NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A SECURITY STRATEGY FOR CYBER THREATS ON NEIGHBOR DISCOVERY IN 6LOWPAN NETWORKS

by

Cheng Hai Ang

December 2017

Thesis Advisor: Preetha Thulasiraman
Second Reader: George Dinolt

Approved for public release. Distribution is unlimited.

Reissued 27 Sep 2018 to reflect updated abstract on pages i and v.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE A SECURITY STRATEGY FOR CYBER THREATS ON NEIGHBOR DISCOVERY IN 6LOWPAN NETWORKS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Cheng Hai Ang			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Wireless sensor networks employ various technologies to facilitate low-power communications. One such technology is the Internet Protocol version 6 Low-Power Local Area Networks (6LoWPAN). One of the key concerns regarding 6LoWPAN networks is the vulnerability of its neighbor discovery (ND) protocol. In this thesis, we study the potential threat vectors against the ND protocol, focusing specifically on replay attacks that can cause denial of service. We propose a combination of hard and soft security approaches to mitigate cyber-attacks against the ND protocol. The hard security approach is based on a Trust-ND option, which includes a Timestamp, Nonce, and SHA-1 hashing function. The soft security approach leverages the social interactions between the nodes in the network to identify malicious nodes. We also propose a time-synchronization mechanism to synchronize the local clock of the nodes in the network. We demonstrate the effectiveness of the Nonce and Timestamp functions against replay attacks using the Contiki Operating System and Cooja network simulator. Via simulations, we also demonstrate the effectiveness of the time-synchronization mechanism. In addition, the data captured during the simulations is further analyzed using Wireshark.

14. SUBJECT TERMS IPv6 neighbor discovery, lightweight security mechanism, 6lowPAN			**15. NUMBER OF PAGES** 121
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

A SECURITY STRATEGY FOR CYBER THREATS ON NEIGHBOR DISCOVERY IN 6LOWPAN NETWORKS

Cheng Hai Ang
Civilian, Defence Science and Technology Agency (DSTA)
B.Eng., Nanyang Technological University of Singapore, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2017

Approved by: Preetha Thulasiraman, Ph.D.
Thesis Advisor

George Dinolt, Ph.D.
Second Reader

R. Clark Robertson, Ph.D.
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Wireless sensor networks employ various technologies to facilitate low-power communications. One such technology is the Internet Protocol version 6 Low-Power Local Area Networks (6LoWPAN). One of the key concerns regarding 6LoWPAN networks is the vulnerability of its neighbor discovery (ND) protocol. In this thesis, we study the potential threat vectors against the ND protocol, focusing specifically on replay attacks that can cause denial of service. We propose a combination of hard and soft security approaches to mitigate cyber-attacks against the ND protocol. The hard security approach is based on a Trust-ND option, which includes a Timestamp, Nonce, and SHA-1 hashing function. The soft security approach leverages the social interactions between the nodes in the network to identify malicious nodes. We also propose a time-synchronization mechanism to synchronize the local clock of the nodes in the network. We demonstrate the effectiveness of the Nonce and Timestamp functions against replay attacks using the Contiki Operating System and Cooja network simulator. Via simulations, we also demonstrate the effectiveness of the time-synchronization mechanism. In addition, the data captured during the simulations is further analyzed using Wireshark

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
A. 6LoWPAN NETWORK....1
B. SECURITY CONCERNS OVER 6LoWPAN NETWORKS....3
C. RESEARCH MOTIVATION AND OBJECTIVES....4
D. THESIS CONTRIBUTION....5
E. THESIS ORGANIZATION....5

II. BACKGROUND ON 6LoWPAN NEIGHBOR DISCOVERY....7
A. IPV6 ND PROTOCOL....7
1. Overview....7
2. ND Protocol Messages....7
3. ND Protocol Mechanism....9
B. ND THREATS AND VULNERABILITIES....10
C. CHAPTER SUMMARY....12

III. RELATED WORK....13
A. EXISTING SOLUTIONS FOR SECURING THE ND PROTOCOL....13
1. IP Security....13
2. Secure Neighbor Discovery (SEND)....13
3. SEND Vulnerabilities....15
4. Further Improvements on SEND....15
B. CHAPTER SUMMARY....17

IV. PROPOSED SECURITY MECHANISM FOR ND PROTOCOL....19
A. DESIGN CONSIDERATIONS....19
B. PROPOSED SECURITY MECHANISM....19
1. Hard Security Implementation....21
2. Soft Security Implementation....25
C. CHAPTER SUMMARY....28

V. EXPERIMENTAL SETUP....29
A. AIM OF EXPERIMENT....29
B. NETWORK SIMULATION SETUP....29
1. Contiki OS and Cooja Network Simulator....29
2. Simulation Environment....29
C. ATTACK SCENARIOS....31

1. Nonce Option against Replay Attacks....31
2. Timestamp Option against Replay Attacks....32
D. MODIFICATIONS TO EXISTING CONTIKI OS CODE....32
E. CHAPTER SUMMARY....33

VI. SIMULATION RESULTS AND ANALYSIS....35
A. ATTACK SCENARIO WITHOUT TRUST-ND OPTION....35
B. ATTACK SCENARIO WITH NONCE OPTION....38
C. ATTACK SCENARIO WITH TIMESTAMP OPTION....42
D. TIMESTAMP OPTION WITH DE-SYNCHRONIZED CLOCKS....44
E. TIMESTAMP OPTION WITH TIME-SYNCHRONIZATION MECHANISM....45
F. CHAPTER SUMMARY....47

VII. CONCLUSION AND FUTURE WORK....49
A. SUMMARY AND CONCLUSIONS....49
B. FUTURE WORK....49
1. Evaluation of the Entire Hard Security Mechanism....50
2. Evaluation of the Soft Security Mechanism....50
3. Scalability of the Security Mechanism....50

APPENDIX. SOURCE CODE....51

LIST OF REFERENCES....99

INITIAL DISTRIBUTION LIST....103

LIST OF FIGURES

Figure 1. Typical Network Architecture between a 6LoWPAN and IPv6 Network. Adapted from [4].....2

Figure 2. Comparison of the 6LoWPAN Protocol Stack and the TCP/IP Protocol Stack.....3

Figure 3. ND Protocol Message Format. Adapted from [14].....8

Figure 4. Router Discovery Mechanism.....9

Figure 5. Structure of Trust-ND Option in an IPv6 Packet. Adapted from [17].....21

Figure 6. Effects of Clock Time Desynchronization in a Network.....23

Figure 7. Replay Attack Mitigated through the Use of a Nonce.....24

Figure 8. Logic Flow for the Soft Security Mechanism.....27

Figure 9. Relative Position of the Nodes in the Simulated Network.....31

Figure 10. Captured Events in the Network without Trust-ND Option (Part One).....36

Figure 11. Captured Events in the Network without Trust-ND Option (Part Two).....37

Figure 12. Correlation between Nodes' Output and 6LoWPAN Network Analyzer (without Nonce Option).....38

Figure 13. Captured Events in the Network with Nonce Option (Part One).....39

Figure 14. Captured Events in the Network with Nonce Option (Part Two).....39

Figure 15. Correlation between Nodes' Output and 6LoWPAN Network Analyzer (with Nonce Option).....40

Figure 16. Trust-ND Option with Nonce Data Breakdown Using Wireshark.....41

Figure 17. Captured Events in the Network with Timestamp Option (Part One).....43

Figure 18. Captured Events in the Network with Timestamp Option (Part Two).....43

Figure 19. Timestamp Option with De-synchronized Clocks Simulation Data.....45

Figure 20. Timestamp Option with Synchronized Clocks Simulation Data.....46

Figure 21. Trust-ND Option with Timestamp Data Breakdown Using Wireshark.....47

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Summary of Attack Scenarios on ND Message Types 11

Table 2. Summary of Outputs Associated with Different Trust Values for the Soft Security Mechanism 27

Table 3. Simulation Parameters 30

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

6LoWPAN	Internet Protocol version 6 Low Power Personal Area Network
ADD	Authorisation Delegation Discovery
AH	Authentication Header
CGA	Cryptographically Generated Address
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
ECC	Elliptic Curve Cryptography
ESP	Encapsulating Security Payload
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IPSec	IP Security
MD5	Message Digest 5
MTU	Maximum Transmission Unit
NA	Neighbor Advertisement
ND	Neighbor Discovery
NS	Neighbor Solicitation
NUD	Neighbor Unreachability Detection
RA	Router Advertisement
RS	Router Solicitation
RSA	Rivest Shamir Adleman
SA	Security Association
SEND	Secure Neighbor Discovery
SHA-1	Secure Hash Algorithim-1
SLAAC	Stateless Address Autoconfiguration
TCP	Transport Control Protocol
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
WSN	Wireless Sensor Networks

