Hindawi
Security and Communication Networks
Volume 2020, Article ID 2408792, 15 pages
https://doi.org/10.1155/2020/2408792

Research Article

Using FDAD to Prevent DAD Attack in SEcure Neighbor Discovery Protocol

Guangjia Song,[1] Hui Wang,[2] and Fuquan Liu[1]

[1]*School of Engineering and Technology, Jiyang College of Zhejiang A&F University, Zhuji 311800, China*
[2]*National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China*

Correspondence should be addressed to Guangjia Song; zhangsanfengajia@163.com

Received 21 September 2019; Revised 8 May 2020; Accepted 14 May 2020; Published 30 May 2020

Academic Editor: Petros Nicopolitidis

Copyright © 2020 Guangjia Song et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The SEND uses CGA as its address configuration method. CGA binds the IPv6 address with multiple auxiliary parameters, thereby making the dependency relationship between IPv6 address and host provable, which prevents address embezzlement. Owing to the considerable overhead in CGA parameter verification, the malicious host can use this point to carry out DoS attacks. To prevent DoS, the paper proposes a new duplicate address detection method in an SDN environment called FDAD. Two additional mechanisms are added to the FDAD, namely, query and feedback; messages used by the new mechanisms are also designed. Through these two mechanisms, on the one hand, the host can query the MAC address of the suspect host to the controller. On the other hand, if the CGA parameter verification fails, the controller will use feedback information to suppress malicious host from its source port in order to prevent subsequent attacks. Experiments show that the CPU overhead of FDAD is much lower than the normal CGA when suffering Denial of Service attack. The increased CPU consumption and memory overhead of the controller are also within acceptable range, and the network communication overhead is greatly reduced.

1. Introduction

People use modular approach in dealing with complex problems. In complex network communication, people utilize a hierarchical method to simplify its design and implementation. Regardless of the TCP/IP structure or the OSI structure, both expressed modularity [1]. To simplify the design, each layer employs different independent communication addresses, such as, Internet layer (layer 3) uses IP address as the communication address of the packet, whereas data-link layer (layer 2) utilizes the Media Access Control (MAC) address as its communication address to forward the frame.

A hierarchical network should solve two problems: one is how the network entity obtains its communication address and ensures its uniqueness, and the other one is how to determine the correspondence of communication addresses between layer N+1 and layer N when data are encapsulated and forwarded. For example, when layer 3 encapsulates the data using the IP address and forwards it to data-link layer, the data-link layer needs to determine what the MAC address should be used to complete the current layer's encapsulation.

At present, there are two main protocols for solving the aforementioned problems: Address Resolution Protocol (ARP) and Neighbor Discovery Protocol (NDP) [1, 2]. The ARP is employed to solve the corresponding relationship between IP and MAC addresses in IPv4. IPv6 uses NDP to carry out the same thing [3]. The NDP protocol has a functional extension to ARP which includes some new functions, such as the Duplicate Address Detection (DAD), Neighbor Unreachable Detection (NUD), and Stateless Address Auto Configuration (SLAAC) [4]. However, many weaknesses in ARP have not been properly solved and persisted in NDP, such as Denial of Service (DoS) in the DAD process.

As shown in Figure 1, when host A is accessed in a LAN, it has to obtain a network address for future communication, and DAD should be completed before a new address is utilized to ensure its uniqueness. Thus, malicious host C can use it to carry out attacks with two basic methods:

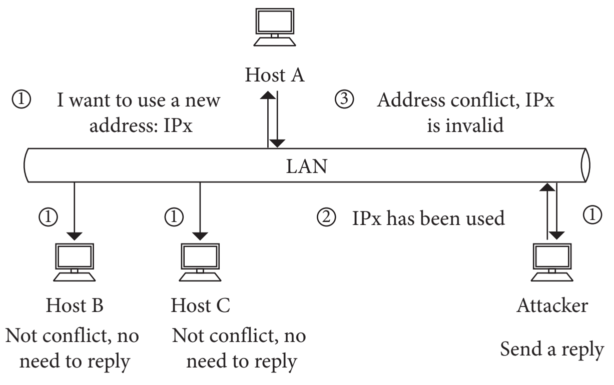
Host A
① I want to use a new address: IPx
③ Address conflict, IPx is invalid
LAN
①
①
② IPx has been used
①
Host B
Not conflict, no need to reply
Host C
Not conflict, no need to reply
Attacker
Send a reply

Figure 1: DAD attack.

(i) Send a reply to host A, declaring that the address has been occupied

(ii) Perform a DAD, and the target address is the same as host A wants to use

Both of these attacks can cause host A to believe that IPx was occupied, and it should choose other addresses and repeat DAD. If the attack of malicious hosts persists, then victim host A will have no address to use.

Although IPv6 recommends using Internet Protocol Security (IPSec) to fully protect IPv6 messages, the chicken-and-egg problem is bothersome when IPSec is employed to protect NDP messages [5]. IPsec should establish a point to point Security Association (SA) before security communication. However, the neighbor discovery process should be completed before SA is set up. Therefore, IPsec cannot be used to protect NDP [6]. In this regard, IETF proposes SEcure Neighbor Discovery (SEND) as an enhancement for NDP. SEND uses four options, namely, Cryptographically Generated Address (CGA), Timestamp, Signature, and Nonce to prevent IPv6 address embezzlement [7]. Even so, DoS attack still bothers SEND [8].

In order to improve the security of DAD, we propose a new method which is called Feedback Duplicate Address Detection (FDAD). FDAD's query and feedback mechanisms have the ability to record information of Neighbor Solicitation (NS) and Neighbor Advertisement (NA) in DAD process and thus make DAD process stateful. The node can use its computing ability to identify the NA with forge CGA parameters, and then feedback the results to the FDAD-Server. These will make controller have ability to inhibit malicious NA from the entrance, thereby preventing DoS attacks. Section 2 introduces the DoS principle in DAD and related research. Section 3 introduces the principle of feedback mechanism and shows the FDAD workflow through an instance. Section 4 is the experiment and result analysis; Section 5 summarizes the article.

2. Related Works

To prevent address deception, the Internet Engineering Task Force (IETF) proposed SEND. To enhance NDP, SEND uses CGA [9, 10], digital signature, and timestamp to protect NDP messages and to prevent IP address embezzlement. CGA is a unique address format for SEND, and its generation method has two steps: first is to find an appropriate Modifier through multiple times hash operations on public key, zero bit, Modifier, and extension field resulting in hahs2; second is to create a hash operation on Modifier, Collision Count, public key, and extension field leading to hash1; the left 64 bits of 160 bit hash1 is then combined with Security Level (Sec) and other parameters to form the final CGA address. The CGA calculation process is presented in Figure 2.

Although SEND uses complex security technologies, some problems still exist. First, the CGA computation and verification process requires a lot of CPU resources. Second, additional options expand the NDP message and increase communication overhead. Third, CGA generation time is related to the Sec bit, in which the larger the Sec value is, the longer the time is. Furthermore, when a new address is generated, DAD is still necessary.

In view of the aforementioned issues, Alsa'deh et al. [11] proposed stopping time algorithm for CGA process, which limits time consumption on CGA, and an appropriate Sec value is obtained by determining the upper limit of resource, to ensure that the CGA address is generated within a specified time. Rafiee et al. [12] indicated a parallel computing algorithm by using a multicore processor to shorten CGA computing time. Cheneau et al. [13] suggested using Elliptic Curve Cryptography (ECC) key and ECDSA to replace the RSA key and corresponding signature algorithm in order to reduce CGA computing time, but the same security can be achieved. Due to the ECC's key is shorter, so the NDP message generated is smaller. Qadir and Umar Siddiqi [14] make a performance evaluation of the deployments of CGA in mobile environment, and it shows that, in addition to the Sec field should not be greater than 0, the choice of public key is also crucial.

Su et al. [15] added a high-performance server in LAN for key computing, and it is recommended to use Dynamic Host Configuration Protocol (DHCP) server to manage CGA. The DHCP protocol is improved so that CGA parameters can be broadcasted in the network. When a host needs to compute CGA, the computing process is handed over to the DHCP server for completion. Certainly, this makes the DHCP server single point of failure. CS-CGA recommends using ECC instead of RSA to speed up calculation, and subnet prefix is also encrypted to counter the spatiotemporal balance algorithm [16]; however, this undoubtedly has a negative impact on routing and forwarding. Reshmi and Murugan [17] used the entropy of system state to generate Interface IDentifier (IID), which overcomes the centralized computation of CGA, reduces address calculation time, and prevents privacy leakage.

