, fake links=01

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	49	01	01	00	100%	100%	00%
2	50	49	01	01	00	100%	100%	00%
3	50	49	01	01	00	100%	100%	00%
4	50	49	01	01	00	100%	100%	00%
5	50	49	01	01	00	100%	100%	00%
6	50	49	01	01	00	100%	100%	00%
7	50	49	01	01	00	100%	100%	00%
8	50	49	01	01	00	100%	100%	00%
9	50	49	01	00	01	00%	98%	02%
10	50	49	01	01	00	100%	100%	00%

Table 5.17 summarizes the values for 20 switches with having 50 total numbers of links. Among 50 total links, there are 45 legitimate links and 05 fake links. The sensitivity for experiment #1, 5, and 9 are observed as 80% for each respectively. It is due to less number of links generated by the switch as compared to its specified threshold value, i.e., L_F .

Table 5.18 summarizes the values for 20 switches with having 50 total numbers of links. Among 50 total links, there are 40 legitimate links and 10 fake links. The sensitivity value is 80 % and 90% for experiment # 1 and 2 with its false alarm rate 4.76% and 2.43% respectively. The suspicious links were considered as legitimate links due to the behavior of the switch that proceed towards legitimate state by generating legitimate links.

Table 5.17: Data collection for switches=20, total links=50, Legitimate links=45, fake links=05

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/(TN+FP))*100$	False Alarm Rate $(FP/(FP+TN))*100$
1	50	45	05	04	01	80%	97.8%	2.17%
2	50	45	05	05	00	100%	100%	00%
3	50	45	05	05	00	100%	100%	00%
4	50	45	05	05	00	100%	100%	00%
5	50	45	05	04	01	80%	97.8%	2.17%
6	50	45	05	05	00	100%	100%	00%
7	50	45	05	05	00	100%	100%	00%
8	50	45	05	05	00	100%	100%	00%
9	50	45	05	04	01	80%	97.8%	2.17%
10	50	45	05	05	00	100%	100%	00%

Table 5.19 summarizes the values for 20 switches with having 50 total numbers of links. Among 50 total links, there are 35 legitimate links and 15 fake links. It has been observed that FoR-Guard identified all the fake links generated from the malicious switches in all the 10 experiments.

Table 5.20 presents the values for 20 switches with having 50 total numbers of links. Among 50 total links, there are 30 legitimate links and 20 fake links. The experiments # 1 and 6 have specificity 93.7% and 88.2% with its false alarm rate as 6.25% and 11.76% respectively. The specificity is less than 100% due to some fake links generated by the switch after some time interval. The attackers use such a strategy to hide its identity by generating more legitimate links as compared to fake links.

Table 5.18: Data collection for switches=20, total links=50, Legitimate links=40, fake links=10

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	40	10	08	02	80%	95.2%	4.76%
2	50	40	10	09	01	90%	97.5%	2.43%
3	50	40	10	10	00	100%	100%	00%
4	50	40	10	10	00	100%	100%	00%
5	50	40	10	10	00	100%	100%	00%
6	50	40	10	10	00	100%	100%	00%
7	50	40	10	10	00	100%	100%	00%
8	50	40	10	10	00	100%	100%	00%
9	50	40	10	10	00	100%	100%	00%
10	50	40	10	10	00	100%	100%	00%

Table 5.19: Data collection for switches=20, total links=50, Legitimate links=35, fake links=15

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	35	15	15	00	100%	100%	00%
2	50	35	15	15	00	100%	100%	00%
3	50	35	15	15	00	100%	100%	00%
4	50	35	15	15	00	100%	100%	00%
5	50	35	15	15	00	100%	100%	00%
6	50	35	15	15	00	100%	100%	00%
7	50	35	15	15	00	100%	100%	00%
8	50	35	15	15	00	100%	100%	00%
9	50	35	15	15	00	100%	100%	00%
10	50	35	15	15	00	100%	100%	00%

Table 5.20: Data collection for switches=20, total links=50, Legitimate links=30, fake links=20

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	50	30	20	18	02	90%	93.7%	6.25%
2	50	30	20	20	00	100%	100%	00%
3	50	30	20	20	00	100%	100%	00%
4	50	30	20	20	00	100%	100%	00%
5	50	30	20	20	00	100%	100%	00%
6	50	30	20	16	04	80%	88.2%	11.76%
7	50	30	20	20	00	100%	100%	00%
8	50	30	20	20	00	100%	100%	00%
9	50	30	20	20	00	100%	100%	00%
10	50	30	20	20	00	100%	100%	00%

5.2.6.4 Data collection of 20 switches and 75 total links

In the current experimental scenario, we have selected 20 switches with having 75 total numbers of links in our network topology. For conducting different experiments to evaluate FoR-Guard performance we have selected a different number of fake links such as 1, 5, 10, 15, and 20 in the network topology.

Table 5.21 shows the values for 20 switches with having 75 total numbers of links. Among 75 total links, there are 74 legitimate links and 01 fake link. The FoR-Guard identifies all the fake links generated from the malicious switches.

Table 5.22 depicts the values for 20 switches with having 75 total numbers of links. Among 75 total links, there are 70 legitimate links and 05 fake links. The experiment # 6 has 60% of sensitivity due to MIR value of the switch is equal to zero.

Table 5.21: Data collection for switches=20, total links=75, Legitimate links=74, fake links=01

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	74	01	01	00	100%	100%	00%
2	75	74	01	01	00	100%	100%	00%
3	75	74	01	01	00	100%	100%	00%
4	75	74	01	01	00	100%	100%	00%
5	75	74	01	01	00	100%	100%	00%
6	75	74	01	01	00	100%	100%	00%
7	75	74	01	01	00	100%	100%	00%
8	75	74	01	01	00	100%	100%	00%
9	75	74	01	01	00	100%	100%	00%
10	75	74	01	01	00	100%	100%	00%

Table 5.22: Data collection for switches=20, total links=75, Legitimate links=70, fake links=05

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	70	05	05	00	100%	100%	00%
2	75	70	05	05	00	100%	100%	00%
3	75	70	05	05	00	100%	100%	00%
4	75	70	05	05	00	100%	100%	00%
5	75	70	05	05	00	100%	100%	00%
6	75	70	05	03	02	60%	97.2%	2.77%
7	75	70	05	05	00	100%	100%	00%
8	75	70	05	05	00	100%	100%	00%
9	75	70	05	05	00	100%	100%	00%
10	75	70	05	05	00	100%	100%	00%

Table 5.23: Data collection for switches=20, total links=75, Legitimate links=65, fake links=10

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	65	10	10	00	100%	100%	00%
2	75	65	10	09	01	90%	98.4%	1.51%
3	75	65	10	07	03	70%	95.5%	4.41%
4	75	65	10	10	00	100%	100%	00%
5	75	65	10	10	00	100%	100%	00%
6	75	65	10	10	00	100%	100%	00%
7	75	65	10	10	00	100%	100%	00%
8	75	65	10	10	00	100%	100%	00%
9	75	65	10	10	00	100%	100%	00%
10	75	65	10	09	01	90%	98.4%	1.51%

Table 5.24: Data collection for switches=20, total links=75, Legitimate links=60, fake links=15

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	60	15	15	00	100%	100%	00%
2	75	60	15	15	00	100%	100%	00%
3	75	60	15	15	00	100%	100%	00%
4	75	60	15	15	00	100%	100%	00%
5	75	60	15	15	00	100%	100%	00%
6	75	60	15	15	00	100%	100%	00%
7	75	60	15	15	00	100%	100%	00%
8	75	60	15	15	00	100%	100%	00%
9	75	60	15	14	01	93.3%	98.3%	1.63%
10	75	60	15	15	00	100%	100%	00%

Table 5.25: Data collection for switches=20, total links=75, Legitimate links=55, fake links=20

Experiments	Total Links	No. of Legitimate Links (TN)	No. of Fake links	Correct Identified (TP)	Incorrect Identified (FP)	Sensitivity $TP/(TP+FP)*100$	Specificity $(TN/TN+FP)*100$	False Alarm Rate $(FP/FP+TN)*100$
1	75	55	20	19	01	95%	98.2%	1.78%
2	75	55	20	20	00	100%	100%	00%
3	75	55	20	20	00	100%	100%	00%
4	75	55	20	20	00	100%	100%	00%
5	75	55	20	20	00	100%	100%	00%
6	75	55	20	20	00	100%	100%	00%
7	75	55	20	17	03	85%	94.8%	5.17%
8	75	55	20	20	00	100%	100%	00%
9	75	55	20	20	00	100%	100%	00%
10	75	55	20	20	00	100%	100%	00%

Table 5.23 presents the values for 20 switches with having 75 total numbers of links. Among 75 total links, there are 65 legitimate links and 10 fake links. The sensitivity of experiment # 2, 3, and 10 is 90%, 70%, and 90% respectively. The number of link frequency is less as compared to the threshold value because the malicious switches maintain its MIR value equal to zero by generating legitimate links as well with different switches in the network.

Table 5.24 summarizes the values for 20 switches with having 75 total numbers of links. Among 75 total links, there are 60 legitimate links and 15 fake links. The experiment # 9 has sensitivity equals to 93.3% as FoR-Guard is not recognized one fake link due to MIR value of the switch which is equal to zero.

Table 5.25 depicts the values for 20 switches with having 75 total numbers of links. Among 75 total links, there are 55 legitimate links and 20 fake links. It has been observed in experiment # 1 and 7 that FoR-Guard fails to identify some fake links. It happens due to link generation frequency of the malicious switch is less as compared to the specified

threshold value. However, if the threshold value is less than link generation frequency of the switch, it means that the malicious switch is generating fake links by making more connection in the network.

5.3 Performance analysis of FoR-Guard with existing solutions

In this section, we collect the data extracted from the experiments to compare the processing time of FoR-Guard with existing solutions. We compare the processing time of FoR-Guard for its first two phases such as trigger and DeSI with TopoGuard (Hong et al., 2015) and Sphinx (Dhawan et al., 2015) because to have a fair comparison. The solutions TopoGuard and Sphinx are not validating the source of the attack in their proposed solutions. However, FoR-Guard provides a validation phase to verify the source of the LFA identified in the DeSI phase. Therefore, we exclude the processing time of validation phase of FoR-Guard during the comparison experiments.

Table 5.26 depicts the processing time comparison of FoR-Guard (trigger + DeSI phases) with TopoGuard and Sphinx. The comparison shows that FoR-Guard has less processing time as compared to the state-of-the-art solutions such as TopoGuard and Sphinx. The TopoGuard has high processing time as compared to FoR-Guard and Sphinx because it has to mark each LLDP packet through cryptographic value to ensure the integrity of the packet. The Sphinx processing time is high as compared to FoR-Guard due to generating network graphs for each update in the network which requires collecting the topological metadata from the OF control messages.

Table 5.26: Processing time comparison in switches=10 and total links=50

Fake Links	FoR-Guard	TopoGuard	Sphinx
1	12.497	38.54	24.21
5	19.009	57.51	37.14
10	24.263	78.04	42.87
15	26.565	84.33	52.67
20	30.031	89.94	68.49

Table 5.27 shows the comparison values of processing time of FoR-Guard with TopoGuard and Sphinx regarding 10 switches and 75 total links. The result depicted in Table 5.27 indicates that the processing time is increased with the increasing number of fake links. FoR-Guard has less processing time as compared to TopoGuard and Sphinx due to its focus on the switch which has generated a new link. However, TopoGuard marks the LLDP packets and Sphinx captures metadata information from the OF control messages which lead them to high processing overhead. For instance, the detection of 5 fake links cost FoR-Guard for 20.481 microseconds as compared to TopoGuard and Sphinx which cost them 89.28 and 48.92 microseconds respectively.

Table 5.27: Processing time comparison in switches=10 and total links=75

Fake Links	FoR-Guard	TopoGuard	Sphinx
1	13.014	57.15	39.21
5	20.481	89.28	48.92
10	29.874	106.21	56.14
15	31.248	135.49	72.91
20	32.962	170.98	83.79

Table 5.28 shows that TopoGuard has high processing time as compared to FoR-Guard and Sphinx. The results for processing time depicted in Table 5.28 are based on 20 switches and 50 total links. The increase in a number of the switches has not a significant impact on the FoR-Guard whereas, it have an impact on the TopoGuard and Sphinx. More switches will cause more LLDP packets to be marked based the connectivity of the switches. The controller has to send LLDP packets to each port of each switch to know about their connectivity with their neighbor switches. It causes a high processing time especially for TopoGuard while Sphinx has to build a network graph which depends on the number of switches as well. For instance, the detection of 20 fake links in 20 switches and 50 total links cost FoR-Guard 30.535 microseconds processing time as compared to TopoGuard and Sphinx which cost 98.21 and 75.49 microseconds respectively.

Table 5.28: Processing time comparison in switches=20 and total links=50

Fake Links	FoR-Guard	TopoGuard	Sphinx
1	12.972	49.75	30.74
5	19.609	63.28	40.07
10	25.305	84.67	48.61
15	26.925	92.47	59.39
20	30.535	98.21	75.49

Table 5.29 shows the processing time of FoR-Guard (trigger + DeSI phases) with TopoGuard and Sphinx regarding 20 switches and 75 total links. The comparison shows that FoR-Guard has less processing time as compared to the state-of-the-art solutions such as TopoGuard and Sphinx. The increase in a number of links has a significant impact on the processing time of TopoGuard and Sphinx. For instance, the detection of 15 fake links in 20 switches and 75 total links cost FoR-Guard 32.865 microseconds processing time as compared to TopoGuard and Sphinx which cost 152.87 and 79.28 microseconds respectively. Thus, the processing time of FoR-Guard is significantly better than processing time of TopoGuard and Sphinx in all experiment setups.

