Hindawi
Security and Communication Networks
Volume 2017, Article ID 5838657, 9 pages
https://doi.org/10.1155/2017/5838657

WILEY | Hindawi

Research Article

An SDN-Based Authentication Mechanism for Securing Neighbor Discovery Protocol in IPv6

Yiqin Lu, Meng Wang, and Pengsen Huang

South China University of Technology, Guangzhou, China

Correspondence should be addressed to Meng Wang; w.m15@mail.scut.edu.cn

Received 18 September 2016; Revised 27 November 2016; Accepted 13 December 2016; Published 24 January 2017

Academic Editor: Jesús Díaz-Verdejo

The Neighbor Discovery Protocol (NDP) is one of the main protocols in the Internet Protocol version 6 (IPv6) suite, and it provides many basic functions for the normal operation of IPv6 in a local area network (LAN), such as address autoconfiguration and address resolution. However, it has many vulnerabilities that can be used by malicious nodes to launch attacks, because the NDP messages are easily spoofed without protection. Surrounding this problem, many solutions have been proposed for securing NDP, but these solutions either proposed new protocols that need to be supported by all nodes or built mechanisms that require the cooperation of all nodes, which is inevitable in the traditional distributed networks. Nevertheless, Software-Defined Networking (SDN) provides a new perspective to think about protecting NDP. In this paper, we proposed an SDN-based authentication mechanism to verify the identity of NDP packets transmitted in a LAN. Using the centralized control and programmability of SDN, it can effectively prevent the spoofing attacks and other derived attacks based on spoofing. In addition, this mechanism needs no additional protocol supporting or configuration at hosts and routers and does not introduce any dedicated devices.

1. Introduction

IPv6 is a protocol designed as the successor to IPv4 protocol [1]. It is used to solve the problems faced by IPv4 in today's Internet, such as IP address space limitation, security, and scalability. Compared with the 32-bit length of the IP address in IPv4, the IPv6 address comprises 128 bits. This is absolutely enough in the foreseeable future as it supports an address space of $O(2^{32})$. The NDP is an auxiliary protocol for IPv6, and it comprises two RFCs (Requests for Comments): Neighbor Discovery for IPv6 [2] and IPv6 stateless address autoconfiguration (SLAAC) [3]. The former is used for discovery of the IPv6 nodes on the same link, and the latter allows the hosts to automatically configure the IPv6 address without the outside help like DHCP (Dynamic Host Configuration Protocol) server. As the IPv6 address is long and its address space is huge, SLAAC is a very convenient function and makes the IPv6 network become plug-and-play. For the normal operation of IPv6, NDP also provides other functions including router/prefix/parameter discovery, address resolution, next-hop determination, neighbor unreachability detection (NUD), duplicate address detection (DAD), and redirection. All of these functions are based on the transmission of NDP messages, which are encapsulated in ICMPv6 (Internet Control Message Protocol version 6) packets. Meanwhile, the NDP messages are confined to a link and only transmitted in the scope of a LAN. This means any router will not forward NDP messages from one network to another. According to [2], NDP uses five kinds of ICMPv6 messages as follows.

Router Solicitation (RS). Hosts send RS messages to find the default router and request for the network information from routers.

Router Advertisement (RA). RA message is sent by routers periodically or responses to the RS message.

Neighbor Solicitation (NS). Nodes send NS message to resolve a neighbor node's IPv6 address to its MAC (Media Access Control) address or to detect the reachability of a neighbor.

Neighbor Advertisement (NA). A node sends NA message to answer solicited NS message or sends unsolicited NA

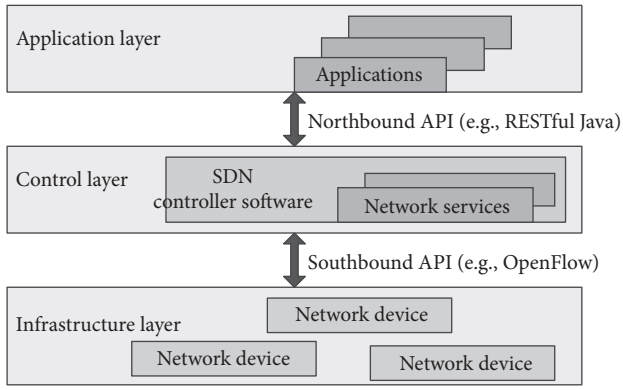

Figure 1: SDN architecture.

message to propagate its changed information, such as the MAC address variation.

Redirect Message (RM). Routers send redirect packets to inform a host of a better first-hop node on the path to a destination.