Owing to technical and cost reasons, most operating systems and communication device vendors have only implemented the partial function of SEND. Thus, the SEND protocol still has abundant work to complete from actual deployment [18, 19]. The CGA verification process is described in Figure 3 as follows:

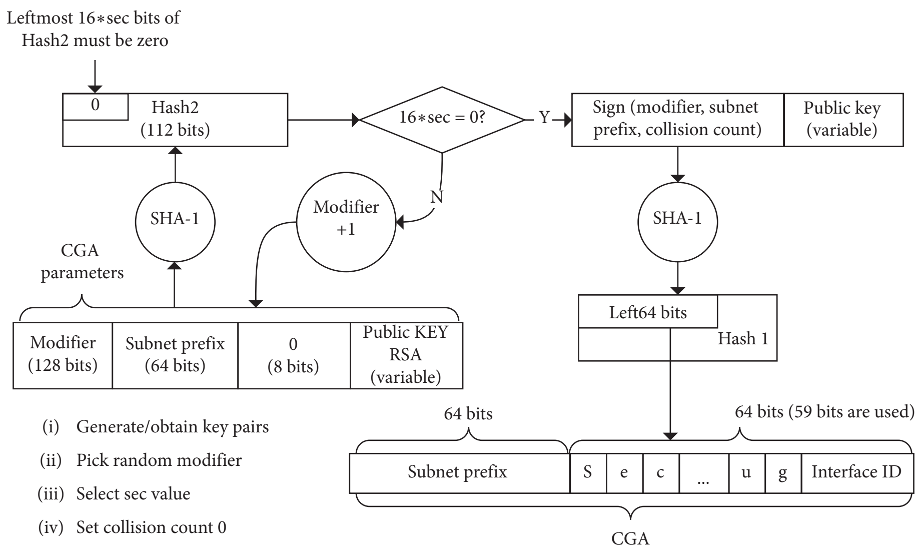

Figure 2: Algorithm of CGA.

(1) If the collision count is not equal to 0, 1, or 2, then CGA verification fails; otherwise, proceed to Step 2.
(2) Check whether the subnet prefix in CGA parameters is equal to the subnet prefix of the address; if it is unequal, then CGA verification fails; else go to Step 3.
(3) Calculate hash1; compare hash1 with IID (ignoring left 1, 2, 3, 6, and 7 bit). If unequal, then CGA verification fails; otherwise, proceed to Step 4;
(4) Calculate hash2, combined with the Sec; if the left 16 × sec bits in hash2 are zero, the verification passed; otherwise, CGA verification fails.

In theory, CGA can effectively prevent address deception. The one-way feature of the hash function means that, for a given y, finding an x to satisfy equation hash(x) = y is computationally infeasible [20]. Therefore, malicious hosts cannot embezzle other hosts' address by forgery parameters; simultaneously, digital signatures further increase deception difficulty. However, a host should complete verification whether CGA parameters are right or not. Therefore, a malicious host can send numerous NA with false CGA parameters to consume the computing resources of a victim, forming DoS [21]. Thus, the manners in which DoS attacks are prevented and host CPU overhead is reduced to remain a major challenge for SEND.

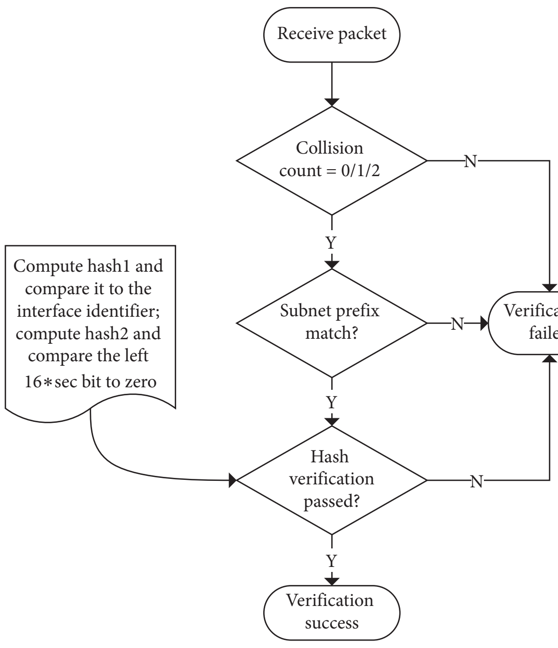
Receive packet
Collision count = 0/1/2
N
Y
Compute hash1 and compare it to the interface identifier; compute hash2 and compare the left 16∗sec bit to zero
Subnet prefix match?
N
Verification failed
Y
Hash verification passed?
N
Y
Verification success

Figure 3: CGA verification process.

3. FDAD

In traditional Ethernet, solving Denial of Service (DoS) attacks in the DAD process is difficult due to the equivalence between hosts and incompleteness of knowledge with single host [22]. However, the emergence and development of Software Defined Network (SDN) have injected new vitality into modern network. The characteristics of its forwarding and control separation and programmability provide new ideas for solving the network problem [23–26], such as the NDP message authentication scheme in SDN environment [27]. In SEND, network device has no ability to distinguish NA which is constructed by malicious host using false CGA parameters, but the host has the ability to distinguish them by hash calculation; therefore, we propose FDAD as a solution. In FDAD, the computing-intensive tasks are performed by various hosts, and the control plane prevents DoS attacks from the data plane. The characteristics of FDAD include the following aspects:

(i) FDAD designs a feedback mechanism for duplicate address detection in SDN environment, which contained two parts: FDAD-S (FDAD-Server) and FDAD-C (FDAD-Client).
(ii) FDAD-C can take advantage of host's computing power to identify malicious message, then feedback to the FDAD-S.

(iii) FDAD-S can then suppress malicious attacks from the source based on feedback information, thereby avoiding meaningless CGA verification consuming the CPU resources of the host.

The architecture of FDAD is shown in Figure 4. It mainly includes four modules: Monitor, Status, FeedBack, and Suppression. The functions of each module are as follows:

Monitor module: it includes two functions. On the one hand, it is responsible for initializing switches to monitor NDP messages; on the other hand, it classifies received messages, forwards NDP messages to Feedback module, and forwards OpenFlow statistical messages to Status module.

Status module: it is responsible for statistical queries of switch flow tables and forwards the query results that satisfying specific conditions to Feedback module for processing.

FeedBack module: it is responsible for maintaining and recording the behavior of DAD process in NDP, initiating MAC address statistics, and sending NDP message suppression rules. To achieve the above functions, it includes four tables: T_{NS}, T_{NA}, T_{Query}, and T_{FeedBack}.

Suppression module: it generates flow tables and sends them to the corresponding switches according to the suppression rules generated by Feedback module.

FDAD-C is mainly responsible for CGA validation, MAC query, and feedback in the SEND process of the client.

In the following description, we assume that all the DAD messages include CGA, Signature, Nonce, and Timestamp options and that the network is consisted by OpenFlow-enabled switches. To implement FDAD, we designed three new messages: FDAD-Request, FDAD-Reply, and FDAD-Feedback. Their formats are basically the same as that of NDP messages, whereas the differences are as follows:

(1) Unlike NS and NA, their ICMPv6 "type" field is 200

(2) Three new options are used, detailed formats are presented in Figure 5, and the descriptions of each field in options are shown in Table 1

3.1. FDAD-S.

3.1. FDAD-S. The FDAD-S is an app running over the SDN controller. In FDAD-S, we added a feedback module to control the FDAD workflow. The feedback module contains four tables: T_{NS}, T_{NA}, T_{Query}, and T_{Feedback}; their formats are the same as in Tables 2–5, respectively. The T_{NS} table is used to record the NS in the DAD and the T_{NA} table is used to record the NA that corresponds to the NS. The T_{Query} table is utilized to note the query initiated by the hosts that previously sent NS, and it is constrained by the T_{NS} and T_{NA}. The T_{Feedback} table is employed to record the feedback of the host that has carried out DAD before, and it is constrained by the T_{Query}.

FDAD-S monitors the NS and NA messages in the DAD process by the preset flow tables in switch. The processing of different messages is as follows:

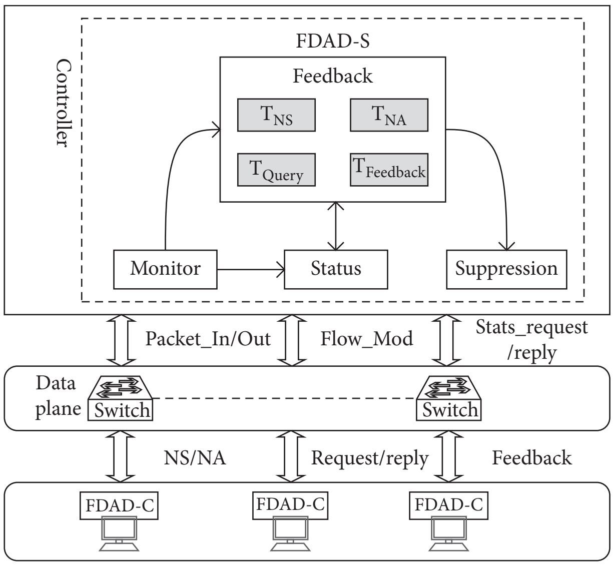

Figure 4: Architecture of FDAD.

(1) NS: if the entry, which meets the condition Entry'.NS.Tgt = = NS.Tgt, does not exist in the T_{NS}, then a new entry is added to the T_{NS} (fields NS.Dpid and NS.Port are extracted from the Dpid and In_Port fields of the Packet_In message. NS.MAC, NS.Tgt, NS.Ts, NS, and Nonce are extracted from the corresponding NA fields carried by the Packet_In message. These methods are also used in T_{NA}). Otherwise, discard the NS.