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank my wife, Ivy, for her support and care during my year of study at NPS. I would also like to thank my son, Yu Zi, for being a sensible boy and offering his help in our daily matters. I would like to express my deep gratitude to my thesis advisor, Professor Preetha Thulasiraman, for providing me the guidance and knowledge needed to write this thesis. She also gave me the motivation and direction to complete the work on time. Without her influence, I would not have been able to finish the thesis in less than six months. I would like to thank Dr. George Dinolt for giving me his guidance as well. Last but not least, I would like to thank the Defence Science and Technology Agency (DSTA) for giving me this opportunity to pursue my postgraduate studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The use of Wireless Sensor Networks (WSN) is growing rapidly in today's military operations, where information dominance is critical for commanders to have better situational awareness. This allows them to make quicker and better decisions in time-critical warfare. Large numbers of remote sensors can be deployed in an area of operations to provide persistent surveillance. Information gathered by the senor node is shared with the command and control center via a secure wireless network. Sensors are usually small in size to avoid detection. They can be self-maintained and self-organized with no need for human intervention or a pre-existing infrastructure. These sensors are usually inexpensive, and thus, they have limited resources in memory, communication range, computational power, and energy storage. Such constraints are the key drivers that require WSNs to have effective communication and security mechanisms. The Institute of Electrical and Electronics Engineers (IEEE) and the Internet Engineering Task Force (IETF) have developed several communication standards to address the constraints in WSNs. These standards are discussed in the following sections.

A. 6LoWPAN NETWORK

The protocols that we use for the Internet are not suitable for WSN applications, as WSNs have varying traffic statistics, dynamic topologies, and limited payload sizes [1]. Moreover, the devices in the WSN are low-powered, with limited computational power and memory. This led to the introduction of the IEEE 802.15.4 standard [2] which defines the operation of low rate wireless personal area networks at the physical and medium access control (MAC) layers. To ensure that each device is Internet Protocol version 6 (IPv6)-addressable, the IETF built upon the IEEE 802.15.4 standard by introducing an IPv6 adaptation layer above the IEEE 802.15.4 MAC sublayer. This adaptation layer is also known as the IPv6 Low Power Personal Area Network (6LoWPAN) adaptation layer [3].

The 6LoWPAN adaptation layer leverages the existing IPv6 infrastructure (i.e., the IPv6 address space and the IPv6 Neighbor Discovery (ND) protocol) while allowing a resource constrained network protocol to communicate with the IPv6 Internet [3].

The 6LoWPAN adaptation layer provides three key functions which include "packet fragmentation and reassembly, header compression, and data link layer routing" [1]. A typical network architecture connecting the IPv6 Internet and a 6LoWPAN network is shown in Figure 1.

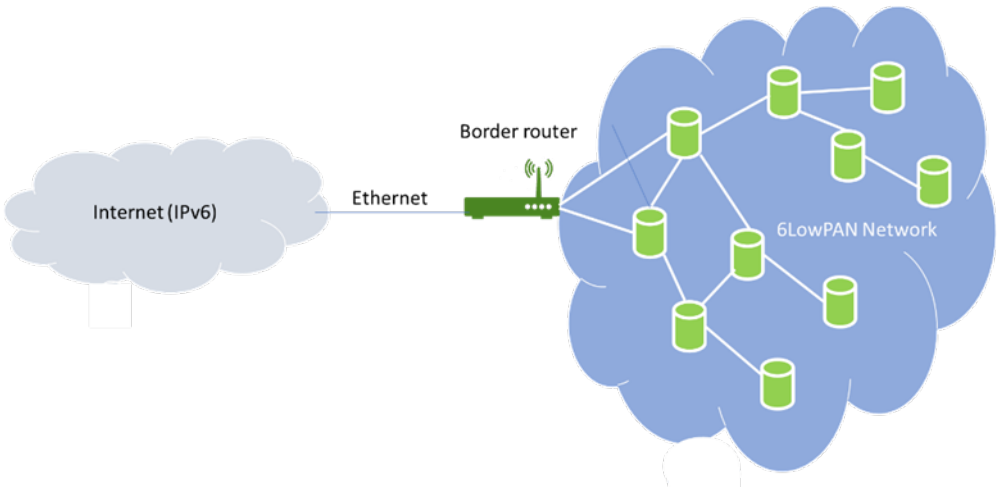


Figure 1. Typical Network Architecture between a 6LoWPAN and IPv6 Network. Adapted from [4].

The differences between the Transport Control Protocol (TCP)/IP protocol stack and the 6LoWPAN protocol stack are shown in Figure 2. In a typical network model, IP is the only protocol that is linked directly to the transport and data link layers. The 6LoWPAN network integrates both IPv6 and the 6LoWPAN adaptation layer to facilitate the transition between a resource constrained network and the typical IPv6 Internet.

To achieve seamless data transition, one of the key functions of 6LoWPAN at the border router is to adapt to the large difference in packet size of 6LoWPAN and IPv6 networks by exercising packet fragmentation at the IPv6 layer and, subsequently, executing packet reassembly at the IEEE 802.15.4 MAC layer. An IPv6 packet has a minimum size of 1280 bytes, while 6LoWPAN has only a size of 127 bytes [5]. For a

6LoWPAN network, the transport layer usually does not use the TCP because TCP is more complex and incurs more overhead as compared to the User Datagram Protocol (UDP) transport protocol [6]. The Internet Control Message Protocol (ICMP) is common in both a typical IP network and in 6LoWPAN networks. It is mainly used for sending error and informational messages between nodes in the network such as ND messages [7].

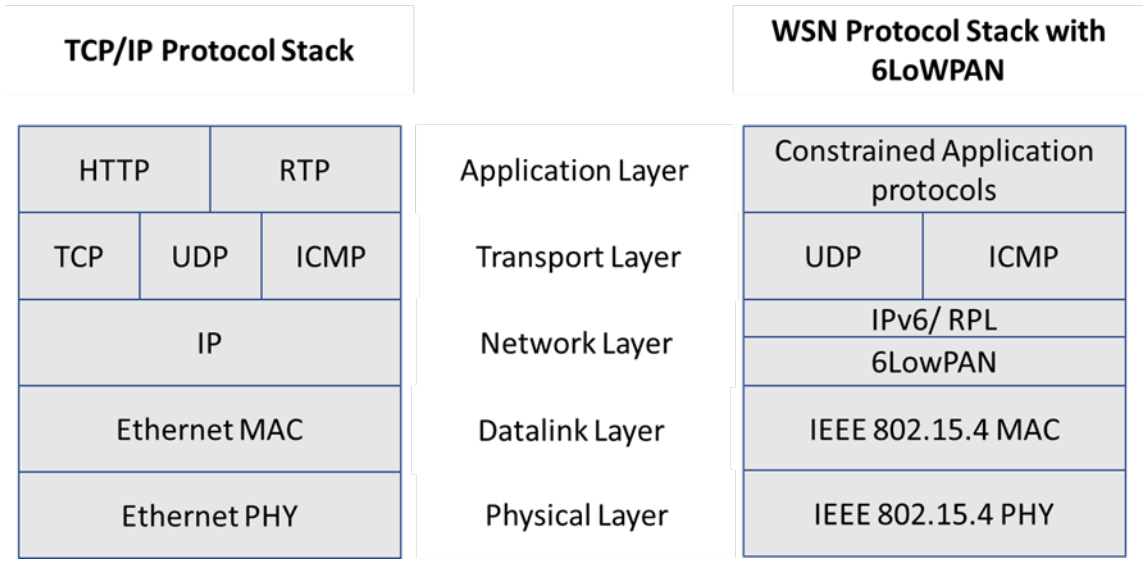


Figure 2. Comparison of the 6LoWPAN Protocol Stack and the TCP/IP Protocol Stack

There are several 6LoWPAN standards offered by the IETF that serve as important references for researchers studying WSNs, namely RFC 4944 (Transmission of IPv6 Packets over IEEE 802.15.4 Networks) [3], RFC 6282 (Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks) [5], and RFC 6775 (Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks) [8].

B. SECURITY CONCERNS OVER 6LoWPAN NETWORKS

The limitations in resources such as memory, computation power, and energy storage impose security challenges on 6LoWPAN networks. The typical security solutions tailored for the Internet are not suitable for 6LoWPAN networks mainly due to

high complexity and high computational power. With limited support from existing security services, the 6LoWPAN network is vulnerable to cyber-attacks at all layers of the protocol stack. Moreover, these attacks can be executed by both internal and external adversaries.

