Architectural Improvements for Secure SDN Topology Discovery

Ph.D. Defence Presentation

Ajay Nehra
2014RCP9553

Under Supervision of
Dr. Meenakshi Tripathi
Associate Professor

Department of Computer Science & Engineering
Malaviya National Institute of Technology Jaipur

July 8, 2019

Agenda progress

1. Coursework and Publication Details

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Agenda progress

Course work details

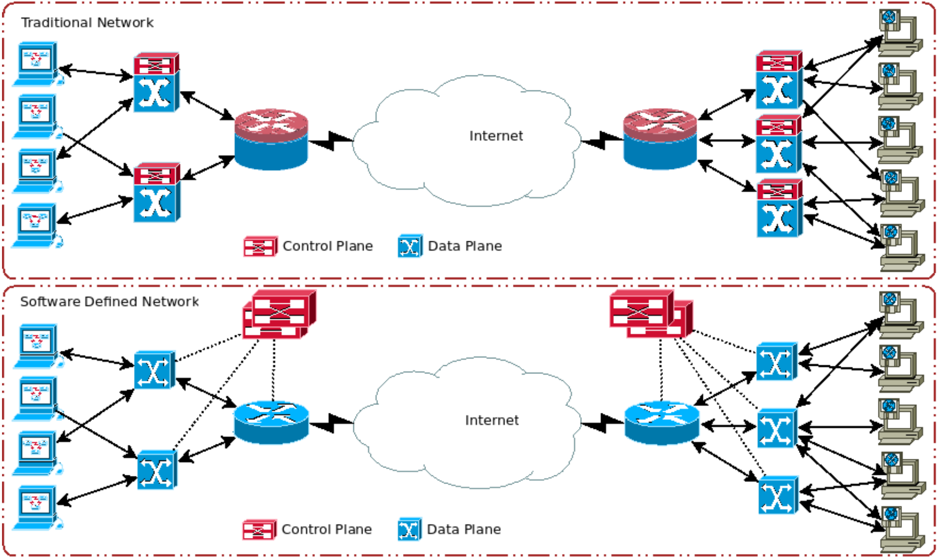
Semester	Session	Course Code	Title	Credit	Grade	Overall CGPA
I	2014-15	CPT-534	Malware Analysis & Detection	3	10	9.25
I	2014-15	HST-601	Building Language Skills	3	9	
I	2014-15	MET-900	Research Methodology & D.O.E	3	9	
I	2014-15	CPP-610	Lab Elective & Detection	3	9	

Publications

- Ajay Nehra, Meenakshi Tripathi, M.S.Gaur, Ramesh Babu Battula and Chhagan Lal, "SLDP: A Secure and Lightweight Link Discovery Protocol for Software Defined Networking", *Computer Networks, Elsevier, 2018*. Published
- Ajay Nehra, Meenakshi Tripathi, M.S.Gaur, Ramesh Babu Battula and Chhagan Lal, "TILAK: A Token based Prevention Approach for Topology Discovery Threats in SDN", *International Journal of Communication Systems, Wiley, 2018*. Published
- Prashant Kumar, Meenakshi Tripathi, Ajay Nehra, Mauro Conti and Chhagan lal, "SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN", *Transactions on Network and Service Management, IEEE, 2018*. Published
- Ajay Nehra, Meenakshi Tripathi and M.S.Gaur "FICUR: Employing SDN Programmability to Secure ARP", *IEEE 7th Annual Computing and Communication Workshop and Conference(9 - 11 January, 2017 Las Vegas, USA)*. Published
- Ajay Nehra, Meenakshi Tripathi and M.S.Gaur "Requirement Analysis for Abstracting Security in Software Defined Network", *8th International Conference on Computing Communication and Networking Technologies 2017 (8th ICCCNT 2017)(3 - 5 July, 2017 IIT Delhi, India)*. Published
- Ajay Nehra, Meenakshi Tripathi and M.S.Gaur ""Global View in SDN: Existing Implementation, Vulnerabilities & Threats", *10th International Conference On Security Of Information And Networks (SIN 2017)(13 - 15 October, 2017 Jaipur, India)*. Published

Agenda progress

Traditional Network vs SDN



Motivation

Limitation of traditional network

- Changing needs require frequent network updation(reconfiguration of switches, routers, ...)
- Network updation requires human resource (Static network)
- Multi-vendor support makes updation more cumbersome for the network administrator
- Slow adaptation to new services

Motivation

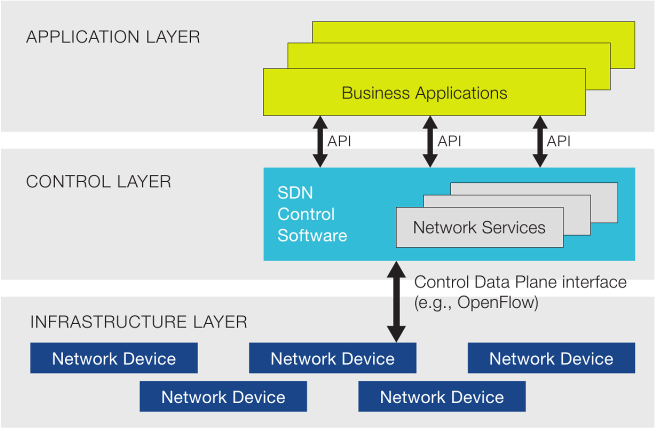
Limitation of traditional network

- Changing needs require frequent network updation(reconfiguration of switches, routers, ...)
- Network updation requires human resource (Static network)
- Multi-vendor support makes updation more cumbersome for the network administrator
- Slow adaptation to new services

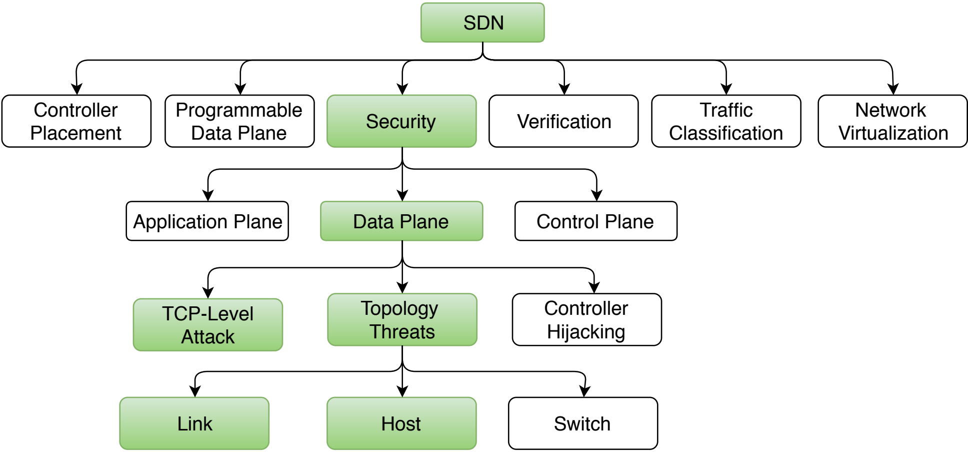
How SDN helps!

- Mastering complexity to Extracting simplicity
- Decouples Control Plane and Data Plane
- Provide Open and programmable network(Dynamic network)
- Openness encourages rapid innovation

SDN architecture[1]



Agenda progress



SDN
Controller Placement
Programmable Data Plane
Security
Verification
Traffic Classification
Network Virtualization
Application Plane
Data Plane
Control Plane
TCP-Level Attack
Topology Threats
Controller Hijacking
Link
Host
Switch

Summary of 'Flow Table Flooding' attack detection

- Every time a switch receives a packet, it checks a flow table to act accordingly
- Flow table flooding attack wastes controller, forwarding element and bandwidth resources.
- OpenFlow provides a mechanism for storage and retrieval of statistical counters
- Packet Flow Ratio(`PFRatio`) is used to detect the aforementioned attack.
- `PFRatio` is defined as the number of packet over the number of flows.
- `PFRatio` uses topology information such as the existence of switch
- To mitigate such attacks some ports information must be known

Ajay Nehra, Meenakshi Tripathi and M.S.Gaur "Requirement Analysis for Abstracting Security in Software Defined Network", *8th International Conference on Computing Communication and Networking Technologies 2017 (8th ICCCNT 2017)(3 - 5 July, 2017 IIT Delhi, India)*

Summary of 'SYN Flooding' attack detection

- An entropy-based early detection and SDN-based mitigation for TCP SYN flooding attacks is proposed.
- Entropy based on traffic features, such as destination IP, destination port and TCP flags is used.
- *SAFETY* is capable of early detection as well as mitigating the attack
- In SAFETY, some parts of topology such as the existence of switch is used
- For mitigation in SAFETY, some ports information must be known to the controller to block a specific type of traffic.
- Switch to Switch links will also play a significant role in preventive approaches.

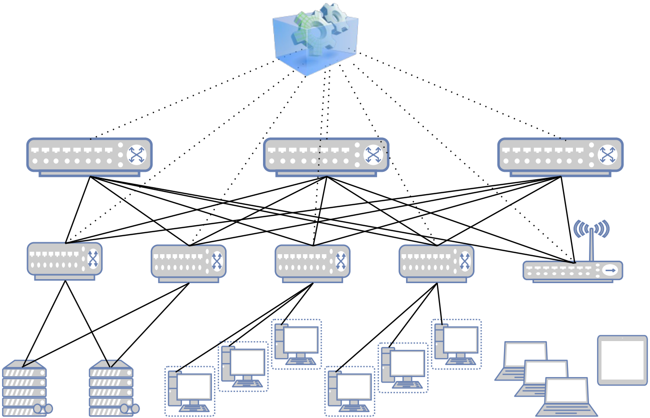
Prashant Kumar, Meenakshi Tripathi, Ajay Nehra, Mauro Conti and Chhagan lal, "SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN", *Transactions on Network and Service Management, IEEE, 2018*

Objectives

- To explore the vulnerability of topology discovery in SDN by performing state art of analysis.
- To develop a preventive solution for LLDP Poison, Flooding and Replay attacks.
- To propose a lightweight and efficient protocol variant for the link discovery process.
- To develop a solution for ARP based threats.