(2) NA: if it satisfies Conditions 1–3, then a new entry is added to T_{NA}; otherwise, discard NA.

Condition 1. The receiving NA is not duplicated with all the entries in T_{NA}.

Condition 2. There is an Entry' in T_{NS} that satisfies Entry'.NS.Tgt = = NA.Tgt.

Condition 3. Nonce and Timestamp field of Entry' is related to the corresponding fields of NA (NA.Nonce = = Entry'.NS.Nonce, and NA.Timestamp-Entry'.NS.Timestamp $\leq$ 3).

(3) FDAD-Request: if there is an Entry' in T_{NS} and an Entry" in T_{NA} satisfy Condition 4–7, then a new entry is added in T_{Query}, and it means the query is legal. The various fields of the new entry are extracted from Entry' and Entry". Then, MAC query is then carried out. The query algorithm is shown in Algorithm 1.

(i) If the query is successful, then the flag of the new entry is set to *T*; otherwise, the flag should be set to *F*. The message is discarded if the conditions are not satisfied.

Condition 4. Entry'.NS.Dpid = = Packet_In.Dpid.

Condition 5. Entry'.NS.Port = = Packet_In.InPort.

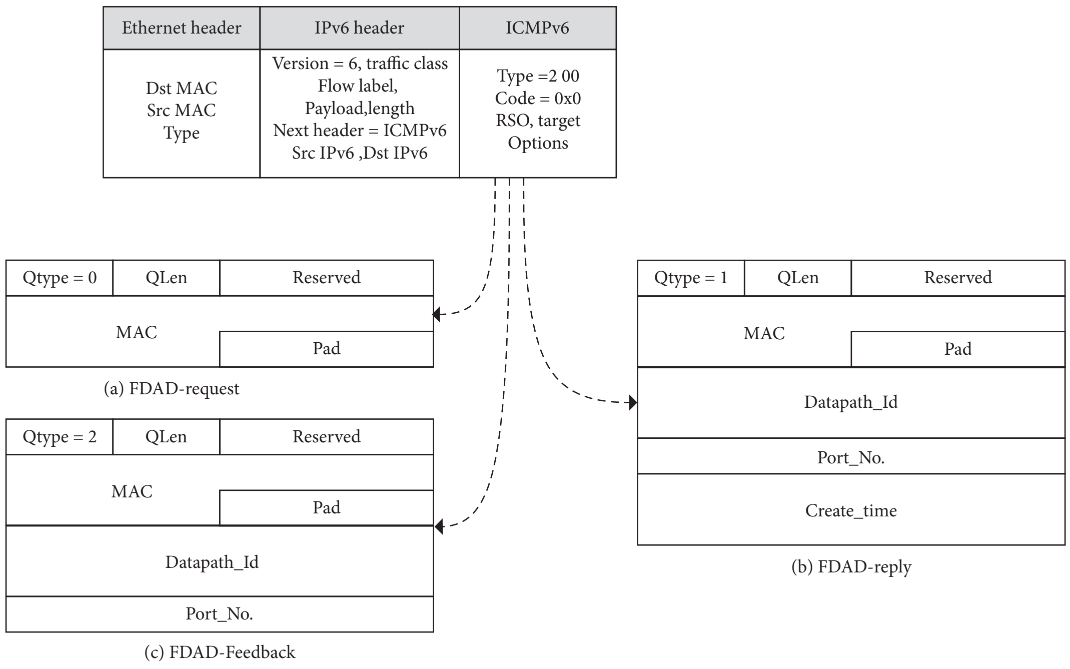

Figure 5: FDAD-request, -reply, and -feedback.

Table 1: Descriptions of each field.

Field	Description
QType	8 bits; 0, query message; 1, reply message; 2, feedback message
QLen	8 bits, the length of the message, and unit is bytes
Reserved	16 bits, reserved, all is zero
MAC	48 bits, the queried MAC address, all zero in reply message indicate MAC not exist
Datapath_Id	64 bits, the ID of OpenFlow switch which MAC table contains the queried MAC
Port_No.	32 bits, the port number corresponding to queried MAC
Create_time	64 bits, creation time of the queried MAC

Table 2: T_{NS} (first update).

NS.Dpid	NS.Port	NS.MAC	NS.Tgt	NS.Ts	NS.Nonce
0 × 1	1	0000-0000-0001	CGA_X	T1	R1

Table 3: T_{NA}.

NA.Dpid	NA.Port	NA.MAC	NA.Tgt
0 × 2	3	MAC_C	CGA_X

Table 4: T_{Query}(first update).

NS.Dpid	NS.Port	NS.MAC	NA.Dpid	NA.Port	NA.MAC	Flag
0 × 1	1	MAC_A	0 × 2	3	MAC_C	

Table 5: T_{Feedback} (first update).

NS.Dpid	NS.Port	NS.MAC	NA.Dpid	Mm,NA.Port	NA.MAC
0 × 1	1	MAC_A	Ox2	3	MAC_C

Condition 6. Entry'.NS.MAC = = FDAD-Request.SrcMAC.

Condition 7. Entry'.NA.MAC = = FDAD-Request.Option. MAC.

(4) FDAD-Feedback: if there is an Entry' in the T_{Query} that satisfies the conditions 8–13, then a new entry is added in T_{Feedback}, and the various fields of the new entry are extracted from Entry'. Related entries in the T_{NS}, T_{NA}, and T_{Query} should be deleted; else discard the FDAD-feedback.

Delete procedure: if feedback is successful, it means that a NA is malicious; then, first it deletes the entries related to the NA in T_{Query}, then deletes the entries related to the NA in T_{NA}, and finally deletes the entries related to the NA in the T_{NS}.

Condition 8. Entry'.NS.Dpid = = Packetin.Dpid.

Condition 9. Entry'.NS.Port = = Packetin.InPort.

```
Algorithm 1: Create time searching
Input: MACx
Output: Create time
Time = MAX
    For each Flow-Table in Switch do
      For each entry in Flow-Table do
        IfMACx exist in entry then
          Ifentry.CreateTime < Time then
            Time = entry.CreateTime
            Switch_Dpid = OVS.ID
            Port_No. = entry.Ingress_Port
          End if
        End if
      End for
    End for
If Time = = MAX then
    Return False
Else
    Return Time, Switch_Dpid, Port_No.
```

Algorithm 1: In FDAD, when the FDAD-S receives the FDAD-Request message from the host, the former inquires the OpenFlow switch and finds the earliest generation time of a specific MAC by traversing flow tables of each switch.

Condition 10. Entry'.NA.Dpid = = FDAD.Feedback.Option.Datapath_Id.

Condition 11. Entry'.NA.Dpid = = FDAD.Feedback.Option.Port_No.

Condition 12. Entry'.NA.MAC = = FDAD.Feedback.Option.MAC.

Condition 13. Flag is *T*.

3.2. FDAD-C. From the view of Ethernet, the DoS can be divided into two categories, such as using real and forged MAC addresses. For the first class, the blacklist will be a good choice. It means that the victim host can write the MAC address of the malicious host in a blacklist to prevent the DoS attack. For the second class, only the blacklist is insufficient because malicious hosts can randomly change values of the fields related to its MAC address.

When the MAC frame is forwarded in Ethernet, the same MAC address will form a forwarding path on multiple switches. If a host uses a new MAC address to communicate, then a new forwarding path will be formed on the switches. Corresponding to the SDN network, if the host adopts a new MAC address, no matter the switch is reactive or proactive, and then it will generate the flow table related to the new MAC address. Therefore, we can determine the authenticity of the MAC address by analyzing the generation time of the flow table related to the MAC address. The DAD process of the host in FDAD-C is shown in Figure 6, described as follows:

(1) After the host generates a new CGA address, suppose it is $\mathrm{CGA_X}$, the host should broadcast NS to carry out DAD and record start time as $T1$.
(2) Within a specified time (usually 3 seconds), if the host receives an NA responding to the NS, then it will record the receiving time as $T2$ and check whether the MAC address in the Option field is consistent with that at the head of NA and whether it is in the blacklist. If any of these conditions are met, then the NA will be discarded, and go back to Step 2. If all the conditions are not satisfied, then go to Step 3.
(3) Send FDAD-Request message to query the switch identity, port number, and MAC address generation time of the NA.
(4) Receive FDAD-Reply message: if no record matches its MAC address, then the NA should be discarded; otherwise, go to Step 5.
(5) Determine whether MAC is added within $T2$-$T1$; if it is, go to Step 6; otherwise, go to Step 7.
(6) Add the switch's Dpid and Port_No. to the blacklist, set its flag to F and then perform the CGA verification.
(7) The host carries out the CGA verification, if the verification fails, then Step 8 is executed; otherwise, Step 9 is executed.
(8) Using NA.srcMAC as a key search in the blacklist, if no Entry' satisfies condition Entry's MAC = = NA.srcMAC, then the NA's SrcMAC, Dpid, and Port_No. should be added to the blacklist, and set the flag as T. If it exists, then its flag should be updated to *T*, and send FDAD-Feedback message to FDAD-S to report the source MAC, switch Dpid and Port_No of the NA, and then go back to Step 2.
(9) DAD failed and $\mathrm{CGA_X}$ is in conflict. If the blacklist has an entry corresponding to the NA.SrcMAC field, then remove it.

