Securing IPv6 Neighbor Discovery and SLAAC in Access Networks through SDN

Daniel Nelle
University of Potsdam,
Potsdam, Germany
dnelle@uni-potsdam.de

Thomas Scheffler
Hochschule für Technik und Wirtschaft Berlin
Berlin, Germany
thomas.scheffler@htw-berlin.de

ABSTRACT

This paper proposes and evaluates a new approach, based on Software Defined Networking (SDN), to secure the IPv6 Neighbor Discovery Protocol (NDP) message exchange and make the Stateless Address Autoconfiguration safer. We created an SDN application on the Ryu SDN framework which functions as an intelligent NDP-Proxy. The SDN application inspects all NDP messages in the data path of the access switch. Once the application has accumulated data about the respective network segment, it performs sanity checking and filtering. We used several relevant attacks from the THC IPv6 toolkit to assert resiliency against attacks on the Neighbor Discovery Protocol. Load tests showed that the overhead for the NDP packet inspection is not neglectable, but once the relevant flow-rules have been installed, subsequent packets are forwarded on the fast-path of the switch and network performance is only minimally affected.

1 INTRODUCTION

One of the differentiating features of IPv6 is the address resolution process, which changed significantly between IPv4 and IPv6. IPv6 hosts support Stateless Address Autoconfiguration (SLAAC) [19] and use it to configure their IP stack. MAC addresses are resolved using the Neighbor Discovery Protocol (NDP) [14] instead of the Address Resolution Protocol (ARP) [17] used by IPv4.

NDP has been diagnosed as a security risk for switched Ethernet networks [2, 3], because routers and hosts implicitly trust all other nodes on the local network. A number of easily executable attacks, such as redirection, man-in-the-middle, resource exhaustion, and starvation have been identified and are easily exploitable. Several approaches have been examined to make the protocols safer. Deployments of *IPsec* [12] or *SEcure Neighbor Discovery (SEND)* [4] suffer from complexity, bootstrap problems, resource hunger and a lack of deployment on relevant platforms [1, 3, 9, 15]. Other methods call for the implementation of monitoring and filtering capabilities on all susceptible network links [6, 13] and require support from the Link-Layer (L2) switching hardware.

1.1 Approach

Threats against the Neighbor Discovery Protocol arise because attackers on the local network can monitor protocol traffic. The protocol requires no authentication and protocol messages can be spoofed or modified using tools such as Scapy[1] or the THC-IPV6-ATTACK-TOOLKIT[2].

This paper explores the possibilities of mitigating threats against NDP in switched Ethernet networks by creating a stateful binding between IPv6-Addresses, MAC-Addresses and physical switch ports on the local network switch. This binding is temporary and behaves similarly to a L2 neighbor cache entry [14]. The binding is used to selectively forward NDP-messages and detect malicious behaviour originating from specific ports (such as MAC-address spoofing and others). The approach does not require any changes to NDP or host implementations and should be transparent for network clients.

SDN offers the ability to implement new functions in the data-path of switched networks not limited by the Link-Layer processing capabilities of the switching hardware. Network functions that have traditionally been implemented by router or switch manufacturers can now be developed and adapted independently and at a much faster pace than previously possible, providing true Network Function Virtualisation (NFV) [8].

We are aiming to secure the neighbor discovery process by deploying an access switch, which intercepts

ANRW '19, July 22, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6848-3/19/07…$15.00
https://doi.org/10.1145/3340301.3341132

[1] https://scapy.net
[2] https://github.com/vanhauser-thc/thc-ipv6

Table 1: Flow table

src_ip	dst_ip	...	priority	counters	instructions	timeouts	cookie	flags
*	1.1.1.1	...	1	0	drop	20	1	...
2.2.2.2	*	...	2	0	dec_ttl	20	2	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
*	*	...	0	6	controller	0	0	...

all ICMPv6 NDP messages and presents them to the SDN-controller for inspection. The controller calls a NDP-proxy application that either generates necessary protocol messages itself or verifies received messages prior to forwarding. The goal is to assure *First Hop security* for IPv6 clients by having the NDP-proxy act as the source and sink of all NDP messages on the local network and thus suppress potential attack vectors. Furthermore, the controller is the only instance allowed to emit IPv6 *Router Advertisements*, blocking all such messages from other sources. In order to distinguish legitimate hosts from potential attackers, the controller maintains its own cache of connected hosts.