Here, we introduce two procedures of the functions to show how the NDP messages are used. The first is address resolution. When a node wants to communicate with another node using IPv6 address without knowing the corresponding MAC address, it will firstly send a multicast NS message to ask all nodes in the LAN who has this IPv6 address. Then, the node occupying this address will send back a unicast NA message to advise its MAC address. The second is DAD procedure. When a node autoconfigures itself with an IPv6 address, it will firstly verify the uniqueness of this address. It orderly sends several NS messages with setting the destination as solicited-node multicast address. Then, if it receives any NA message in response to this solicitation, this address is already used. Otherwise, this address could be issued on the network by this node. From these two examples, we could find that they are vulnerable to be attacked through spoofing. A fake reply to address resolution may lead to MITM (man-in-the-middle) attacks, and forged NAs to DAD will result in DoS (denial-of-service) attacks. Therefore, an effective authentication mechanism is very important for securing the NDP.

SDN is a different network architecture compared with the traditional distributed network. In SDN architecture, the control plane and data plane are decoupled [4], and SDN is designed to have a logical centralized controller and distributed forwarding devices. The whole architecture can be divided into three layers: application layer, control layer, and infrastructure layer, as depicted in Figure 1. The application layer is composed of various applications that are programmed by developers, and the control layer abstracts the underlying infrastructure to provide programmability to the upper layer through northbound API (Application Program Interface). The infrastructure layer contains the physical and virtual network devices, which are conducted by the controller through southbound API. This way, SDN presents many benefits (e.g., the programmability of network, the rise of virtualization, device configuration, and troubleshooting) to solve the problems and challenges faced in legacy distributed networks [5]. According to the survey of [6], the ability to programmatically control network behavior and view global network state in real time gives SDN exciting possibilities for the network security. For example, the applications in SDN can have a "god view" ability profited from the global view of the controller, but, in traditional networks, the distributed control components only have local information, and it is very difficult to build a global view because of the distribution and autonomy of many network devices. Therefore, we are inspired to think about securing NDP in SDN environment.

Match fields	Priority	Counters	Instructions	Timeouts	Cookie

Figure 2: A flow entry.

OpenFlow (OF) is one of the many southbound API specifications. It is based on an Ethernet switch, with an internal flow table and a standardized interface to add and remove flow entries [7]. The OF specification defines how the packets are processed in OF-enabled switches, and Figure 2 gives an example of a flow entry in a flow table [8]. When the switch receives a packet, it matches the packet header with the flow entries in a flow table. If the packet matches a flow entry, it will be processed according to the actions specified in the entry. Otherwise, it will be sent to the controller as a packet-in message, and the controller will handle it according to the strategies implemented by control components and applications. Then, the controller will send a packet-out message to command the switch to continue processing this packet. According to [8], a packet can trigger a packet-in message through the "send to controller" action, and this packet can be included in the data portion of this packet-in. After the controller handled this packet-in message, a packet-out message will be sent to the switch, and then the switch processes this packet according to the action field of this packet-out. The packet included in packet-in is also included in the data portion of this packet-out. We use this feature to capture the NDP packets transmitted in a LAN and extract the header of these packets from the packet-in messages to authenticate and then send the packets back to switch through packet-out messages.

The rest of this paper is organized as follows. In Section 2, we analyse the security issues in NDP and the existing solutions. Section 3 describes the details of our proposed mechanism. Then, we give a validation experiment and discussion in Section 4. At last, we draw a conclusion and talk about the future works in Section 5.

2. Security Issues and Solutions in NDP

In IPv6 network, NDP is an essential component in a LAN. However, there are many security issues that can be used by attackers to impact the legitimate communication of users. This section analyses the common security issues and talks about the existing solutions.

2.1. Security Issues in NDP. Although the NDP defined many rules for the nodes to send or receive NDP messages legitimately, there is no compulsive method to guarantee the node

behaves normally. Therefore, malicious nodes can launch attacks through illegally using NDP messages. Referring to the work of [2, 9–12], we conclude that there are mainly six basic kinds of attacks in NDP as follows.

Spoofing. This attack is executed by using a forged address and may cause a false entry in a node's neighbor cache. Meanwhile, spoofing is often used to leverage other attacks, such as MITM attacks, DoS attacks, and redirect attacks.

MITM Attack. This attack hijacks the communication between two nodes. When node A sends a NS message to resolve the MAC address of node B, the attacker can pretend to answer this NS with spoofed NA. Then, the attacker will receive the subsequent packets from node A and forward them to node B also using spoofed packets. This way, although nodes A and B seem to be normally communicating with each other, the attacker has taken over all the traffic flows between them without being perceived.