The blacklist contains five fields: Switch Dpid, Port No., MAC, Idle_time, and Flag. The Idle_time field is used to record the idle time of the entry, field value plus 1 per second; if the entry is not matched within 3 minutes, then the entry will be removed from the blacklist. The Idle_time field is cleared each time the entry is matched.

3.3. FDAD Instance. Suppose the network is composed of one SDN controller, two Openflow switches, and three hosts. The network has been running for a period of more than 3 minutes. The topology is shown in Figure 7, and the configuration of each host is presented in Table 6.

First, the controller should dispatch the flow table toOpen vSwitch1 (OVS1) and OVS2 in order to monitor DAD message in the network. The items in the flow table related to monitor are shown in Table 7.

Assume that host A generates a new link-local address $\mathrm{CGA_X}$ using CGA as the address configuration method (for illustrative purposes, supposing the last 32 bits of $\mathrm{CGA_X}$ is

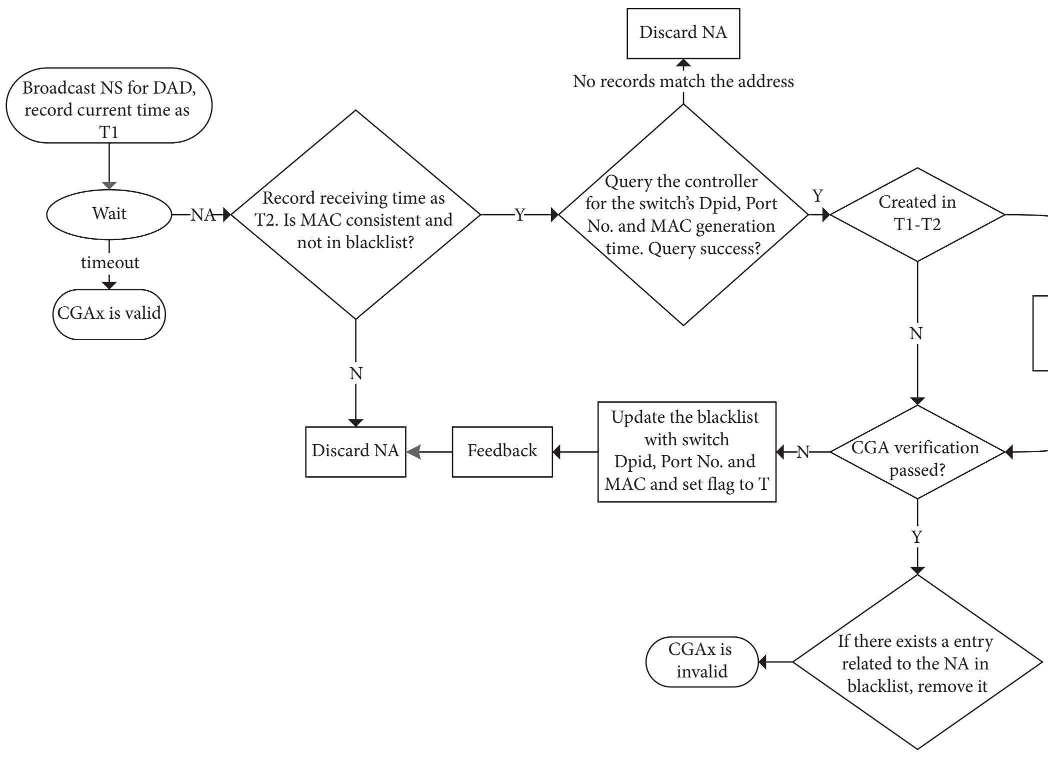
Discard NA
No records match the address
Broadcast NS for DAD, record current time as T1
Wait
NA
timeout
CGAx is valid
Record receiving time as T2. Is MAC consistent and not in blacklist?
Y
N
Query the controller for the switch's Dpid, Port No. and MAC generation time. Query success?
Y
Created in T1-T2
Y
N
Add a new entry to blacklist and set its flag to F
Discard NA
Feedback
Update the blacklist with switch Dpid, Port No. and MAC and set flag to T
N
CGA verification passed?
Y
CGAx is invalid
If there exists a entry related to the NA in blacklist, remove it

Figure 6: Workflow of FDAD-C.

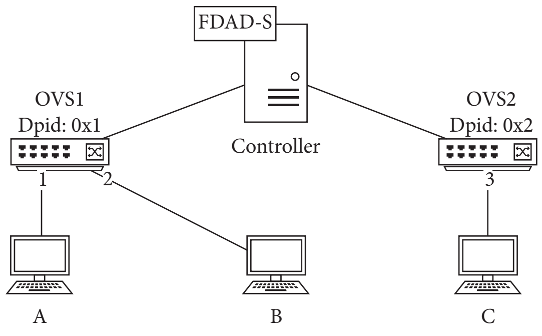

Figure 7: Network topology.

Table 7: Flow table of the OVS1.

OFPST_FLOW (OF1.3) (Dpid = 0 × 1):	
1	cookie = 0 × 0, duration = 100s, table = 0, n_packets = 0, n_bytes = 0, priority = 50,icmp6,icmp_type = 200 actions = CONTROLLER:65535
2	cookie = 0 × 0, duration = 100s, table = 0, n_packets = 0, n_bytes = 0, icmp6, dl_dst = 33 : 33 : 00 : 00 : 00 : 00/33 : 33 : 00 : 00 : 00 : 00, icmp_type = 135 actions = CONTROLLER: 65535,resubmit(,1)
3	cookie = 0 × 0, duration = 100s, table = 0, n_packets = 0, n_bytes = 0, icmp6, icmp_type = 136 actions = CONTROLLER: 65535, resubmit(,1)

Table 6: Basic information of hosts A, B, and C.

Host	IPv6 address	MAC address	Connection	Remarks
A	IPv6_A	MAC_A	OVS1-Port1	Normal host
B	IPv6_B	MAC_B	OVS1-Port2	Normal host
C	IPv6_C	MAC_C	OVS2-Port3	Attack host

"cc00-aabb"), then host A broadcasts NS for DAD, and the details of the NS are shown in Figure 8.

When OVS1 receives this NS, it will encapsulate the message and send a Packet_In to FDAD-S. After the FDAD-S gets the NS contained in Packet_In, it will check it and add a new entry in T_{NS}. The result is shown in Table 4.

After host B receives NS, it finds that CGA_X does not conflict with its own IPv6 address, thus, host B discards the NS. After the host C receives the NS, considering that C is malicious, C forges the NA to attack host A, and the NA detail is shown in Figure 9. According to the flow table, OVS2 generates a Packet_In message and forwards it to FDAD-S after receiving the NA. After FDAD-S checks the NA, it is found that Conditions 1–3 are satisfied, and then it updates the T_{NA}. The result is shown in Table 3.

After the NA is received, host A does not verify the parameters immediately but checks the consistency of the MAC and blacklist; if the check is passed, the FDAD-Request message will be sent. The detail of the message is shown in Figure 10. After OVS1 receives the message, it uses Packet_In to forward the message to the FDAD-S.

FDAD-S checks the T_{NS} and T_{NA}, and it finds two entry corresponding to the FDAD-Request and satisfies Conditions 4–7, and thus, it updates the table T_{Query} (see the update results in Table 4). Then, FDAD-S queries to the OVS2 for the generation time of MAC_C. Once the query is

Ethernet Header	Dest MAC	3333-ff00-aabb
	Src MAC	MAC_A
	Type	0x0806
IPv6 Header	Src IP	$IPv6_A$
	Dest IP	ff02::1:ff00:aabb
	Next header	0x3a
ICMPv6	Type	135
	Target address	CGA_X
	Option	MAC_A
		CGA parameters
		Time stamp: T1
		Nonce:R1
		Signature

Figure 8: The NS sent by host A.

Ethernet Header	Dest MAC	MAC_A
	Src MAC	MAC_C
	Type	0x0806
IPv6 Header	Src IP	$IPv6_C$
	Dest IP	$IPv6_A$
	Next header	0x3a
ICMPv6	RSO	O = 1
	Type	136
	Target address	MAC_A
	Option	CGA parameters
		Time stamp: T2
		Nonce:R1
		Signature

Figure 9: The NA sent by host C.

Ethernet Header	Dest MAC	3333-ff00-0001
	Src MAC	MAC_A
	Type	0x0806
IPv6 Header	Src IP	$IPv6_A$
	Dest IP	ff02::1
	Next header	0x3a
ICMPv6	Type	200
	Option	Qtype: 0
		Qlen: 12
		MAC: MAC_C

Figure 10: FDAD-request.

successful, the flag field of the entry in T_{Query} is set to T (see Table 8). The controller then sends FDAD-reply message to OVS1 and indicates OVS1 use port 1 to forward the

Table 8: T_{Query} (second update).