1.2 OpenFlow

OpenFlow is the communication protocol between the control and forwarding plane of an SDN architecture. It enables a controller application to alter the configuration of network hardware, based on decisions reached by analyzing incoming traffic and other criteria. OpenFlow enabled switches maintain a *flow table* as well as an API allowing the insertion and deletion of *flow rules*. Incoming packets are *matched* to one or several *flow rules* by comparing packet data with *match fields*. Those fields can contain actual data or be wildcarded. Subsequently, *actions* stored in the *instructions* field of the matching rule with the highest *priority* are executed, such as forwarding the packet to a certain port or to the controller for further inspection. Flow rules furthermore contain a *cookie*, which is a numerical field used for identification purposes and a *timeout* value, allowing for automatic removal of a rule after a certain amount of time has elapsed. Each rule can be associated with a *meter*, which can limit traffic forwarded by that rule. Table 1 shows a schematic representation of such a table, for a more detailed insight, please refer to the specification [16].

1.3 RYU

Ryu[3] is an open-source, component-based SDN framework written in Python. It enables developers to write controller applications, called Ryu apps. Those can be used to process packets forwarded by an SDN-enabled switch and trigger actions, such as altering the behavior of the switch or sending alerts to a monitoring system. Communication with the switch is facilitated via one of several protocols, with OpenFlow being the most widely supported one [11]. While Ryu is not the fastest available SDN framework [21], it has full OpenFlow support and a very active developer community.

[3]https://osrg.github.io/ryu/

1.4 IPv6 Address Configuration

IPv6 hosts support the stateless autoconfiguration of addresses (SLAAC) specified in [19]. A host willing to join a particular network generates itself a link-local address for the given interface by combining the link-local prefix `fe80::/10` with an identifier, usually the MAC address of the interface.

The resulting address is considered *tentative* until *Duplicate Address Detection* (DAD) has been performed. DAD verifies the uniqueness of the address in a network segment and is conducted by sending out an ICMPv6 *Neighbor Solicitation* (NS) message with the Network-Layer source set to the unspecified address `::` and the target field set to the tentative address. Should a *Neighbor Advertisement* (NA) for the tentative address be received, another host on the network segment signals that this address is already in use and the joining host can not use this address. If no such NA is received, the address is considered to be unique on the segment and can be used, permitting local communication on the given network segment.

The host also needs a global IPv6 address to be fully reachable. Global address prefix information is usually distributed via a *Router Advertisement* (RA) message. RAs are emitted by routers periodically and can be requested via *Router Solicitation* (RS) messages for faster configuration.

If a host needs the L2 address of another host on the same segment, it sends a NS to the destination's *Solicited Node Multicast* address, with the target field set to the link-local IP of the destination host. If a host with the target IP address is present, it responds with a NA to the origin of the request, providing it with its