DoS Attack. This attack aims to prevent the nodes from normally running the functions provided by NDP. For example, when a node executes the DAD, an attacker can snoop the NS messages sent by this procedure and send back forged NA saying "I have occupied this address." Hence, the victim cannot finish the DAD to get an IPv6 address for the following communication. Similarly, an attacker can send forged RA and NA messages to create DoS attacks on router/prefix/parameter discovery and NUD procedure.

Redirect Attack. An attacker can fabricate RMs to redirect the packets away from the correct path and take over the packets transmitted to a router. Then, the attacker can act as MITM or hinder the normal communication to a remote node.

Replay Attack. The attackers can capture the multicast packets and then resend these packets on the link to confuse the hosts or routers with false information. All neighbor/router discovery messages are prone to replay attacks.

Rogue Router. In this attack, a malicious node can pretend to be a router and send fake RA messages. If a node selects it as default router, it can siphon off the traffic of this node or act as MITM. An attacker can also inject rogue information to poison the routing tables in a good router to prevent victims from accessing the desired network.

Some other sophisticated attacks are the combination of the above attacks. From the above content, we conclude that all attacks rely on the spoofing or abusing of the NDP messages. If there is a perfect authentication mechanism to verify the NDP messages, this protocol can be protected fundamentally and have strong resistibility to various attacks. Many works of securing NDP are making efforts toward this direction, and several related works will be talked about next.

2.2. Existing Solutions of Securing NDP. The SEND (SEcure Neighbor Discovery) protocol [13] is developed by the IETF (Internet Engineering Task Force) to specify security mechanisms for NDP. Actually, NDP intended to use IPsec (IP security) to protect itself through IP layer authentication, but IPsec is not suited for the autoconfiguration in SLAAC, as there is a bootstrapping problem. Therefore, SEND proposed three mechanisms to protect NDP messages. The first is router authorization. SEND uses authorization delegation discovery (ADD) procedure to validate and authorize the IPv6 routers. This is based on a trusted third party, called trust anchor, to issue the certifications. Only after the router is authorized can it act as a router, and every node must certify the router via the trust anchor before setting the router as a default router. The second is Cryptographically Generated Addresses (CGA). A node cryptographically generates IPv6 address by using a one-way hash function from the node's public key and some other parameters. CGA is used to make sure that the sender of NDP packets is the "owner" of the claimed address. The third mechanism aims to protect the integrity of the messages and authenticate the identity of their sender. These three mechanisms consume vast computation resources for computing the cryptographic algorithm. Moreover, the new options add more than one Kbyte to each NDP packet. SEND also introduced some new vulnerabilities and attacks, such as CGA verification vulnerability and DoS attacks on router authorization. In a word, SEND has many limitations including computation, deployment, and security.

The paper [14] proposed an attack detection mechanism for the spoofing of NS and NA messages. The mechanism aims to ensure the genuineness of the IP-MAC pairing through an active verification procedure. It defines six data tables to maintain the information obtained from an IDS (Intrusion Detection System) and uses two main modules named NS-Handler and NA-Handler to handle the NS and NA messages, respectively. For example, when a host sends a NA message to another host, the NA will be checked by the authenticated bindings table, which records the IP-MAC bindings that have been verified by IDS, and if the IP-MAC pair of this message exists in the table, it is judged as genuine. Otherwise, the IDS will send a NS probe packet to verify the source MAC address claimed by this message, and then it will judge the address as genuine or spoofed according to the response. Although this mechanism can prevent the spoofing of NS and NA messages, it still has some drawbacks. On the one hand, the probe analyser module can only detect the existence of spoofing when it receives more than one response but cannot find out which one is genuine. On the other hand, a smart attacker may poison the authenticated binding table through sophisticated spoofing, which will result in false negatives and positives. In addition, it introduces a new IDS device which is connected to a mirror port to monitor all traffic on the network. This is costly and needs the device to have high performance.

The paper [15] proposed a rule-based mechanism to detect the DoS attacks in the DAD process. This method introduced a trusted controller scheme machine to execute the detection rules to verify the generated IPv6 address. It firstly screens out NS/NA packets through ip6tables rules and then uses the control scheme to do detection. For example, when a new host sends NS messages to perform DAD, it will wait for the controller machine to verify the uniqueness of the address, and the host only uses the address after receiving the verification reply from the controller machine. The paper [16]

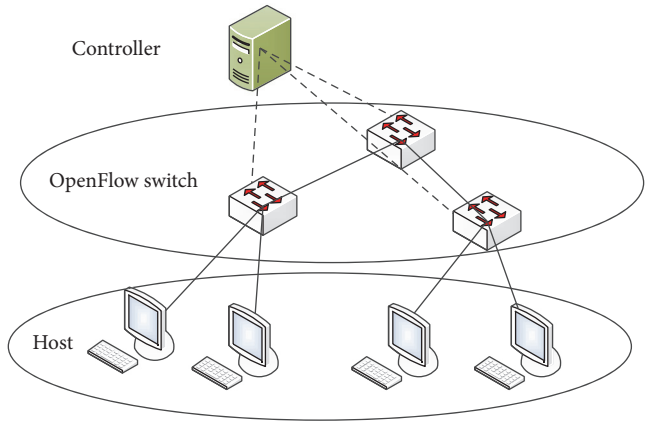

Figure 3: Deployment scenario.