A robust and efficient security mechanism is desired to enable the 6LoWPAN network to mitigate different cyber-attacks; however, a robust security mechanism is difficult to achieve, as this usually requires highly complex software and computational power which then leads to a system with higher latency and low efficiency. Implementation of security protocols create a trade-off between robustness and efficiency. The challenge is to find the right balance, which usually varies depending on the relative importance of security and performance requirements of the network.

The authors of [4] studied the potential security threats and vulnerabilities of 6LoWPAN networks. They catalogued suitable security mechanisms to counter specific security weaknesses in 6LoWPAN. According to [4], one of the key security concerns for the 6LoWPAN network is the potential cyber-attack on the IPv6 ND Protocol. The ND protocol is used by nodes to join a new network and to establish communication between nodes and routers. Without an appropriate security mechanism to protect the ND protocol, cyber-attackers can disguise themselves as legitimate hosts or routers and cause denial of service (DoS) to the nodes in the network or redirect ND messages to the wrong destinations.

C. RESEARCH MOTIVATION AND OBJECTIVES

The original IPv6 ND specification recommended the use of IP Security (IPSec) to protect the ND messages; however, due to the impracticality of having to manually configure security associations between all the sensor nodes, using IPSec is infeasible. Threat models on IPv6 ND have been discussed in RFC 3756 [9]. This RFC defines the requirements for a Secure Neighbor Discovery (SEND) protocol without using IPSec. The SEND protocol was developed to counter potential cyber vulnerabilities in RFC 3971 [10]; however, we assessed the SEND protocol to be computational intensive due to its use of a Cryptographically Generated Address (CGA) and digital signatures. This

approach may not be suitable for a 6LoWPAN network as it has limited resources in terms of memory, computational power, and energy storage.

In this thesis, we study the security mitigating measures for the ND protocol based on existing literature and defines an appropriate security mechanism that is efficient and effective in protecting the ND protocol against cyber-attack. Effectiveness of the security measure is evaluated through simulation.

D. THESIS CONTRIBUTION

To meet the previously stated objectives, we study and compare several existing security solutions for the ND protocol. We then define a set of design considerations for the security mechanism of ND suitable for a 6LoWPAN network. We propose a combination of hard and soft security mechanisms for the ND protocol to provide a balance in terms of network performance and security protection. Hard security mechanisms are functions that prevent attackers from entering the network, while soft security can tolerate attackers entering the network but use social interactions with the other nodes in the network to identify the malicious nodes and avoid communicating with them. The hard security mechanism uses Secure Hash Algorithim-1 (SHA-1), Timestamp, and the Nonce function, while the soft security mechanism uses a centralised trust management scheme to identify the malicious nodes in the network. We evaluate the effectiveness of the proposed security mechanism by simulating attack scenarios against the key security functions using the Contiki operating system (OS) and Cooja network simulator.

E. THESIS ORGANIZATION

The remainder of this thesis is organized as follows. In Chapter II, we cover the relevant background information of the ND protocol of both IPv6 and 6LoWPAN and their associated security issues. In Chapter III, we discuss the related work that has been done to address the security concerns in the 6LoWPAN ND protocol. In Chapter IV, we discuss the proposed neighbor discovery security algorithm for the 6LoWPAN network. In Chapter V, we describe our simulation environment for the attack scenarios. In

Chapter VI, we present and discuss the results of the simulations. In Chapter VII, we conclude the thesis and discuss directions for future work. The code developed for the network simulation is included in the Appendix.

II. BACKGROUND ON 6LoWPAN NEIGHBOR DISCOVERY

The overview of the ND process in IPv6 and 6LoWPAN networks is described in the following sections. We discuss the various ICMP messages exchanged between hosts and routers during the ND process in addition to the actual ND mechanism. We also highlight the vulnerabilities of the ND process.

A. IPV6 ND PROTOCOL

1. Overview

The IPv6 ND protocol is one of the main building blocks in IPv6 wireless systems. Its critical functions include discovering neighbor nodes on the same link, detecting if a neighbor node is reachable, detecting duplicate IP addresses, determining the link-layer addresses, and finding routers [11]. Without the ND protocol, the network is not able to function.

The ND protocol serves an important role in mobile IPv6 networks by removing the need for third party devices such as a Dynamic Host Configuration Protocol (DHCP) server. DHCP is not required as new nodes are able to self-configure their own IPv6 addresses [12]. Another protocol named ND proxy is only applicable for mobile nodes. Mobile nodes may not be reachable from nodes on the home network. The ND proxy protocol allows the home agent to act as a proxy to the off-link mobile node and perform the ND operations on the node's behalf. The ND proxy protocol has its own security challenges [13] which are not covered in this thesis. In this thesis, we assume that the deployment of the mobile nodes does not require proxies because the nodes are expected to stay within the range of a single 6lowPAN network.

2. ND Protocol Messages

The ND protocol messages are formatted as ICMP messages. The message is encapsulated by IPv6 and IEEE 802.15.4 headers. The ND protocol message format is shown in Figure 3. Five key ICMP version 6 (ICMPv6) message types are used in the ND protocol to facilitate ND operations: Router Solicitation (RS), Router Advertisement

(RA), Neighbor Solicitation (NS), Neighbor Advertisement (NA), and Redirect Message (RM).

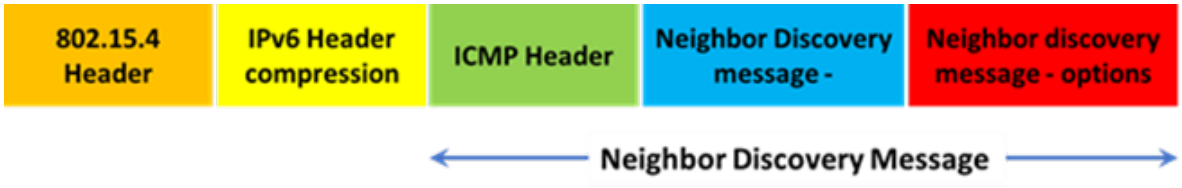


Figure 3. ND Protocol Message Format. Adapted from [14].

The multicast RS message is generated by a new host to find routers on the network it is physically connected to and to gain information about the network. Information learned by a router includes the prefix and IP address of the host [11].

IPv6 Routers send periodic RA messages to advertise their presence on the link as well as to communicate router parameters. The router parameters include link prefixes, a link Maximum Transmission Unit (MTU), and a hop limit. An RA message is also used as a response to an RS message [11].

A multi-cast NS message is sent by a host to obtain the link-layer addresses of the neighbor nodes. It is also used to check if the neighbor node is still within reachable range [11]. A multi-cast message is sent from a source to a group of hosts that subscribe to the specific multicast address.

A unicast NA message is mainly used to respond to a NS message. A unicast message is sent from a host to a specific destination. Unsolicited NA messages (can be unicast or multicast) can also be sent if there are changes within the node, (i.e., if there is a change in the link-layer address, the host sends an unsolicited NA message to inform the network [11]).

The redirect message is used by a router to inform the neighbor node to improve their route to reach a certain destination node [11].

3. ND Protocol Mechanism

According to [11], there are several key IPv6 ND protocol mechanisms that allow the interaction between nodes that are attached to the same link. These key mechanisms are described in the following subsections.

a. Router Discovery

Based on [11], the Router discovery mechanism is required for the host to a) locate neighboring routers, b) learn the IP prefix address ranges that reside on the same link and c) discover the capabilities of the link and router. This function is executed by the exchanges of RS and RA messages between the host and router. The host sends a multicast RS message to find routers in the network. When the router receives the RS message, it responds to the host with a multicast RA message containing a set of prefix information that belongs to the on-link IP addresses. Also, link parameters such as hop limit and link MTU are also sent in the RA message. The router discovery mechanism is illustrated in Figure 4. In addition to solicited RA messages, the router also sends periodic RA messages to neighbor hosts [11].