Table 5.29: Processing time comparison in switches=20 and total links=75

Fake Links	FoR-Guard	TopoGuard	Sphinx
1	15.516	72.64	46.89
5	21.551	99.23	55.19
10	31.634	125.41	69.88
15	32.865	152.87	79.28
20	34.723	198.21	99.45

5.4 Conclusion

This chapter presents the experimental tools and setup used to evaluate FoR-Guard proposed method to determine the fake links in SDN. The Floodlight controller is briefly elaborated which use to generate a centralized controlled environment in SDN. The data collection method is presented to examine FoR-Guard based on performance metrics including processing time (effectiveness), sensitivity, specificity, and false alarm rate.

The data of comparison among FoR-Guard, TopoGuard, and Sphinx is presented with its critical description.

The data collection is performed based on different experimental setups depending on the number of switches, the total number of links, and the number of fake links. The processing time data is collected for each experiment in the sample space of 10 values. The 95% confidence interval is used to calculate the mean value of the sample space. The data collection for sensitivity, specificity, and false alarm rate shows the accuracy of FoR-Guard regarding detecting fake links in SDN. Thus, it has been shown that FoR-Guard has less processing time in executing the proposed method and have high detection accuracy rate in detecting the fake links as compared to TopoGuard and Sphinx in SDN.

University of Malaya

CHAPTER 6: RESULTS AND DISCUSSIONS

This chapter evaluates the performance of FoR-Guard and presents the experimental results in a graphical format to have a logical conclusion. We start by evaluating each phase of FoR-Guard by evaluating the controller processing time. We derive the points to know which phase of FoR-Guard has a high impact on the controller processing time. Subsequently, we evaluate the performance of FoR-Guard by using ROC graphs. We perform a comparative analysis of FoR-Guard with TopoGuard and Sphinx in measuring the processing time. The comparative experimental data is validated by using paired sample t-tests. Finally, we show that FoR-Guard is effective in having less processing time to detect and identify the fake links in SDN.

The rest of the chapter is organized as follows. Section 6.1 presents the performance analysis of each phase of FoR-Guard by measuring the processing time. Section 6.2 compares the processing time of trigger and DeSI phases with the validation phase of FoR-Guard. Section 6.3 investigates the performance of FoR-Guard by using ROC graphs. Section 6.4 discusses the comparison between FoR-Guard and the existing solutions. Finally, Section 6.5 concludes the chapter by highlighting the significance of FoR-Guard.

6.1 Performance Analysis of FoR-Guard

In the section, we analyzed the computational burden of the controller due to proposed forensic investigation method for LFA in SDN. The processing time of the controller is analyzed for each FoR-Guard phase such as a trigger, DeSI, and validation phase.

6.1.1 Trigger phase processing time

In our early discussion about FoR-Guard in section 4.3.1 of chapter 4; we discuss that the trigger phase is mainly responsible for two tasks, i.e., discovering network topology and triggering the DeSI phase of FoR-Guard. The time taken by the controller to discover the network topology depends on the de-facto standard for SDN controllers. However,

triggering a trigger message to the DeSI phase utilize the controller time by computing and collecting MIR value of the switches.

In our experimental evaluation, we performed each experiment 10 times, and its average value is recorded for the processing time. We have performed our experiments on two network data traces having 10 and 20 switches which contain 50 and 75 total number of links respectively. Each network trace is tested for 1, 5, 10, 15, and 20 numbers of fake links.

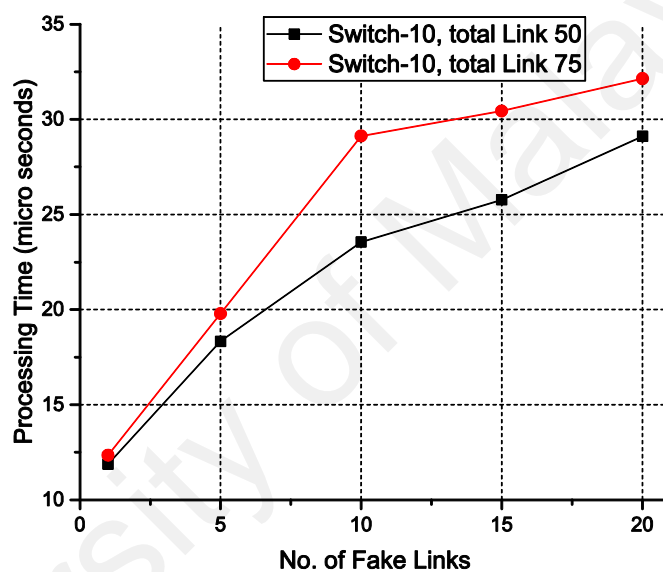


Figure 6.1: Processing time of trigger phase for switches=10 and total links=(50,75)

Figure 6.1 presents the processing time of the controller to trigger a message based on various numbers of fake links. The y-axis shows the processing time in microseconds and x-axis represents the number of fake links. The graph shows that the processing time increases due to increase in the number of fake links. For instance, when there are 10 switches and 01 fake links than the processing time is recorded as 11.876 microseconds as compared to 25.781 microseconds for 10 switches and 15 fake links out of 50 total links. The processing time increases with the number of fake links because the controller has to collect MIR value information from the switches. Similarly, the processing time

increases when the total number of links in the network topology increases. For instance, 10 switches and 10 fake links out of 50 total links in the network topology utilize 23.550 microseconds of the controller as compared to 10 switches and 10 fake links out of 75 total links which use 29.132 microseconds of the controller time.

The processing time of the controller increases because topology manager has to send LLDP packets to each switch to inquire the links between the switches. The more links in the network topology result in a more time of the controller to get information of the links from the switches. Thus, the network topology of 75 links will have more processing time as compared to 50 links in the network topology. Similarly, processing time trend for the controller have 20 switches and (50,75) total links is shown in Figure 6.2.

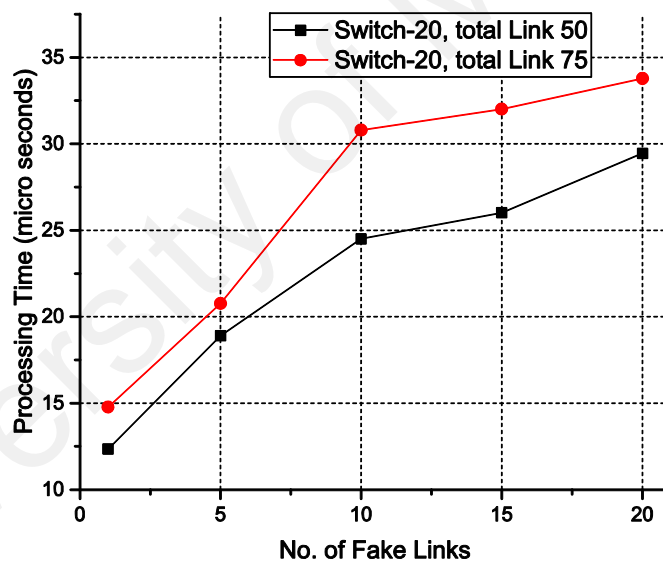


Figure 6.2: Processing time of trigger phase for switches=20 and total links=(50,75)

6.1.2 DeSI phase processing time

Figure 6.3 shows the processing time of the controller to investigate the fake links in the network topology. The controller investigates the fake links by detecting and finding the real source of the attack. The y-axis shows the processing time in microseconds and x-axis represents the number of fake links. The processing time of the DeSI phase of FoR-

Guard is less as compared to the triggering phase. The trigger phase involves de-facto standard network topology discovery to update the controller view. Thus, the processing time of the controller becomes high as compared to the DeSI phase. However, other than network topology discovery, the controller spends very less time to measure and collects the MIR value of switches in the trigger phase.

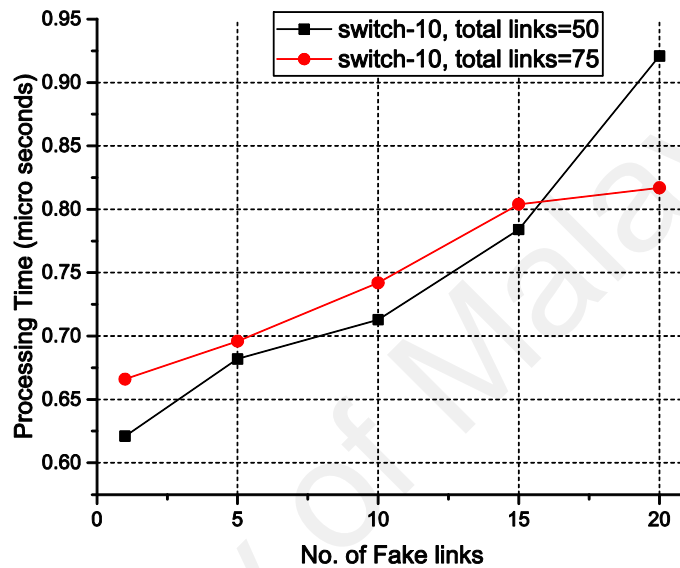


Figure 6.3: Processing time of DeSI phase for switches=10 and total links=(50,75)

The processing time of the controller in DeSI phase increases with the increasing number of fake links due to finding LCD for each suspicious link which needs to be investigated. For instance, when there are 10 switches and 05 fake links than the processing time is recorded as 0.682 microseconds as compared to 0.921 microseconds for 10 switches and 20 fake links out of 50 total links. Similarly, the processing time increases when the total number of links in the network topology increases. For instance, 10 switches and 15 fake links out of 50 total links in the network topology utilize 0.784 microseconds of the controller as compared to 10 switches and 15 fake links out of 75 total links which use 0.804 microseconds of the controller time. However, we have observed in Figure 6.3, the processing time of 20 fake links within 50 total links is high

as compared to the 20 fake links within 75 total links. This trend has been observed due to generation of different fake links at different switches rather than different fake links at the single switch in the network topology. If fake links are scattered in the network topology that generates from different switches than the processing time for 20 fake links within 50 total links would be less as compare to 75 total links. Similarly, the processing time of the controller having 20 switches and (50,75) total links are shown in Figure 6.4.

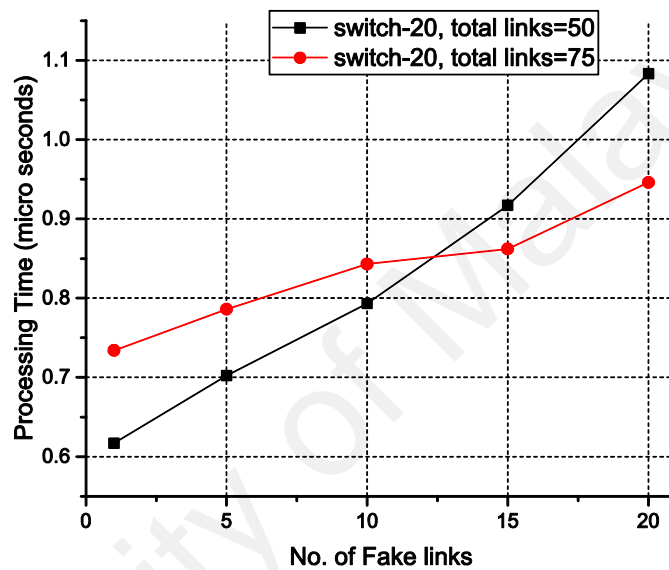


Figure 6.4: Processing time of DeSI phase for switches=20 and total links=(50,75)

6.1.3 Validation phase processing time

Figure 6.5 illustrates the graph showing the processing time of the controller to verify the source of fake links identified in the DeSI phase of FoR-Guard. The SV module is used in the controller to verify the source of the fake links. The y-axis shows the processing time in microseconds and x-axis represents the number of fake links. The processing time of the validation phase of FoR-Guard is less as compared to both the phases including trigger and DeSI phase. The validation phase has only to apply measurement entropy calculation on the already build adjacency matrices. This task has

been performed by SV module which collects information from other modules responsible for DeSI phase of the FoR-Guard.

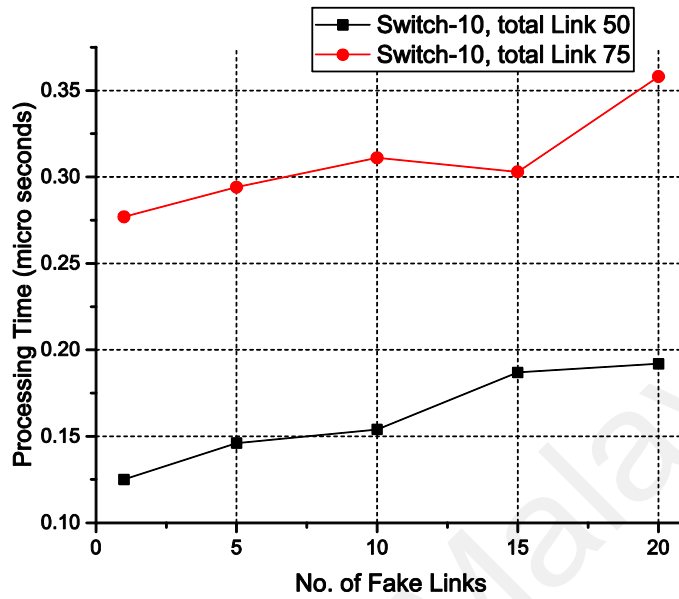


Figure 6.5: Processing time of validation phase for switches=10 and total links=(50,75)

The processing time of the controller in validation phase increases with the increasing number of fake links due to verifying more sources for generated fake links. For instance, when there are 10 switches and 01 fake links than the processing time is recorded as 0.125 microseconds as compared to 0.192 microseconds for 10 switches and 20 fake links out of 50 total links. Similarly, the processing time increases when the total number of links in the network topology increases. For instance, 10 switches and 10 fake links out of 50 total links in the network topology utilize 0.154 microseconds of the controller as compared to 10 switches and 10 fake links out of 75 total links which use 0.311 microseconds of the controller time. This trend has been observed due to calculating probabilistic values for adjacency matrices and measurement of the entropy values for each switch along with its comparison with other switches. Similarly, the processing time of the controller having 20 switches and (50,75) total links is shown in Figure 6.6.