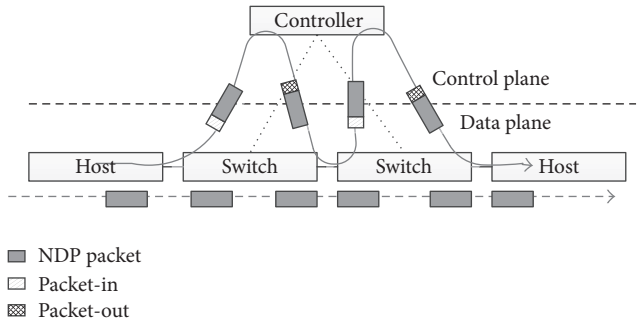

Figure 4: Transmission of NDP packet.

proposed a pull model to improve the reliability and security of DAD by changing the solicitation model. This proposal reduces the overhead of DAD and enhances the flexibility in address generation, but it is vulnerable to brute force and inverting attacks.

From the above related works, we could see that it is most important to build an effective authentication mechanism to verify the identity of NDP packets for defending the attacks on NDP. SEND and [16] build this mechanism through cryptography and hash algorithm. References [14, 15] introduce a trust component to monitor NDP packets and execute authentication. All of these methods either proposed new protocols that need to be supported by all nodes or built defense mechanisms that require the cooperation of all nodes. This is inevitable in the traditional networks because of the distribution and autonomy of network devices. Moreover, these kinds of methods are usually difficult and costly to deploy in a network. Nevertheless, the SDN provides a new perspective to think about how to build the authentication mechanism. In SDN environment, we can realize the authentication mechanism as a program based on the APIs provided by the controller. It has a global view of the network. Meanwhile, as the network is centralized and controlled, it does not need any new protocol support on distributed devices. Furthermore, the interactive communication of authentication is at the control plane of SDN, which means the mechanism is transparent to all nodes and does not need any additional configuration on the hosts and routers, totally satisfying the fundamental goal of zero configuration indicated in RFC 3756 [17].

3. The Proposed Mechanism

This section describes the details of the proposed mechanism. We start with an overview of the workflow and then talk about the module and algorithm.

3.1. Workflow Overview. Our proposed mechanism is deployed in an OpenFlow-based SDN network, as depicted in Figure 3. When a switch receives a NDP packet, it will be included in a packet-in message and sent to the controller. Meanwhile, the mechanism listens to packet-in messages and extracts the NDP packet from desired packet-in. Then, it verifies the identity of this packet according to the global information of the network. After this, the NDP is encapsulated in a packet-out message and resent to the switch. At last, the switch will process this packet according to the action specified in packet-out. If the packet successfully passed the verification, the action is to output the packet to the suitable port based on the global topology; otherwise, the action is to drop this packet. The transmission of the NDP packets is depicted in Figure 4; we could see that the mechanism is transparent to the nodes at data plane.

3.2. Module and Algorithm. The whole mechanism contains five modules as depicted in Figure 5. The function of every

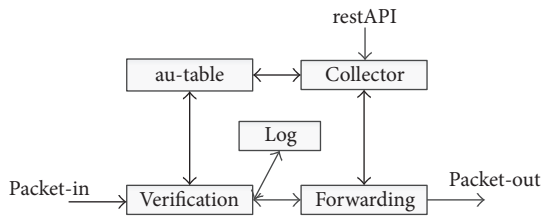

Figure 5: Modules of the mechanism.


```
{
  "entityClass":"DefaultEntityClass",
  "mac":[
     "00:00:00:00:00:03"
 ],
  "ipv4": []
  "ipv6": [
     "fe80::200:ff:fe00:3"
 ],
  "vlan":[
     "0x0"
 ],
  "attachmentPoint": [
     {
        "switchDPID": "00:00:00:00:00:00:00:01"
        "port": 3,
        "errorStatus":null
     }
 ],
  "lastSeen": 1472470896206
},
```


Figure 6: JSON data of device information.

module and the details of the algorithm are presented as follows.