NS.Dpid	NS.Port	NS.MAC	NA.Dpid	NA.Port	NA.MAC	Flag
0×1	1	MAC_A	0×2	3	MAC_C	T

Ethernet Header	Dest MAC	MAC_A
	Src MAC	3333-0000-0001
	Type	0x0806
IPv6 Header	Src IP	$IPv6_A$
	Dest IP	ff02::1
	Next header	0x3a
ICMPv6	Type	200
	Option	Qtype: 1
		Qlen: 32
		MAC: MAC_C
		Datapath_Id: 0x02
		Port_No.: 3
		Ctreate_time:180s

Figure 11: FDAD-Reply.

Ethernet Header	Dest MAC	3333-0000-0001
	Src MAC	MAC_A
	Type	0x0806
IPv6 Header	Src IP	$IPv6_A$
	Dest IP	ff02::1
	Next header	0x3a
ICMPv6	Type	200
	Option	Qtype: 2
		Qlen: 24
		MAC: MAC_C
		Datapath_Id: 0x02
		Port_No.: 3

Figure 12: FDAD-Feedback.

message, and the detail of FDAD-Reply is shown in Figure 11.

Once host A receives the FDAD-reply, it finds that the MAC_C existed for a long time and a new address does not appear in time $T2$-$T1$. Thus, host A verifies CGA parameters contained in NA. Because the parameters are fabricated by host C, the result fails, and so it sends a FDAD-feedback message which will be forwarded to FDAD-S by OVS1, and the detail of FDAD-feedback is shown in Figure 12.

After receiving the FDAD-Feedback, FDAD-S finds that the message meets Conditions 8–13, so it updates T_{Feedback} (see Table 5). It then delivers the flow tables to suppress the NA that accessed in through OVS2 port 3 and

Table 9: Configuration of network environment components.

Operating system	Network simulation	Switch	Controller
Ubuntu mate16.04	Mininet 2.30d1	Open vSwitch 2.5.2	RYU 4.2

deletes the entries that related to the NA in the T_{NS}, T_{NA}, and T_{Query}.

4. Experiment and Analysis

4.1. Experiment. In order to verify FDAD, we implement it in Mininet. OpenFlow switch is Open vSwitch, and the controller is RYU. The operating system is Ubuntu Mate (virtual machine: CPU $2\,\mathrm{GHz} \times 2$, Memory 2 GB). The specific version of each software is shown in Table 9. The network has three hosts, namely, A, B, and C. Hosts A and C simulate normal and malicious hosts, and B is used for monitoring.

C is assumed to have the following computing and communication capabilities.

(i) Computing capability:

(1) The computing power is limited. For any hash value y, C cannot find the original image x satisfying the equation hash $(x) = y$ in a limited time (3 seconds).

(2) C can build NA with false CGA parameters in accordance with the NS but cannot make a forged NA pass the CGA verification of the other side.

(ii) Communication capability:

(1) C can receive broadcast and unicast.

(2) C can change its protocol stack to send any NDP message, e.g., sending NA with false MAC address and CGA parameters and sending a large number of NA to consume the target's resource and carry out DoS.

(3) C cannot sniff peer-to-peer communication, such as the switch forwards a unicast frame from port 1 to port 2.

(4) C is aware of the FDAD mechanism and has the ability to send fake FDAD-Request and FDAD-Feedback to the controller or fake FDAD-Reply.

Scenario 1. Testing overhead of DAD in NDP and SEND and in SEND under attack

Scenario 1 simulates the host which continuously generates a new IPv6 address and performs DAD. It records CPU consumption and carries out the following three experiments.

Experiment 1. A conducts address configuration and DAD using NDP method; B and C monitor.

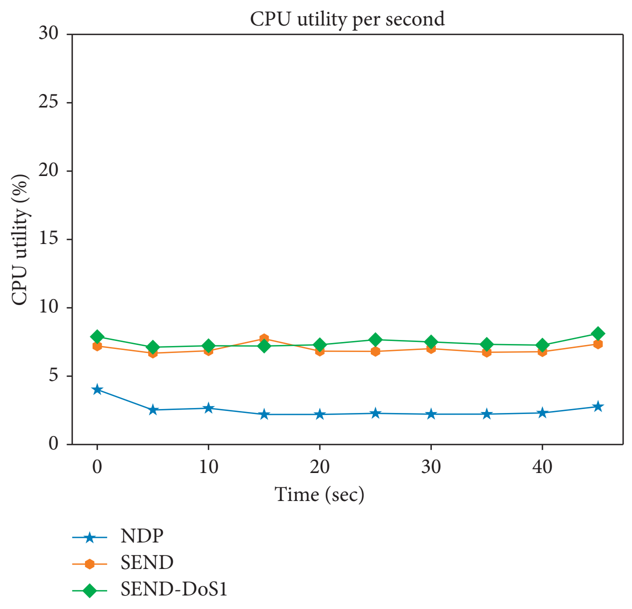

Figure 13: CPU utility comparisons among NDP, SEND, and SEND under DoS1.

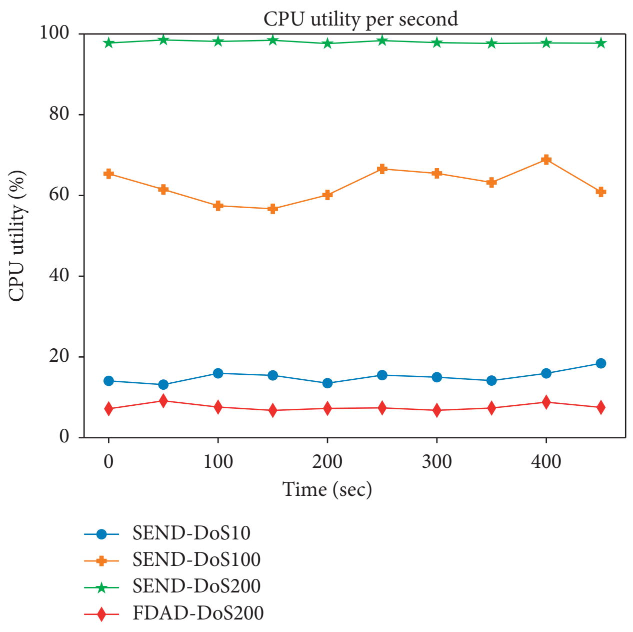

Figure 14: CPU utility comparisons among FDAD and SEND under attacks.

Experiment 2. A carries out address configuration and DAD utilizing SEND method; B and C monitor.

Experiment 3. A conducts address configuration and DAD using SEND method; C sends a false NA with random MAC address and fake CGA parameters to attack (we define this kind of attack as DoS1). The experiment results are shown in Figure 13. The figure presents that the CPU overhead of NDP is the lowest. CGA which is used by SEND will cause CPU overhead slight increase of approximately 4.15%. When SEND

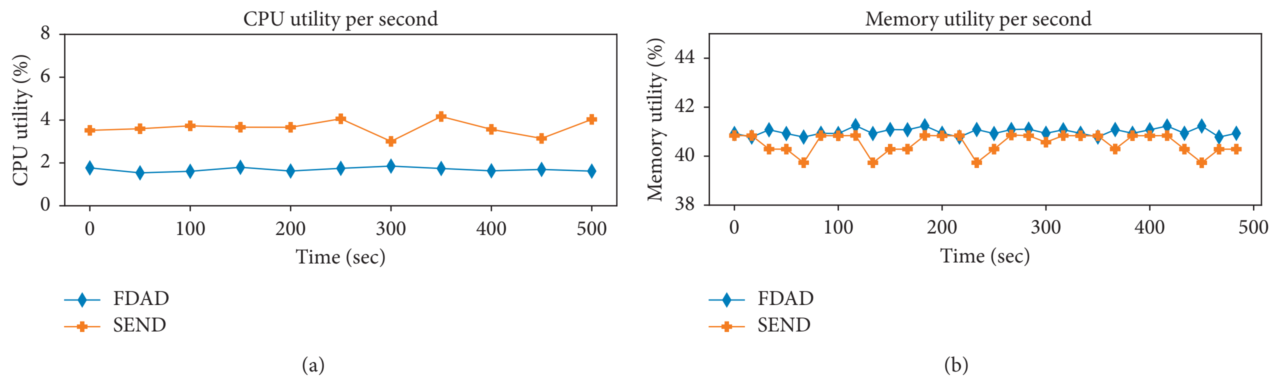

Figure 15: Comparison of controller CPU and memory overhead between Experiments 6 and 7.