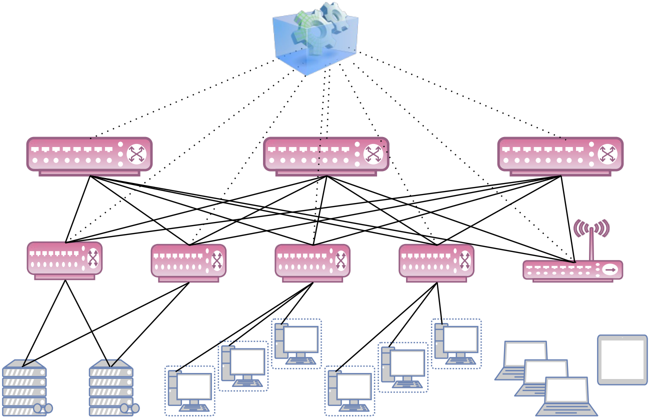
Agenda progress

Global view



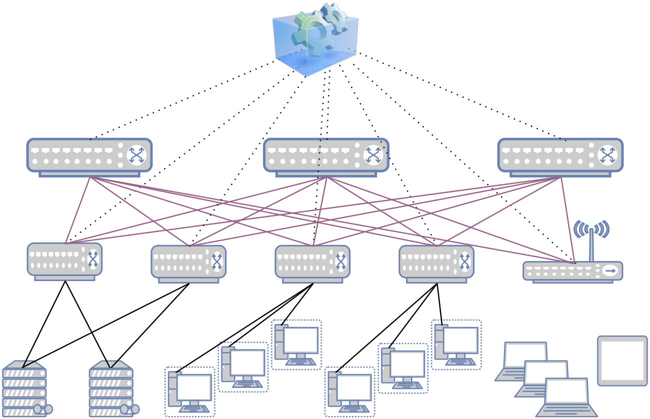
- In SDN, controller ‘can’ maintain global view
- Global view facilitate some services or applications like shortest path routing, network load balancer

Global view: Switch(s)



- On Switch startup, OFPT_Hello packets are exchanged between switch and controller
- Required for successful implementation of network policy

Global view: Link(s) between Switch(s)



- LLDP/BDDP packets, initiated by controller, pass through switches and confirm link between switches
- Required for topology-aware application or services

Link Discovery

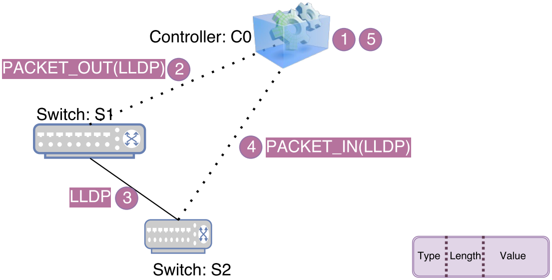


Figure: LLDP movement

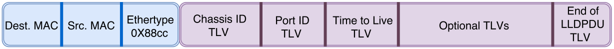


Figure: LLDP packet format

Formal Description

- Controller C $= \{Vs, \delta\}$
- View states Vs $= \{\phi, GV', GV\}$
- Transition function δ : $< LLDP >$ X $Vs \rightarrow Vs$

- Global view GV $= < S, L >$
- Partial global view GV' $= < S, L' >$
- $< LLDP > = < CTlv, PTlv, TTlv, UK1Tlv, UK2Tlv...UKnTlv, ELLDP >$

- Switch set S $= \{s_1, s_2, s_3....s_n\}$
- Link set L $= \{l_{12}, l_{21}, l_{23}, l_{32}....l_{mn}, l_{nm}\}$
- Partial link set $L' : L' \subset L$
- TLV $= < Type, Length, Value >$

Construction of global view with δ

- $\delta(< LLDP >, \phi) \rightarrow GV'$
- $\delta(< LLDP >, GV') \rightarrow GV'$
- $\delta(< LLDP >, GV') \rightarrow GV$

Attack vector for link discovery

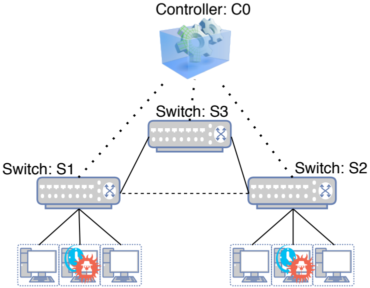


Figure: Sample topology for attack vector

- Replay Attack (RA) : A malicious host at switch $S1$ receives LLDP and shares it with malicious host attached to $S2$.
- Poison Attack (PA): An attacker at $S1$ creates an LLDP packet containing information like Chassis id $= 2$ and port $= 3$. Afterwards the switch would forward it to the controller.
- Flooding Attack (FA): A controller receives huge fake crafted link discovery packets.

LLDP Threats

LLDP Poisoning

- $\delta(< LLDP^P >, GV') \rightarrow GV^P$
- $\delta(< LLDP^P >, GV) \rightarrow GV^P$ where $GV^P \not\subset GV$
- $GV^P = < S, L^P > \bigcup < S^P, L^P >$ where $S^P \not\subset S$ and $L^P \not\subset L$

LLDP Flooding

- $< LLDP >$ moves from $DataPlane \rightarrow ControlPlane$ iff matching flow entry Fe exists
- $Fe = < M, A >$ where M $= \{m_1, m_2, ..m_n\}$ and A $= \{a_1, a_2, ..a_n\}$
- iff Fe exists then all packet with M moves from $DataPlane \rightarrow ControlPlane$

LLDP Replay

- $< LLDP >$ packet generated at the time 't' is $< LLDP_t >$
- $[\delta(< LLDP_t >, GV') \rightarrow GV^P]_{t'}$
- where $t' > t + latency$

Summary Table for Different Controllers

Controller(Ver.)	Conditional Tlvs					Information Source		Vulnerability	LLDP Attack			Attack
	TlvCnt	CTlv	PTlv	TTlv	SDTlv	Dpid	Port		LP	LF	LR	Type
POX(0.2.0)	✓	✓	✓	✓		*CTlv*	*PTlv*	No hash	✓	✓	✓	*RC*
Ryu(4.12)		✓	✓			*CTlv*	*PTlv*	No hash	✓	✓	✓	*RC*
OpenDayLight(3.0.7)					✓	*CTlv*	*PTlv*	Partial static hash	✓	✓		*LC*
FloodLight(1.2)			✓			*Uk1Tlv*	*PTlv*	Static hash	✓	✓		*LC*
Beacon(1.0.4)			✓		✓	*Uk1Tlv*	*PTlv*	No hash	✓	✓	✓	*RC*
ONOS(1.9.0)		✓				*CTlv*	*PTlv*	No hash	✓	✓	✓	*RC*
HP VAN(2.7.18)			✓			*Uk1Tlv*	*PTlv*	Static hash	✓	✓	✓	*RC*

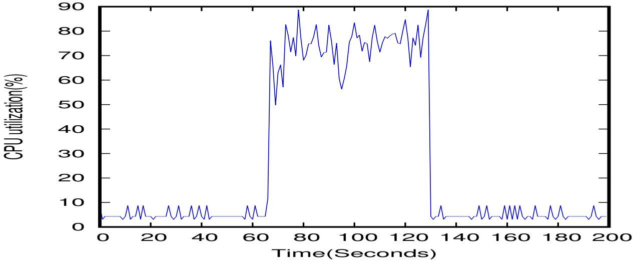


Figure: CPU utilization during LLDP flood

Ajay Nehra, Meenakshi Tripathi and M.S.Gaur "'Global View in SDN: Existing Implementation, Vulnerabilities & Threats", *10th International Conference On Security Of Information And Networks (SIN 2017)(13 - 15 October, 2017 Jaipur, India)*

Agenda progress

Assumptions

- OVS and RYU neither have bugs nor are malicious
- OVS and RYU are working as per specification
- The designed solution will handle wired network

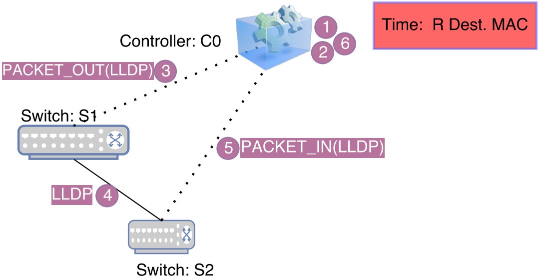


Figure: LLDP movement

R Dest. MAC	Src. MAC	Ethertype 0X88cc	Chassis ID TLV	Port ID TLV	Time to Live TLV	Optional TLVs	End of LLDPDU TLV