Collector. This module is responsible for invoking the restAPIs provided by the controller to get the global information of the network, including device information, global topology, and routing path. Then, it sends the returned JSON (JavaScript Object Notation) data to the module that has sent the invoking command. The device information lists all devices tracked by the controller and their details including MAC, IP, and attachment point. Figure 6 gives an example of the JSON data of this information.

au-Table. This module builds and maintains an authentication table through processing the JSON data of device information returned from the collector. The items of this table comprise MAC address, IPv6 address, and attachment point, as depicted in Table 1. The attachment point item has two subitems named DPID (datapath ID) and port: the former identifies a switch and the latter indicates the accessing port of a device. This item points out the location of a node in the network topology. This table is indexed by the MAC address item, so that it is not allowed to have repetitive entries in this item. Strictly speaking, the duplication of IPv6 address item is also not allowed, but this is the IP conflict problem, which is out of the scope of this paper. In addition, this table is updated periodically every t_0 seconds. This parameter determines the sensitivity to the topology changing, such as a mobile node moving from an attachment point to another one, and this parameter is configured by the administrator according to the experience.

Table 1: Authentication table.

MAC	IPv6	Attachment point	
		DPID	Port

Verification. This module is responsible for verifying the NDP packets and is the core component of this mechanism. At the initial stage, this module sends a flow entry to all switches to capture the ICMPv6 packets. According to [8], this flow entry is configured with the OXM_OF_ETH_TYPE match field setting to match $0x86dd$ (IPv6 type), the OXM_OF_IP_PROTO field setting to match 58 (ICMPv6 type), the *priority* field setting to a high value, and the *instructions* field setting as "send to controller" action. This way, every ICMPv6 packet will be included in a packet-in message and sent to the controller. Here, why we do not capture the NDP packets directly through the OXM_OF_ICMPV6_TYPE match field is because this field is not a required match field, which means it may not be supported by the switch. This module listens to the packet-in messages and screens out the NDP packets according to the value of type field of ICMPv6 packet. If it is between 133 and 137, which means a NDP packet, the process will build an entry according to the packet-in constructed as [sMAC,sIPv6,DPID,port], where sMAC and sIPv6 mean source MAC address and source IPv6 address, respectively. Then, this entry is checked to match the entries in the au-table to finish verification. If the packet is not NDP, this module will do nothing with it, and it will be processed by other default modules of the controller. The pseudocode of the algorithm is shown in Algorithm 1.

According to the algorithm, the process will first check whether the entry [sMAC,sIPv6] exists in re-table. This table records the historical authentication and path-finding and is used to avoid unnecessary cost in look-up and path-finding when a NDP packet passes multiple switches (the following paragraphs will give more details). If the entry [sMAC,sIPv6] exists in re-table, then this NDP packet will be sent to forwarding module immediately. Only when the entry is not in re-table will the process turn to look up in au-table and continue the authentication. The IPv6 address 0:0:0:0:0:0:0:0 is the uncertain address used to specify the default of address when a node is at the initial stage and does not have an address. In this situation, the *ipv6* value of the entry in au-table mapped to this node will be *null*, and we regard these two values as matching. At last, the process executes the forwarding function according to the verification result flagged by the value of $flag$. If $flag$ equals two, which means fail, the process also invokes *add_log*() function and adds this event to log module.

```
procedure verification (packet-in,au-table,re-table)
extract information from packet-in
DPID ≔ dpid of the switch sent this packet-in
port ≔ OFPXMT_OFB_IN_PORT {ingress port
of the ICMPv6 packet}
packet ≔ data[] {data field of
packet-in which contains the packet}
header ≔ packet.header
type ≔ header.ICMPv6Header.type
flag ≔ 0
if (type ∉ {133, 134, 135, 136, 137})
   exit
else
   sMAC≔header.sourceMAC
   sIPv6≔header.sourceIPv6
   dMAC≔header.destinationMAC
   dIPv6≔header.destinationIPv6
   if ([sMAC,sIPv6] exists in re-table)
     flag ≔ 0
   else if (sMAC exists in au-table)
     read entry [sMAC,ipv6,dpid,p] from au-table
     if ([sIPv6,DPID,port]==[ipv6,dpid,p])
          flag ≔ 1
     else if (sIPv6==0:0:0:0:0:0:0:0 && ipv6==null
      && [DPID,port]==[dpid,p])
          flag ≔ 1
     else
          flag ≔ 2
   else
          flag ≔ 2
comb≔[sMAC,sIPv6,dMAC,dIPv6,DPID,port]
if (flag == 2)
   add_log ([sMAC,sIPv6,DPID,port])
execute forwarding (packet-in,comb,flag,re-table)
```