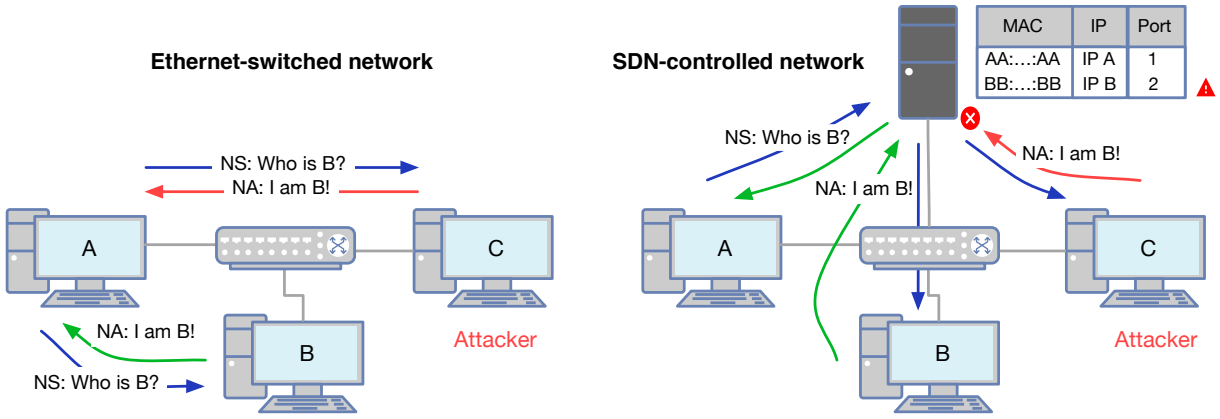

Figure 1: Attacking a hosts Neighbor Cache via IPv6 Neighbor Spoofing.

L2 address. Address information is stored in a host's neighbor cache [19].

All these procedures are vulnerable to attacks, because any host on the network can craft and send the necessary protocol messages and hosts accept the contained information without being able to assess the authenticity of the messages.

Attacks can be mounted by sending spoofed NS messages to a target host which leads to service disruption and/or man-in-the-middle attacks. The left network in Figure 1 depicts a situation, where an attacker (Host C) may be able to spoof the identity of a victim (Host B). Tools such as `parasite6` from the THC-Toolkit allow for the easy execution of such attacks with only minimal knowledge about the process. Another threat are bogus *Router Advertisements*, that can disrupt service or allow an attacker to compromise the network [5].

2 NDP PROXY

This section provides a brief introduction to the concept and architecture of our Ryu-app.

The access-switch in any network is in a unique position to monitor and control network access by connected hosts. Traditional Ethernet switches, with their hardware-based data plane, focused on fast L2 processing, can fulfill this task only imperfectly. Software-defined networking allows for a flexible split between data traffic processed by the switch hardware and control traffic presented to the controller for further inspection, not limited by the hardware and software capabilities of the switch.

Our main component, the NDP proxy app, connects to the switch and sets up *flow rules* to intercept all ICMPv6 / IPv6 messages. It then processes the packets matched by said rules. Our main interest lies in the NS/NA messages that allow the app to learn the state of the network and stop malicious activities before they reach other hosts.

Incoming packets are processed sequentially by Ryu and categorized according to their *icmp_type* [7]. We usually want to switch ordinary IPv6 traffic directly via the switch. However, before a dedicated flow-rule has been established (or after it may have been expired) other IPv6 traffic may reach the controller. We make use of these packets to track whether a host is still active in the network.

Non-IPv6 packets are used to learn their origin port and Media Access Control (MAC) address. Afterwards they are forwarded to the switch and a dedicated flow-rule may be installed. This is the standard behavior of a MAC-learning switch and permits regular IPv4 operation on the network. ICMPv6 / IPv6 traffic originating from a host already learned is still caught, as the corresponding flow rules for IPv6 use a higher priority than the MAC-learning rules.

Certain messages, such as *redirect* [14] are simply ignored in our setup, since they are significant only in multi-router scenarios [7]. A configuration file allows to set certain parameters, such as the prefix and DNS server advertised within the *Router Advertisements*, timeout values and others.

2.1 Router Advertisement Guard

Our app is the only instance allowed to emit *Router Advertisement* (RA) messages into this network segment. RAs emitted from other sources are dropped and not forwarded through the subnet. *Router Solicitation* messages are answered by the app itself, providing the requesting host with information preset in the configuration file. This behavior constitutes a *Router Advertisement*