TILAK: Token-based authentication to prevent LLDP threats

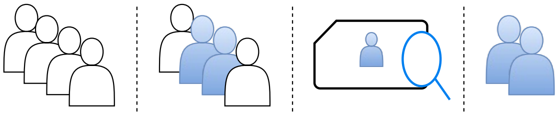


Figure: TILAK overview

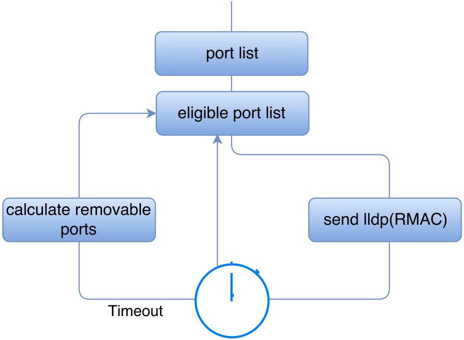


Figure: Eligible port selection

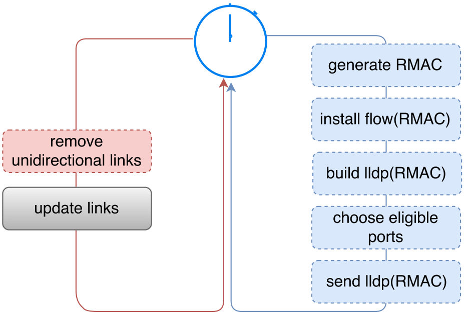


Figure: TILAK flow diagram

Topology

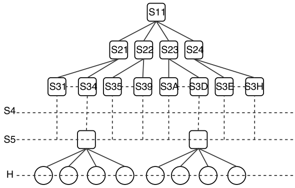


Figure: Topology 1: tree,4,4

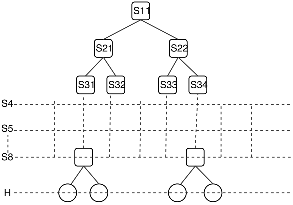


Figure: Topology 2: tree,7,2

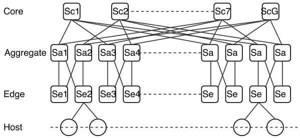


Figure: Topology 3: Fat tree

Topology	Switch	Link	Port	Host
Tree,4,4	85	340	424	256
Tree,7,2	127	254	380	128
Fat tree	80	384	705	64

Table: Number of Switches, Links, Ports and Hosts

Results

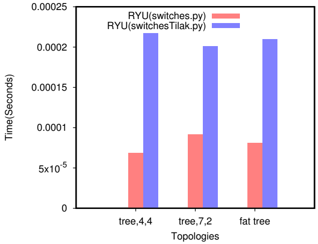


Figure: Packet construction time

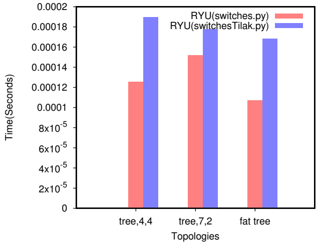


Figure: Packet verification time

Inferences from results

- Time taken by TILAK is higher than original
- It is because of frequent LLDP packet construction and verification

Results

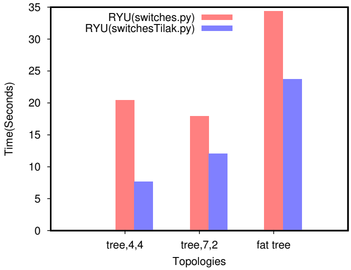
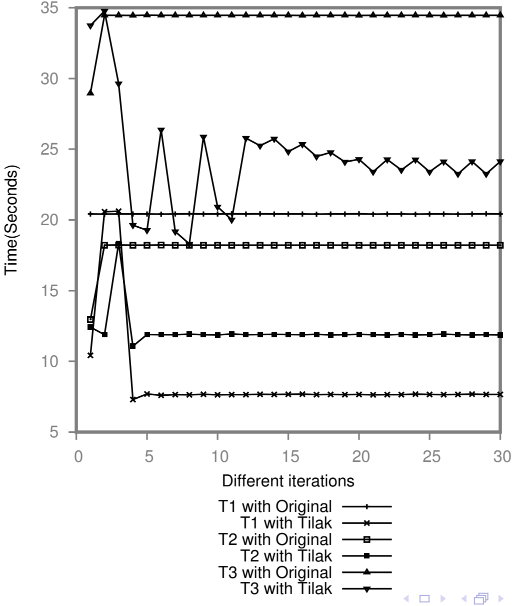


Figure: Performance comparison

Inferences from results

- Resource consumption by the original implementation is higher than TILAK irrespective of the topology
- Initially the time taken for both scenarios in each topology is same
- But after few iterations, TILAK takes less time when compared to scenario without solution

Results



Time(Seconds)
35
30
25
20
15
10
5
0
5
10
15
20
25
30
Different iterations
T1 with Original
T1 with Tilak
T2 with Original
T2 with Tilak
T3 with Original
T3 with Tilak

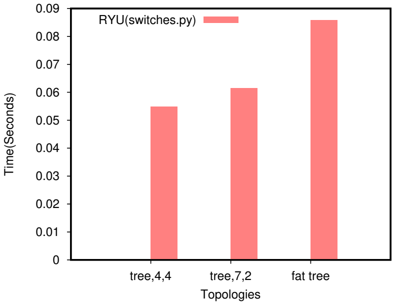


Figure: Initial overhead

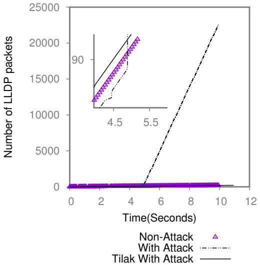


Figure: Evidence for LLDP flooding attack prevention

Inferences from results

- Initial overhead is directly proportional to the number of switches in which a flow entry will be installed and the number of ports for which LLDP packets will be created
- All forged LLDP packets are blocked on the port because of unmatched flow entry for those packets

Existing approaches

Link discovery in SDN

- *TopoGuard* [2] and *OFDP_HMAC* [3] uses HMAC authentication with dynamic key
- *SPHINX* [4] uses an abstraction of flow graphs, used to validate all given constraints

Major Shortcomings

- Few are not addressing LLDP flooding attack
- Resource penalty to produce HMAC for each LLDP packet
- Static switch-port bindings

Approach	Authentication	Integrity	LLDP broadcast	Detection	Prevention	Poison	Flood	Replay
TopoGuard[2]		y	y	y		y		
SPHINX[4]			y	y		y		
OFDP_HMAC[3]	y	y	y	y		y		y
TILAK	y				y	y	y	y

Table: Comparison in different solutions of LLDP packet based threats.

Ajay Nehra, Meenakshi Tripathi, M.S.Gaur, Ramesh Babu Battula and Chhagan Lal, "TILAK: A Token based Prevention Approach for Topology Discovery Threats in SDN", *International Journal of Communication Systems, Wiley, 2018*

Agenda progress

Motivation for security in link discovery

Controller(Ver.)	Vulnerability	LLDP Attack			Attack Type
		LP	LF	LR	
POX(0.2.0)[5]	No hash	✓	✓	✓	*RC*
RYU(4.12)[6]	No hash	✓	✓	✓	*RC*
OpenDayLight(3.0.7)[7]	Static hash	✓	✓		*LC*
FloodLight(1.2)[8]	Static hash	✓	✓		*LC*
ONOS(1.9.0)[9]	No hash	✓	✓	✓	*RC*
HP VAN(2.7.18)[10]	Static hash	✓	✓	✓	*RC*

Table: Different controllers with attack vector

Approach	Authentication	Integrity	LLDP broadcast	Poison	Flood	Replay
TopoGuard[2]		y	y	y		
SPHINX[4]			y	y		
OFDP_HMAC[3]	y	y	y	y		y
ESLD[11]		y		y		y

Table: Comparison of different research proposals for security

Motivation for lightweight packets

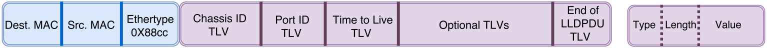


Figure: LLDP/BDDP packet format

Deployments	Size of link discovery frames(bytes)
POX(0.2.0)	41
Ryu(4.12)	40
OpenDayLight(3.0.7)	85
FloodLight(1.2)	75
Beacon(1.0.4)	47
ONOS(1.9.0)	66
HP VAN(2.7.18)	67
Target	14+8+4=26

Table: Size in Bytes for different LLDP packets

Motivation for efficient link discovery process

Topology	Switch	Link	Port	Host	eligible ports
tree,4,4	85	340	424	256	168
tree,7,2	127	254	380	128	252
fat tree	80	384	705	64	641

Table: Number of Switches,Links,Ports and Hosts

Non eligible ports

- LLDP packets are generated for each port on each switch
- LLDP packets for hosts attached ports are of no use

Desired characteristics of the proposed SLDP protocol

Secure

- SLDP will probably secure against replay, poison, and flooding attacks
- The controller will work as intended and will prevent the poisoned discovery packet processing. Hence better CPU resource utilization is achieved

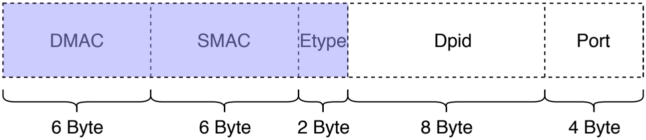
Lightweight

- SLDP packet size must be minimum in the deployed link discovery solutions
- Use less bandwidth and light traffic on network interfaces

Efficient

- SLDP will perform link discovery with less number of packets
- Requires less bandwidth and less CPU resources to generate discovery packets

SLDP frame



Frame values

- DMAC → ff:ff:ff:ff:ff:ff
- SMAC → Random MAC address
- EType → 0xabcd
- dpid → Source dpid
- port → Source port

Correctness for SLDP

Assumptions

- OVS and controller neither have bugs nor are malicious
- The designed solution will handle wired network

SLDP

- SLDP must discover the link between two OpenFlow enabled switches.
- SLDP must discover the link between two OpenFlow enabled switches separated with a non-OpenFlow switch.
- SLDP must provide latency for each discovered link.
- SLDP should secure against replay, poison and flooding attacks. **(Secure)**
- SLDP packet size must be kept to minimum. **(Lightweight)**
- SLDP must perform link discovery with less number of packets. **(Efficient)**

Formal notations

Desired characteristics

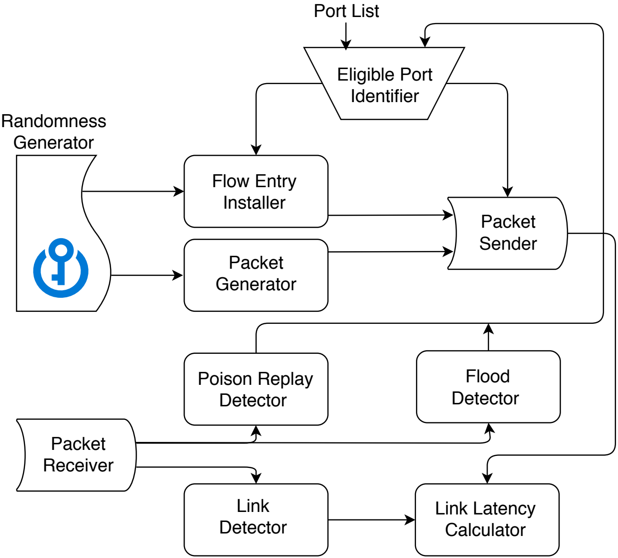
- SLDP must discover the link between two OF-switches
- All unidirectional links in a topology L $= \{l_{12}, l_{21}, l_{23}, l_{32}....l_{mn}, l_{nm}\}$
- SLDP's discovered links D $= \{d_{12}, d_{21}, d_{23}, d_{32}....d_{mn}, d_{nm}\}$.
- $\forall x[x \in L \iff x \in D]$, i.e., all elements which belong to L must also belong to D.