Algorithm 1

Forwarding. This module processes the packet-in messages according to the verification result. It also maintains a table called re-table to simplify the process of verification and forwarding when the NDP packets traverse multiple switches. Because these packets will be verified and forwarded repeatedly, we record the verification result and path in re-table when a packet is firstly verified and forwarded, and when this packet is sent to the controller again, it will be verified and forwarded immediately according to the information stored in re-table. So it reduces the time cost compared to look-up in au-table, which is usually much bigger than re-table, and avoids repeatedly finding path to the destination. Every entry in this table is constructed as ([smac,sipv6],([dmac,dipv6],path)), where smac/sipv6 and dmac/dipv6 represent the source and destination MAC/IPv6 address, respectively, and *path* indicates the path from the source to the destination node. This table is indexed by the key [smac,sipv6], and every key can map to multiple values because a node may send different NPD packets to different destinations. When the au-table updates an entry (e.g., host changes its connection to another attachment point), the re-table will look up all values of [dmac,dipv6] to examine whether a value related to the changed entry, and if it does exist, the re-table will find a new path to update the value of *path*. Besides, every entry in re-table only lasts for t_1 seconds, which controls the size of this table. Here, we firstly give the pseudocode of the forwarding algorithm and then explain the details with an example (see Algorithm 2).

Suppose a network is as the topology as shown in Figure 7, and the au-table and re-table have been initialized. Assume all hosts have run for a certain time so that the au-table has three entries and the re-table is empty. The three entries are denoted as [mac1,ip1,s1,p1], [mac2,ip2,s1,p3], and [mac3,ip3,s2,p1]. If host 1 wants to send a NA response packet to host 3, the steps are as follows:

(1) Host 1 sends this packet to port 1 of switch 1.

(2) The packet matches the flow entry and is sent to the controller through a packet-in message.

(3) Verification module verifies this packet. Because the re-table is empty, it looks up in au-table. Then, the packet is matched to the entry [mac1,ip1,s1,p1] and passes the verification with the value of *flag* being one.

```
procedure forwarding (packet-in,comb,flag,re-table)
sMAC:=comb.sMAC
sIPv6:=comb.sIPv6
dMAC:=comb.dMAC
dIPv6:=comb.dIPv6
DPID:=comb.DPID
port:=comb.port
if (flag == 2) {authentication failed}
    send packet-out with drop action
else if (flag == 1)
    path:=collector_find_path ([sMAC,sIPv6,dMAC,dIPv6])
    add ([sMAC,sIPv6],([dMAC,dIPv6],path)) to re-table
    determine action based on path
    send packet-out with the action
else if (flag == 0)
    if ([dMAC,dIPv6] exists in re-table.([sMAC,sIPv6])
    && [DPID,port] exists in path)
      determine action based on stored path
    else
      path:=collector_find_path ([sMAC,sIPv6,dMAC,dIPv6])
      add ([dMAC,dIPv6],path) to re-table.([sMAC,sIPv6])
      determine action based on path
      send packet-out with the action
else
    error value of flag
```

Algorithm 2

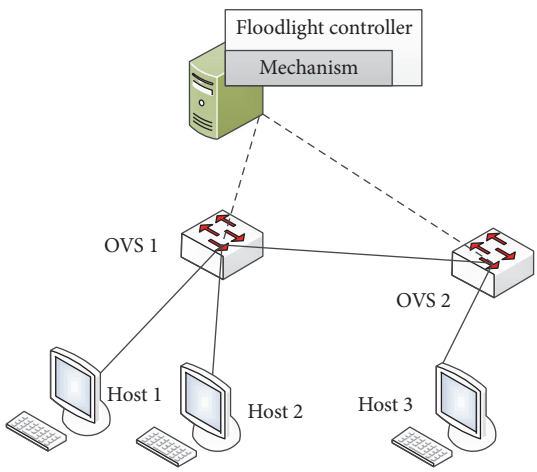

Figure 7: Topology of testbed.

(4) Forwarding module invokes function to find a path for the NDP packet, and, according to the topology, the value of *path* is denoted as [(s1,p1),(s1,p2),(s2,p2),(s2,p1)]. The entry ([mac1,ip1],([mac3,ip3],[(s1,p1),(s1,p2),(s2,p2),(s2,p1)])) is added to the re-table. Then, this packet is sent out through port 2 of switch 1 (corresponding to (s1,p2)).

(5) Switch 2 receives the packet at its port 2 and sends it to the controller again.

(6) Verification module firstly looks up in re-table and finds a matching entry. So the process does not check in au-table and executes the forwarding function immediately with the value of $flag$ being zero.

(7) Forwarding module firstly checks whether [mac3,ip3] and (s2,p2) both exist in the entry related to the [mac1,ip1] in re-table. As we can see in step (4), they do exist and the next point on the path is (s2,p1), so that the packet is sent out to port 1 of switch 2 immediately. If either of them does not exist in re-table, which may be because host 1 sent a new packet to a new destination or host 3 moved to another attachment point or the link between switches on the path changed, the module will invoke the function to find a new path and update it to the re-table.