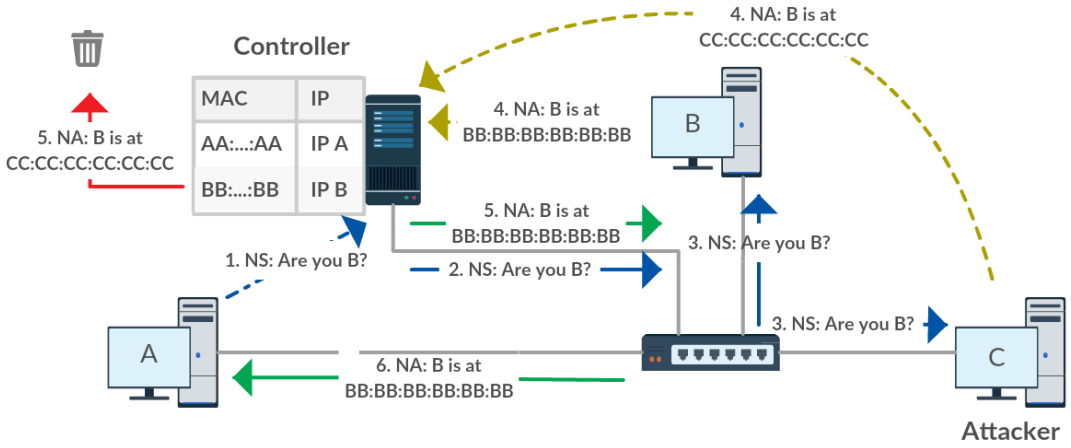

Figure 2: The app verifies NS and NA messages by checking them against the cache.

Guard [13], which further secures network operation in local IPv6 subnets.

2.2 Generating a Neighbor Cache

Securing SLAAC and NDP is challenging because it is difficult to distinguish legitimate NDP messages from spoofed ones, aimed at compromising the communication between hosts. However, if we can establish a trustworthy mapping between a hosts MAC-address, IP-addresses and physical port in the access network switch, we will be able to assess the legitimacy of those messages without limiting the auto-configuration process.

As explained in Section 1.4, each host is required to verify the uniqueness of configured addresses using DAD, once it becomes active on a network segment.[4] We decided to use the DAD messages to build a cache of trustworthy neighbors within the NDP proxy app, where we bind an IP address to a specific L2 address until the cache entry is deleted. Subsequent changes to the L2 address associated with an IP address will be blocked[5] and reported (see Figure 1, right network).

DAD messages are not forwarded through the subnet. This increases security, because an attacker does no longer see other hosts entering the network. However, it shifts the responsibility for detecting address collisions from the hosts themselves to the app. When a DAD message is received, the app checks the cache for an existing entry with the address in question. In case of a collision, a NA is sent to the requesting host, notifying it that the address is already taken. Otherwise a new cache entry is added. This approach is as reliable as the original behavior, even if two hosts perform the check at the same time. Since Ryu processes packets sequentially, one request will always be the first to get permission to use the address.

2.3 Using the Neighbor Cache to control secure operation

Figure 2 shows the normal operation for a network where DAD has already been performed for all hosts. *Neighbor Solicitation* messages are send by hosts to resolve unknown L2 addresses for network targets. NS messages are only forwarded in the network, if a cache entry for the given IP address exists (Steps 1, 2, 3). The app could theoretically answer this request itself (with information from the Neighbor Cache), but we decided to forward this message in order to verify the reachability of the host (Step 4).

If a corresponding *Neighbor Advertisement* is received, the contents of its fields are checked against the cache entry. Should Network- and Link-Layer addresses match, the NA is deemed safe to be forwarded and communication between the two hosts is permitted via a temporary *flow rule* (Step 5 green, 6). If the data does not match, the packet is discarded (Step 5 red). This prevents the insertion of a false L2 address into the neighbor cache of hosts, but also provides protection against flooding the Neighbor Cache of the app with random NAs. Other IPv6 traffic is used in a similar way - the first packet is forwarded to the controller and used to record activity by the sending host. If IP and L2 address match, a new temporary flow rule is inserted and the packet

[4]The standard states this as a 'should' and allows hosts to disable DAD. However, this is not recommended and in our case would exclude the host from communicating on the network.

[5]This may break certain failover-scenarios for servers where multiple NICs share one IP-address, which is not within the use-case of our app.

is forwarded to the destination, otherwise the traffic is dropped.