Desired gain

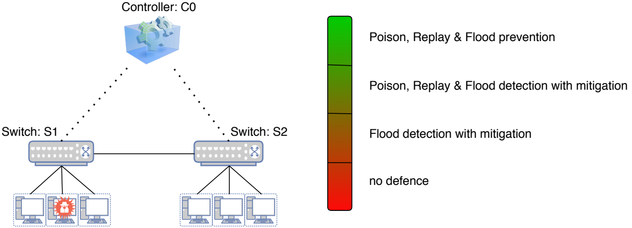
- A controller's OFDP implementation is generating 'o' bytes link discovery packets
- An SLDP implementation for the same controller is taking 's' bytes
- $p_1, p_2, p_3...p_n$ are ports of switches in a topology
- $h_1, h_2, h_3..h_n$ are the hosts attached to switches on some of the ports
- The total profit in terms of bytes in every five seconds.

$$\left\{ \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} h_i \right\} \left\{ o - s \right\} \tag{1}$$

SLDP system architecture



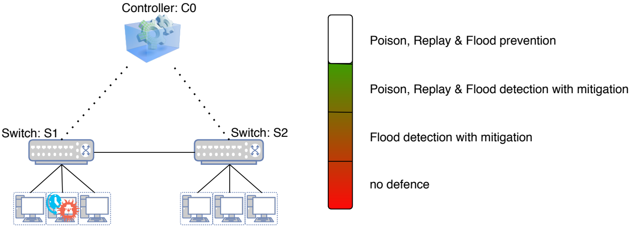
Test case # 1



Description

- An attacker or malicious application is attached to an OpenFlow enabled switch.
- SDLP prevents Poison, Replay and Flooding attacks.

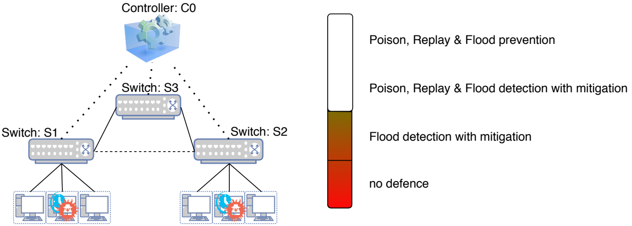
Test case # 2



Description

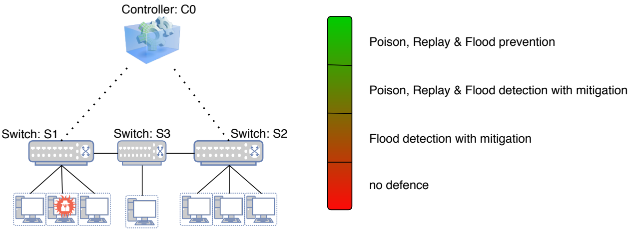
- An attacker or malicious application is attached to an OpenFlow enabled switch.
- The infected host is running before a switch starts
- SDLP detects and mitigates Poison, Replay and Flooding attacks.

Test case # 3



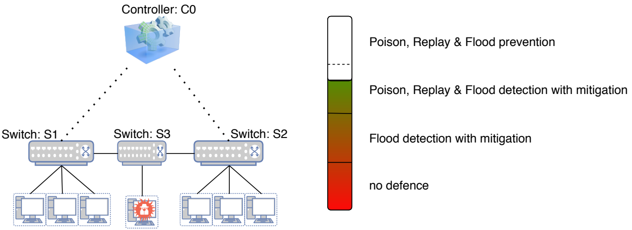
Description

- Two attacker or malicious application is attached to an OpenFlow enabled switch.
- SDLP detects and mitigates Flooding attacks.



Description

- A non OpenFlow switch separates link of two OpenFlow switches
- An attacker or malicious application is attached to an OpenFlow enabled switch.
- SDLP prevents Poison, Replay and Flooding attacks.



Description

- An attacker or malicious application is attached to an non OpenFlow enabled switch.
- SDLP detects and mitigates Poison, Replay and Flooding attacks.

Results

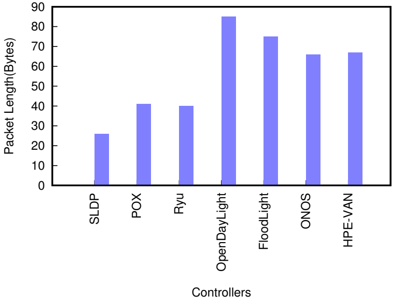


Figure: Link discovery packet length among all controllers

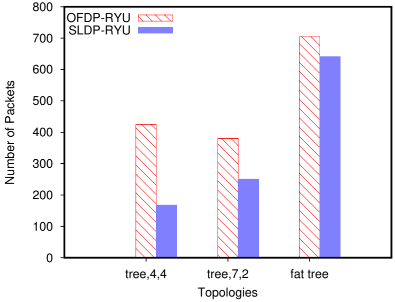


Figure: Number of packets in link discovery with three topologies

Inferences from results

- SLDP removes some unnecessary fields and restructures the packet to reduce the size
- RYU-SLDP always require lesser packets irrespective of topology

Results

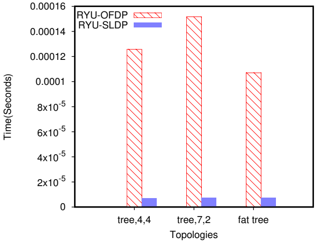


Figure: Link discovery packet verification overhead

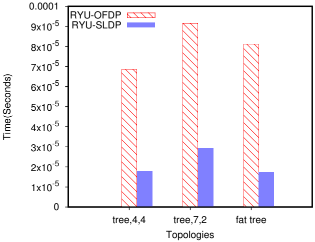


Figure: Link discovery packet construction overhead

Inferences from results

- SLDP use lighter and simple packets which are parsed on a less complex algorithm
- lighter packet takes lesser time to be constructed
- SLDP packets are generated and sent periodically

Results

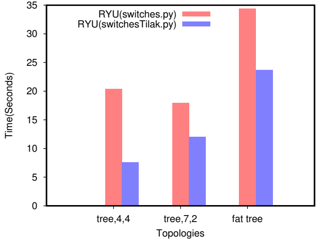


Figure: Computation overhead for link discovery

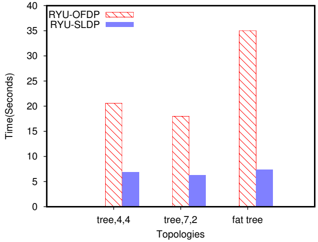


Figure: Topology discovery time

Inferences from results

- SLDP is doing so well because of lightweight and lesser number of the packets
- Initially, all ports are eligible ports but later some ports are declared as non-eligible
- Topology discovery time is the time taken by the controller to build entire topology

Ajay Nehra, Meenakshi Tripathi, M.S.Gaur, Ramesh Babu Battula and Chhagan Lal, "SLDP: A Secure and Lightweight Link Discovery Protocol for Software Defined Networking", *Computer Networks, Elsevier, 2018*. Published

Agenda progress

ARP translation process

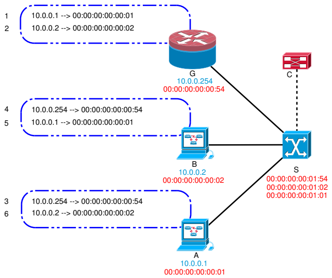


Figure: IP address to MAC address translation with ARP

ARP translation process

- A host needs to send ARP request to initiate communication process
- A genuine ARP reply consists of a valid IP-MAC binding
- Forge ARP will force all the traffic to pass trough an invalid host leading to eavesdropping or packet dropping

An ARP frame

```
---------------------------------------------------------------------
|<--Byte-->|<--Byte-->|<--Byte-->|<--Byte-->|<--Byte-->|<--Byte-->|
---------------------------------------------------------------------
|ETHERNET FRAME HEADER:   --------Destination MAC Address--------|
---------------------------------------------------------------------
|ETHERNET FRAME HEADER:     --------Source MAC Address--------   |
---------------------------------------------------------------------
|EFH: --Frame Type--  | 0806 for arp req/rep  and 0800 for ip(tcp/udp)
-----------------------
|AH:    --H/W Type--  | 0001 for ethernet
-----------------------
|AH: --Protocol Type--| 0800 for IPv4
-----------------------
|AH: -HAL-| hardware address length 06 for ethernet nic address
-----------
|AH: -PAL-| protocol address length 04 for IPv4 address
-----------------------
|AH: -Operation Code--| 0001 for ARP request and 0002 for ARP reply
---------------------------------------------------------------------
|ARP HEADER:        --------Source MAC Address--------           |
---------------------------------------------------------------------
|ARP HEADER:    --Source IP Address--      |
---------------------------------------------------------------------
|ARP HEADER:      -------Destination MAC Address--------         |
---------------------------------------------------------------------
|ARP HEADER:  --Destination IP Address--   |
---------------------------------------------
```

‘

Formal description

- System S $= \{ARP_L,\ \delta\}$
- ARP list $ARP_L = \{ARP_{E1},\ ARP_{E2}\ ...\ ARP_{En}\}$
- $ARP_{Ei} = i^{th}$ ARP entry in ARP_L
- Transition_function $\delta : ARP_L \times ARP_E \rightarrow ARP_L$

- $\delta(ARP_L, ARP_E) \rightarrow ARP_L \cup ARP_E$

ARP Poison

- $\delta(ARP_L, ARP_{E(p)}) \rightarrow ARP_L \cup ARP_{E(p)}$

ARP Flooding

- If it is true that $\forall[\delta(ARP_L,\ ARP_E) \rightarrow ARP_L \cup ARP_E]$
- Then it must be true that $\nexists[\delta(ARP_L,\ ARP_{E(p)}) \not\rightarrow ARP_L \cup ARP_{E(p)}]$

ARP Poison

Use Cases

- Man in the Middle Attack
- Denial of Service Attack
- ...