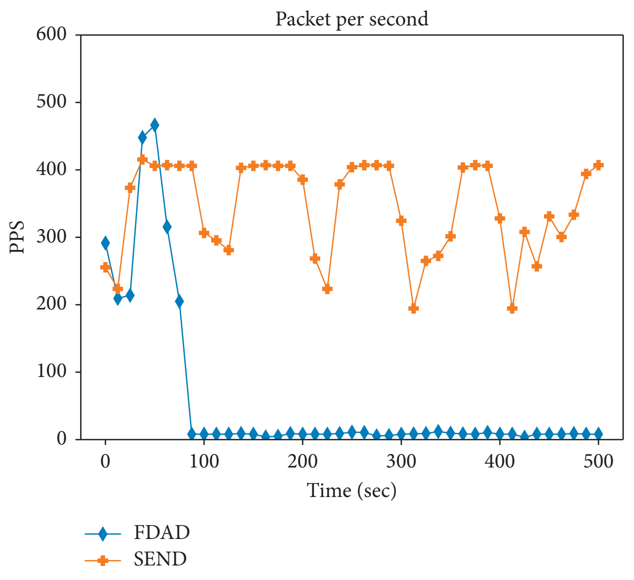

Figure 16: Comparison of network communication overhead between Experiments 6 and 7.

receives a false NA, its CPU consumption has no evident increase compared with normal state, and an increase of about 0.369%.

Scenario 2. Testing CPU overhead of SEND and FDAD when suffering DoS

Scenario 2 simulates the host that suffered DoS attacks during CGA address configuration and DAD, and the attacker uses random MAC address and forged CGA parameters, and records its CPU overhead; it also carries out the following four experiments.

Experiment 4. Host A carries out CGA address configuration and DAD. For each NS that host C receives, it sends ten times NA which contains false parameters to respond (DoS10).

Experiment 5. Host A conducts CGA address configuration and DAD. C sends out NA 100 times which contains false parameters for each NS it received (DoS100).

Experiment 6. Host A carries out CGA address configuration and DAD; C sends out NA 200 times which contains false parameters for each NS it received (DoS200).

Experiment 7. Host A conducts address configuration and DAD using FDAD; C sends out NA 200 times which contains false parameters for each NS it received (DoS200).

The experiment results are shown in Figure 14. We can see that when host A suffers DoS attack in the CGA DAD process, its CPU consumption increases as attack frequency increases. When the attack frequency reaches 200, the host's CPU has been exhausted. In contrast, the figure also shows that the CPU utility of FDAD is much lower, stable, and insensitive to high-frequency attacks.

Figure 15 shows the comparison of controller's CPU and memory overhead between Experiments 6 and 7. As the figure shows, the controller's CPU overhead in FDAD is stable but fluctuant in SEND, with average increases of approximately 1.59%. In FDAD, memory overhead average increases about 0.402% because the controller needs to maintain four additional state tables.

In terms of communication overhead, at the beginning of the experiment, FDAD-S needs to query the switch for MAC information and host communicate with switch frequently; these lead to FDAD communication overhead a slightly higher than normal environment. However, when the suppression mechanism of FDAD works, it effectively suppresses the DoS attack packets sent by the malicious host C. As a result, network traffic is reduced significantly, far below the normal environment, as shown in Figure 16.

Figure 17 shows the overhead comparison of switch bandwidth and RTT in Experiments 6 and 7. The FDAD's bandwidth is relatively stable when suffering from DoS. Even though some fluctuations exist, the amplitude is very small. Under the same attack intensity, SEND's bandwidth gradually increases over time and finally reaches the peak. The RTT of SEND also increases significantly higher than that of FDAD due to

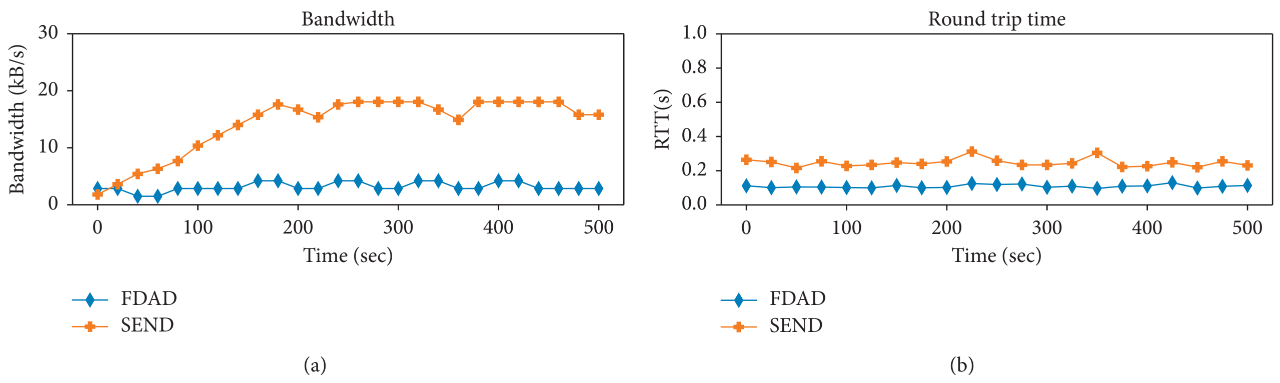

FIGURE 17: Comparison of bandwidth and RTT between Experiments 6 and 7.

hundreds of times increase in network packets caused by DoS.

4.2. Security Analysis

4.2.1. Attack Mode Analysis

(1) Malicious hosts use real MAC addresses to attack.

Attack mode: when the attack host receives a DAD NS sent by a normal host, it sends a fake NA which contains its real MAC in response to declare a conflict.

In FDAD, after the host receives the NA which declares an address conflict, it will not carry out parameter verification immediately. Instead, it will query the controller for the generation time of NA's source MAC. If MAC exists on a switch for more than 1 second, the host will further verify the CGA parameters. Given that the NA is a forgery, the parameter verification step will fail, and the host will write the malicious host's information to the blacklist and feed it back to the FDAD-S. The result is that the subsequent NA of the malicious host will be blocked by a flow table and cannot enter the network. Even if the subsequent forged NA reaches the normal host, it will also be shielded from host's blacklist.

(2) Malicious host uses fake Mac to perform Denial of Service attack.

Attack mode: when the attack host receives the DAD NS sent by normal host, it sends a fake NA which contains random MAC for responding to declare a conflict.

Its residence time on switch is less than 1 second because the random MAC address is newly generated. In FDAD, if the host receives the NA containing the new generated MAC, then it will immediately write it into the blacklist, then verify and discard it, and send feedback to the FDAD-S. Thus, the subsequent NA of the malicious host will be suppressed and cannot cause an effective attack.

(3) Host uses the FDAD-Request message to consume the controller resources.

Attack mode: the malicious host uses FDAD mechanism to send a large number of FDAD-Request messages to consume controller resources.

In FDAD, the controller does not allow a host which has not carried out DAD before to query other hosts' MAC information. For a host that has initiated DAD, MAC queries will not be allowed and only a host who responds the NS exists. The query is limited to specific entries in the T_{Query}. When an entry is queried, it will be marked up, that is, repeated queries are prohibited.

(4) Host uses FDAD-Feedback messages to attack other hosts.

Attack mode: the malicious host uses FDAD mechanism to send a large number of FDAD-Feedback messages to suppress normal host's communication.

The feedback is allowed only when Conditions 8–13 are satisfied because the T_{Feedback} is limited by T_{NS}, T_{NA}, and qT_{Query}, and even if the feedback is successful, the related entries in these three tables will be cleared, and all these means, for a specific NA, the host can feedback only once in the DAD process.

(5) C sends forged FDAD-Reply to other hosts.

Attack mode: in the process of DAD, if C receives a NS, it sends a forged NA and FDAD-Reply to the response, which contains forged Dpid and port information.

The switch does not forward the FDAD-Reply generated by the normal host. Thus, the FDAD-Reply does not arrive at host A but directly arrives at the controller and is discarded. Furthermore, the fake NA cannot pass the CGA verification and is fed back to the controller by host A. Therefore, this kind of attack is invalid. Subsequent attack message is suppressed; it cannot enter the network.

4.2.2. Storage and Communication Overhead Analysis

(1) Switch Storage Overhead. In Ethernet, the switch is capable of learning the MAC address. When a frame arrives,

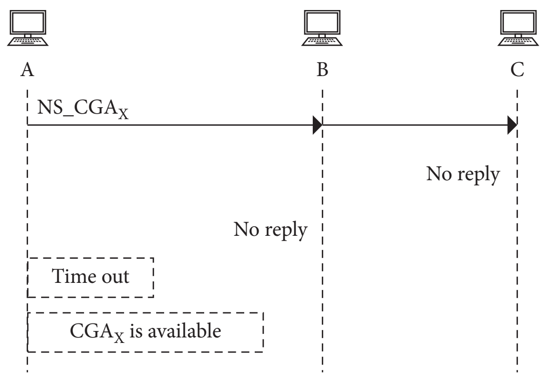

Figure 18: Normal DAD process with CGA.