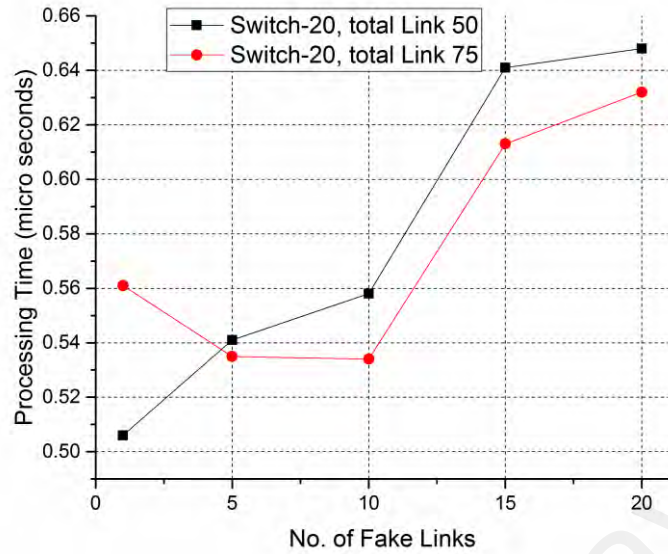


Figure 6.6: Processing time of validation phase for switches=20 and total links=(50,75)

6.2 Comparison of FoR-Guard phases in different scenarios

In this section, we analyze the processing time of the validation phase by comparing it with first two phases of FoR-Guard such as trigger and DeSI phases. The comparison is important to observe the effect of the validation phase regarding processing time of the controller by integrating it with trigger and DeSI phases of FoR-Guard. The validation phase verifies the source identified at the DeSI phase by computing the entropy measurement based on previous and current adjacency matrices. There is no state-of-the-art solution available that use a validation mechanism to verify the sources of LFA. Thus, the overall comparison shows that the validation phase imposes low processing time overhead on the controller as compared to the trigger and DeSI phases of FoR-Guard.

Figure 6.7 compares the processing time of first two phases (trigger + DeSI) with the third phase (validation) of the FoR-Guard. The x-axis and y-axis coordinates show a different number of fake links and processing time of the controller respectively. The aim of this comparison is to highlight how much processing time is increased by integrating the validation phase to the FoR-Guard.

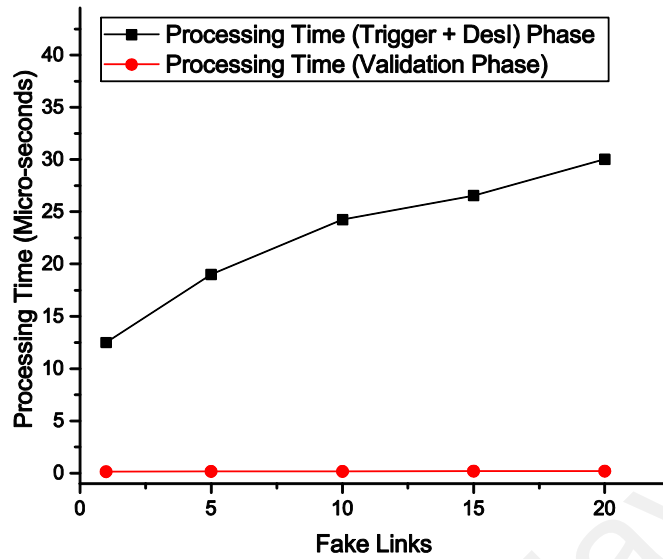


Figure 6.7: Processing time comparison of FoR-Guard phases in switches=10 and total links=50

Figure 6.7 clearly shows that only some part of a micro-second is added to the processing time of the controller regarding a different number of fake links. It happens because the FoR-Guard used an SV module which used an entropy measurement to validate the source of the attack. The SV compute the entropy weighted value based on the previous and current adjacency matrices which are already maintained by the LH module in the controller. The increase in the number of fake links increases the processing time of validation phase but not significantly. For instance, 0.146 and 0.192 microseconds are added to the processing time of the controller for verifying the sources for 05 and 20 fake links respectively. Thus, the validation phase plays an important role in FoR-Guard to verify the source of the attack by integrating the acceptable amount of processing time to the first two phase of the FoR-Guard running in the controller.

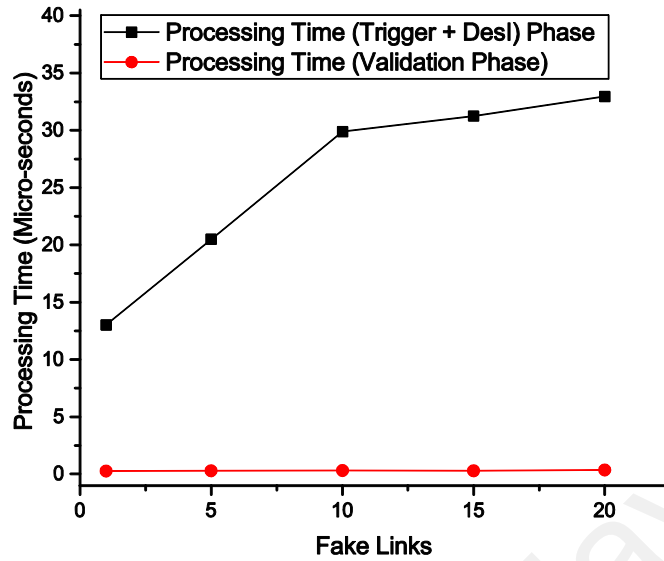


Figure 6.8: Processing time comparison of FoR-Guard phases in switches=10 and total links=75

Figure 6.8 depicts the comparison of validation phase processing time with trigger and DeSI phases processing time in terms of 10 switches and 75 total links. It is observed that increasing total number of links does not affect the processing time of the validation phase. For instance, the processing time of the validation phase was 0.146 and 0.192 microseconds for 05 and 20 fake links regarding 10 switches and 50 links as shown in Figure 6.7 above. However, the processing time of the validation phase is 0.294 and 0.358 microseconds for 05 and 20 fake links regarding 10 switches and 75 links.

Figure 6.9 shows the comparison of validation phase processing time with trigger and DeSI phase processing time regarding 20 switches and 50 total numbers of links. The result shows that the validation phase has less processing time as compared to trigger and DeSI phases of FoR-Guard.

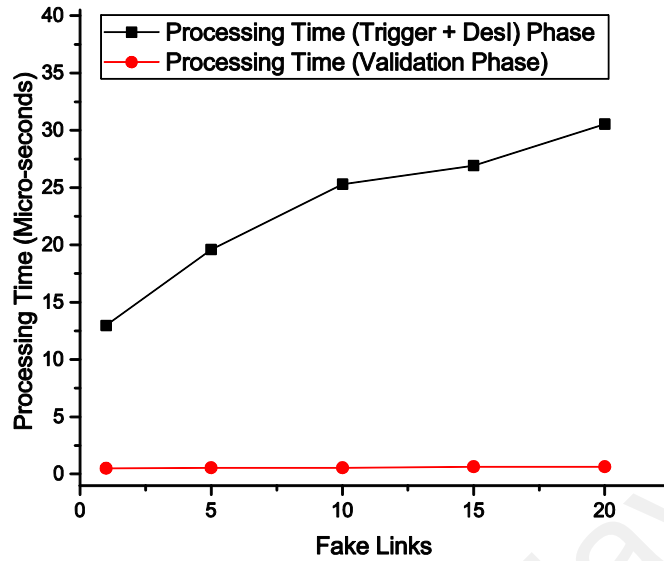


Figure 6.9: Processing time comparison of FoR-Guard phases in switches=20 and total links=50

Figure 6.10 shows that validation phase has processing time less than one microsecond regarding different fake links whereas, the trigger and DeSI phase have collective processing time up to 34 microseconds. The trigger phase has high processing time as compared to rest of the phases of FoR-Guard. The processing time is high because the topology manager module running in the controller discovers the network topology after specified interval of time which prolongs the processing time of the controller in the trigger phase. Therefore, the processing time for the validation phase has no negative impact on the overall performance of the FoR-Guard.

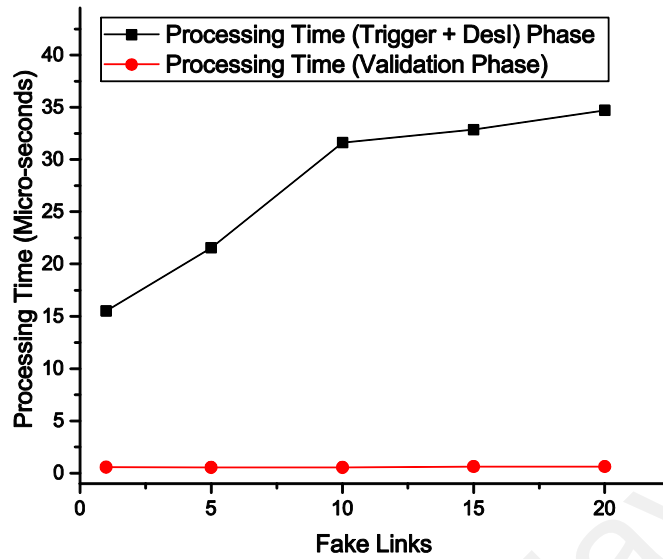


Figure 6.10: Processing time comparison of FoR-Guard phases in switches=20 and total links=75

6.3 Performance analysis of FoR-Guard through ROC analysis

In this section, we have used the Receiver Operating Characteristic (ROC) graphs to evaluate the classification capabilities of our proposed method FoR-Guard. The ROC graph is a useful tool to evaluate and visualize the classifiers. The ROC graph is a handy two-dimensional graph to depict the degree of trade-off between true positive rates (sensitivity) and false positive rates (1- specificity). The graph plots the true positive rate on y-coordinates representing the sensitivity and false positive rate on x-coordinates representing 1-specificity.

The curves in the ROC graph is of worth importance, the lower left point (0,0) represents that there is a gain in true positives and the classifier not commits any false positive errors. However, the upper left point in the ROC space depicts no false negatives and false positive indicating in 100% sensitivity and specificity. The diagonal line in the ROC graph represents the (0.5,0.5) changes for the values such as 50% sensitivity and specificity respectively. Thus, the points plotted on the curve above the diagonal line is better than the points plotted on the curves below the diagonal line in the ROC graph.

The ROC graph fits our evaluation criteria, as our focus is to determine the fake links as a fake link in SDN. We used ROC graphs because it provides better measures of classification performance as compared to different scalar measures including accuracy, error cost, and error rate. Moreover, ROC graphs have more advantages over precision-recall graphs and lift curves.

6.3.1 ROC graphs for 50 total number of links

The ROC graphs illustrated in figures (Figures 6.11-6.20) have been plotted based on the data collected from our experiments as shown in Tables (Table 5.6 – 5.25) in chapter 5. We have plotted the ROC graphs for the worst case of each experiment. For instance, Table 5.9 depicts the 10 experimental values for FoR-Guard to detect 15 fake links out of 50 total number of links. In this situation, FoR-Guard has correctly identified 15 fake links in the 9 experiments, however, in one experiment (experiment # 2) it detects 13 fake links. Thus, sensitivity remains 86.6%, specificity 94.5%, and false alarm rate 5.40%. The values have been converted into a percentage for the sake of simplicity. In the above example, we have only plotted ROC graphs for the lowest percentage value, i.e., experiment # 2 in the data collection table (Tables 5.9) presented in Chapter 5 same applies to all ROC graphs. The ROC graphs will be same for all the values which have 100% sensitivity.

Figure 6.11 shows the ROC graph for 10 and 20 switches with having one fake link in the pool of 50 total number of links. The x-axis and y-axis represent the true positive rate (sensitivity) and false positive rate (1- specificity) respectively. The figure 6.11 is plotted base on the experiment # 3 and experiment # 9 of Table 5.6 and 5.16 respectively. These experiments have been selected because of the minimum value of sensitivity present in the respective tables as discussed above. The ROC curves for 10 switches have 100% sensitivity with zero false positive rates. However, the ROC curve for 20 switches is far from the topmost left corner in the ROC space indicating with less sensitivity as compared

to 10 switches, i.e., 98%. Thus, FoR-Guard as a classifier identifies the single fake link in the case of 10 switches while in 20 switches it not identifies the fake link during the experiment # 9. It is due to the value of MIR value of the switch is equal to zero at the time the controller queries the switch.

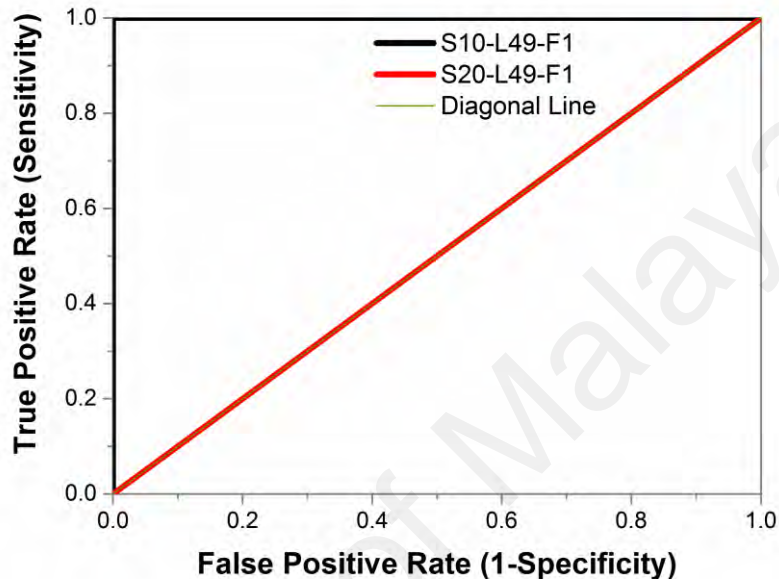


Figure 6.11: ROC graph for single fake link in total links=50

Figure 6.12 shows the detection accuracy of FoR-Guard by using ROC graph for 10 and 20 switches with having 05 fake links in the pool of 50 total number of links. The number of switches such as 10 and 20 and a total number of links, i.e., 50 and 75 is common for all of our experiments while the number of fake links varies. The ROC curves of both 10 and 20 switches show the better results as it is plotted above the diagonal reference line in the graph. However, the detection accuracy for 20 switches is better than 10 switches as it is closer to the left side of the ROC space. The closer to the left side, better the sensitivity is observed for the classifiers. It is observed that upper part of the curve is touching the upper side of the ROC space because our FoR-Guard identifies all the legitimate links. Our focus in experiments is only to check that how accurate our proposed method (FoR-Guard) is to identify fake links as a fake link in SDN.