Possible Reason

- No Authentication and Integrity check of ARP packets

Few Observations

```
[pktType, pktSubType, eventPort, srcIP, srcMAC, dstIP, dstMAC]
```

```
['ARP', 1, 1, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01')]
['IP', 0, 1, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02')]
['IP', 0, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01')]
['ARP', 1, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 1, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02')]
['ARP', 1, 1, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01'), IPAddr('10.0.0.3'), EthAddr('00:00:00:00:00:00')]
```

Figure: Normal sequence for IP and ARP packet in communication

```
['ARP', 1, 3, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:03'), IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 1, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:01'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:03')]
['ARP', 1, 3, IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:03'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.0.0.1'), EthAddr('00:00:00:00:00:03')]
```

Figure: ARP packets sequence in case of Man in the Middle attack

```
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.185.191.149'), EthAddr('e0:69:95:aa:04:4a')]
['ARP', 1, 1, IPAddr('10.93.32.88'), EthAddr('e0:69:95:31:4c:7d'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.93.32.88'), EthAddr('e0:69:95:31:4c:7d')]
['ARP', 1, 1, IPAddr('10.111.49.233'), EthAddr('e0:69:95:f6:58:a9'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.111.49.233'), EthAddr('e0:69:95:f6:58:a9')]
['ARP', 1, 1, IPAddr('10.84.7.151'), EthAddr('e0:69:95:d3:e2:80'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.84.7.151'), EthAddr('e0:69:95:d3:e2:80')]
['ARP', 1, 1, IPAddr('10.86.187.2'), EthAddr('e0:69:95:4c:45:3a'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.86.187.2'), EthAddr('e0:69:95:4c:45:3a')]
['ARP', 1, 1, IPAddr('10.89.252.42'), EthAddr('e0:69:95:82:33:f3'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.89.252.42'), EthAddr('e0:69:95:82:33:f3')]
['ARP', 1, 1, IPAddr('10.22.101.179'), EthAddr('e0:69:95:38:18:c6'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.22.101.179'), EthAddr('e0:69:95:38:18:c6')]
['ARP', 1, 1, IPAddr('10.1.13.244'), EthAddr('e0:69:95:01:05:30'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.1.13.244'), EthAddr('e0:69:95:01:05:30')]
['ARP', 1, 1, IPAddr('10.24.94.92'), EthAddr('e0:69:95:a4:10:8d'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.24.94.92'), EthAddr('e0:69:95:a4:10:8d')]
['ARP', 1, 1, IPAddr('10.131.209.137'), EthAddr('e0:69:95:09:a5:c8'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
['ARP', 2, 2, IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:02'), IPAddr('10.131.209.137'), EthAddr('e0:69:95:09:a5:c8')]
['ARP', 1, 1, IPAddr('10.83.68.201'), EthAddr('e0:69:95:f8:05:b6'), IPAddr('10.0.0.2'), EthAddr('00:00:00:00:00:00')]
```

Figure: ARP packets sequence in case of ARP flood

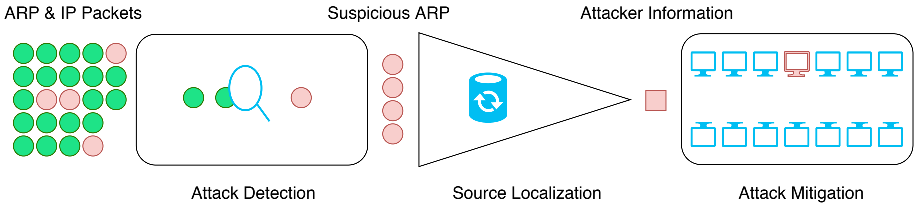


Figure: FICUR block diagram

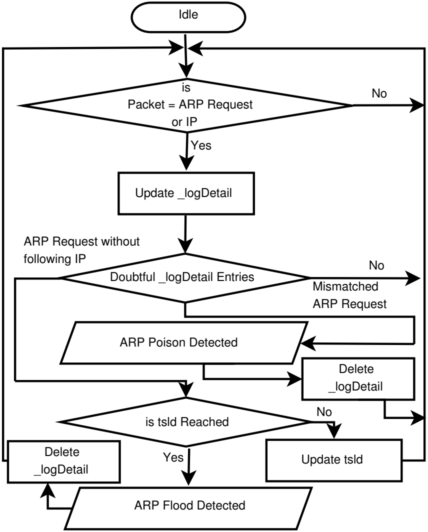
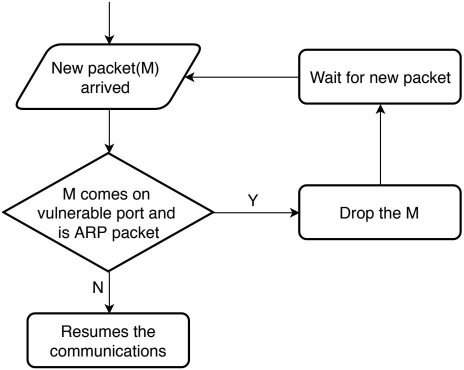


Figure: Overall flowchart for FICUR

FICUR mitigation



Attack detection observations

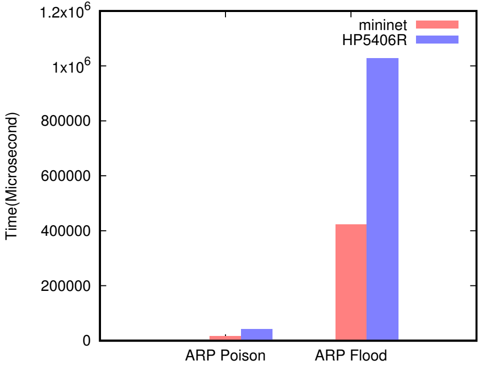


Figure: Attack detection time

Statistics

- ARP poison attacks are detected in 16056, 42840 microseconds
- ARP flood attacks are detected in 422517, 1028160 microseconds

In traditional network and SDN

- Pre-stored MAC/IP binding(DHCP,SNMP,...)[12][13][14][15][4]
- Cryptographic solution[16][17][18]
- Statistical predictions[19][20]

Major shortcomings

- Protocol dependent
- lookup process becomes very time-consuming
- resource intensive solutions

FICUR limitations

- Each ARP packet is traveling to controller
- Periodic probe to calculate ratio is resource intensive

Ajay Nehra, Meenakshi Tripathi and M.S.Gaur "FICUR: Employing SDN Programmability to Secure ARP", *IEEE 7th Annual Computing and Communication Workshop and Conference(9 - 11 January, 2017 Las Vegas, USA)*

Conclusion

- LLDP Poison, LLDP Flooding, and LLDP Replay attacks are possible on current deployments.
- Available literature for secure link discovery is inadequate.
- A novel solution called TILAK can prevent LLDP based security threats.
- The current version of link discovery is using borrowed LLDP frame.
- SDLP is a secure, efficient and lightweight link discovery protocol for SDN.
- Without securing data link layer host discovery, secure topology discovery is hard to imagine.
- FICUR utilizes the match and filter-ability of SDN to detect and mitigate the ARP-based attacks.

Future directions

- If the controller or switch is malicious, it is more challenging to prevent, detect or mitigate Poison attack
- The controller assumes that connected switch follows OpenFlow specifications. But if not, then what?
- Optimal time for SLDP period is yet to be proved practically.
- Distributed link discovery is challenging
- FICUR is not scalable for the enterprise network
- FICUR with DoS detection system can be investigated.

Agenda progress

1A. Which data set researcher has used? Which environment proposed algorithms are suitable?

Response: Mostly, data sets are used for analysis, processing, training, and classification of pre-stored data. In our research, no such data is used for training and classification. Instead of creating a data trained classification model, our approach use algorithm restricted classification model. The recent orthogonal researches agree with the same. To prove the correctness we used a theoretical approach. To generate traffic, valid LLDP, SLDP and ARP packets are created and emulated on the mininet environment. Results are validated on three different topologies with the following specification.

Topology	Switch	Link	Port	Host
Tree,4,4	85	340	424	256
Tree,7,2	127	254	380	128
Fat tree	80	384	705	64

Table: Number of Switches, Links, Ports and Hosts

1B. How the researcher compares his findings with present state-of-the-art?

Response: Our work can be divided into three parts to compare the results with present state-of-the-art.

TILAK:

Controller(Ver.)	Vulnerability	LLDP Attack			Attack Type
		LP	LF	LR	
POX(0.2.0)[5]	No hash	✓	✓	✓	*RC*
Ryu(4.12)[6]	No hash	✓	✓	✓	*RC*
OpenDayLight(3.0.7)[7]	Static hash	✓	✓		*LC*
Floodlight(1.2)[8]	Static hash	✓	✓		*LC*
ONOS(1.9.0)[9]	No hash	✓	✓	✓	*RC*
HPE-VAN(2.7.18)[10]	Static hash	✓	✓	✓	*RC*
Beacon(1.0.4)[21]	No hash	✓	✓	✓	*RC*

Table: Different controllers with attack vector

Approach	Authentication	Integrity	LLDP broadcast	Detection	Prevention	Poison	Flooding	Replay
TopoGuard[2]		y	y	y		y		
SPHINX[4]			y	y		y		
OFDP_HMAC[3]	y	y	y	y		y		y
TILAK[22]	y				y	y	y	y

Table: Comparison in different solutions of LLDP packet based threats

How the researcher compares his findings with present state-of-the-art?