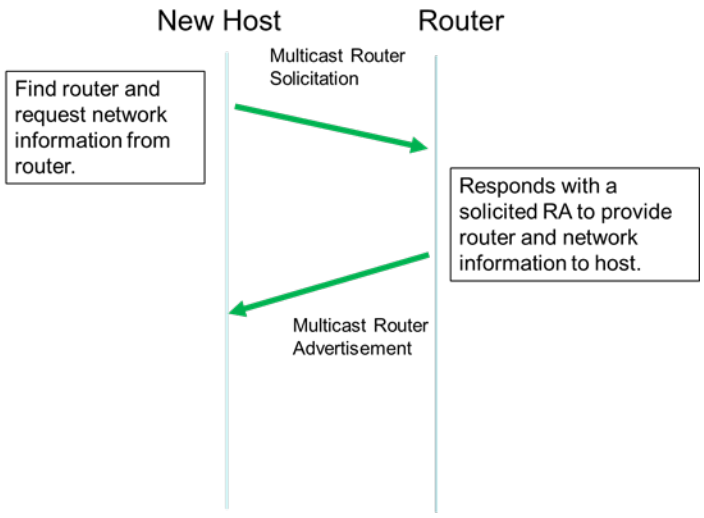


Figure 4. Router Discovery Mechanism

b. Address Resolution Mechanism

The authors of [11] also introduce the address resolution mechanism. When a node needs to send a unicast packet to its neighbor, it needs to have both its neighbor's IP and link-layer addresses. If the node has only its neighbor's IP address, it needs to perform address resolution to obtain the neighbor's link-layer address. To achieve this, the node sends an NS message to the neighbor to request its link-layer address. The neighbor then responds with a unicast NA message which includes its link-layer address [11].

c. Neighbor Unreachability Detection (NUD)

The neighbour unreachability detection (NUD) function is used to determine whether a node's neighbour is reachable [11]. The node sends a unicast NS message to its neighbour, and if the neighbor receives the NS message, it responds with a unicast NA message. This confirms that its neighbor is reachable. The reachability status of a neighboring node is then updated in the neighbor cache [11]. The neighbor cache is a table of information that consists of all the neighbors' IP addresses with their associated link-layer addresses and the status of the neighbor's reachability state [14].

d. Redirect

The router sends a redirect message to inform originating hosts that there is a better first-hop neighbor where packets can be forwarded to a specific destination. When the originating host receives the Redirect message, it sends subsequent packets addressed to the same destination via the better route determined by the router [11].

B. ND THREATS AND VULNERABILITIES

The authors of [9] surveyed various threats and vulnerabilities to the ND protocol with the aim to develop a more secure ND operation. All the ND protocol mechanisms described in Section A are subject to cyber-attacks, including DoS, redirect attacks, and replay attacks. These attacks can be easily executed during the ND phase by forging any of the ND messages including RS, RA, NS, NA and Redirect messages; thus, it is important to implement a security mechanism that can protect the ND messages, detect

cyber-attacks, and identify malicious nodes. A summary of the cyber-attacks against the ND protocol is shown in Table 1.

Table 1. Summary of Attack Scenarios on ND Message Types

	ND Message Types	ND Mechanism	Attack Scenarios
1	Router Solicitation (RS)	Router Discovery	Denial of Service Redirect attack Replay attack
2	Router Advertisement (RA)	Router Discovery	Denial of Service Redirect attack Replay attack
3	Neighbor Solicitation (NS)	Duplicate Address Detection NUD Address resolution	Denial of Service Redirect attack Replay attack
4	Neighbor Advertisement (NA)	Duplicate Address Detection NUD Address resolution	Denial of Service Redirect attack Replay attack
5	Redirect	Redirect	Replay attack Redirect attack

In [9], the potential threats to the router discovery mechanism are discussed. RA messages are important for nodes to learn about the on-link prefix address and the network configuration parameters including MTU, router lifetime, router link-layer address, etc. Based on [7], there is a checksum in the ICMPv6 header to ensure the integrity of the entire message; however, the attacker can simply alter the details of the message to keep the same checksum code. An attacker can modify one or more fields in the RA messages to cause a DoS attack to the hosts who received and processed the RA messages. With the fake prefix address, the hosts will configure an incorrect source address and end up being suspected as an attacker as it does not belong to the subnet. With a fake link MTU that is larger than the actual link MTU, the victim hosts are unable to send packets out to the link.

The attacker can set the router lifetime to zero, and the victim hosts will not regard the legitimate router as a default router and would then need to find an alternative router. The hosts may end up not being able to send packets out of its subnet as there are

no other routers available. At this point the host has been deceived by the attacker into believing that the actual router is disabled. This allows the attacker to masquerade as a legitimate router which the host uses to send data. The attacker can then redirect the host's packets to other destinations [9].

It is also discussed in [9] that the router discovery mechanism is subject to replay attacks. An attacker can duplicate the RA messages and replay them later to the host. With no security mechanism to detect replay messages, the host accepts the replay messages with outdated information which may cause confusion between the host and the network. The attacker can also send multiple replay messages to overwhelm the victim, and as a result, the victim is denied service. In this attack, the attacker does not need to modify the messages [9]; therefore, securing the integrity of the ND messages is insufficient. Additional security measures are needed to detect and prevent replay attacks.

C. CHAPTER SUMMARY

In this chapter, the basic messaging mechanism of the ND protocol in IPv6 and 6LoWPAN were discussed. The threats and vulnerabilities of ND were also covered. Security mitigation techniques against the above threats and vulnerabilities are discussed in the next chapter.

III. RELATED WORK

In literature, various solutions to secure the IPv6 ND protocol have been proposed. In this chapter, we discuss existing security mechanisms for ND, focusing on those approaches which our proposed solution is based.

A. EXISTING SOLUTIONS FOR SECURING THE ND PROTOCOL

1. IP Security

The IPSec protocol [15] was initially introduced to provide security protection for the ND protocol via the IP layer. IPSec consists of a set of services and protocols that offer data integrity, confidentiality, and protection against certain types of cyber security attacks such as replay attacks.

There are two key security mechanisms in IPSec, namely the Authentication Header (AH) and the Encapsulating Security Payload (ESP). The AH is used to provides authentication for all or parts of the packet, while ESP provides encryption on the messages. Both AH and ESP use common hash algorithms such as SHA-1 and Message Digest 5 (MD5). Prior to the use of AH and ESP, the two devices need to manually set up bidirectional Security Associations (SA) so that they can exchange information securely. As a result, the effort to setup an SA configuration can become tedious and almost impractical when a large network is being used. IPSec is not beneficial for ND operations where nodes need to auto configure themselves to connect to the network [10]. In addition, the authors of [16] assessed that IPSec consumes a large amount of resources that leads to poor network performance and high network administration cost.

2. Secure Neighbor Discovery (SEND)

Due to the above-mentioned limitations in the IPSec protocol, the IETF developed the SEND protocol [10] to secure the various functions in the ND protocol. The security measures of SEND include: a) address ownership proof mechanism, b) message integrity function, and c) discovery of trusted routers. These measures are either in the form of ND message options or new ICMPv6 messages. Four ND options are introduced in SEND,

namely Cryptographically Generated Address (CGA), RSA signature, Timestamp, and Nonce. Two new ICMPv6 messages between hosts and router including Certification Path Solicitation (CPS) and Certification Path Advertisement (CPA) are implemented. The details of the security implementation are discussed in the following paragraphs..

In [10], it states that the CGA option is used to ensure that the sender of a ND message is the legitimate owner of the source address stated in the message. All the nodes' registered IP addresses are combined with a public key to form a CGA IPv6 address. The public key and its associated parameters are in turn hashed (using SHA-1) in the ND message option so that no one can modify it. The receiver can then re-calculate the hash value to confirm that the sender is the actual owner of the claimed address. The key benefit of CGAs is that they are independent and do not need additional infrastructure or a third party to form IPv6 addresses. The drawback of CGA is its complexity and high computational cost that may lead to CPU exhaustion attacks [17].

The Rivest Shamir Adleman (RSA) signature is another option to ensure that the sender's identity is genuine [10]. The sender uses its own private key to generate the signature which in turns mitigates against stealing of CGA addresses. The generation and verification of the RSA signature are also assessed to be computationally expensive [9].

A Nonce option can prevent replay attacks in the solicited messages such as NS/NA and RS/RA messages [10]. The Nonce option generates a unique random number in the sender's solicitation message. The recipient responds with an advertisement message with the same unique random number. As such, when a node receives the advertisement message, it is able to detect if the message is a fresh response to its last solicitation message.

In [10], the Timestamp option was proposed to mitigate unsolicited advertisements against replay attacks. For the Timestamp option to be effective, all the nodes must have synchronized clocks. According to [10], the "Timestamp is based on a 64-bit unsigned integer that represent the number of seconds since January 1, 1970, 00:00 UTC. The first 48 bits of the field indicate the number of seconds while the remaining 16 bits represent the number of 1/64K fractions of a seconds".

Router authorization is another security feature of [10]. When a new host wants to join a network, it does not know who to trust. A rogue router can send fake RA messages to the new host and make it believe that the rogue router is a legitimate router. To prevent this, [10] suggest the deployment of an Authorisation Delegation Discovery (ADD) to allow hosts to validate if the router is authorised by a trust anchor.