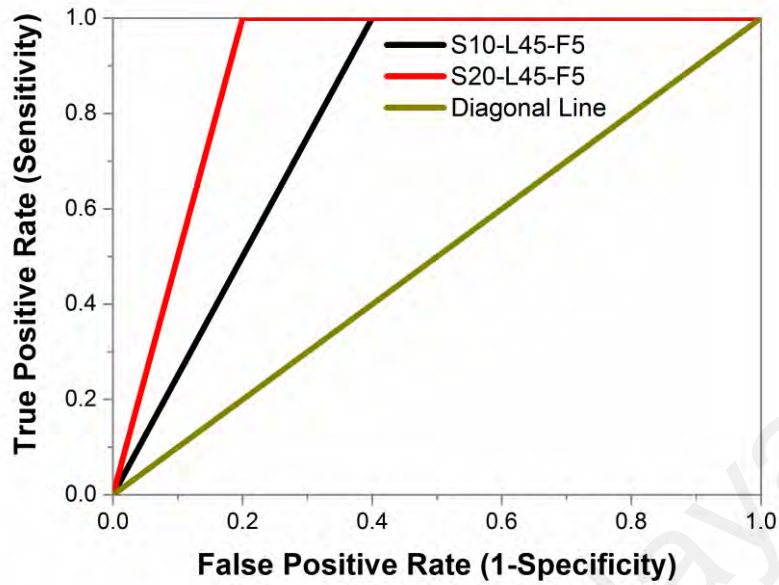


Figure 6.12: ROC graph for fake links=05 in total links=50

Figure 6.13 shows the ROC graph for 10 fake links in the pool of 50 total number of links. The both ROC curves of 10 and 20 switches are plotted above the diagonal reference line which indicates the better performance regarding the fake links. It is observed that increasing fake links do not affect much the sensitivity and false positive rate. It is due to adaptive trigger mechanism proposed in FoR-Guard which triggers an alarm message after checking the behavior of the switch, i.e., MIR value. Thus, it assists the controller in the DeSI phase to use the MIR value of the switch in detecting the fake links. The detection accuracy of 20 switches is better than 10 switches as it identifies 08 fake links as compared to 07 fake links identified by the 10 switches.

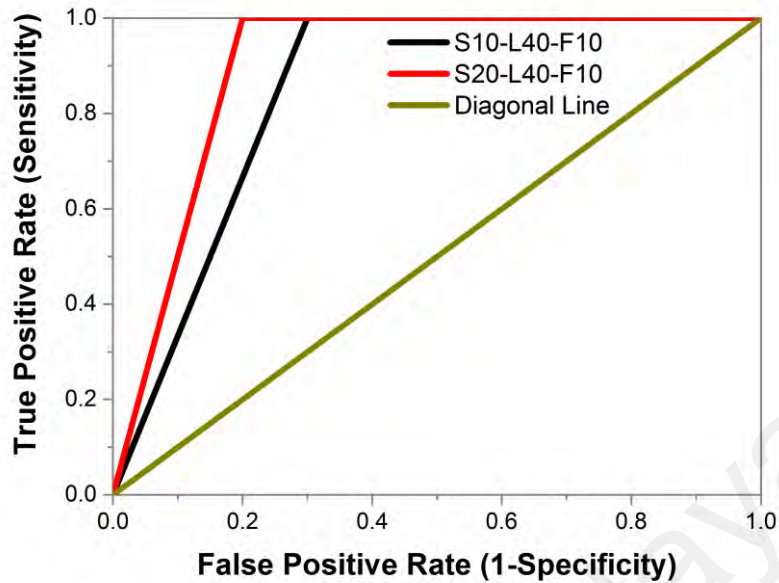


Figure 6.13: ROC graph for fake links=10 in total links=50

Figure 6.14 represents the ROC graph for 15 fake links. It is observed that in 20 switches experiment, FoR-Guard identifies all the fake links, i.e., 15 fake links with resulting in 100% sensitivity. However, FoR-Guard in the 10 switches identifies 13 fake links out of 15 links in its worst case. Thus, increase in the number of switches in SDN does not affect FoR-Guard in detecting the fake links.

Figure 6.15 shows the ROC graph generated base on the data collected from the Table 5.10 and 5.20 respectively. It is observed that curve for 20 switches is closer to the diagonal reference line while curve for 10 switches is closer to the left most side of the ROC space. The sensitivity is higher for 10 switches as compared to 20 switches. The FoR-Guard finds 18 and 16 fake links in the case of 10 and 20 switches respectively. However, in most of the case, FoR-Guard has 100% sensitivity and zero false alarm rate in different experimental setup. The Figure 6.15 is representing the worst case as fake links are generated randomly between different switches. The FoR-Guard may not detect the fake link generated by the switch if the switch has MIR value equal to zero or the link has a less frequent number of links generated (explained in chapter 4).

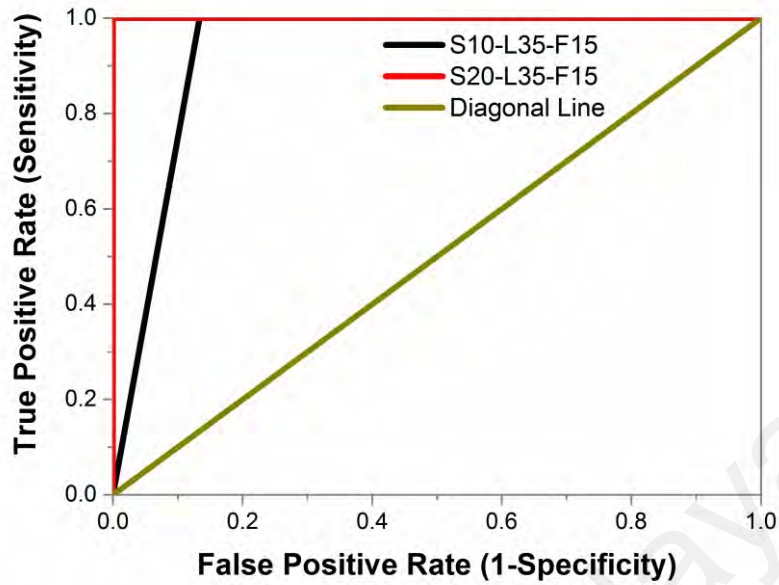


Figure 6.14: ROC graph for fake links=15 in total links=50

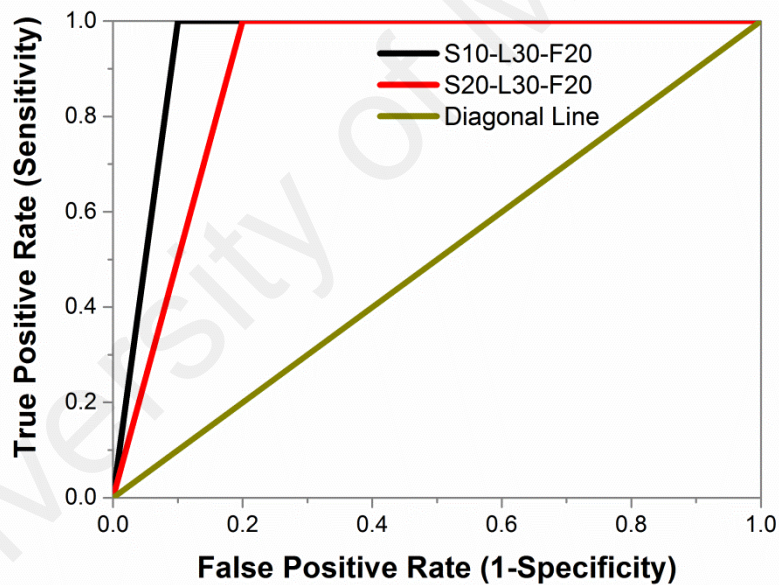


Figure 6.15: ROC graph for fake links=20 in total links=50

6.3.2 ROC graphs for 75 total number of links

In this section, we have explained the ROC graphs for 10 and 20 switches which have 75 total number of links. We have increased the total number of links such as from 50 to 75 links as discuss in the previous section. We have built the ROC graphs based on the data shown in Tables (5.6- 5.25) in chapter 5. We have generated the ROC graph for the

lowest percentage value of sensitivity in each table as the highest value is same for all experiments regarding fake links.

Figure 6.16 depicts the ROC graph for 10 and 20 switches with having one fake link in the pool of 75 total number of links. The experiment # 5 in Table 5.11 of Chapter 5 shows that FoR-Guard does not detect the fake link because the MIR value of the switch was zero at the time of the investigation. Therefore, the curves are plotted on top of the diagonal reference line of the ROC graph. However, all experiments performed for 20 switches have detected the fake link which results in the curve to be plotted on top of the leftmost line of the ROC space.

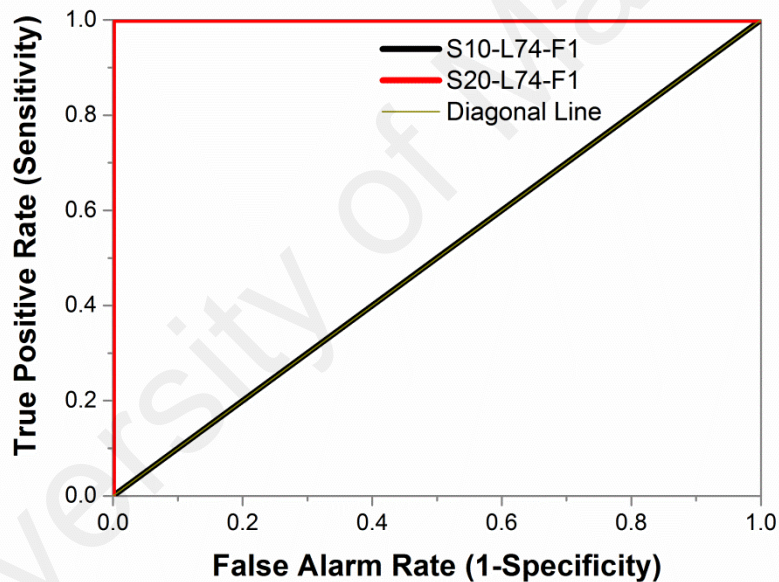


Figure 6.16: ROC graph for single fake link in total links=75

Figure 6.17 shows the comparison between the ROC curves for 05 fake links in 10 and 20 switches environment. The sensitivity of 10 switches is high than 20 switches. Therefore, its curve is closer to the left most side of the ROC space. The curve for 20 switches is closer to the diagonal reference line. However, the sensitivity value of 20 switches is still good as its curve is plotted above the diagonal line. The top part of both the curves in Figure 6.17 is horizontal to the upper part of the ROC graphs has FoR-Guard determines all the legitimate links as legitimate links in SDN.

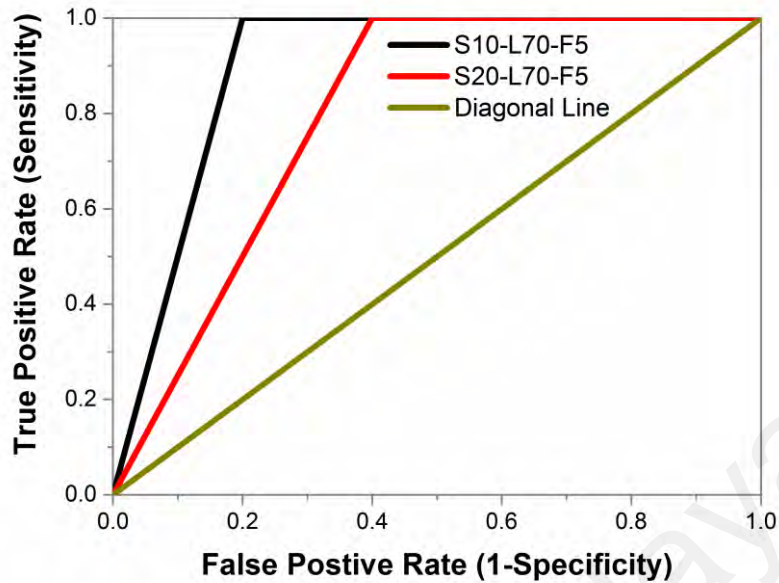


Figure 6.17: ROC graph for fake links=05 in total links=75

Figure 6.18 illustrates the ROC graphs generated based on the data of the Table 5.13 and 5.23 respectively. We can see that both the curves are plotted at the same points. The true positive rate is same for both the curves. The curves have plotted points above the diagonal reference line which indicates the FoR-Guard has good sensitivity rate with the acceptable false positive rate.

Figure 6.19 shows the ROC curves generated from the data selected from the Tables 5.14 and 5.24. The FoR-Guard has detected all the fake links regarding 20 switches. However, it does not detect 01 fake links in the pool of 15 fake links for 10 switches. It is due to MIR value is zero because the attacker hides its identity by generating legitimate links within SDN. Both the curves are plotted above the diagonal reference line which indicates the better sign of the classification.

Figure 6.20 represents the graphical representation of the experiment # 9 and 7 in 10 and 20 switches respectively. The ROC curve of 10 switches is above the 20 switches curve and much closer to the left side of the ROC space. The result shows higher true positive rate for the 10 switches as compared to the 20 switches. Moreover, we have concluded from our experiments that the detection of fake links is not affected by

increasing number of the switches, but it has some effect based on the total number of fake links in SDN.