- Overhead: TopoGuard[2] 4.56%, OFDP_HMAC[3] 8%, TILAK (-)40.32%.

SLDP:

Deployments	Size of link discovery frames(bytes)
POX(0.2.0)[5]	41
Ryu(4.12)[6]	40
OpenDayLight(3.0.7)[7]	85
Floodlight(1.2)[8]	75
ONOS(1.9.0)[9]	66
HPE-VAN(2.7.18)[10]	67
SLDP	14+8+4=26

Table: Length of different LLDP packets

Topology	Switch	Link	Port	Host	eligible ports
Tree,4,4	85	340	424	256	168
Tree,7,2	127	254	380	128	252
Fat tree	80	384	705	64	641

Table: Eligible ports in different topologies

FICUR: is not using pre-stored MAC/IP bindings or cryptographic hash

1C. The English is generally correct. There are still few mistakes, the document required to be checked.

Response: The thesis is edited as per the given suggestions and also incorporated some additional grammatical corrections.

2A. Pg. no. 3, "Lets start with two problems and unique solutions to examine whether topology information can make help to design an attractive solution or not?..............." Clearly mention the number with the problems and "attractive solution" for what??

Response: Exactly two problems i.e. flow entry flooding and SYN flooding are investigated and the solutions are provided for the same. The thesis is modified accordingly.
Lets start with two security problems and their unique solutions to examine whether topology information can help in designing an attractive solution? Problems are related to flow entry flooding and SYN flooding.

Reviewer 2

2B. Pg. no. 4 "a particular service irrespective of its location will generate packets with same destination IP and port address........." will it always be considered the malicious situation or it can be genuine also, specify???

Response: Following text is added to the thesis.
To perform an attack with the highest possible strength, packets with the same destination IP and destination port have to be generated by a single malicious host or a group of malicious hosts. The attacker may also perform a low-intensity attack from the network and wait for other attack probes from outside the network.

2C. Pg. no. 64, " in production network, usually Drop all others rule is applied for unknown traffic......." provide citation for this claim.

Response: It is an unintentional grammatical error so the text is modified as follows.
Thus, in the production network, usually Drop all others rule can be applied for unknown traffic.

2D. Pg. no. 92, "Packet generator takes random source MAC and creates SLDP packet." how the random source address is generated??

Response: Following text is added to the thesis.
To generate random MAC addresses, a python based cryptographically secure pseudo-random number generator i.e. os.urandom() is used.

2E. Pg. no. 94, " If a port is not receiving an SLDP packet for a long time........." what is the duration of long time?

Response: Following text is added to the thesis.
tflow is taken as 10 seconds as an observational threshold. The optimal tflow calculation is one of the future works.

2F. Pg. no. 99, "B: Poison, Replay, and Flooding detection and mitigation C: Flooding attacks detection and mitigation........." how two flooding attacks are different or why to put them in 2 categories?

Response: These two are a different level of security provided by SLDP. In case B, Poison, Replay and Flooding attacks can be detected and mitigated. In case C, Only Flooding attack detection and mitigation can be done.

2G. Pg. no. 130, "Consider the case, that the attacker also producing IP traffic with ARP traffic to fool the solution. But this will leads to degrading the attack magnitude. " not finding any flow between these two claims, rewrite them.

Response: Statements are rewritten in the thesis.
"Consider the case, that the attacker also produces IP traffic along with ARP traffic to fool the solution. To generate IP traffic the attacker has to invest resources. But this will lead to degradation of the attack magnitude."

2H. The thesis has minor English mistakes so it is suggested correct them before final printing.

Response: The thesis is edited as per the given suggestions and also incorporated some additional grammatical corrections.

Agenda progress

For Further Reading I

"Software-defined networking: The new norm for networks". In: *White Paper, Open Networking Foundation* 2 (2012), pp. 2–6. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

Sungmin Hong et al. "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures". In: *Proceedings 2015 Network and Distributed System Security Symposium* February (2015), pp. 8–11.

T. Alharbi, M. Portmann, and F. Pakzad. "The (in)security of Topology Discovery in Software Defined Networks". In: *2015 IEEE 40th Conference on Local Computer Networks (LCN)*. Oct. 2015, pp. 502–505.

Mohan Dhawan et al. "SPHINX: Detecting Security Attacks in Software-Defined Networks." In: *NDSS*. The Internet Society, 2015.

POX. URL: https://github.com/noxrepo/pox (visited on 01/01/2019).

RYU. URL: https://osrg.github.io/ryu/ (visited on 01/01/2019).

OpenDaylight. URL: https://www.opendaylight.org/ (visited on 01/01/2019).

Floodlight. URL: http://www.projectfloodlight.org (visited on 01/01/2019).

ONOS. URL: http://onosproject.org/ (visited on 01/01/2019).

HPE-VAN. URL: https://community.arubanetworks.com/t5/HPE-VAN-SDN-Controller-OVA-Free/ct-p/HPEVANSDNControllerOVAFreeTrial (visited on 01/01/2019).

Zhao Xin, Yao Lin, and Wu Guowei. "ESLD: An efficient and secure link discovery scheme for software defined networking". In: *International Journal of Communication Systems* 31.10 (), e3552.

Lawrence Berkeley. *arpwatch(8) - Linux man page*. URL: https://linux.die.net/man/8/arpwatch (visited on 01/01/2019).

M. Carnut and J. Gondim. "switched Ethernet networks: A feasibility study". In: *Proceedings of the 5th Simposio Seguranca em Informatica*. 2003.

Cisco. *Dynamic ARP Inspection*. 2013. URL: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dynarp.html (visited on 01/01/2019).

M. Z. Masoud, Y. Jaradat, and I. Jannoud. "On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm". In: *Applied Electrical Engineering and Computing Technologies (AEECT), 2015 IEEE Jordan Conference on*. Nov. 2015, pp. 1–5.

Vipul Goyal and Rohit Tripathy. "An Efficient Solution to the ARP Cache Poisoning Problem". In: *Proceedings of the 10th Australasian Conference on Information Security and Privacy*. ACISP'05. Brisbane, Australia: Springer-Verlag, 2005, pp. 40–51.

D. Bruschi E. Rosti A. Ornaghi. *S-ARP: a Secure Address Resolution Protocol*. 2003. URL: https://www.acsac.org/2003/papers/111.pdf (visited on 01/01/2019).

Wesam Lootah, William Enck, and Patrick McDaniel. "TARP: Ticket-based address resolution protocol". In: *Computer Networks* 51.15 (2007), pp. 4322–4337.

Han-Wei Hsiao, Cathy S. Lin, and Ssu-Yang Chang. "Constructing an ARP Attack Detection System with SNMP Traffic Data Mining". In: *Proceedings of the 11th International Conference on Electronic Commerce*. ICEC '09. Taipei, Taiwan: ACM, 2009, pp. 341–345.

For Further Reading IV

H. Ma et al. "Bayes-based ARP attack detection algorithm for cloud centers". In: *Tsinghua Science and Technology* 21.1 (Feb. 2016), pp. 17–28.

Beacon. 2011. URL: https://github.com/bigswitch/BeaconMirror (visited on 01/01/2019).

Ajay Nehra et al. "TILAK: A token-based prevention approach for topology discovery threats in SDN". In: *International Journal of Communication Systems* (), e3781.

T. Alharbi et al. "Securing ARP in Software Defined Networks". In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. Nov. 2016, pp. 523–526.

Cisco. *Address Resolution Protocol*. 2013. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_arp/configuration/xe-3se/3850/arp-xe-3se-3850-book/arp-config-arp.pdf (visited on 01/01/2019).

For Further Reading V

SANS. *Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks.* 2006. URL: https://www.sans.org/reading-room/whitepapers/threats/address-resolution-protocol-spoofing-man-in-the-middle-attacks-474 (visited on 01/01/2019).

S. An, B. Lee, and D. Shin. "A Survey of Intelligent Transportation Systems". In: *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks.* July 2011, pp. 332–337.

A. Gohil, H. Modi, and S. K. Patel. "5G technology of mobile communication: A survey". In: *2013 International Conference on Intelligent Systems and Signal Processing (ISSP).* Mar. 2013, pp. 288–292. DOI: 10.1109/ISSP.2013.6526920.

A. Gupta and R. K. Jha. "A Survey of 5G Network: Architecture and Emerging Technologies". In: *IEEE Access* 3 (2015), pp. 1206–1232. ISSN: 2169-3536.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Comput. Netw.* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010. URL: http://dx.doi.org/10.1016/j.comnet.2010.05.010.

- J. Lin et al. "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications". In: *IEEE Internet of Things Journal* 4.5 (Oct. 2017), pp. 1125–1142. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2683200.
- L. Ochoa-Aday, C. Cervellfffdfffd-Pastor, and A. Fernfffdfffdndez-Fernfffdfffdndez. "Self-Healing Topology Discovery Protocol for Software-Defined Networks". In: *IEEE Communications Letters* 22.5 (2018), pp. 1070–1073.
- P. Thorat et al. "Rapid recovery from link failures in software-defined networks". In: *Journal of Communications and Networks* 19.6 (2017), pp. 648–665.
- A. Azzouni et al. "sOFTDP: Secure and efficient OpenFlow topology discovery protocol". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. Apr. 2018, pp. 1–7. DOI: 10.1109/NOMS.2018.8406229.
- I. Ahmad et al. "Security in Software Defined Networks: A Survey". In: *IEEE Communications Surveys Tutorials* 17.4 (Oct. 2015), pp. 2317–2346.