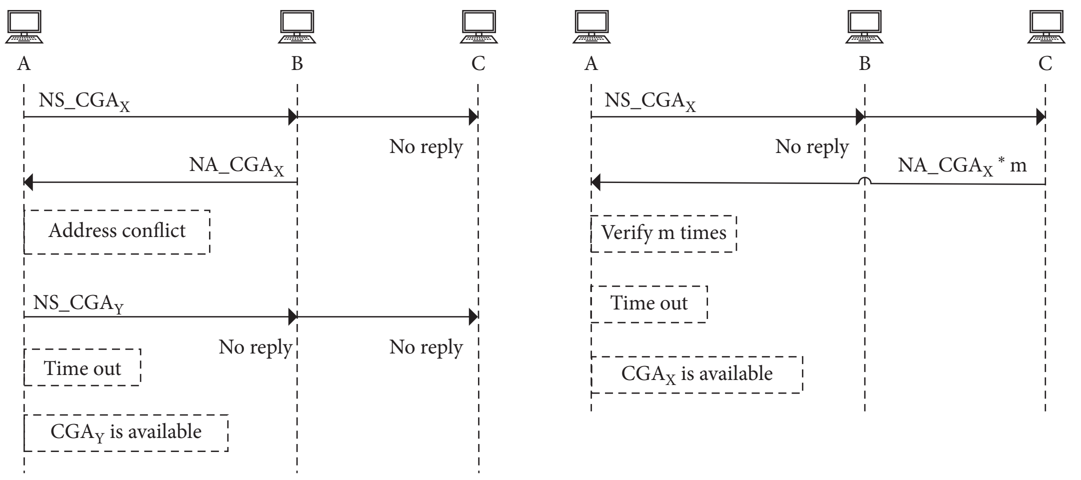

Figure 19: Address conflict and DoS suffering in SEND. (a) Address conflict. (b) Under DoS.

the switch learns the source MAC address and accesses the port of the frame. A forwarding entry is then generated in the cache; its format is like < mac, inport, ttl, type>. Then, the frame is forwarded in unicast or broadcast. In SDN, the frame forwarding mode of OpenFlow-enabled switch is similar to that of Ethernet switch. Therefore, in SDN, when the host uses a random MAC address for DoS, a large number of garbage flows are generated in the switch, occupying a large amount of storage space.

In FDAD, if a malicious node responds to NS with a random MAC address, since the forwarding entry is newly generated, the time of existence on the switch is less than 1 second. Therefore, the node is directly written in the blacklist of the host. After CGA verification, the host discards it and feeds back to the controller. Therefore, the subsequent NA of the malicious host is suppressed at its entry port by the controller. Even if the malicious host sends numerous frames with random MAC address, it does not form a DoS attack and generate additional flows on the switch.

(2) Communication Overhead. The three options undoubtedly increase the size of NDP packets and the communication overhead of DAD in some cases. The size of the original NDP message is 78 bytes, and the sizes of FDAD-Request, FDAD-Reply, and FDAD-Feedback are 90, 102, and 110 bytes, respectively. The size of NDP with CGA option is 238 bytes. Therefore, the new messages adopted by FDAD are larger than the original NDP, with increased ratios of 15.4%, 30.8%, and 41%, respectively, but still less than the messages of SEND. The increased overhead is mainly used to reduce the additional consumption caused by DoS. The communication overhead is analyzed in several typical scenarios.

(1) Normal DAD process (no address conflict):

The normal FDAD process is shown in Figure 18. Host A generates the address CGA$_X$ and then broadcasts the NS_CGA$_X$. No host answers because an address conflict does not exist. After the timeout, the DAD process is completed and CGA$_X$ is available. In this case, the communication overheads of SEND and FDAD are the same, that is, both have one NS broadcast. The specific number is determined by number n, which refers to the hosts in the LAN:

$$T_{\text{SEND\_Normal}} = T_{\text{FDAD\_Normal}} = \text{NS} \times n. \quad (1)$$

(2) Address conflict in SEND:

(i) When the address of B is also CGA$_X$, an address conflict exists. The DAD process is shown in

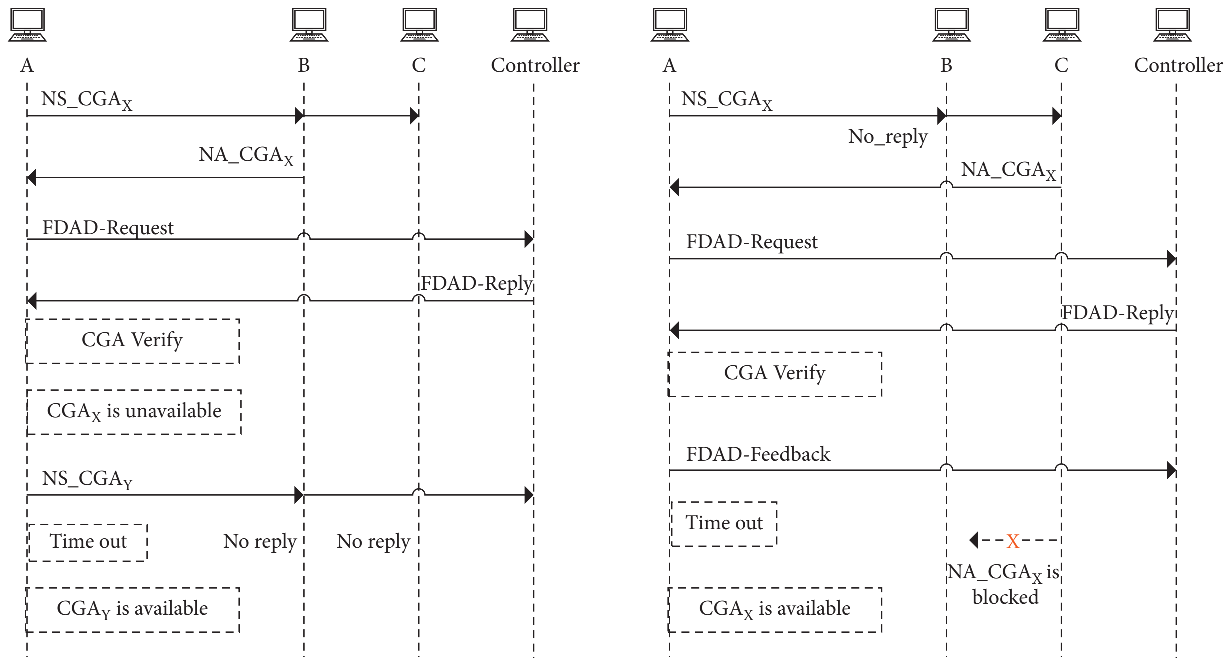

Figure 20: Address conflict and DoS suffering in FDAD. (a) Address conflict. (b) Under DoS.

Figure 19(a). A sends NS_CGA$_X$ in broadcast, B replies to NA_CGA$_X$ after detecting the address conflict, and then A regenerates a new address CGA$_Y$ and broadcasts NS_CGA$_Y$ again without conflict. The communication overhead of the whole process is as follows:

$$T_{\text{SEND\_Conflict}} = \text{NS} \times n + \text{NA} + \text{NS} \times n. \quad (2)$$

(3) DoS attack in SEND:

The DoS process in SEND is shown in Figure 19(b). In DAD, C sends m NA to reply and consume A's resources after receiving the NS_CGA$_X$ sent by A. The communication overhead is as follows:

$$T_{\text{SEND\_DoS}} = \text{NS} \times n + \text{NA} \times m. \quad (3)$$

(4) Address conflict in FDAD

The communication process is shown in Figure 20(a). In DAD, A sends the FDAD-Request message to the controller and receives the FDAD-Reply after receiving the NA_CGA$_X$ sent by B. Then, CGA parameters are verified for NA. After the address conflict is found, host A needs to generate a new address CGA$_Y$ and then broadcasts NS_CGA$_Y$. At this time, no address conflict exists, and the DAD process is completed. The communication overhead is as follows:

$$T_{\text{FDAD\_Conflict}} = \text{NS} \times n + \text{FDAD}_{\text{Request}} + \text{FDAD}_{\text{Reply}} + \text{NS} \times n. \quad (4)$$

(5) DoS attack in FDAD:

The communication process is shown in Figure 20(b). Host A receives the forged NA_CGA$_X$ sent by C after NS_CGAX broadcast. Then, host A sends FDAD-Request to the controller and receives FDAD-Reply message. Host A sends the FDAD-Feedback to the controller because the forged NA cannot pass the CGA verification, and then FDAD-S performs NDP message suppression on the port connected to C. Therefore, the subsequent NA_CGA$_X$ can no longer enter the network. The communication overhead is as follows:

$$\text{T}_{\text{FDAD}_{\text{DoS}}} = \text{NS} \times \text{n} + \text{NA} + \text{FDAD}_{\text{Request}} + \text{FDAD}_{\text{Reply}} + \text{FDAD}_{\text{Feedback}}. \quad (5)$$

The following is a quantitative analysis of the abovementioned situations.