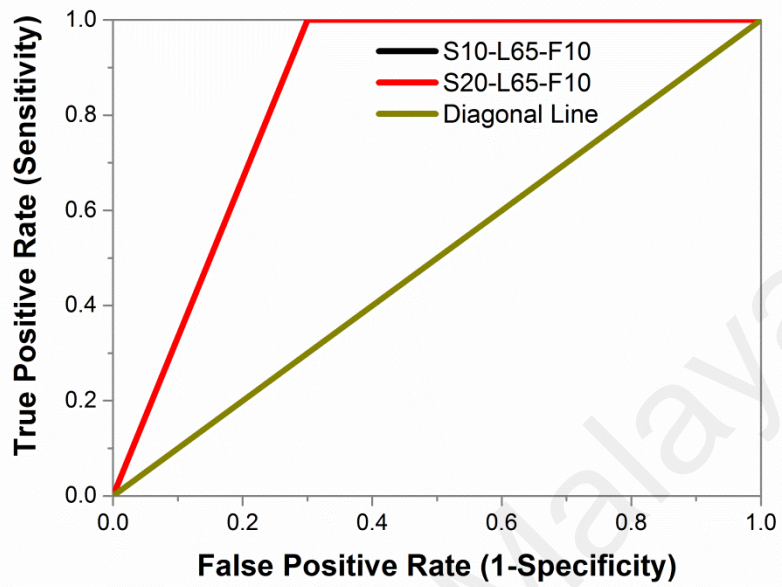


Figure 6.18: ROC graph for fake links=10 in total links=75

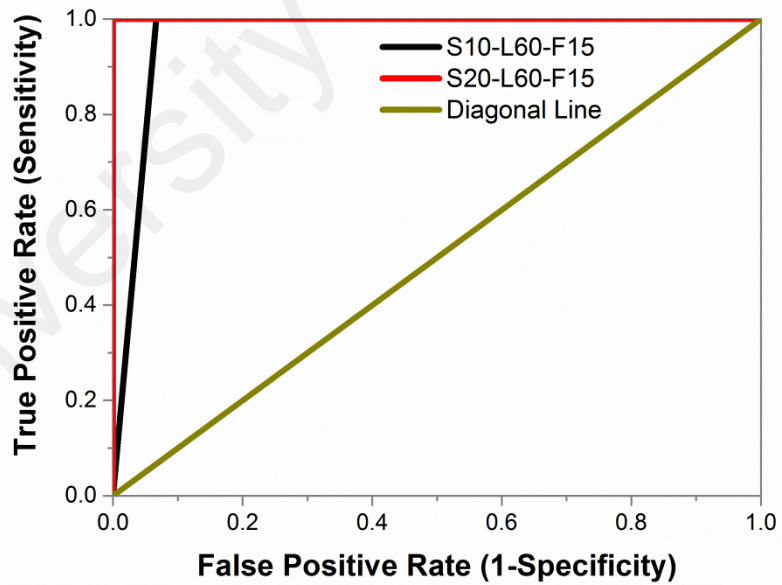


Figure 6.19: ROC graph for fake links=15 in total links=75

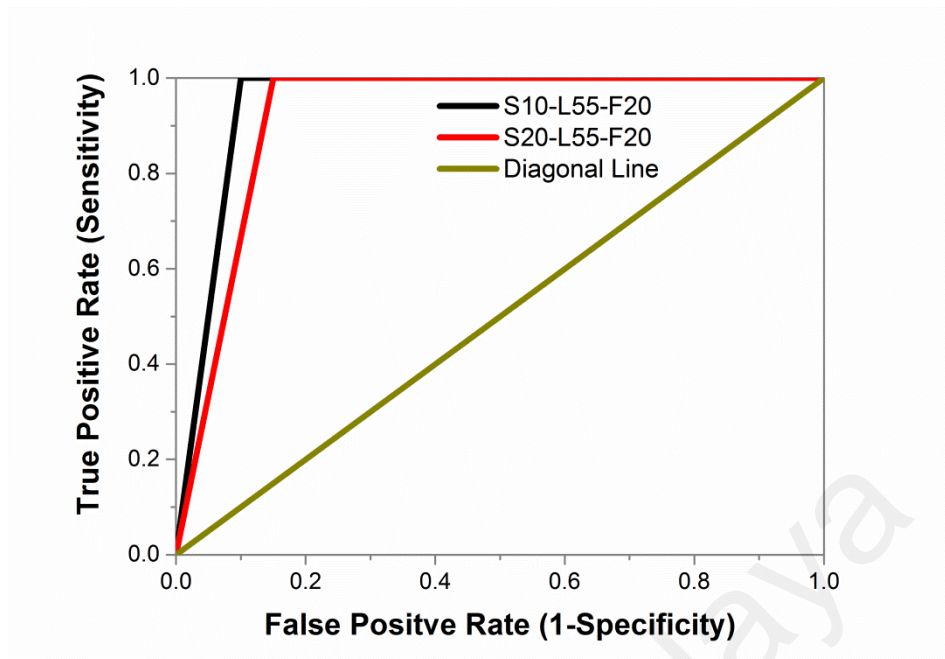


Figure 6.20: ROC graph for fake links=20 in total links=75

6.4 Comparison of FoR-Guard with existing solutions

In this section, we compare the performance of For-Guard with state-of-the-art solutions such as TopoGuard and Sphinx based on the processing time. The TopoGuard and Sphinx proposed solutions to detect LFA in SDN which are briefly explained in Sections 2.4.3.1 and 2.4.3.2 of Chapter 2 respectively.

6.4.1 Processing time

In this section, we compare the processing time of our proposed method FoR-Guard with the TopoGuard and Sphinx solutions. Herein, we present the results for 5 different network traces having a different number of fake links. Each network trace is used for 10 and 20 number of switches to analyze the processing time of FoR-Guard, TopoGuard, and Sphinx respectively. The processing time is less for FoR-Guard in all the network traces.

6.4.1.1 Empirical results of 10 switches and 50 total links

Figure 6.21 presents the comparison of FoR-Guard processing time, with the Sphinx and TopoGuard solutions in case of 10 switches and 50 total number of links. In this case, FoR-Guard takes 12.497 microseconds, whereas processing time for TopoGuard and

Sphinx is 38.54 and 24.21 microseconds in identifying one fake link in 10 switches and 50 total number of links respectively. The increased in processing time of TopoGuard is because the topology update checker module verifies the integrity of LLDP packet by placing signature TLV value in LLDP packets which is the cryptographic value of DPID and Port ID. Moreover, it also checks the LLDP propagation path to verify the LLDP is generated from the switch.

The processing time of Sphinx is less than TopoGuard but higher than FoR-Guard because it has to capture all the control OF messages to extract topological metadata from it to detect the malicious behavior observe in the traffic. Moreover, it builds the network graph from the metadata to detect the diversion of the flow behavior. Similarly, FoR-Guard has 30.031 microseconds processing time as compared to TopoGuard and Sphinx which have 89.94 and 68.49 microseconds respectively regarding 20 fake links.

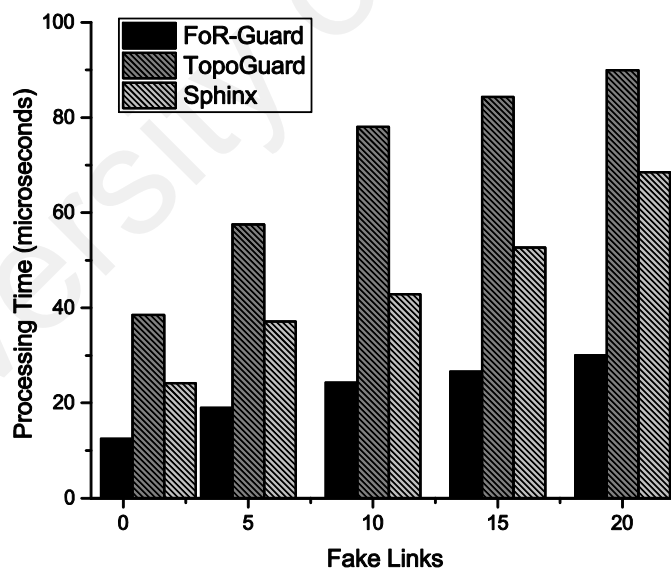


Figure 6.21: Processing time comparison of FoR-Guard with TopoGuard and Sphinx in switches=10 and total links=50

Statistical Validation

In the above section, we empirically analyze the processing time of FoR-Guard with the available existing solutions such as TopoGuard and Sphinx regarding 10 switches and 50 total links. In this section, we analyze the comparison results obtained from the experiments for FoR-Guard, TopoGuard, and Sphinx through statistical analysis. The statistical analysis allows us to know how much FoR-Guard is significant regarding processing time as compared to TopoGuard and Sphinx.

We used paired sample t-tests to validate the correctness of our empirical results (Lakens, 2013). Table 6.1 shows the results for paired sample t-tests. The value of (t-Stat) is 7.284325 and value of (p) are 0.001888. The result is significant as the value of $p \leq 0.05$.

Table 6.1: Paired sample t-tests for switches=10 and total links=50

Comparison	t-Stat	P(T<=t) two-tail
FoR-Guard <-> TopoGuard	7.28432507	0.001887591
FoR-Guard <-> Sphinx	4.942772	0.007801

6.4.1.2 Empirical results of 10 switches and 75 total links

In Figure 6.22, we compare FoR-Guard with TopoGuard and Sphinx by increasing the total number of links from 50 to 75. The results have shown that increasing total number of links have more effect on the TopoGuard and Sphinx but have less effect on the FoR-Guard. The TopoGuard processing time increases with the increasing number of the total link because it has to insert the cryptographic values in the LLDP packets. However, Sphinx processing time increases because it has to capture all OF control messages to extract the topological information. The control messages increase with the increase in total links. The FoR-Guard processing time is significantly less as compared to TopoGuard and Sphinx due to checking MIR value of the switch that generates the link.

Statistical Validation

Table 6.2 shows the results for paired sample t-tests to validate the correctness of our empirical comparison results. The value of P should be less than or equal to 0.005 to have statistically significant difference between the two comparisons. The value of P is 0.005 and 0.002195 in comparison of FoR-Guard <-> TopoGuard and FoR-Guard <-> Sphinx respectively. Thus, FoR-Guard has statistically significant values regarding processing time with a comparison of TopoGuard and Sphinx.

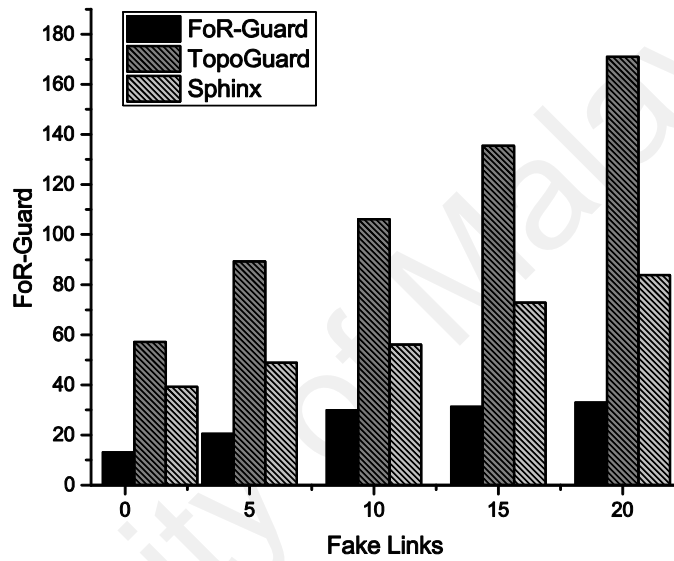


Figure 6.22: Processing time comparison of FoR-Guard with TopoGuard and Sphinx in switches=10 and total links=75

Table 6.2: Paired sample t-tests for switches=10 and total links=75

Comparison	t-Stat	P(T<=t) two-tail
FoR-Guard <-> TopoGuard	5.362788	0.005835
FoR-Guard <-> Sphinx	6.997278	0.002195

6.4.1.3 Empirical results of 20 switches and 50 total links

Figure 6.23 shows the processing time comparison of FoR-Guard with Sphinx and TopoGuard solutions regarding 20 switches and 50 total number of links. In this case, FoR-Guard takes 12.972 microseconds, whereas processing time for TopoGuard and

Sphinx is 49.75 and 30.74 microseconds in finding one fake link in 20 switches and 50 total number of links.

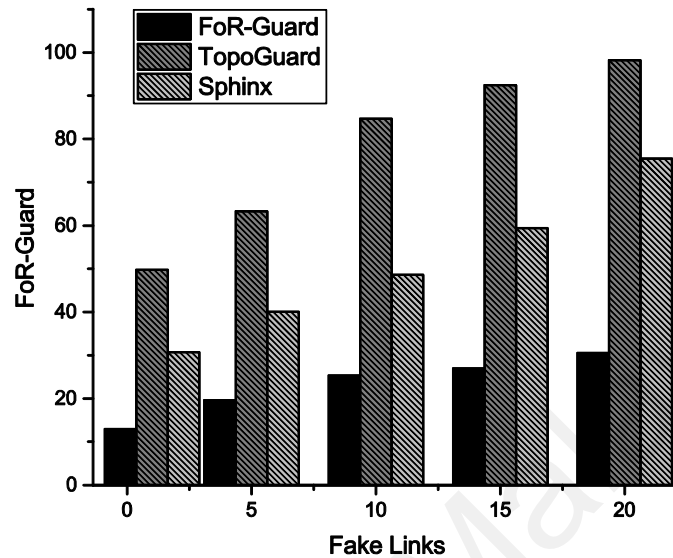


Figure 6.23: Processing time comparison of FoR-Guard with TopoGuard and Sphinx in switches=20 and total links=50

The increase in a number of switches has no significant effect on the processing time of FoR-Guard however, TopoGuard and Sphinx processing time is increased due to marking the LLDP packets and building network graphs for each switch respectively. The processing time of FoR-Guard for 15 fake links is 26.92 microseconds whereas, TopoGuard and Sphinx have 92.47 and 59.39 microseconds respectively.

Statistical Validation

Table 6.3 depicts the statistical results of paired sample t-tests to validate the correctness of comparison results of FoR-Guard with TopoGuard and Sphinx. The value of t-Stat is 8.914161 and 5.610224 for FoR-Guard <-> TopoGuard and FoR-Guard <-> Sphinx respectively. Also, the value of P is 0.000875 and 0.004959 for FoR-Guard <-> TopoGuard and FoR-Guard <-> Sphinx respectively. The value of t-Stat and P shows that the comparison results is valid regarding the significance difference available between FoR-Guard and existing solutions such as TopoGuard and Sphinx.