For Further Reading VII

Brandon Heller, Rob Sherwood, and Nick McKeown. "The Controller Placement Problem". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 7–12. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342444. URL: http://doi.acm.org/10.1145/2342441.2342444.

Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. "Towards Secure and Dependable Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491199. URL: http://doi.acm.org/10.1145/2491185.2491199.

Y. Hu et al. "Reliability-aware controller placement for Software-Defined Networks". In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. May 2013, pp. 672–675.

S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. "On scalability of software-defined networking". In: *IEEE Communications Magazine* 51.2 (Feb. 2013), pp. 136–141. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6461198.

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.0.0.* Dec. 31, 2009. URL: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.1.0.* Feb. 28, 2011. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.2.0.* Dec. 5, 2011. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.0.* June 25, 2012. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf (visited on 01/01/2019).

For Further Reading IX

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.1*. Sept. 6, 2012. URL: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.2*. Apr. 25, 2013. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.3*. Sept. 27, 2013. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.3.pdf (visited on 01/01/2019).

Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.4*. Mar. 27, 2014. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.4.pdf (visited on 01/01/2019).

- Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.5*. Mar. 26, 2015. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf (visited on 01/01/2019).
- Open Networking Foundation. *OpenFlow Switch Specification, Version 1.4.0*. Oct. 14, 2013. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf (visited on 01/01/2019).
- Open Networking Foundation. *OpenFlow Switch Specification, Version 1.4.1*. Mar. 26, 2015. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.4.1.pdf (visited on 01/01/2019).
- *OpenFlow Switch Specification, Version 1.5.0*. Dec. 19, 2014. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf (visited on 01/01/2019).

- Open Networking Foundation. *OpenFlow Switch Specification, Version 1.5.1*. Mar. 26, 2015. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (visited on 01/01/2019).
- "Thomas Nadeau and Ken Gray". *SDN: Software Defined Networks. An Authoritative Review of Network Programmability Technologies*. O'Reilly, 2013.
- "Paul Goransson and Chuck Black". *Software Defined Networks. A Comprehensive Approach*. Morgan Kaufmann, 2014.
- Siamak Azodolmolky. *Software Defined Networking with OpenFlow*. Packt Publishing, 2013.
- William Stallings. *Foundations of Modern Networking. SDN, NFV, QoE, IoT, and Cloud*. Pearson Education, 2016.
- HP White Paper. *HP SDN hybrid network architecture*. 2015. URL: https://community.arubanetworks.com/aruba/attachments/aruba/SDN/43/1/4AA5-6738ENW.PDF (visited on 01/01/2019).

Cisco White Paper. *Software-Defined Networking: Why We Like It and How We Are Building On It*. 2013. URL: https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/cis13090_sdn_sled_white_paper.pdf (visited on 01/01/2019).

Dell White Paper. *Open Networking: Dellfffdfffdfffds Point of View on SDN*. 2015. URL: https://i.dell.com/sites/csdocuments/Business_solutions_whitepapers_Documents/en/us/dell-networking-sdn-pov.pdf (visited on 01/01/2019).

AMD White Paper. *Enabling Smart Software Defined Networks*. 2015. URL: https://www.amd.com/Documents/SDN-Whitepaper.pdf (visited on 01/01/2019).

Juniper White Paper. *Network Transformation with NFV and SDN*. 2017. URL: https://www.juniper.net/assets/fr/fr/local/pdf/whitepapers/2000628-en.pdf (visited on 01/01/2019).

Microsoft White Paper. *Software-Defined Networking*. 2015. URL: http://download.microsoft.com/download/4/a/0/4a01102f-c83a-4cf6-824b-c7e21d6a0160/software_defined_networking_white_paper.pdf (visited on 01/01/2019).

For Further Reading XIII

ONF White Paper. *Software-Defined Networking: The New Norm for Networks*. 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf (visited on 01/01/2019).

Linux Foundation White Paper. *Harmonizing Open Source and Standards in the Telecom World*. 2017. URL: https://go.pardot.com/l/6342/2017-04-27/3tjbm4/6342/173369/LF_StandardsOpenSource_Whitepaper.pdf (visited on 01/01/2019).

Intel White Paper. *Adopting Software-Defined Networking in the Enterprise*. 2014. URL: https://www.intel.com/content/dam/www/public/us/en/documents/best-practices/adopting-software-defined-networking-in-the-enterprise-paper.pdf (visited on 01/01/2019).

P. Kumar et al. "SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN". In: *IEEE Transactions on Network and Service Management* 15.4 (Dec. 2018), pp. 1545–1559. ISSN: 1932-4537.

- Ajay Nehra et al. "SLDP: A secure and lightweight link discovery protocol for software defined networking". In: *Computer Networks* 150 (2019), pp. 102–116. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.12.014. URL: http://www.sciencedirect.com/science/article/pii/S1389128618307916.
- A. Nehra, M. Tripathi, and M. S. Gaur. "FICUR: Employing SDN programmability to secure ARP". In: *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2017, pp. 1–8.
- Ajay Nehra, Meenakshi Tripathi, and M. S. Gaur. "'Global View' in SDN: Existing Implementation, Vulnerabilities & Threats". In: *Proceedings of the 10th International Conference on Security of Information and Networks*. SIN '17. Jaipur, India: ACM, 2017, pp. 303–306.
- A. Nehra, M. Tripathi, and M. S. Gaur. "Requirement analysis for abstracting security in software defined network". In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. July 2017, pp. 1–8. DOI: 10.1109/ICCCNT.2017.8204161.

For Further Reading XV

- David C. Plummer. *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. RFC 826. RFC Editor, Nov. 1982.
- T. Alharbi, M. Portmann, and F. Pakzad. "The (in)security of Topology Discovery in Software Defined Networks". In: *2015 IEEE 40th Conference on Local Computer Networks (LCN)*. 2015, pp. 502–505.
- Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly Media, 2005. Chap. 29, pp. 699–748.
- S. Y. Nam, D. Kim, and J. Kim. "Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks". In: *IEEE Communications Letters* 14.2 (Feb. 2010), pp. 187–189.
- Bob Lantz, Brandon Heller, and Nick McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 19:1–19:6.
- *Production Quality, Multilayer Open Virtual Switch, Version 2.5.0*. URL: `http://openvswitch.org/` (visited on 01/01/2019).

- *HPE 5400 zl Switch Series*. URL: http://h17007.www1.hpe.com/docs/whatsnew/spyglass/4AA2-6511ENW.pdf (visited on 01/01/2019).
- Thomas Ball et al. "VeriCon: Towards Verifying Controller Programs in Software-defined Networks". In: *SIGPLAN Not.* 49.6 (June 2014), pp. 282–293. ISSN: 0362-1340. DOI: 10.1145/2666356.2594317. URL: http://doi.acm.org/10.1145/2666356.2594317.
- Ahmed Khurshid et al. "VeriFlow: Verifying Network-wide Invariants in Real Time". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 49–54. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342452. URL: http://doi.acm.org/10.1145/2342441.2342452.
- Ehab Al-Shaer and Saeed Al-Haj. "FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures". In: *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*. SafeConfig '10. Chicago, Illinois, USA: ACM, 2010, pp. 37–44. ISBN: 978-1-4503-0093-3. DOI: 10.1145/1866898.1866905. URL: http://doi.acm.org/10.1145/1866898.1866905.

Marco Canini et al. “A NICE Way to Test Openflow Applications”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=2228298.2228312.

V. A. Zakharov, R. L. Smelyansky, and E. V. Chemeritsky. “A formal model and verification problems for software defined networks”. In: *Automatic Control and Computer Sciences* 48.7 (2015), pp. 398–406. ISSN: 1558-108X. DOI: 10.3103/S0146411614070165. URL: http://dx.doi.org/10.3103/S0146411614070165.

M. Kang et al. “Formal Modeling and Verification of SDN-OpenFlow”. In: *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. Mar. 2013, pp. 481–482. DOI: 10.1109/ICST.2013.69.

Richard Skowyra et al. “A Verification Platform for SDN-Enabled Applications”. In: *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*. IC2E '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 337–342. ISBN: 978-1-4799-3766-0. DOI: 10.1109/IC2E.2014.72. URL: http://dx.doi.org/10.1109/IC2E.2014.72.

For Further Reading XVIII

Nikhil Handigol et al. "Where is the Debugger for My Software-defined Network?" In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 55–60. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342453. URL: http://doi.acm.org/10.1145/2342441.2342453.

Nikhil Handigol et al. "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 71–85. ISBN: 978-1-931971-09-6. URL: http://dl.acm.org/citation.cfm?id=2616448.2616456.

Peyman Kazemian, George Varghese, and Nick McKeown. "Header Space Analysis: Static Checking for Networks". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 9–9. URL: http://dl.acm.org/citation.cfm?id=2228298.2228311.

Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. "Towards Secure and Dependable Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491199. URL: http://doi.acm.org/10.1145/2491185.2491199.

K. Sabnani. "An algorithmic technique for protocol verification". In: *IEEE Transactions on Communications* 36.8 (Aug. 1988), pp. 924–931. ISSN: 0090-6778. DOI: 10.1109/26.3772.

G. G. Xie et al. "On static reachability analysis of IP networks". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 3. Mar. 2005, 2170–2183 vol. 3. DOI: 10.1109/INFCOM.2005.1498492.

Urs Hengartner et al. "Detection and Analysis of Routing Loops in Packet Traces". In: *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*. IMW '02. Marseille, France: ACM, 2002, pp. 107–112. ISBN: 1-58113-603-X. DOI: 10.1145/637201.637217. URL: http://doi.acm.org/10.1145/637201.637217.

H. Yang and S. S. Lam. "Real-Time Verification of Network Properties Using Atomic Predicates". In: *IEEE/ACM Transactions on Networking* 24.2 (Apr. 2016), pp. 887–900. ISSN: 1063-6692. DOI: 10.1109/TNET.2015.2398197.