Cache entries time out when no network activity is detected from a host. This preserves the transparent switching property of Ethernet networks.

3 IMPLEMENTATION

This section contains an overview of the app as well as a non-exhaustive description of implementation details which may be of particular interest.

3.1 Overview

The app consists of several modules, which all inherit from `RyuApp`. Separating functionality of the app results from the need to let the respective parts run concurrently.

`ndp_proxy.py` is the heart of the app and manages the connection with the switch, handling of incoming traffic, and processing events sent by OpenFlow.

`cache_manager.py` iterates over the neighbor cache, checking and potentially deleting entries. `ra_sender.py` emits *Router Advertisement* messages at configurable intervals and `flood_checker.py` monitors switch ports for flooding. `ndp_proxy_controller.py` exposes a REST API that gives access to statistics and several runtime configuration settings.

The app is started by running `ndp_proxy_runner.py`.

3.2 Packet matching

On startup, the app inserts flow rules with a higher priority than regular rules into the flow table of the switch, with the aim to catch all ICMPv6 traffic. In each rule, the cookie field (see Section 1.2) is set to the `icmp_type` of the respective message type it is supposed to match. The switch sends the cookie information of the matching rule with each Packet-In message to the controller. The cookie can thus be used to conveniently categorize the messages, without having to decode the packet first.

Another rule with a medium priority is added to catch all remaining IPv6 traffic. It acts as a fall back to forward IPv6 packets to the controller, should no other flow rule explicitly allow communication between two hosts. If the `meter_flag` in `config.py` is set, an OpenFlow meter [16] is associated with each rule on switches that support this feature. This provides a limited form of DoS protection, as it shields the controller against attacks, where a host floods it with more ICMPv6 traffic than it can handle.

3.3 Additional features

The app provides some additional features that were deemed useful during development, testing and operational troubleshooting.

A counter for cache misses as well as statistics regarding received ICMPv6 messages can be retrieved via a REST call. The app generates warnings for address collisions, cache misses and NDP messages with content not matching the neighbor cache. This provides valuable insight into the operation of the network segment and allows to easily identify misconfigured or malicious hosts.

Traffic going through the app can be logged to a `pcap`-file. The logging can be toggled via REST and provides two modes: One for all ICMPv6 traffic going through the controller and one for packets generated by the app itself.

The app also keeps track of the rate at which packets are forwarded from the switch to the controller. Together with the support for OpenFlow meters this provides a basic anti-flood mechanism. If the Packet-In rate for a certain switch port exceeds a configurable threshold, a warning is generated. This message contains the exact period during which abnormal traffic conditions were observed. Future versions of the app could also provide an option to (temporarily) disable the switch-port to stop flooding-attacks at the source.

4 EVALUATION

A number of tests have been conducted to assess the functionality of the app as well as its performance.

4.1 Functional Tests

Basic IPv6 functionality has been verified in a virtual set-up using Mininet[6] running Open vSwitch 2.10.1 as well as in a lab setting using a mix of dedicated Windows and Linux hosts using an Edgecore AS4610-30T switch running PICOS 2.10.2.

A second test used the THC toolkit to simulate attacks on a virtual network topology created with Mininet. The `parasite6` tool spoofs NA and NS messages in order to establish a man-in-the middle position for the attacking host. Spoofing attempts by this program could be successfully prevented by the proxy app. Unlike in tests performed without the controller, the integrity of neighbor caches on all hosts is preserved. `flood_router6` and `flood_advertise6` were used to simulate flooding with ICMPv6 messages. The metering feature proved to be successful in mitigating such attacks, provided it kept the rate of arriving packets below the threshold of what Ryu itself can handle on the given hardware. We also

[6] http://mininet.org

Table 2: Average first ping times for the NDP proxy measured over 25 runs

Network Size	2	5	10	25	50	100
Latency in ms	12.39	12.98	16.32	21.63	34.62	58.21

tested `fake_router6`, a program announcing itself as the router with highest priority on the network. The attack was rendered useless since its *Router Advertisements* are caught and dropped.