3. SEND Vulnerabilities

While SEND provides security protection on the threats highlighted in Chapter II, Section B, the authors of [18] assessed that its RSA and CGA mechanisms consume excessive computational resources and bandwidth. In addition, SEND also introduces some new vulnerabilities that make the network susceptible to attacks.

a. CGA Vulnerability

In [18], it states that while the CGA mechanism can prevent address spoofing, it cannot be certain that the node with the CGA address is a legitimate node. Attackers can use their own public keys to create new CGA addresses.

b. DoS Attacks on Router Authorization

The router authorization mechanisms may be subject to a DoS attack by sending a number of unnecessary CPA messages to the target host and forcing it to spend its resources to process and verify the forged certification path [18].

4. Further Improvements on SEND

As mentioned previously, while the CGA option has improved security on the ND protocol, it created a few loopholes for other potential threats to enter the network. In [19], threats on CGA were highlighted, namely global time-memory trade-off attacks and the lack of authentication in the CGA verification process. They have developed an enhancement of CGA option, named CGA++, to mitigate these identified potential threats; however, these enhancements add more complexity and make the CGA algorithm less efficient. In the context of 6lowPAN devices, which have constrained power and computation limits, these CGA enhancements or even CGA itself may not be suitable.

Another solution was proposed in [20] to improve the security of the ND protocol by designing a central detection mechanism to detect NS and NA spoofing. The mechanism included a) setting up and monitoring of several data logging tables related to the NS/NA messages and b) introducing probe requests in the form of an NS message format. One of the key logging tables records all the authenticated IP-MAC binding addresses based on previous interactions between the nodes and the detection mechanism. While spoofing is detected, this mechanism has no means to know which message is spoofed.

In [21], SEND was improved by replacing the CGA algorithm with a more efficient solution to randomise the nodes' IP address. The solution included the integration of a randomly generated IP address with a signature to the ND messages. The random generator used an Elliptic Curve Cryptography (ECC) algorithm which is able to generate keys within a short time and to randomize the IP address using less steps as compared to the CGA algorithm. Based on experimental results, it was shown that while there is an improvement in the computation time by using the ECC algorithm as compared to CGA algorithm, the improvement dropped significantly when the security level parameter used in the algorithm is raised.

A distributed trust-based mechanism, Trust-ND, was developed to secure the ND protocol in [17]. To reduce computational resources and complexity, the mechanism removed both the CGA and RSA options proposed by SEND. A hybrid approach consisting of hard and soft security measures was implemented. For hard security, three techniques were included: a) unkeyed hash function using SHA-1, b) Timestamp, and c) Nonce.

SHA-1 was used to ensure message integrity, while the Timestamp and the Nonce were used to ensure timeliness and uniqueness of every ND message. For this mechanism, a trust option was generated on top of the ND messages, and both the Timestamp and Nonce information were stored in the option. This single option reduced hundreds of bytes from the SEND options and, therefore, aided in reducing the bandwidth consumption while still possessing the security protection provided by both Timestamp and Nonce.

For soft security, the concept is to accept that there will be malicious nodes in the network regardless of how strong the hard security is, which may include encryption, certification, and digital signatures. The aim of soft security is to identify which are the malicious nodes in the network and prevent them from causing damage to the network by sharing information with the nodes in the network. Trust-ND is based on a distributed trust management scheme using a probabilistic trust model to identify and determine if the nodes are trustworthy. The experiments in [17] showed that Trust-ND is more efficient than SEND and, therefore, it is assessed that it could be used for a 6LoWPAN network. Due to sleeping nodes in 6LoWPAN, we believe a centralised trust mechanism, instead of distributed approach, may be more effective.

B. CHAPTER SUMMARY

In this chapter, various solutions from existing researchers to improve the security for the ND process for 6LoWPAN networks were discussed. From the above solutions, it is understood that one cannot develop a perfect solution that can keep the 6LoWPAN network safe from all security attacks on ND. The solutions usually come with trade-offs and there are always loopholes in the network which the attacker can exploit to disrupt the network communications. More security measures may improve the situation but may also lead to poorer network performance (e.g., higher power consumption, higher bandwidth, higher computation complexity, higher latency, and lower throughput) or create new gaps for an attacker to disrupt the network; thus, we need to find the right balance between security protection and network performance.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PROPOSED SECURITY MECHANISM FOR ND PROTOCOL

As was discussed in Chapter III, there are many solutions that have been proposed to secure the ND protocol. Each solution has its own advantages and disadvantages. In this chapter, we discuss the components of our security mechanism for the ND protocol using hard and soft security. First, we discuss our design considerations for the developed ND security mechanism. Secondly, we adapt and modify existing solutions in the literature to meet our design considerations.

A. DESIGN CONSIDERATIONS

Key design considerations identified for designing a suitable security mechanism for the 6LoWPAN ND protocol are as follows:

- It should leverage the existing ND protocol and minimize changes to the ND message format and operation.
- It should not require regular software upgrades or installation of software in all systems in the network.
- It should require low power consumption, computational power, and latency.
- It should not require an external infrastructure or a third party to support the network.
- It should be able to secure the ND protocol against cyber-threats identified in Chapter II. Specifically, in this thesis, we focus on replay attacks which can cause DoS.

B. PROPOSED SECURITY MECHANISM

The proposed security mechanism for the ND protocol of 6LoWPAN networks is adapted from [17] which adopts both hard and soft security measures. This security approach is more efficient when compared to SEND as it removes the use of complex

security algorithms such as CGA and RSA and relies only on a simpler SHA-1 algorithm for data integrity. SHA-1 requires smaller key size, less computational power, and less processing time as compared to RSA [22]. While SHA-1 provides weaker security protection as compared to CGA and RSA, it provides a balance in terms of network performance and security protection. The addition of further hard security protections will only overload the network. The inclusion of a soft security mechanism (as proposed by [17]) offers an alternative solution in securing the network. Instead of trying to protect the network by preventing attackers from entering, the soft security mechanism can tolerate attackers entering the network by using social interactions with the other nodes in the network to identify the malicious nodes; thereby, the node avoids communicating with malicious devices. As a result, the attackers are isolated and their influence in the network is reduced.

This solution leverages the existing ICMPv6 ND messages by adding a Trust-ND option on top of these messages [17], and no changes are made to either the ND process or the ICMPv6 message format. This solution addresses the constraints faced by 6LoWPAN networks by keeping it simple with minimal resources yet efficient and effective.

The hard security mechanism is built into the ICMPv6 messages as an option, while the soft security mechanism lies in the interactions between nodes and data logging in the neighbor cache at the existing central default router. As a result, external infrastructure or additional parties to aid in the security mechanism are not needed.

We make two modifications to the soft security approach to better suit 6LoWPAN. These modifications are as follows: a) we replace the distributed trust-based mechanism with a centralized trust-based mechanism and b) we adopt a simple trust value calculation for soft security instead of using a probabilistic trust model based on beta reputation function [23]. In the following sections, we will provide the details of the proposed security mechanism.

1. Hard Security Implementation

A new ICMPv6 option named Trust-ND option is added to all the ICMPv6 ND messages. The ICMPv6 ND message with the Trust-ND option is referred to as the Trust-ND message [17]. The structure of the Trust-ND message within an IPv6 packet is illustrated in Figure 5.

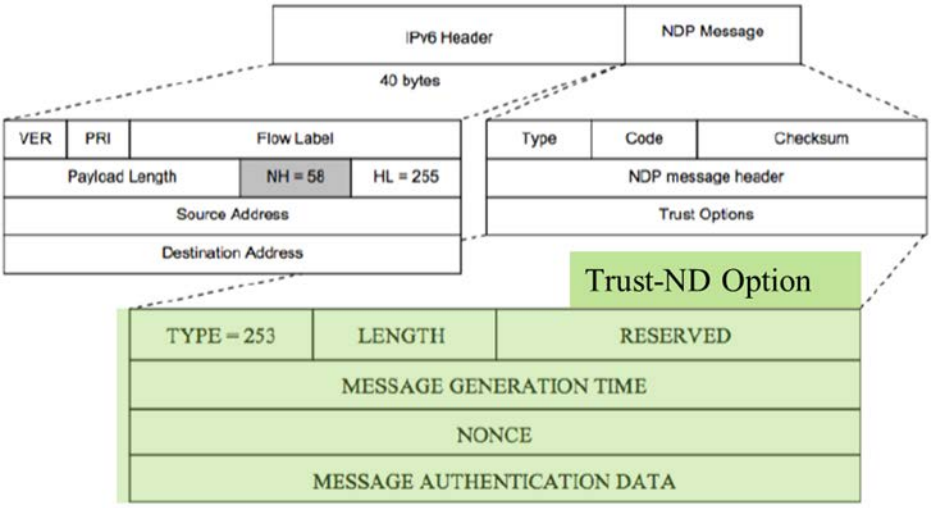


Figure 5. Structure of Trust-ND Option in an IPv6 Packet. Adapted from [17].