Table 6.3: Paired sample t-tests for switches=20 and total links=50

Comparison	t-Stat	P(T<=t) two-tail
FoR-Guard <-> TopoGuard	8.914161	0.000875
FoR-Guard <-> Sphinx	5.610224	0.004959

6.4.1.4 Empirical results of 20 switches and 75 total links

As mentioned above, the increase in a number of switches and links have no significant impact on the processing time of FoR-Guard whereas, it affect the processing time of TopoGuard and Sphinx. In the case of 10 fake links, FoR-Guard has a processing time of 31.63 microseconds whereas TopoGuard and Sphinx have 125.41 and 69.88 microseconds respectively as shown in Figure 6.24. In the case of 20 fake links, For Guard has a processing time of 34.72 microseconds whereas TopoGuard and Sphinx have 198.21 and 99.45 microseconds respectively.

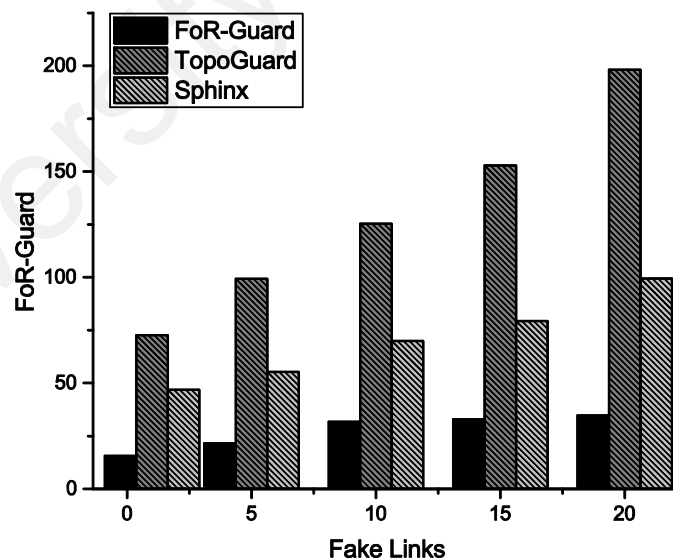


Figure 6.24: Processing time comparison of FoR-Guard with TopoGuard and Sphinx in switches=20 and total links=75

Statistical Validation

Table 6.4 presents the statistical results of paired sample t-tests to validate the correctness of comparison results of FoR-Guard with TopoGuard and Sphinx in 20 switches and 75 total links. The value of t-Stat is 5.563618 and 7.102225 for FoR-Guard <-> TopoGuard and FoR-Guard <-> Sphinx respectively. Also, the value of P is 0.005 and 0.002076 for FoR-Guard <-> TopoGuard and FoR-Guard <-> Sphinx respectively. The value of P is less than or equal than 0.005 which indicates the comparison difference between FoR-Guard and existing solutions such as TopoGuard and Sphinx is significant.

Table 6.4: Paired sample t-tests for switches=20 and total links=75

Comparison	t-Stat	P(T<=t) two-tail
FoR-Guard <-> TopoGuard	5.563618	0.005111
FoR-Guard <-> Sphinx	7.102225	0.002076

6.5 Conclusion

This chapter presents the graphical representation of the experimental results for the FoR-Guard method regarding controller processing time and detection accuracy regarding detecting and identifying fake links in SDN. The experiment was conducted based on specific criteria such as (a) number of switches, (b) number of total links, and the (c) number of fake links. The processing time of each phase of FoR-Guard was tested based on the criteria mentioned above. The processing time of the trigger phase is high as compared to other two phase, i.e., DeSI and validation phases because the controller has to discover network nodes in each topology discovery round. The validation phase of FoR-Guard has less processing time because the controller has to compute entropy value for each switch based on the information available from the DeSI phase.

Moreover, the performance of the FoR-Guard in detecting fake links was analyzed through plotting ROC graphs to know about the tradeoff between the sensitivity and false alarm rate. Mostly, the ROC curves plotted above the diagonal curve in ROC graphs,

mentioning that the detection rate of FoR-Guard is effective. To validate the performance of FoR-Guard, we perform a comparison with TopoGuard and Sphinx by measuring their processing time. The processing time of trigger and DeSI phases is used for FoR-Guard in comparing the results with TopoGuard and Sphinx because the validation phase is not available in the latter solutions.

We also validated the results by using the paired sample t-tests to study whether there is a significant difference between the mean values of the experimental results or not. The result concludes that there is a significant difference between FoR-Guard and existing solutions such as TopoGuard and Sphinx regarding the controller processing time. Thus, it makes FoR-Guard efficient from TopoGuard and Sphinx in detecting fake links in SDN.

University of Malaya

CHAPTER 7: CONCLUSION

This chapter concludes the overall research presented in this thesis and highlights the future research directions. We re-examine the research objectives by re-evaluating them again to ensure that we have accomplished it in our thesis. Moreover, the contribution of the research has been highlighted, and research scope with its limitations has been discussed. Finally, the future work has been highlighted to improve this research.

The complete organization of this chapter is as follows. Section 7.1 describes the different ways that how the research objectives were fulfilled. Section 7.2 highlights the main contribution of the research. Section 7.3 examines the research scope and limitation of the research work. Section 7.4 highlights the future work which can further improve this research work.

7.1 Achievement of research objectives

The main purpose of this study is to investigate the problem of LFA in SDN. The proposed method is capable of having a dynamic triggering mechanism, efficient detection and source identification method, and validation process. In following, it has been shown that four research objectives have been achieved which was highlighted in Section 1.4 of Chapter 1.

The first research objective was to review the state-of-the-art topology poisoning attack techniques to have insight about the LFA in SDN. First, topology discovery process of SDN is studied by devising a thematic taxonomy. Second, different topology discovery threats are classified to gain enough knowledge about how the adversary performs the malicious behavior to affect the network topology. It enables to focus specifically on fake links generated in the network topology during the time of LFA. Third, we have studied different research articles related topology discovery security collected from various digital libraries including Science Direct, IEEE, Springer, ACM, and Elsevier. Also, we continue to read and study various SDN security web pages and web blogs to have

updated knowledge about SDN security including topology discovery, LFA, and controller vulnerabilities. A qualitative analysis is performed on the collected research materials which lead to identifying the open research issues for the security of topology discovery in SDN.

The second objective was to formally investigate and analyze the impact of fake links in the network topology of SDN. We first formally represent the SDN infrastructure through HOL formal methods. After that, we represent SDN as a service through HOL with unique identification protocols such as service name, service version, service protocol, the port used for service communication, and access policy. Moreover, LFA is formally represented with a HOL and analyzed through various parameters including effectiveness, sensitivity, specificity, and false alarm rate. It establishes the problem of LFA in SDN as non-trivial.

The third objective was to design and develop the method to detect the fake links in SDN along with its identification of malicious sources. A forensic investigation method has been proposed to address the issue of LFA in SDN. The proposed method used a dynamic triggering mechanism to trigger the DeSI phase to minimize the processing time of the controller in investigating fake links. The trigger phase of the proposed method reduces the controller overhead by only triggering a message for the links generated from the malicious switches as compared to the current techniques which investigate all the new inserted links in the network topology. The detection and source identification phase of the proposed method investigate the suspicious link by looking towards the LCD and MIR of the switch. It reduces false alarm rates by investigating metadata of the malicious links and switches to detect the fake links in the network topology. The proposed method also reduce the computation overhead of the controller by avoiding complex algorithms used to detect the fake links. After the detection of the fake link, a traceback technique is used to identify the real source of the attack to mitigate the attack in the future. The source

of the fake link is verified in the validation phase of our proposed method by using entropy measurement. The entropy determines the uncertainty in the network generated through the fake links which lead towards source identification of the LFA.

The final objective was to evaluate the performance of the proposed method with the state-of-the-art LFA solutions. The proposed solution has been implemented in SDN emulated environment by using Mininet and Floodlight controller version 1.0. The performance verification of the proposed method was done through different metrics namely, effectiveness, sensitivity, specificity, and false alarm rates. The experimental results have shown that the proposed method performs better than existing solutions regarding controller processing time, computation overhead, and false alarm rate as compared to the state-of-the-art solutions.

7.2 Contribution of the research

In this section, we explain some contributions to the body of knowledge which are summarized as follows.

7.2.1 Thematic taxonomy

We devise a thematic taxonomy of topology discovery in SDN to have a conceptual knowledge of topology discovery by grouping the concepts that participate in the same scenario. The taxonomy is classified into four different categories by topology discovery features namely, discovery entities, controller platform, topology-dependent services, and objectives.

We aim to analyze the critical aspect of the state-of-the-art topology discovery security techniques and compare the techniques by significant parameters. The literature review helps us to identify open issues of the topology discovery security solutions that need to be addressed.

7.2.2 Formal representation of SDN

We contribute to the body of knowledge by representing SDN as an HOL formal method. The HOL assist network operators in protecting the systems or network from unexpected errors that might not be detected at the time of development and deployment. Similarly, it is important to verify the security properties of the programs that control the network before it is deployed in SDN for its execution.

Thus, an effective approach to check and ensure the security of these programs or systems is to use HOL formal methods. Thus, we formally modelled the LFA using HOL formal methods. The LFA is analyzed through various performance metrics namely, effectiveness, sensitivity, specificity, and false alarm rate.

7.2.3 Dynamic trigger mechanism for fake links

The next contribution of this research is a proposed dynamic trigger mechanism which triggers an alarm message to the DeSI phase (second phase of our proposed method) on the observation of the suspicious link. The dynamic trigger mechanism reduces the processing time of the controller by investigating only the links which have been generated by the malicious switches.

The malicious switches are identified through the MIR value which has been generated based on their historical link generation stats. Thus, dynamic trigger mechanism allows the controller to have less computational cost due to only investigating suspicious links rather than investigating all newly generated links in the network topology.

7.2.4 Algorithms for detection and source identification of fake links

We have designed and developed three algorithms to detect and determine the real source of fake links in SDN. The developed modules assist the controller to identify the status of the link such as a legitimate or fake link. The proposed algorithms acquire malicious switch information (MIR) from the previous phase and identify the link communication direction (LCD) of the link which is under investigation.

The integration of MIR and LCD enables the controller to identify the status of the link. If the status of the link is declared as a fake link then the proposed algorithm trace back the malicious source of the fake link. Otherwise, the controller is informed to update its network topology due to the legitimate link.

7.2.5 Validation of source identification

This contribution strengthens our research work by validating the source of the fake link that has been identified in the detection phase of our proposed method. The source identification is an important characteristic of the forensic investigation which leads the controller to prevent the attack in the future. We validate the source of the fake links through the concept of information theory such as entropy measurement.

The entropy is used as a tool to confirm that the source of the fake link identified previously is correct or not. It may create some extra computational burden on the controller, but it assists the controller to update the network topology with correct information.

7.2.6 FoR-Guard Evaluation

We contribute to the knowledge by evaluating our proposed method with the state-of-the-art solutions which detects the fake links in the network topology. The results verify that FoR-Guard used less processing time of the controller to detect the fake links as compared to the current solutions. Moreover, the forensic aspect of the FoR-Guard leads to identify the real source of the fake links which can be prevented through efficient mitigation techniques.

7.2.7 Accepted journal articles from thesis

1. **Suleman Khan**, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Mohsen Guizani, and Muhammad Khurram Khan, Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-art, IEEE Communications Surveys & Tutorials, Accepted 19 July 2016 (ISI, Q1, Impact Factor: 9.22)

2. **Suleman Khan**, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Ahmed Abdelaziz, Muhammad Khurram Khan, and Mohsen Guizani, Software-defined Network Forensics: Motivation, Potential Locations, Requirements, and Challenges, IEEE Networks, Accepted 23 August 2016 (**ISI Q1, Impact Factor: 2.88**)

7.2.8 Accepted journal articles in other research areas

1. **Suleman Khan**, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, Mustapha Aminu Bagiwa, Samee U. Khan, Raj Kumar Buyya, and Albert Y. Zomaya, Cloud Log Forensics: Foundations, State-of-the-art, and Future Directions, ACM Computing Surveys, Volume 49 Issue 1, June 2016 Article No. 7 (**ISI Q1, Impact Factor: 3.37**)

2. **Suleman Khan**, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad, Network Forensics: Review, Taxonomy, and Open Challenges, Journal of Network and Computer Applications, Volume 66, May 2016, Pages 214–235 (**ISI, Q1, Impact Factor: 2.22**)

3. **Suleman Khan**, Muhammad Shiraz, Ainuddin Wahid Abdul Wahab, Abdullah Gani, Qi Han, and Zulkanain Bin Abdul Rahman, A Comprehensive Review on Adaptability of Network Forensics Frameworks for Mobile Cloud Computing, The Scientific World Journal, July 2014, Pages 27 (**ISI Q1, Impact Factor 1.73**)

7.2.9 Accepted journal articles in collaboration with group members

1. Thomas, Bimba Andrew, Norisma Idris, Ahmed Al-Hnaiyyan, Rohana Binti Mahmud, Ahmed Abdelaziz, **Suleman Khan**, and Victor Chang. "Towards Knowledge Modeling and Manipulation Technologies: A Survey." International Journal of Information Management, Volume 36, Issue 6, Part A, December 2016, Pages 857–871 (**ISI, Q1, Impact Factor: 2.69**)

2. Mahdi, Omar Adil, Ainuddin Wahid Abdul Wahab, Mohd Yamani Idna Idris, Ammar Abu Znaid, Yusor Rafid Bahar Al-Mayouf, and **Suleman Khan**. WDARS: A

Weighted Data Aggregation Routing Strategy with Minimum Link Cost in Event-Driven WSNs, Journal of Sensors, Accepted 25 May 2016 (**ISI, Q3, Impact Factor: 1.18**)

3. Mustapha Aminu Bagiwa, Ainuddin Wahid Abdul Wahab, Mohd Yamani Idna Idris, and **Suleman Khan**, Digital Video Inpainting Detection Using Correlation Of Hessian Matrix, Malaysian Journal of Computer Science, Accepted 11 May 2015 (**ISI, Q4, Impact Factor: 0.40**)

4. Shiraz, Muhammad, Abdullah Gani, Azra Shamim, **Suleman Khan**, and Raja Wasim Ahmad. Energy efficient computational offloading framework for mobile cloud computing, Journal of Grid Computing, Vol 13, Issue 1, March 2015, Pages 1-18. (**ISI, Q1, Impact Factor: 1.50**)

5. Qi, Han, Muhammad Shiraz, Abdullah Gani, Md Whaiduzzaman, and **Suleman Khan**, Sierpinski triangle based data center architecture in cloud computing, The Journal of Supercomputing, Vol 69, Issue 2, August 2014, Pages 887-907 (**ISI, Q2, Impact Factor: 0.84**)

6. Gani, Abdullah, Golam Mokatder Nayeem, Muhammad Shiraz, Mehdi Sookhak, Md Whaiduzzaman, and **Suleman Khan**. A review on interworking and mobility techniques for seamless connectivity in mobile cloud computing, Journal of Network and Computer Applications, Vol 43, August 2014, Pages 84–102 (**ISI, Q1, Impact Factor: 1.77**)

7.3 Research scopes and limitations

The proposed forensic investigation method of LFA is effective for all types of SDN controllers. The proposed modules in the proposed method are capable enough to detect single and multiple fake links in the network topology. Moreover, the abstract level of design and development of the modules enables to be adapted to the multiple SDN controller environments. Thus, it increases the scalability factor of the forensic investigation in SDN with an incorporating minimum level of design efforts.