4.2 Performance

We also evaluated the impact of the proxy app on network performance. We chose to compare ping times between `ndp_proxy` and the MAC-learning switch application `simple_switch13` included in Ryu. Only the switching delay for the first ICMPv6 packet per host is of interest. It is the only packet that is forwarded via the controller and triggers the installation of a flow-rule. Subsequent packets are directly forwarded via the switch to the destination host, until the corresponding flow rule times out.

We created networks of different size. For each topology size, the setup has been started 25 times on a laptop computer equipped with a Intel i7-5500U CPU and 8 GB of RAM running Ubuntu 17.10. After starting the app and creating the network in Mininet, we waited 5 seconds for the address autoconfiguration to finish. Afterwards the command `ping6 -c1 fe80::200:ff:fe00:2 -I h1-eth0` was issued and the output parsed.
Table 2 lists the average timings for the `ndp_proxy` app. For `simple_switch13`, the time for the first ping has constantly been between 5 and 7 milliseconds, regardless of network size.

The results indicate that the impact on first ping performance is a function of the network size. Our implementation uses Python dictionaries to store data for identified network hosts. In IPv6 multiple IP addresses are associated with one MAC (link-local, global, private) and in turn need to be associated with the same switch port and flow-entry. This forces the implementation to iterate over all entries in the dictionary in order to retrieve/ update information because Python dictionaries do not support multiple keys. `simple_switch13` uses a Python dictionary to record a MAC to port table that can be accessed via a key value.

The focus of this work has been the on the functionality rather than performance. In future versions, the current data structure holding the cache could be replaced by a proper in-memory database and the performance reevaluated.

5 RELATED WORK

Cisco offers similar functionality, as presented in this paper, in its wireless LAN controllers [6]. However their solution can not be used to secure switched Ethernet LANs and is not based on open network standards.

Faucet[7] is an open source SDN controller application that is developed as a drop-in replacement for switched Ethernet enterprise networks. While Faucet offers some basic security features, it lacks the specific IPv6 NDP proxy functionality provided by our solution.

6 CONCLUSION

We could show that our original goal, improving NDP security through the implementation of an intelligent SDN-based proxy app, can be achieved with relative ease. Our prototypical solution is able to harden the security of a switched Ethernet network segment by sucessfully mitigating common attack scenarios against NDP and SLAAC.

Our approach does not require any changes to the Neighbor Discovery Protocol or host implementations. If each SDN switch is paired with a dedicated controller, the fully distributed nature of the IPv6 address resolution and neighbor discovery process is retained. Currently, we drop all Multicast Listener Discovery (MLD) [20] messages. Filtering MLD messages in semi-public access networks can actually enhance the security of the network [18]. So far we we did not test any attacks using specific evasion techniques such as Extension Header Chaining [10].

An SDN-application can provide additional features, that are either lacking from regular Ethernet switches or may be otherwise costly to implement. Our app is capable of granting comprehensive insight into the state of the network and generates intelligent notifications concerning network events. The current solution could easily be extended to provide additional functionality, such as automatically blocking a port from being flooded for some configurable interval.

During development and testing we found a few problems when dealing with OpenFlow. Hardware and software switch implementations do not always support all features specified in the standard. Further tests are necessary in order to assess whether the measured performance and functionality are adequate for a production network.

Our implementation is available on a public Git repository hosted by the University of Potsdam[8].

[7] https://faucet.nz
[8] https://gitup.uni-potsdam.de/dnelle/ryu_ipv6_ndp_proxy

REFERENCES

[1] Ahmad Alsa'deh and Christoph Meinel. 2012. Secure Neighbor Discovery: Review, Challenges, Perspectives, and Recommendations. *IEEE Security and Privacy* 10, 4 (July 2012), 26–34. https://doi.org/10.1109/MSP.2012.27

[2] Mohammed Anbar, Rosni Abdullah, Redhwan M. A. Saad, Esraa Alomari, and Samer Alsaleem. 2016. Review of Security Vulnerabilities in the IPv6 Neighbor Discovery Protocol. In *Information Science and Applications (ICISA) 2016*, Kuinam J. Kim and Nikolai Joukov (Eds.). Springer Singapore, Singapore, 603–612.