(8) Host 3 receives the packet at port 1 of switch 2.

The steps of transmitting other kinds of NDP packets are similar to the above. The main difference is when the NDP packet is a multicast packet; the process will use a special value of *path* to direct the transmission of this packet, and here we do not give more details of the process.

Log. This module is used to record the event of false verification. When the verification module invokes *add_log*() function, this module searches the au-table according to the value of [MAC,DPID,port], and if it finds that an entry in the table is the same as this value, it will regard the device owning this MAC address at attachment point [DPID, port] as a suspicious node. When this situation happens, log module will alarm the administrator to take further actions, such as informing the owner to scan for viruses or set an ACL (Access Control List) for this device.

Table 2: Testing result.

Number of packets		Host 1		Host 3		Detection %	False %
Legitimate	Spoofed	Sent NS	Received NA	Received NS	Sent NA		
10	0	10	10	10	10	100%	0
20	10	30	20	20	20	100%	0
200	100	300	200	200	200	100%	0

4. Validation Experiment and Discussion

In this section, we give some results of the experiment to validate the effectiveness of the proposed mechanism. And then we discuss some problems of it.

4.1. Validation Experiment. In this experiment, we use Floodlight [18] as a controller. The topology of the testbed is depicted in Figure 6. The testbed is deployed based on Open vSwitch (OVS) [19] and VirtualBox [20] installed in two servers, respectively. The OVS is virtual OpenFlow-enabled switch software and VirtualBox is virtual machine (VM) software. One server is installed with two OVS VMs and a controller VM, and the other one is installed with three host VMs. Both servers possess 8 CPUs of Intel Core i7-4790 at 3.60 GHz, 32 GB RAM, and multiple NICs (Network Interface Cards).

We use the THC-IPv6 toolkit [21] to generate NDP packets and Wireshark to capture the packets on host 1 and host 3. We send legitimate and spoofed NS packets in different rates from host 1 to host 3 to simulate the address resolution procedure and then check the captured packets to assess the detection rate. The result is shown in Table 2. From this result, we could see that the proposed mechanism filtered all spoofed packets and successfully transmitted the legitimate packets, so it is effective to verify the NDP packets. As the NDP packets belong to the tiny traffic in a LAN, here we do not take stress testing with thousands of NDP packets.

4.2. Discussion. Strictly speaking, this experiment only validates the effectiveness of the proposed mechanism. Its performance and scalability greatly depend on the performance of the controller. As far as we know, some benchmark results of Floodlight show that it can handle more than 30,000 flows every second with 3 switches and more than 7,000 flows every second with 200 switches. As the number of NDP packets sent per unit time is very small, the controller is fully competent to handle the NDP packets timely in a common LAN. In addition, there is a significant problem that we should discuss: why do we not push flow entry to process the subsequent NDP packets at switches but choose to process all of them at the controller? There are two main reasons. Firstly, because the flow entry we used to capture the NDP packets has a very high priority and the action is "send to controller," if we push a new flow entry to process subsequent NDP packets after verification, the packet will be processed starting at the first flow table, so that the packets will match the prior flow entry and will be sent to the controller again. According to [8], this may lead to a controller-to-switch loop and make the new flow entry and the proposed mechanism invalid. Secondly, the memory resource of the switch is small and precious but the memory of the controller server is sufficient and cheap. Suppose there are m switches and n hosts; in order to process the packet at the granularity that could identify every NDP packet from a host to another host, the number of the flow entries in a switch is $O(n^2 - n)$, but in our solution it is a constant one, which saves much more memory resource of the switch. The size of au-table is determined by the active hosts in a network and is no more than n, and the size of re-table is much smaller than au-table because it only records NDP packets lately sent in t_1 seconds. So the proposed solution takes very little memory resource and the time cost of verification is small. If we do not consider the transition from traditional network to SDN, this mechanism, compared with the mechanism in [13–16], is very easy to deploy and has a low cost, and it is totally transparent to the hosts/routers and does not introduce any dedicated devices.