Based on [17], the Trust-ND option follows the ICMPv6 option format and has a total size of 32 bytes. The TYPE field consist of one-byte data to define the type of ICMPv6 option that the ND message is carrying. A value of 253 is assigned to show that the Trust-ND option is an experimental option to the ND message [24]. A value of four is assigned to the one-byte length field as the total size of the Trust-ND option is 32 bytes. There is a two-byte field that is reserved. There are three main security fields: Message Generation Time, Nonce, and Message Authentication Data. Message Authentication Data is 20 bytes, while the Message Generation Time and Nonce fields are four bytes each [17]. The details of these three fields are described in the following subsections.

a. Message Generation Time

Message Generation Time is a 4-byte Timestamp that registers the time that the Trust-ND message was sent by the sender. An elapsed time window is set for the Trust-ND message. Once the sender sends out the Trust-ND message, the elapsed time starts. When the receiver receives the Trust-ND message, it checks if the elapsed time has expired by comparing the Timestamp at the point the message is received with the elapsed time. The message is processed if

$$T_{send} < T_{receive} < T_{send} + \text{elapsed time window} \quad (4.1)$$

where T_{send} is the Timestamp at the point the Trust-ND message is sent and $T_{receive}$ is the Timestamp at the point the message is received. Otherwise, the receiver discards the message. This concept is based on [17] and is a simple secure solution to mitigate replay and DoS attacks by ensuring the timeliness and freshness of the Trust-ND messages.

Based on [17], the Timestamp function can only work effectively in a time-synchronized environment. If the nodes in the network are using their own local times that are not synchronized, the time differences between the clocks, if significant, may lead to a DoS as the check on timeliness may not be valid anymore. The effect of nodes experiencing de-synchronized clock time in the network is shown in Figure 6.

The authors of [17] proposed using Coordinated Universal Time (UTC) to replace the local clocks as the reference time for the Timestamp function; however, due to the limitation in our simulation platform, UTC time could not be implemented. As a result, we propose an alternative time synchronization process (adapted from [25]) to synchronize the local clock of each nodes. We assume that the router is connected to a secure global time source. The router is the time master for its local network and sends its time to the nodes in the network. The nodes then synchronize their own local clock with the router's clock time. Time synchronization is conducted at the initial phase of joining a network.

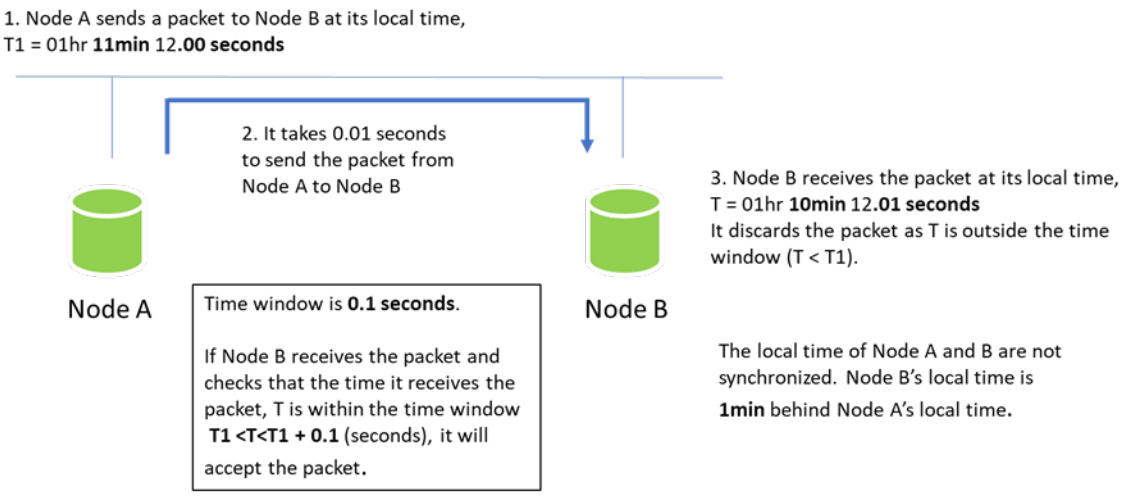


Figure 6. Effects of Clock Time Desynchronization in a Network

When a new host wants to join a network, it sends an RS message to find the router in the network. The router receives the RS message, checks if it has a Trust-ND option, and determines if the sender is a new node by examining its neighbor cache. If the node's local link address is not in the neighbor cache, it is regarded as a new node. If the RS message does not contain a Trust-ND option, the router discards it. If the RS message contains a Trust-ND option and the sender is a new node, it is possible that the Timestamp of the RS message is not valid as the local time of the sender may not be synchronized with the router's clock time. As such, the router does not check the validity of the Timestamp in the RS message. It updates the node's local link address in the neighbor cache and sends an RA message with the router's Timestamp to the new node. The new node receives it and synchronizes its local clock with the router's clock time by using the router's Timestamp and the expected delay between the router and the new node. Expected delay can be estimated from

$$\text{Expected delay} = \frac{1}{2}(TOA_{RA} - T_{RA}) \qquad (4.2)$$

where TOA_{RA} is the time-of-arrival of the RA message based on the local clock of the new node and T_{RA} is the Timestamp of the RS message.

The advantage of this approach is that the time-synchronization of the new node is only done once in the first exchange of RS and RA messages between the new hosts and the router. We believe this approach to be more secure compared to getting time synchronization from an external open source which may be subject to a time spoofing attack; however, the limitation of this approach is that periodic time synchronization cannot be implemented.

b. Nonce

The Nonce is a 4-byte field that contains a number randomly generated by the sender of the Trust-ND message to make sure that every ND message is unique in order to prevent replay attacks. The Nonce is only applied to solicited pair messages, i.e., NS/NA and RS/RA pairs. For every NS or RS message sent, a Nonce is inserted in the message as part of the Trust-ND option. NA and RA messages that respond to the associated NS and RS messages contain the same Nonce in their messages. The use of a Nonce to mitigate a replay attack is illustrated in Figure 7. Unsolicited RA and NA messages are not required to have a Nonce. Replay attacks on unsolicited RA and NA messages are mitigated by the Timestamp mechanism. This concept is adapted from [17].

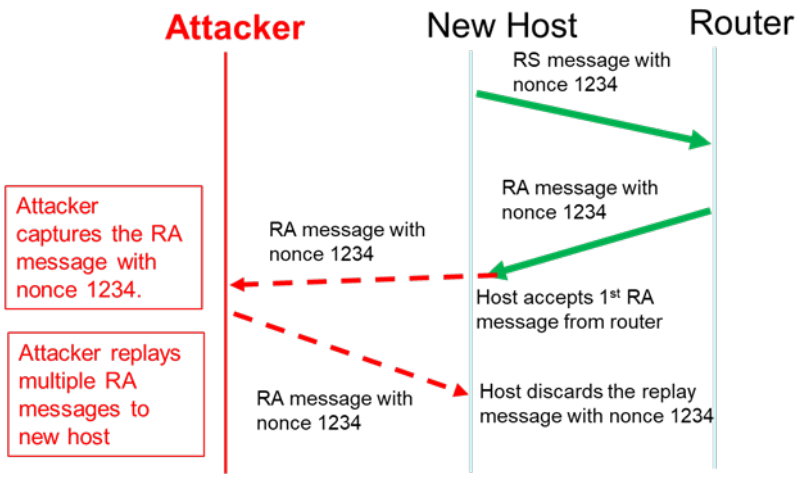


Figure 7. Replay Attack Mitigated through the Use of a Nonce

c. Message Authentication Data

This concept is based on [17]. The sender's ICMPv6 ND message is hashed by the SHA-1 function, and the output value of the SHA function is stored in the 20-byte field in the Trust-ND option on each ICMPv6 message. It is used to ensure data integrity. The receiver inputs the ICMPv6 ND message via the same SHA-1 function and checks if the hashed output value is the same as the one in the option. If it is the same, the message is intact and the system continues to process the message; otherwise, the message is discarded.

2. Soft Security Implementation

These soft security measures are modified from [17] and are in the form of a trust management system controlled by a trusted default router in the network. The trust management includes calculating the trust value of every node in the network and, subsequently, recording and updating the neighbor cache with the trust value associated with each node. The trust calculation is based on two factors, namely the message verification result and the existing trust value of the sender stored in the neighbor cache. The logic of the trust calculation is described in the following paragraphs.