The third phase of the proposed method, i.e., validation phase verifies the real source of the fake links with an execution and computational cost of the controller. The validation phase has been integrated into the proposed solution to verify the real source as it is considered to be an important factor in investigating the attack. Also, the proposed method depends on both MIR switch information and LCD of the link and assumes that each of the information will be untampered from any modification. The fake link cannot be identified correctly when the switch information is spoofed which provides incorrect information to the modules running in the controller. Finally, the proposed modules have been executed in the Floodlight controller 1.0 in the Mininet emulation environment and are not guaranteed to work on other versions of the controllers.

7.4 Future work

In this section, we enlist a possible future work that can be elevated from this research study.

1. In this research study, we have only target LFA which poison the topological view of the controller in SDN. However, there include other possible ways to poison the topological view of the controller. For instance, host hijacking attacks in which the controller is informed with spoof host location information which pretends the host is migrated to another new location in the network. However, the host remains at its previous location and the controller start sending the information to the wrong location that has been pretended by the malicious host. It may cause eavesdropping of information which may further assist in other attacks.

2. We aim to detect the LFA in heterogeneous OF environment where OF switches communicate with non-OF switches. Our research study has focused the fake links in the network having only OF switches. In future, we are going to implement our proposed modules in heterogeneous OF environment to validate the performance of our proposed method.

3. We also aim to address the issue of predicting the communication patterns of the link to assist the controller in identifying the fake links in SDN. The communication pattern of the link traffic will enable the controller to identify the behavior of the malicious switches, especially in highly dynamic SDN environment.

4. The authentication mechanism of the LLDP packets can significantly reduce the fake links in SDN. The integration of such a mechanism can improve the efficiency and reliability of the topology discovery process in SDN. It will lead towards an improvement in the throughput of the application-dependent services running on top of the controller.

University of Malaysia

REFERENCES

- Akyildiz, Ian F, Lee, Ahyoung, Wang, Pu, Luo, Min, & Chou, Wu. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, 71, 1-30.
- Akyildiz, Ian F, Lee, Ahyoung, Wang, Pu, Luo, Min, & Chou, Wu. (2016). Research challenges for traffic engineering in software defined networks. *IEEE Network*, 30(3), 52-58.
- Alharbi, Talal, Portmann, Marius, & Pakzad, Farzaneh. (2015). *The (In) Security of Topology Discovery in Software Defined Networks*. Paper presented at the Local Computer Networks (LCN), 2015 IEEE 40th Conference on.
- Antikainen, Markku, Aura, Tuomas, & Särelä, Mikko. (2014). Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch *Secure IT Systems* (pp. 229-244): Springer.
- Aslan, Mohamed, & Matrawy, Ashraf. (2016). On the Impact of Network State Collection on the Performance of SDN Applications.
- Bakshi, Kapil. (2013). *Considerations for software defined networking (SDN): Approaches and use cases*. Paper presented at the Aerospace Conference, 2013 IEEE.
- Banikazemi, Mohammad, Olshefski, David, Shaikh, Ali, Tracey, John, & Wang, Guohui. (2013). Meridian: an SDN platform for cloud network services. *Communications Magazine, IEEE*, 51(2), 120-127.
- Berde, Pankaj, Gerola, Matteo, Hart, Jonathan, Higuchi, Yuta, Kobayashi, Masayoshi, Koide, Toshio, . . . Snow, William. (2014). *ONOS: towards an open, distributed SDN OS*. Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking.
- Blenk, A., Basta, A., Reisslein, M., & Kellerer, W. (2016). Survey on Network Virtualization Hypervisors for Software Defined Networking. *IEEE Communications Surveys & Tutorials*, 18(1), 655-685. doi: 10.1109/COMST.2015.2489183
- Braun, Wolfgang, & Menth, Michael. (2014). Software-Defined Networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet*, 6(2), 302-336.
- Cho, Sungryung, Chung, Sungmoon, Lee, Wooyeob, Joe, Inwhae, Park, Jeman, Lee, Soohyung, & Kim, Wontae. (2015). An Software Defined Networking Architecture Design Based on Topic Learning-Enabled Data Distribution Service Middleware. *Advanced Science Letters*, 21(3), 461-464.
- Choudhary, Jagjit. (2010). Distributed BPDU processing for spanning tree protocols: Google Patents.

- Chowdhury, Shubhajit Roy, Bari, M Faizul, Ahmed, Rizwan, & Boutaba, Raouf. (2014). *Payless: A low cost network monitoring framework for software defined networks*. Paper presented at the Network Operations and Management Symposium (NOMS), 2014 IEEE.
- Chung, Chun-Jen, Khatkar, Pankaj, Xing, Tianyi, Lee, Jeongkeun, & Huang, Dijiang. (2013). NICE: Network intrusion detection and countermeasure selection in virtual network systems. *Dependable and Secure Computing, IEEE Transactions on*, 10(4), 198-211.
- Civanlar, Seyhan, Lokman, Erhan, Kaytaz, Bulent, & Murat Tekalp, A. (2015). *Distributed management of service-enabled flow-paths across multiple SDN domains*. Paper presented at the Networks and Communications (EuCNC), 2015 European Conference on.
- Clausen, Thomas, Jacquet, Philippe, Adjih, Cédric, Laouiti, Anis, Minet, Pascale, Muhlethaler, Paul, . . . Viennot, Laurent. (2003). Optimized link state routing protocol (OLSR).
- Cui, Laizhong, Yu, F Richard, & Yan, Qiao. (2016). When big data meets software-defined networking: SDN for big data and big data for SDN. *Network, IEEE*, 30(1), 58-65.
- Darren, Bounds (2016). Packet Network Injection and Capture. Retrieved 18-09-2016, 2016, from <http://packetfactory.openwall.net/projects/packit/>
- de Oliveira, Rogério Leão Santos, Schweitzer, Christiane Marie, Shinoda, Ailton Akira, & Prete, Ligia Rodrigues. (2014). *Using mininet for emulation and prototyping software-defined networks*. Paper presented at the Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on.
- Dhawan, Mohan, Poddar, Rishabh, Mahajan, Kshiteej, & Mann, Vijay. (2015). *SPHINX: Detecting Security Attacks in Software-Defined Networks*. Paper presented at the NDSS.
- Dixit, Advait, Hao, Fang, Mukherjee, Sarit, Lakshman, TV, & Kompella, Ramana. (2013). Towards an elastic distributed SDN controller. *ACM SIGCOMM Computer Communication Review*, 43(4), 7-12.
- Dixon, Colin, Olshefski, David, Jain, Vinesh, Decusatis, Casimer, Felter, Wes, Carter, Jenny, . . . Recio, Renato. (2014). Software defined networking to support the software defined environment. *IBM Journal of Research and Development*, 58(2/3), 3: 1-3: 14.
- Dover, Jeremy M. (2013). A denial of service attack against the Open Floodlight SDN controller: Dover Networks LCC.
- Drutskoy, Dmitry, Keller, Eric, & Rexford, Jennifer. (2013). Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, 17(2), 20-27.

- Feamster, Nick, Rexford, Jennifer, & Zegura, Ellen. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2), 87-98.
- Fonseca, Paulo, Bennesby, Ricardo, Mota, Edjard, & Passito, Alexandre. (2012). *A replication component for resilient OpenFlow-based networking*. Paper presented at the Network Operations and Management Symposium (NOMS), 2012 IEEE.
- Gani, Abdullah, Nayeem, Golam Mokatder, Shiraz, Muhammad, Sookhak, Mehdi, Whaiduzzaman, Md, & Khan, Suleman. (2014). A review on interworking and mobility techniques for seamless connectivity in mobile cloud computing. *Journal of Network and Computer Applications*, 43, 84-102.
- Goncalves, Patricia, Martins, Alfredo, Corujo, Daniel, & Aguiar, Rui. (2014). *A fail-safe SDN bridging platform for cloud networks*. Paper presented at the Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International.
- Hakiri, Akram, Gokhale, Aniruddha, Berthou, Pascal, Schmidt, Douglas C, & Gayraud, Thierry. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, 75, 453-471.
- Haleplidis, Evangelos, Pentikousis, Kostas, Denazis, Spyros, Salim, J Hadi, Meyer, David, & Koufopavlou, Odysseas. (2015). Software-defined networking (sdn): Layers and architecture terminology.
- Hamel, Danielle. (2014). New Juniper Networks Study Finds U.S. Companies Split on Adopting Software-Defined Networking. Retrieved from <http://newsroom.juniper.net/press-releases/new-juniper-networks-study-finds-u-s-companies-sp-nyse-jnpr-1134411>
- Heller, Brandon, Sherwood, Rob, & McKeown, Nick. (2012). *The controller placement problem*. Paper presented at the Proceedings of the first workshop on Hot topics in software defined networks.
- Hollander, Jeremy. (2007). *A Link Layer Discovery Protocol Fuzzer*: Citeseer.
- Hong, Sungmin, Xu, Lei, Wang, Haopei, & Gu, Guofei. (2015). *Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures*. Paper presented at the NDSS.
- Hu, Hongxin, Han, Wonkyu, Ahn, Gail-Joon, & Zhao, Ziming. (2014). *FLOWGUARD: building robust firewalls for software-defined networks*. Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking.
- Hu, Zhiyuan, Wang, Mingwen, Yan, Xueqiang, Yin, Yueming, & Luo, Zhigang. (2015). *A comprehensive security architecture for SDN*. Paper presented at the Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on.
- Hwang, Sungmin, & Kim, Kyungbaek. Middlebox Driven Security Threats in Software Defined Network.

- Jajodia, Sushil, Noel, Steven, & O'Berry, Brian. (2005). Topological analysis of network attack vulnerability *Managing Cyber Threats* (pp. 247-266): Springer.
- Jarraya, Yosr, Madi, Taous, & Debbabi, Mourad. (2014). A survey and a layered taxonomy of software-defined networking. *Communications Surveys & Tutorials, IEEE, 16*(4), 1955-1980.
- Jarschel, Michael, Wamser, Florian, Hohn, Thomas, Zinner, Thomas, & Tran-Gia, Phuoc. (2013). *Sdn-based application-aware networking on the example of youtube video streaming*. Paper presented at the Software Defined Networks (EWSN), 2013 Second European Workshop on.
- Jim, Metzler. (2012). Understanding Software-Defined Networks *InformationWeek Software-Defined Networking Survey* (pp. 1-26). North America: InformationWeek Reports.
- Kamisiński, Andrzej, & Fung, Carol. (2015). *FlowMon: Detecting Malicious Switches in Software-Defined Networks*. Paper presented at the Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense.
- Karakus, Murat, & Duresi, Arjan. (2015). *A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN)*. Paper presented at the Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on.
- Karimzadeh, Morteza, Valtulina, Luca, & Karagiannis, Georgios. (2014). Applying sdn/openflow in virtualized lte to support distributed mobility management (dmm).
- Katz, Dave, Kompella, K, & Yeung, D. (2003). Traffic engineering (TE) extensions to OSPF version 2: RFC 3630, September.
- Kempf, James, Bellagamba, Elisa, Kern, András, Jocha, David, Takács, Attila, & Sköldström, Pontus. (2012). *Scalable fault management for OpenFlow*. Paper presented at the Communications (ICC), 2012 IEEE International Conference on.
- Khan, S., Gani, A., Wahab, A. Abdul, Guizani, M., & Khan, M. K. (2016). Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-art. *IEEE Communications Surveys & Tutorials, PP*(99), 1-1. doi: 10.1109/COMST.2016.2597193
- Khan, Suleman, Ahmad, Ejaz, Shiraz, Muhammad, Gani, Abdullah, Wahab, Ainuddin Wahid Abdul, & Bagiwa, Mustapha Aminu. (2014). *Forensic challenges in mobile cloud computing*. Paper presented at the Computer, Communications, and Control Technology (I4CT), 2014 International Conference on.
- Khan, Suleman, Gani, Abdullah, Wahab, Ainuddin Wahid Abdul, & Bagiwa, Mustapha Aminu. (2015). *SIDNFF: Source identification network forensics framework for cloud computing*. Paper presented at the Consumer Electronics-Taiwan (ICCE-TW), 2015 IEEE International Conference on.