[3] Jari Arkko, Tuomas Aura, James Kempf, Vesa-Matti Mäntylä, Pekka Nikander, and Michael Roe. 2002. Securing IPv6 neighbor and router discovery. In *WiSE '02: Proceedings of the 1st ACM workshop on Wireless security*. ACM, New York, NY, USA, 77–86. https://doi.org/10.1145/570681.570690

[4] J. Arkko, J. Kempf, B. Zill, and P. Nikander. 2005. *SEcure Neighbor Discovery (SEND)*. RFC 3971. Internet Engineering Task Force. http://tools.ietf.org/html/rfc3971

[5] Tim Chown and Stig Venaas. 2011. *Rogue IPv6 Router Advertisement Problem Statement*. RFC 6104. Internet Engineering Task Force. http://tools.ietf.org/html/rfc6104

[6] Cisco. n.d.. *Cisco Wireless LAN Controller Deployment Guide*. Technical Report. Cisco Systems. https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-0/IPV6_DG.pdf

[7] A. Conta, S. Deering, and M. Gupta. 2006. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443. Internet Engineering Task Force. http://tools.ietf.org/html/rfc4443

[8] ETSI. 2012. *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action*. Technical Report. European Telecommunications Standards Institute. http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[9] Niels Ferguson and Bruce Schneier. 2000. *A Cryptographic Evaluation of IPsec*. Technical Report. Counterpane Internet Security, Inc. https://www.schneier.com/academic/paperfiles/paper-ipsec.pdf

[10] Fernando Gont. 2014. *Implementation Advice for IPv6 Router Advertisement Guard (RA-Guard)*. RFC 7113. Internet Engineering Task Force. https://tools.ietf.org/html/rfc7113

[11] F. Hu, Q. Hao, and K. Bao. 2014. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys Tutorials* 16, 4 (Fourthquarter 2014), 2181–2206. https://doi.org/10.1109/COMST.2014.2326417

[12] S. Kent and K. Seo. 2005. *Security Architecture for the Internet Protocol*. RFC 4301. Internet Engineering Task Force. http://tools.ietf.org/html/rfc4301

[13] Eric Levy-Abegnoli, Gunter Van de Velde, Ciprian Popoviciu, and Janos Mohacsi. 2011. *IPv6 Router Advertisement Guard*. RFC 6105. Internet Engineering Task Force. http://tools.ietf.org/html/rfc6105

[14] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. 2007. *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861. Internet Engineering Task Force. http://tools.ietf.org/html/rfc4861

[15] Pekka Nikander. 2001. Denial-of-Service, Address Ownership, and Early Authentication in the IPv6 World. In *In Proc. 9th International Workshop on Security Protocols, volume 2467 of LNCS*. Springer, 25–27.

[16] OpenFlow 2014. *OpenFlow Switch Specification Version 1.3.4*. Technical Report. Open Networking Foundation. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.4.pdf

[17] David C. Plummer. 1982. *An Ethernet Address Resolution Protocol*. RFC 826. Internet Engineering Task Force. http://tools.ietf.org/html/rfc826

[18] Enno Rey, Antonios Atlasis, and Jayson Salazar. 2016. MLD Considered Harmful. https://ripe72.ripe.net/presentations/74-ERNW_RIPE72_MLD_Considered_Harmful_v1_light_web.pdf

[19] S. Thomson, T. Narten, and T. Jinmei. 2007. *IPv6 Stateless Address Autoconfiguration*. RFC 4862. Internet Engineering Task Force. http://tools.ietf.org/html/rfc4862

[20] R. Vida and L. Costa. 2004. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810. Internet Engineering Task Force. https://tools.ietf.org/html/rfc3810

[21] Y. Zhao, L. Iannone, and M. Riguidel. 2015. On the performance of SDN controllers: A reality check. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 79–85. https://doi.org/10.1109/NFV-SDN.2015.7387410