When Node A receives an incoming Trust-ND message from Node B, it first verifies the existence of the Trust-ND option that is attached to the ND message. If there is no Trust-ND option, the receiver considers the message to be insecure and discards the message; otherwise, it proceeds to the next verification. The second verification is to check the value of each of the fields in the Trust-ND option, which includes message-generation time (including time window), Nonce, and the SHA-1 hashed value. If any of the data in these fields are not valid, the message is discarded.

After the verification, Node A performs a trust calculation for the message sender based on two parts, namely the results of the second verification and the existing trust status of the sender which is stored in the neighbor cache.

For part one, Node B (the sender) is allocated a value of one if its message is a valid Trust-ND message. It is allocated a value of zero if the message is not valid.

For part two, Node A checks the neighbor cache to obtain the existing trust value of Node B. Node B is allocated a value of zero if it is not registered in the neighbor cache as it is regarded as a new node joining the network. If Node B is already registered in the neighbor cache, there is an existing value that is associated to Node B based on past interactions with other nodes. The existing value can be zero, one, or two. If Node B has an existing trust value of zero, a value of zero is allocated to Node B. If Node B has an existing trust value of one or two, a value of one is allocated to Node B. Node B's new trust value is equal to the summation of the values allocated to Node B in part one and two of the verification process. The new trust value is then updated in the neighbor cache.

The neighbor cache is updated with the new trust value associated with each node in the network after every interaction between the nodes in the network. The nodes in the network can then leverage the trust value to determine which nodes they should interact with. If Node B has a zero value, it means that it has failed its past interactions with the nodes in the network and is likely to be a malicious node. If Node B has a value of one, it means that it was either a new node in the network in the previous interaction with a valid Trust-ND option or it was a trustworthy node but failed its message verification in the previous interaction. This trust value is considered uncertain, and we need more successful interactions with Node B before we consider the node to be trustworthy. If Node B has a value of two, it means that Node B has successfully interacted with the other nodes in the network and is regarded as trustworthy.

Nodes should refrain from sending messages to nodes with trust values of zero or one as these nodes are either malicious or uncertain. The output associated with different trust values using the soft security mechanism is shown in Table 2. The logic of the soft security mechanism is shown in Figure 8.

Table 2. Summary of Outputs Associated with Different Trust Values for the Soft Security Mechanism

	Node B trust value=0	Node B trust value=1	Node B trust value=2
Incoming messages from Node B	Discard incoming messages	Accept incoming messages if only message verification value =1	Accept incoming messages
Outgoing messages via Node B	Avoid sending messages to Node B	Avoid sending messages to Node B	Allow sending messages to node B

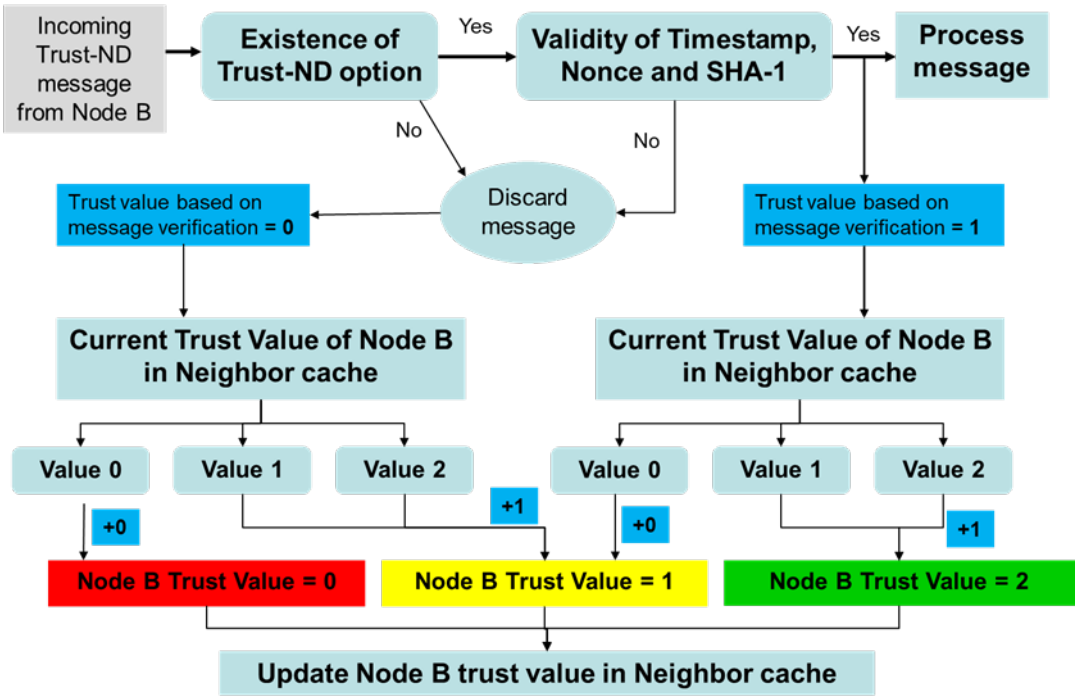


Figure 8. Logic Flow for the Soft Security Mechanism

C. CHAPTER SUMMARY

In this chapter, key considerations for the design of the security mechanism for the 6LoWPAN ND protocol were presented followed by a discussion of the proposed security mechanism that uses a combination of hard and soft security approaches. The hard security approach includes having a Trust-ND option on the ICMPv6 ND messages which includes Timestamp, Nonce, and SHA-1 hashing functions. A time-synchronization mechanism is also introduced to address time de-synchronized networks. The soft security approach leverages the social interactions between the nodes in the network to identify malicious nodes using a trust management scheme.

V. EXPERIMENTAL SETUP

In this chapter, we describe the simulation setup used to run various experiments to test the effectiveness of the hard security mechanisms against replay attacks. The simulation parameters and the attack scenarios that were simulated are also discussed.

A. AIM OF EXPERIMENT

The aim of the experiments was to evaluate the effectiveness of the proposed security mechanism in a 6LoWPAN network against replay attacks. In this thesis, we conducted three sets of experiments using the Contiki operating system (OS) version 3.0 and Cooja network simulator [26]. The first and second experiment tested the effectiveness of the Nonce and Timestamp against replay attacks. The third experiment examined the impact of de-synchronized time in the network and evaluated the time-synchronization solution proposed in Chapter IV.

B. NETWORK SIMULATION SETUP

1. Contiki OS and Cooja Network Simulator

Contiki OS version 3.0 and its Cooja network simulator were selected as the simulation platform for the experiments in this thesis. According to [26], Contiki is an open source operating system that is designed for Internet of Things (IoT), which are low-powered, wireless networks. It has built in 6lowPAN protocols which allow us to modify existing code or add in new code to suit our system requirements. The Contiki system includes a network simulator named Cooja. It is a powerful tool for Contiki development as it allows developers to evaluate their software through network simulation before implementing on the actual hardware. There are also other useful websites that guide the use of Contiki OS and the Cooja network simulator [27]–[29].

2. Simulation Environment

Simulation parameters set in the Cooja network simulator are shown in Table 3. The single 6LoWPAN network was set up with sky mote sensors (which is a type of sensor available in Contiki) and a router. An attacker node was added to the network. Sky

mote sensors are common sensors used in IoT, and their hardware is emulated in the Cooja simulation. The nodes can be placed in any position within a two-dimensional space. A node can be moved manually during the simulation. The communication range of the sensors can be configured, and the antennas of the sensors are omni-directional. The ND messages sent within the network are monitored and displayed in the Cooja network simulator. The behaviour of the nodes in handling these messages were also observed in our simulation. This data was captured during the simulation and was further analyzed using the network protocol analyzer Wireshark.

Table 3. Simulation Parameters

	Simulation Parameters	Descriptions
1	Network	A single 6lowPAN network with Sky mote sensors consisting of a router, hosts (including a attacker node)
2	Sensor Communication range	Transmission range: 50m Interference range : 100m
3	Transmission and Reception success ratio	100%
4	Antenna	Omni directional

An 80 m × 80 m ground plane was set up with three legitimate hosts, one attacker, and one router. The respective positions of these nodes are shown in Figure 9. The green node with ID: 1 is the attacker. The orange nodes with ID:2, 3, and 4 are the legitimate hosts. The purple node with ID:5 is the router. The arrangement of the nodes is made such that every node is within the communication range of one another. Once the simulation starts, the nodes create their link local addresses and start interacting with each other to discover their neighbors.