Actually, this mechanism made an assumption that the controller and switch are trusted and the network global information obtained from the controller is credible and reliable. As the network of switches and controller is usually a separate dedicated network and the connection between switch and controller is through security channel such as TLS connection, this assumption is reasonable and acceptable in real network. If the controller is trusted, all nodes at data plane can be verified by this controller to act as a trustworthy node. This way, we transfer the trust problems at data plane to the control plane. According to the analysis of [8], many threats are not a concern if NDP works in the trust model that all authenticated nodes trust each other to behave correctly at the IP layer and not send spoofed NDP packets. So, if the controller can authenticate the nodes to trust each other, it could provide an ideal trust model for the NDP to work as supposed. As the controller is logically centralized, the security policies for it are easy to be realized by the administrator compared with the complicated configuration and deployment in traditional distributed networks. The problem of single point of failure can be solved by the solutions like master/slave method, which is out of the scope of this paper. In addition, the network that connects the controller and switches is usually physically isolated from the outside networks, and this prevents the compromise from the Internet. In a word, transferring the authentication problems of NDP to the controller provides us with a new perspective to secure the NDP in a way with zero configuration and transparent to hosts and routers.

5. Conclusion and Future Works

In this paper, we proposed an authentication mechanism to verify the source of NDP packets for securing the NDP protocol. This mechanism is based on the functions of SDN controller, and it handles the NDP packets based on the global view of the network. Compared with the solutions in traditional networks, our mechanism does not need any new protocol support or configuration on hosts and routers. This is because we transfer the trust problem in data plane to the control plane, and, with the help of the logical centralization of SDN, we could solve this problem at a single point. The programmability of SDN also makes it easy to realize the solution, and we believe SDN can provide more exciting possibilities for solving the IPv6 security issues.

In the future, we mainly consider three problems. The first problem is the detection of attacks that use the real IPv6 address, such as flooding DoS attacks on address resolution. Secondly, the potential vulnerabilities of this mechanism also need to be researched, and, for example, this mechanism may incur DoS attacks on the controller. At last, as different applications may adopt different strategies for the packet-in messages, the rules or actions conflict should be treated carefully.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

This work is supported by the CERNET Innovation Project NGII20150401.

References

[1] S. Deering and R. Hinden, *Internet Protocol Version 6 (IPv6) Specification*, Internet rfc 2460 edn, 1998.

[2] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," Internet rfc 2461 edn. 2007.

[3] S. Thomson, T. Narten, and T. Jinmei, "IPv6 stateless address autoconfiguration," Internet rfc 4862, 2007.

[4] Open Networking Foundation(ONF), *Software-Defined Networking: The New Norm for Networks*, 2012.

[5] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: state of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[6] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 1086–1097, 2015.

[7] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[8] Open Networking Foundation(ONF), *OpenFlow Switch Specification Version 1.3.0*, 2012.

[9] F. Xiaorong, L. Jun, and J. Shizhun, "Security analysis for IPv6 neighbor discovery protocol," in *Proceedings of the 2nd International Symposium on Instrumentation and Measurement (IMSNA '13)*, pp. 303–307, Toronto, Canada, December 2013.

[10] A. AlSa'deh and C. Meinel, "Secure neighbor discovery: review, challenges, perspectives, and recommendations," *IEEE Security and Privacy*, vol. 10, no. 4, pp. 26–34, 2012.

[11] C. E. Caicedo, J. B. D. Joshi, and S. R. Tuladhar, "IPv6 security challenges," *Computer*, vol. 42, no. 2, pp. 36–42, 2009.

[12] A. S. Ahmed, R. Hassan, and N. E. Othman, "Improving security for IPv6 neighbor discovery," in *Proceedings of the International Conference on Electrical Engineering and Informatics (ICEEI '15)*, pp. 271–274, IEEE, Denpasar, Indonesia, August 2015.

[13] J. Arkko, J. Kempf, B. Zill, and P. Nikander, "Secure Neighbor Discover (SEND)," Internet rfc 3971 edn. 2005.

[14] F. A. Barbhuiya, S. Biswas, and S. Nandi, "Detection of neighbor solicitation and advertisement spoofing in IPv6 neighbor discovery protocol," in *Proceedings of the 4th International Conference on Security of Information and Networks (SIN '11)*, pp. 111–118, November 2011.

[15] S. U. Rehman and S. Manickam, "Rule-based mechanism to detect Denial of Service (DoS) attacks on Duplicate Address Detection process in IPv6 link local communication," in *Proceedings of the 4th ICRITO (Trends and Future Directions)*, pp. 1–6, 2015.

[16] G. Yao, J. Bi, S. Wang, Y. Zhang, and Y. Li, "A pull model IPv6 duplicate address detection," in *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN '10)*, pp. 372–375, Denver, Colo, USA, October 2010.

[17] P. Nikander, J. Kempf, and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats," Internet rfc 3756 edn. 2004.

[18] floodlight, http://www.projectfloodlight.org/.

[19] OpenvSwitch, http://openvswitch.org/.

[20] Virtualbox, https://www.virtualbox.org/.

[21] THC-IPv6, https://www.thc.org/thc-ipv6/.