- Khan, Suleman, Gani, Abdullah, Wahab, Ainuddin Wahid Abdul, Shiraz, Muhammad, & Ahmad, Iftikhar. (2016). Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 66, 214-235.
- Khan, Suleman, Shiraz, Muhammad, Abdul Wahab, Ainuddin Wahid, Gani, Abdullah, Han, Qi, & Bin Abdul Rahman, Zulkanain. (2014). A comprehensive review on adaptability of network forensics frameworks for mobile cloud computing. *The Scientific World Journal*, 2014.
- Klaedtke, Felix, Karame, Ghassan O, Bifulco, Roberto, & Cui, Heng. (2015). *Towards an access control scheme for accessing flows in SDN*. Paper presented at the Network Softwarization (NetSoft), 2015 1st IEEE Conference on.
- Kloti, Rowan, Kotronis, Vasileios, & Smith, Paul. (2013). *OpenFlow: A security analysis*. Paper presented at the Network Protocols (ICNP), 2013 21st IEEE International Conference on.
- Kotronis, Vasileios, Dimitropoulos, Xenofontas, & Ager, Bernhard. (2012). *Outsourcing the routing control logic: Better Internet routing based on SDN principles*. Paper presented at the Proceedings of the 11th ACM Workshop on Hot Topics in Networks.
- Kreutz, Diego, Ramos, Fernando MV, Esteves Verissimo, P, Esteve Rothenberg, C, Azodolmolky, Siamak, & Uhlig, Steve. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14-76.
- Kreutz, Diego, Ramos, Fernando, & Verissimo, Paulo. (2013). *Towards secure and dependable software-defined networks*. Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.
- Lakens, Daniël. (2013). Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs. *Frontiers in psychology*, 4, 863.
- Lee, Seungsoo, Yoon, Changhoon, & Shin, Seungwon. (2016). *The Smaller, the Shrewder: A Simple Malicious Application Can Kill an Entire SDN Environment*. Paper presented at the Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization.
- Li, He, Li, Peng, Guo, Song, & Nayak, Amiya. (2014). Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *Cloud Computing, IEEE Transactions on*, 2(4), 436-447.
- Li, Li Erran, Mao, Z Morley, & Rexford, Jennifer. (2012). *Toward software-defined cellular networks*. Paper presented at the Software Defined Networking (EWSN), 2012 European Workshop on.
- Mahdi, Omar Adil, Wahab, Ainuddin Wahid Abdul, Idris, Mohd Yamani Idna, Znaid, Ammar Abu, Al-Mayouf, Yusor Rafid Bahar, & Khan, Suleman. WDARS: A Weighted Data Aggregation Routing Strategy with Minimum Link Cost in Event-Driven WSNs.

- Mayoral, A, Vilalta, R, Muñoz, R, Casellas, R, & Martinez, R. (2015). *Experimental Seamless Virtual Machine Migration Using a SDN IT and Network Orchestrator*.
- McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru, Peterson, Larry, Rexford, Jennifer, . . . Turner, Jonathan. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2), 69-74.
- Mizrak, Alper Tugay, Cheng, Yu-Chung, Marzullo, Keith, & Savage, Stefan. (2006). Detecting and isolating malicious routers. *Dependable and Secure Computing, IEEE Transactions on*, 3(3), 230-244.
- Mogul, Jeffrey C, AuYoung, Alvin, Banerjee, Sujata, Popa, Lucian, Lee, Jeongkeun, Mudigonda, Jayaram, . . . Turner, Yoshio. (2013). *Corybantic: towards the modular composition of SDN control programs*. Paper presented at the Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks.
- Monaco, Matthew, Michel, Oliver, & Keller, Eric. (2013). *Applying operating system principles to SDN controller design*. Paper presented at the Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks.
- Moshref, Masoud, Bhargava, Apoorv, Gupta, Adhip, Yu, Minlan, & Govindan, Ramesh. (2014). *Flow-level state transition as a new switch primitive for SDN*. Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking.
- Moy, J, Pillay-Esnault, Padma, & Lindem, Acee. (2003). Graceful OSPF restart. *RFC3623, November*.
- Moy, John T. (1998). *OSPF: anatomy of an Internet routing protocol*: Addison-Wesley Professional.
- Nadeau, Thomas D, & Gray, Ken. (2013). *SDN: software defined networks*: " O'Reilly Media, Inc."
- Nakagawa, Shinichi, & Cuthill, Innes C. (2007). Effect size, confidence interval and statistical significance: a practical guide for biologists. *Biological Reviews*, 82(4), 591-605.
- Namal, Suneth, Ahmad, Ishtiaq, Gurtov, Andrei, & Ylianttila, Mika. (2013a). *Enabling secure mobility with openflow*. Paper presented at the Future Networks and Services (SDN4FNS), 2013 IEEE SDN for.
- Namal, Suneth, Ahmad, Ishtiaq, Gurtov, Andrei, & Ylianttila, Mika. (2013b). *Sdn based inter-technology load balancing leveraged by flow admission control*. Paper presented at the Future Networks and Services (SDN4FNS), 2013 IEEE SDN for.
- Nelson, Tim, Ferguson, Andrew D, Scheer, Michael JG, & Krishnamurthi, Shriram. (2014). *Tierless programming and reasoning for software-defined networks*. Paper presented at the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).

- Nunes, Bruno AA, Mendonca, Manoel, Nguyen, Xuan-Nam, Obraczka, Katia, & Turletti, Thierry. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3), 1617-1634.
- Nunes, Bruno Astuto A, Mendonca, Marc, Nguyen, Xuan-Nam, Obraczka, Katia, & Turletti, Thierry. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3), 1617-1634.
- Nunes, Bruno, Mendonca, Manoel, Nguyen, Xuan-Nam, Obraczka, Katia, & Turletti, Thierry. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3), 1617-1634.
- Ochoa Aday, Leonardo, Cervelló Pastor, Cristina, & Fernández Fernández, Adriana. (2015). Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks.
- Ornaghi, Alberto, & Valleri, Marco. (2003). *Man in the middle attacks*. Paper presented at the Blackhat Conference Europe.
- Padmanabhan, Venkata N, & Simon, Daniel R. (2003). Secure traceroute to detect faulty or malicious routing. *ACM SIGCOMM Computer Communication Review*, 33(1), 77-82.
- Pakzad, Farzaneh, Portmann, Marius, Tan, Wee Lum, & Indulska, Jadwiga. (2014). *Efficient topology discovery in software defined networks*. Paper presented at the Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on.
- Pakzad, Farzaneh, Portmann, Marius, Tan, Wee Lum, & Indulska, Jadwiga. (2015). Efficient topology discovery in OpenFlow-based Software Defined Networks. *Computer Communications*.
- Pei, Dan, Massey, Dan, & Zhang, Lixia. (2003). *Detection of invalid routing announcements in rip protocol*. Paper presented at the Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE.
- Phemius, Kévin, Bouet, Mathieu, & Leguay, Jérémie. (2014). *Disco: Distributed multi-domain sdn controllers*. Paper presented at the Network Operations and Management Symposium (NOMS), 2014 IEEE.
- Qazi, Zafar Ayyub, Lee, Jeongkeun, Jin, Tao, Bellala, Gowtham, Arndt, Manfred, & Noubir, Guevara. (2013). *Application-awareness in SDN*. Paper presented at the ACM SIGCOMM Computer Communication Review.
- Qi, Han, Shiraz, Muhammad, Gani, Abdullah, Whaiduzzaman, Md, & Khan, Suleman. (2014). Sierpinski triangle based data center architecture in cloud computing. *The Journal of Supercomputing*, 69(2), 887-907.

- Rohan. (2016). Cyber Security Market worth 202.36 Billion USD by 2021. Retrieved 18-09-2016, 2016, from <http://www.marketsandmarkets.com/PressReleases/cyber-security.asp>
- Rotsos, Charalampos, Sarrar, Nadi, Uhlig, Steve, Sherwood, Rob, & Moore, Andrew W. (2012). *OFLOPS: An open framework for OpenFlow switch evaluation*. Paper presented at the Passive and Active Measurement.
- Saha, Anish Kumar, Sambyo, Koj, & Bhunia, CT. (2016). *Topology Discovery, Loop Finding and Alternative Path Solution in POX Controller*. Paper presented at the Proceedings of the International MultiConference of Engineers and Computer Scientists.
- Saini, Hemraj, Rao, Yerra Shankar, & Panda, TC. (2012). Cyber-crimes and their impacts: A review. *International Journal of Engineering Research and Applications*, 2(2), 202-209.
- Scott-Hayward, S., Natarajan, S., & Sezer, S. (2016). A Survey of Security in Software Defined Networks. *IEEE Communications Surveys & Tutorials*, 18(1), 623-654. doi: 10.1109/COMST.2015.2453114
- Scott-Hayward, Sandra, O'Callaghan, Gemma, & Sezer, Sakir. (2013). *SDN security: A survey*. Paper presented at the Future Networks and Services (SDN4FNS), 2013 IEEE SDN For.
- Scott, Colin, Wundsam, Andreas, Raghavan, Barath, Panda, Aurojit, Or, Andrew, Lai, Jefferson, . . . Whitlock, Sam. (2014). *Troubleshooting blackbox SDN control software with minimal causal sequences*. Paper presented at the ACM SIGCOMM Computer Communication Review.
- SDxCentral. (2015). SDxCentral SDN and NFV Market Size Report *SDN and NFV Market Size Report 2015*. Sunnyvale, CA, USA.
- SDxCentral, Team. (2015). SDx Infrastructure Security Report *SDx Infrastructure Security Report 2015* (pp. 43). Sunnyvale, CA, USA.
- Sezer, Sakir, Scott-Hayward, Sandra, Chouhan, Pushpinder-Kaur, Fraser, Barbara, Lake, David, Finnegan, Jim, . . . Rao, Neeraj. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7), 36-43.
- Sharma, Parmanand, Banerjee, Sean, Tandel, Sébastien, Aguiar, Rui, Amorim, Ronan, & Pinheiro, David. (2013). *Enhancing network management frameworks with SDN-like control*. Paper presented at the Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on.
- Shenker, Scott, Casado, M, Koponen, Teemu, & McKeown, N. (2011). The future of networking, and the past of protocols. *Open Networking Summit*, 20.
- Sherwood, Rob, Gibb, Glen, Yap, Kok-Kiong, Appenzeller, Guido, Casado, Martin, McKeown, Nick, & Parulkar, Guru. (2009). Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 1-13.

- Shin, Seungwon, Yegneswaran, Vinod, Porras, Phillip, & Gu, Guofei. (2013). *Avant-gard: Scalable and vigilant switch flow management in software-defined networks*. Paper presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.
- Shu, Zhaogang, Wan, Jiafu, Li, Di, Lin, Jiayang, Vasilakos, Athanasios V, & Imran, Muhammad. Security in Software-Defined Networking: Threats and Countermeasures. *Mobile Networks and Applications*, 1-13.
- Staff, CACM. (2016). A purpose-built global network: Google's move to SDN. *Communications of the ACM*, 59(3), 46-54.
- Stewart III, John W. (1998). *BGP4: inter-domain routing in the Internet*: Addison-Wesley Longman Publishing Co., Inc.
- Suleman, Khan, Abdullah, Gani, Ainuddin, Wahid Abdul Wahab, Ahmed, AbdelAziz, & Mustapha, Aminu Bagiwa. (2016). *FML: A novel Forensics Management Layer for Software Defined Networks*. Paper presented at the 6th International Conference on Cloud System and Big data Engineering, Confluence-2016, 14-15 Jan, 2016, Amity University, , Noida, UP India.
- Syrivelis, Dimitris, Parisis, George, Trossen, Dirk, Flegkas, Paris, Sourlas, Vasilis, Korakis, Thanasis, & Tassioulas, Leandros. (2012). *Pursuing a software defined information-centric network*. Paper presented at the Software Defined Networking (EWSN), 2012 European Workshop on.
- Thomas, Bimba Andrew, Idris, Norisma, Al-Hnaiyyan, Ahmed, Binti Mahmud, Rohana, Abdelaziz, Ahmed, Khan, Suleman, & Chang, Victor. (2016). Towards Knowledge Modeling and Manipulation Technologies: A Survey. *International Journal of Information Management*.
- Tootoonchian, Amin, Gorbunov, Sergey, Ganjali, Yashar, Casado, Martin, & Sherwood, Rob. (2012). On Controller Performance in Software-Defined Networks. *Hot-ICE*, 12, 1-6.
- Van Benthem, Johan, & Doets, Kees. (2001). Higher-order logic *Handbook of Philosophical Logic* (pp. 189-243): Springer.
- Wallner, Ryan, & Cannistra, Robert. (2013). An SDN approach: quality of service using big switch's floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35, 14-19.
- Wang, Haopei, Xu, Lei, & Gu, Guofei. (2014). Of-guard: A dos attack prevention extension in software-defined networks. *The Open Network Summit (ONS)*.
- Wang, Haopei, Xu, Lei, & Gu, Guofei. (2015). *FloodGuard: a dos attack prevention extension in software-defined networks*. Paper presented at the Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on.
- Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2016). Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing

Environments: A Survey, Some Research Issues, and Challenges. *IEEE Communications Surveys & Tutorials*, 18(1), 602-622. doi: 10.1109/COMST.2015.2487361

You, Wanqing, Qian, Kai, He, Xi, Qian, Ying, & Tao, Lixin. (2014). *Towards security in virtualization of SDN*. Paper presented at the Proceedings of the International Conference on Computer Communications and Networks Security, ser. ICCCN.

Yu, Fei, & Leung, Victor. (2002). Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks. *Computer Networks*, 38(5), 577-589.

Zaalouk, Adel, Khondoker, Rahamatullah, Marx, Ronald, & Bayarou, Kpatcha. (2014). *OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions*. Paper presented at the Network Operations and Management Symposium (NOMS), 2014 IEEE.

Zhou, Wei, Li, Li, Luo, Min, & Chou, Wu. (2014). *REST API design patterns for SDN northbound API*. Paper presented at the Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on.

Zou, Ting, Xie, Haiyong, & Yin, Hongtao. (2013). Supporting software defined networking with application layer traffic optimization: Google Patents.