E. Al-Shaer et al. "Network configuration in a box: towards end-to-end verification of network reachability and security". In: *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*. Oct. 2009, pp. 123–132. DOI: 10.1109/ICNP.2009.5339690.

R. W. Ritchey and P. Ammann. "Using model checking to analyze network vulnerabilities". In: *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*. 2000, pp. 156–165. DOI: 10.1109/SECPRI.2000.848453.

G. J. Holzmann. "The model checker SPIN". In: *IEEE Transactions on Software Engineering* 23.5 (May 1997), pp. 279–295. ISSN: 0098-5589. DOI: 10.1109/32.588521.

Andrew Hinton et al. “Tools and Algorithms for the Construction and Analysis of Systems: 12th International Conference, TACAS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006. Proceedings”. In: ed. by Holger Hermanns and Jens Palsberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. Chap. PRISM: A Tool for Automatic Verification of Probabilistic Systems, pp. 441–444. ISBN: 978-3-540-33057-8. DOI: 10.1007/11691372_29. URL: http://dx.doi.org/10.1007/11691372_29.

Shriram Krishnamurthi. “Tierless Programming and Reasoning for Networks”. In: *Proceedings of the 10th ACM Workshop on Programming Languages and Analysis for Security*. PLAS'15. Prague, Czech Republic: ACM, 2015, pp. 42–42. ISBN: 978-1-4503-3661-1. DOI: 10.1145/2786558.2786559. URL: http://doi.acm.org/10.1145/2786558.2786559.

Jean-Claude Fernandez et al. “CADP - A Protocol Validation and Verification Toolbox”. In: *Proceedings of the 8th International Conference on Computer Aided Verification*. CAV '96. London, UK, UK: Springer-Verlag, 1996, pp. 437–440. ISBN: 3-540-61474-5. URL: http://dl.acm.org/citation.cfm?id=647765.735840.

Alessandro Cimatti et al. "NUSMV: A New Symbolic Model Verifier". In: *Proceedings of the 11th International Conference on Computer Aided Verification*. CAV '99. London, UK, UK: Springer-Verlag, 1999, pp. 495–499. ISBN: 3-540-66202-2. URL: http://dl.acm.org/citation.cfm?id=647768.733923.

Johan Bengtsson et al. "Hybrid Systems III: Verification and Control". In: ed. by Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. Chap. UPPAAL — a tool suite for automatic verification of real-time systems, pp. 232–243. ISBN: 978-3-540-68334-6. DOI: 10.1007/BFb0020949. URL: http://dx.doi.org/10.1007/BFb0020949.

Heini Wernli et al. "SALfffdfffdfffdA Novel Quality Measure for the Verification of Quantitative Precipitation Forecasts". In: *Monthly Weather Review* 136.11 (2008), pp. 4470–4487. DOI: 10.1175/2008MWR2415.1. eprint: "http://dx.doi.org/10.1175/2008MWR2415.1". URL: http://dx.doi.org/10.1175/2008MWR2415.1.

J. Qadir and O. Hasan. "Applying Formal Methods to Networking: Theory, Techniques, and Applications". In: *IEEE Communications Surveys Tutorials* 17.1 (Firstquarter 2015), pp. 256–291. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2345792.

Rob Sherwood et al. *FlowVisor: A Network Virtualization Layer*. 2009. URL: https://www.gta.ufrj.br/ensino/cpe717-2011/openflow-tr-2009-1-flowvisor.pdf (visited on 01/01/2019).

Anat Bremler-Barr et al. "Deep Packet Inspection As a Service". In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia: ACM, 2014, pp. 271–282. ISBN: 978-1-4503-3279-8. DOI: 10.1145/2674005.2674984. URL: http://doi.acm.org/10.1145/2674005.2674984.

Tim Nelson et al. "Tierless Programming and Reasoning for Software-defined Networks". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 519–531. ISBN: 978-1-931971-09-6. URL: http://dl.acm.org/citation.cfm?id=2616448.2616496.

Chen Chen et al. "Proof-based Verification of Software Defined Networks". In: *Presented as part of the Open Networking Summit 2014 (ONS 2014)*. Santa Clara, CA: USENIX, 2014. URL: https://www.usenix.org/conference/ons2014/technical-sessions/presentation/chen.

D. Sethi, S. Narayana, and S. Malik. "Abstractions for model checking SDN controllers". In: *Formal Methods in Computer-Aided Design (FMCAD), 2013.* Oct. 2013, pp. 145–148. DOI: 10.1109/FMCAD.2013.6679403.

Nate Foster et al. "Frenetic: A Network Programming Language". In: *SIGPLAN Not.* 46.9 (Sept. 2011), pp. 279–291. ISSN: 0362-1340. DOI: 10.1145/2034574.2034812. URL: http://doi.acm.org/10.1145/2034574.2034812.

Christopher Monsanto et al. "Composing Software-defined Networks". In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. nsdi'13. Lombard, IL: USENIX Association, 2013, pp. 1–14. URL: http://dl.acm.org/citation.cfm?id=2482626.2482629.

Andreas Voellmy and Paul Hudak. "Nettle: Taking the Sting out of Programming Network Routers". In: *Proceedings of the 13th International Conference on Practical Aspects of Declarative Languages*. PADL'11. Austin, TX, USA: Springer-Verlag, 2011, pp. 235–249. ISBN: 978-3-642-18377-5. URL: http://dl.acm.org/citation.cfm?id=1946313.1946339.

For Further Reading XXV

- Timothy L. Hinrichs et al. "Practical Declarative Network Management". In: *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*. WREN '09. Barcelona, Spain: ACM, 2009, pp. 1–10. ISBN: 978-1-60558-443-0. DOI: 10.1145/1592681.1592683. URL: http://doi.acm.org/10.1145/1592681.1592683.
- Hyojoon Kim et al. "Kinetic: Verifiable Dynamic Network Control". In: *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. NSDI'15. Oakland, CA: USENIX Association, 2015, pp. 59–72.
- Christopher Monsanto et al. "A Compiler and Run-time System for Network Programming Languages". In: *SIGPLAN Not.* 47.1 (Jan. 2012), pp. 217–230.
- Katta Naga Praveen, Jennifer Rexford, and David Walker. *Logic Programming for Software-Defined Networks*. URL: https://www.cs.princeton.edu/~dpw/papers/xldi-2012.pdf (visited on 01/01/2019).

Open Networking Foundation. *Principles and Practices for Securing Software-Defined Networks*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles_and_Practices_for_Securing_Software-Defined_Networks_applied_to_OFv1.3.4_V1.0.pdf (visited on 01/01/2019).

S. Scott-Hayward, G. O'Callaghan, and S. Sezer. "Sdn Security: A Survey". In: *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. Nov. 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

Kevin Benton, L. Jean Camp, and Chris Small. "OpenFlow Vulnerability Assessment". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. 2013, pp. 151–152.

HP5406R Switch: Product documentation. URL: https://www.hpe.com/h20195/v2/default.aspx?cc=emea_africa&lc=en&oid=7430783 (visited on 01/01/2019).

Open vSwitch. URL: http://openvswitch.org/ (visited on 01/01/2019).

D. Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

B. A. A. Nunes et al. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks". In: *IEEE Communications Surveys Tutorials* 16.3 (2014), pp. 1617–1634.

S. Scott-Hayward, G. O'Callaghan, and S. Sezer. "Sdn Security: A Survey". In: *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. Nov. 2013, pp. 1–7.

Nivindar Kaur and Shashank Srivastava. "Performance Impact of Topology Poisoning Attack in SDN and its Countermeasure". In: (2016), pp. 1–6.

Hewlett-Packard Development Company. "HPN SDN Controller Link Discovery". In: (2014). URL: http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=c04495141.

Leonardo Ochoa Aday, Cristina Cervelló Pastor, and Adriana Fernández Fernández. "Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks". In: *International Journal of Distributed Sensor Networks* 5.2 (2015), pp. 1–6.

- Tri Hai Nguyen and Myungsik Yoo. "Analysis of link discovery service attacks in SDN controller". In: *International Conference on Information Networking* (2017), pp. 259–261.
- George Tarnaras, Evangelos Haleplidis, and Spyros Denazis. "SDN and ForCES based optimal network topology discovery". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)* (2015), pp. 1–6.
- A. Doria et al. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. RFC 5810. RFC Editor, Mar. 2010. URL: http://www.rfc-editor.org/rfc/rfc5810.txt.
- S. Khan et al. "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art". In: *IEEE Communications Surveys Tutorials* 19.1 (2017), pp. 303–324.
- Bob Lantz, Brandon Heller, and Nick McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. New York, NY, USA: ACM, 2010, 19:1–19:6.

Link Layer Discovery Protocol and MIB. URL: http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf (visited on 01/01/2019).

F. Pakzad et al. “Efficient topology discovery in software defined networks”. In: *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*. Dec. 2014, pp. 1–8.

L. Ochoa-Aday, C. Cervello-Pastor, and A. Fernandez-Fernandez. “Self-Healing Topology Discovery Protocol for Software-Defined Networks”. In: *IEEE Communications Letters* 22.5 (May 2018), pp. 1070–1073.

Leonardo OCHOA-ADAY, Cristina CERVELLO-PASTOR, and Adriana FERNANDEZ-FERNANDEZ. “Discovering the Network Topology: An Efficient Approach for SDN”. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 5.2 (2016).

Y. Jimenez, C. Cervello-Pastor, and A. Garcia. “Dynamic Resource Discovery Protocol for Software Defined Networks”. In: *IEEE Communications Letters* 19.5 (May 2015), pp. 743–746.

E. Rojas et al. "TEDP: An Enhanced Topology Discovery Service for Software-Defined Networking". In: *IEEE Communications Letters* 22.8 (Aug. 2018), pp. 1540–1543. ISSN: 1089-7798.