(1) Normal DAD process (no address conflict):

$$T_{\text{SEND\_Normal}} = T_{\text{FDAD\_Normal}} = \text{NS} \times n = 238 \times n. \quad (6)$$

(2) Address conflict:

$$T_{\text{FDAD\_Conflict}} > T_{\text{SEND\_Conflict}}. \quad (7)$$

The difference between the two is

$$T_{\text{FDAD}_{\text{Conflict}}} - T_{\text{SEND}_{\text{Conflic}}} = \text{FDAD}_{\text{Request}} + \text{FDAD}_{\text{Reply}} = 200. \quad (8)$$

(3) Under DoS attack:

$$\begin{aligned}
T_{\mathrm{SEND}_{\mathrm{DoS}}} &= \mathrm{NS} \times n + \mathrm{NA} \times m = 238 \times (m+n),\\
T_{\mathrm{FDAD}_{\mathrm{DoS}}} &= \mathrm{NS} \times n + \mathrm{NA} + \mathrm{FDAD}_{\mathrm{Request}} + \mathrm{FDAD}_{\mathrm{Reply}}\\
&\quad + \mathrm{FDAD}_{\mathrm{Feedback}},\\
T_{\mathrm{FDAD}_{\mathrm{DoS}}} &= 238 \times n + 302.
\end{aligned} \tag{9}$$

For a LAN with n nodes, the total length of CGA address is 64 bits. If Sec bits (3 bits), u bit, and g bit are removed, the remaining 59 bits are random values. If each node generates k IPv6 addresses for a specific network prefix, then the probability of address conflict P is as follows:

$$\begin{aligned}
\mathrm{P} &= 1 - \prod_{i=1}^{n\times k}\left(1 - \frac{i}{2^{59}}\right),\\
\mathrm{P} &= 1 - e^{-n\times k\times(n\times k-1)/2^{60}}.
\end{aligned} \tag{10}$$

The size of a general LAN is less than $2^9$to prevent broadcast storm. For the same network prefix, the number of IPv6 addresses generated by each node does not exceed 2^5 (only one address is usually generated). When n is set to $2^9$and k to 2^5,

$$P = 1 - e^{-\left(1/2^{32}\right)}. \tag{11}$$

The probability of real address conflict is very small. Thus, the communication overheads of FDAD and SEND are almost the same.

Assuming h attack nodes exist in the LAN and each node sends m NA during the DAD process, the overhead comparison is as follows:

$$\begin{aligned}
T_{\mathrm{SEND\_DoS}} &= 238 \times n + h \times m \times 238,\\
T_{\mathrm{FDAD\_DoS}} &= 238 \times n + 302 \times h.
\end{aligned} \tag{12}$$

As long as $m > 1.3$, the expression $T_{\mathrm{SEND\_DoS}} > T_{\mathrm{FDAD\_DoS}}$ is satisfied. Therefore, the communication overhead of FDAD is much less than that of SEND when attacked by DoS.

4.3. Strengths and Limitations of FDAD. FDAD is the combination of the host's computing capability and SDN's control capability. Although network protocols can be implemented in hosts and network devices, in the traditional network, both of them detect attacks from their own point of view and adopt different methods. For example, distributed hosts have powerful computing power, but they are unable to prevent the generation and access of attack packets in DoS attack. The network device has the ability to prevent the packets from coming into the network but lacks the computing ability to verify the attack. FDAD combines the both abilities effectively, and its advantages are as follows:

(1) It protects the security of the host's DAD process for the host to generate IPv6 address smoothly

(2) It could suppress DoS attack from the source and reduce the generation of useless flow in SDN

(3) When suffering from DoS attack, the amount of LAN packets could be greatly reduced

FDAD also has some limitations:

(1) It adds three new NDP messages to the network, and the new message is larger than the original NDP message. Thus, in some cases, the communication overhead increases.

(2) It is not suitable for traditional Ethernet because its mechanism needs the support of a control layer. Otherwise, FDAD-Query, FDAD-Reply, and FDAD-Feedback messages could not play their roles.

(3) FDAD mechanism increases the memory and CPU cost of the controller, but this cost is greatly reduced when DoS attack occurs in the network.

5. Conclusion

DoS attacks are difficult to eliminate, largely because the attack hosts are concealed. In the DAD process, for the CGA parameters in NA, the host cannot identify its authenticity before verification is completed. However, even if the host identifies a false NA through verification, it is unable to prevent subsequent attacks from the same host because a malicious host can send a message using other false source address. In FDAD, the host can use the FDAD-Request to retrieve specific MAC's generation time and evaluate the authenticity of the NA in advance. For a false NA, the host can use FDAD-Feedback message to send the feature of the attack host to the control part. The controller can suppress the attack message from the source by dispatching a flow table, breaking the concealment of the attack host. Experiments show that FDAD greatly reduces the CPU and communication overhead of the host that suffering DoS. Of course, feedback mechanism causes the controller's CPU and memory consumption to increase slightly in s some scenarios, but its security cannot be matched by the traditional method.

Data Availability

All data are included in the manuscript.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] A. Tanenbaum and D. Wetherall, *Computer Networks: Pearson New International Edition*, Pearson Schweiz Ag, Zug, Switzerland, 2013.

[2] D. Plummer, "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48. Bit Ethernet Address for Transmission on Ethernet Hardware," 1982.

[3] S. Cheshire, "IPv4 address conflict detection," RFC 5227, 2008.

[4] T. Narten, "Neighbor discovery for IP version 6 (IPv6)," RFC 4861, 2007.

[5] J. Arkko and P. Nikander, "Limitations of IPsec policy mechanisms,"*Proceedings of the International Workshop on Security Protocols*, Springer, Cambridge, UK, April 2003.

[6] S. Ul. Rehman and S. Manickam, "Denial of service attack in IPv6 duplicate address detection process," *International Journal of Advanced Computer Science & Applications*, vol. 7, pp. 232–238, 2016.

[7] J. Arkko, J. Kempf, B. Zill, and P. Nordmark, "Secure neighbor discovery (SEND)," RFC 3971, 2005.

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

[9] T. Aura, "Cryptographically generated addresses (CGA),"*Proceedings of the 6th International Conference Information Security, ISC 2003*, Bristol, UK, October 2003.

[10] M. Bagnulo, "Hashased addresses (HBA) IETF," RFC 5535, 2009.

[11] A. Alsa'deh, H. Rafiee, and C. Meinel, "Stopping time condition for practical IPv6 cryptographically generated addresses," in *Proceedings of the 2012 International Conference on Information Networking (ICOIN)*, IEEE, Bangkok, Thailand, February 2012.

[12] H. Rafiee, A. Alsa'deh, and C. Meinel, "Multicore-based auto-scaling secure neighbor discovery for windows operating systems," in *Proceedings of the 2012 International Conference on Information Networking (ICOIN)*, IEEE, Bangkok, Thailand, February 2012.

[13] T. Cheneau, A. Boudguiga, and M. Laurent, "Significantly improved performances of the cryptographically generated addresses thanks to ECC and GPGPU," *Computers & Security*, vol. 29, no. 4, pp. 419–431, 2010.

[14] S. Qadir and M. Umar Siddiqi, "Cryptographically generated addresses (CGAs): a survey and an analysis of performance for use in mobile environment," *IJCSNS International Journal of Computer Science and Network Security*, vol. 11, no. 2, pp. 24–31, 2011.

[15] G. Su, W. Wang, X. Gong, X. Que, S. Jiang, and X. Gao, "A quick CGA generation method," in *Proceedings of the 2010 2nd International Conference on Future Computer and Communication (ICFCC)*, vol. 1, IEEE , Wuhan, China, May 2010.

[16] A. F. C. Alsa'deh and C. Meinel, "CS-CGA: compact and more secure CGA," in *Proceedings of the 2011 17th IEEE International Conference on Networks (ICON)*, IEEE, Singapore, December 2011.

[17] T. R. Reshmi and K. Murugan, "Light weight cryptographic address generation (LW-CGA) using system state entropy gathering for IPv6 based MANETs," *China Communications*, vol. 14, no. 9, pp. 114–126, 2017.

[18] S. Hogg and E. Vyncke, *IPv6 Security*, Pearson Education, London, UK, 2008.

[19] H. C. Liu and Q. G. Dai, "Design of security neighbor discovery protocol," in *Proceedings of the 2013 International Conference on Communication Systems and Network Technologies (CSNT)*, IEEE, Gwalior, India, 2013.

[20] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, FL, USA, 2005.

[21] Y. E. Gelogo, R. D. Caytiles, and B. Park, "Threats and security analysis for enhanced secure neighbor discovery protocol (SEND) of IPv6 NDP security," *International Journal of Control & Automation*, vol. 4, 2011.

[22] S. Guangjia and J. Zhenzhou, "Research on equivalence between address resolution and duplicate address detection," in *Proceedings of the Fifth International Conference on Instrumentation & Measurement IEEE*, Anaheim, CA, USA, September 2016.

[23] A. S. Ahmed, R. Hassan, and N. E. Othman, "IPv6 neighbor discovery protocol specifications, threats and countermeasures: a survey," *IEEE Access*, vol. 5, pp. 18187–18210, 2017.

[24] C. Elliott, "Geni—global environment for network innovations," in *Proceedings of the IEEE Conference on Local Computer Networks, 2008. LCN 2008*, p. 8, IEEE, Montreal, Canada, October 2008.

[25] N. Mckeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[26] M. Paliwal, D. Shrimankar, and O. Tembhurne, "Controllers in SDN: a review report," *IEEE Access*, vol. 6, pp. 36256–36270, 2018.

[27] Y. Lu, M. Wang, and P. Huang, "An SDN-based authentication mechanism for securing neighbor discovery protocol in IPv6," *Security and Communication Networks*, vol. 2017, Article ID 5838657, 9 pages, 2017.