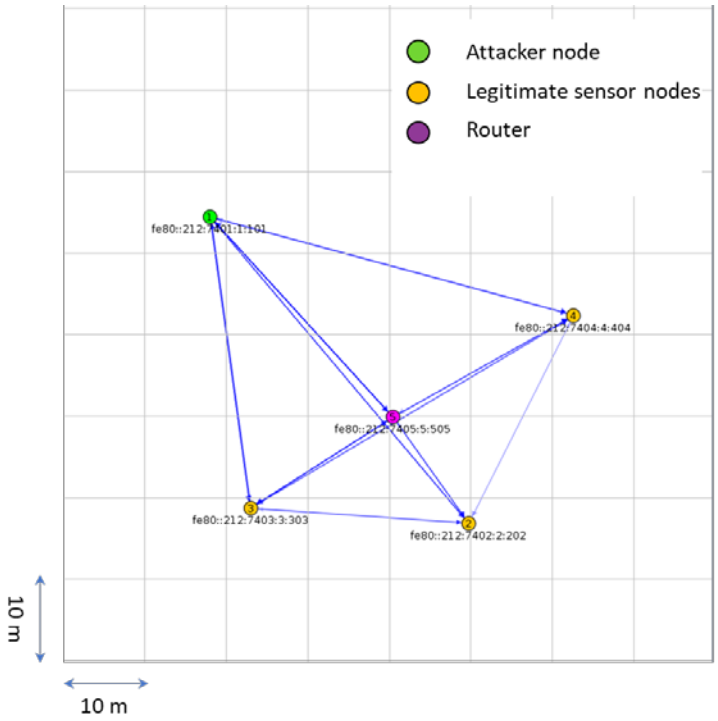


Figure 9. Relative Position of the Nodes in the Simulated Network

C. ATTACK SCENARIOS

The attack scenarios focused mainly on the effectiveness of the Nonce and Timestamp options against replay attacks during the router discovery phase. Attack scenarios that included the Nonce and Timestamp options are described in the following subsections.

1. Nonce Option against Replay Attacks

A Trust-ND option with four-byte Nonce field is implemented on the ICMPv6 RA and RS messages. The Nonce function generates a random unique number and stores it in the Trust-ND option of every RS message created by the sender nodes. A four-byte Nonce can generate a range of 2^{32} numbers. In our simulation we set the range of numbers to be between 0 to 10000 so that it is easier to monitor the Nonce numbers generated by the nodes during the simulation. The attack scenario is based on Figure 7 (shown in Chapter IV) where the attacker sends replay RA messages to the host victims during the router discovery phase. Host victims with the Nonce option in their RS and

RA messages are expected to detect replay RA messages from the attacker and discard them. A scenario without the Trust-ND option was also simulated. The nodes are expected to accept the replay RA messages from the attacker as they have no means to detect and identify replay messages.

2. Timestamp Option against Replay Attacks

A Trust-ND option with a four-byte Timestamp field is implemented in the ICMPv6 RA and RS messages. The clock resolution is 128 ticks per second (7.8 ms per tick) and is limited by the emulated hardware of sky mote sensors in the Contiki OS. We set up the attack scenario similar to the previous scenario with the Nonce option where the attacker replays RA messages to the host victims. Host victims with the Timestamp option in their RS and RA messages are expected to detect replay RA messages sent from attacker and then discard them.

On top of demonstrating the effectiveness of Timestamp against replay attacks, the impact of de-synchronization of local clocks between the nodes in the network was also simulated. The effectiveness of the proposed time synchronization mechanism was demonstrated.

D. MODIFICATIONS TO EXISTING CONTIKI OS CODE

Modifications to the code of existing files in Contiki OS version 3.0 are needed to implement the Trust-ND option which consists of the Nonce and Timestamp field. Moreover, new code was added to simulate the attacks on the network. The key modifications to the existing code are described in the following paragraphs. The code is found in the Appendix.

Uip-nd6.c file is the ND protocol specified in [11]. It covers the processes of the ND operation. Under this file, we have modified four functions. The first function named uip_nd6_rs_output is responsible for generating RS messages in host nodes. A random function was inserted to generate a Nonce and the generated Nonce was stored for future checks under the function named expected_nonce. The Timestamp of the RS message was also inserted under this function. The second function named rs_input is responsible

for processing the RS messages that are sent to the router. A routine was inserted into this function so that the router uses the same Nonce number from the received RS message on the RA message that the router is going to send to the host node.

Another routine was added for the router to check if the local link address of the sender of the RS message is in the neighbor table cache. If it is not in the neighbor cache, the router synchronises the sender's local clock with its own time using a RA message.

The third function named uip_nd6_ra_output is responsible for generating the RA messages from the router. A RA message with a Trust-ND option was created in this function so that the router can insert the same Nonce number as the received RS message and send it back to the host node. In addition, we included the attack function of copying the RA message when the attacker intercepted the RA message.

The fourth function named ra_input is responsible for processing the RA packets received by the host node or attacker. The attacker mode of sending multiple replay RA messages to the host node was included in this function. Two scenarios were implemented in this function. One was without the Trust-ND option, while the other was with the Trust-ND option. With the Trust-ND option, the host node is able to detect the replay RA messages and discard them. Without the Trust-ND option, the host node accepts the RA messages without realising that they are replay RA messages.

E. CHAPTER SUMMARY

In this chapter, we described the simulation environment used to evaluate the hard security mechanisms proposed in Chapter IV. The experiments evaluated the effectiveness of the Nonce and Timestamp against replay attacks and evaluated the time-synchronization solution proposed in Chapter IV. The simulation was conducted using Contiki OS version 3.0 and its Cooja network simulator. The simulation setup for the attack scenarios and the modifications done on the existing code files were also discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

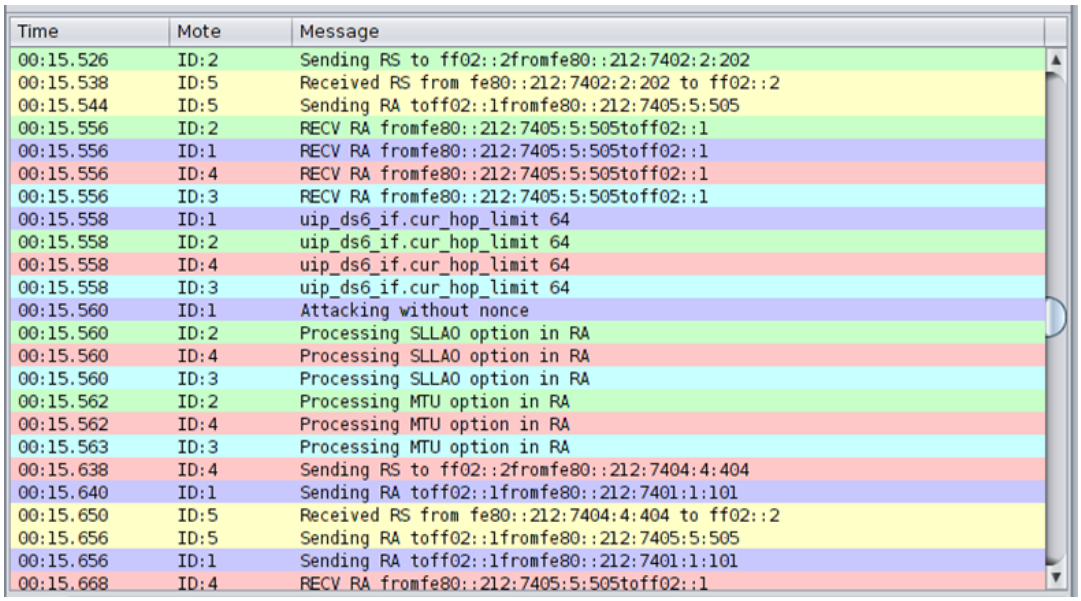
VI. SIMULATION RESULTS AND ANALYSIS

Five simulations were conducted: 1) an attack scenario without the Trust-ND option, 2) an attack scenario with Nonce, 3) an attack scenario with the Timestamp option, 4) a scenario in which the Timestamp option operates with de-synchronized clocks, and 5) a scenario in which the Timestamp option operates with the proposed time synchronization mechanism. Events with respect to time were captured in the simulation and were analyzed to evaluate the effectiveness of the Nonce and Timestamp options and the time-synchronization mechanism. Wireshark was also used to analyze the data captured during the simulation. The details of the observations and analysis of the simulation results are covered in the following sections.

A. ATTACK SCENARIO WITHOUT TRUST-ND OPTION

The scenarios discussed in this section follow the network topology shown in Figure 9 (see Chapter V). After the sensor nodes received their local link addresses, they proceeded to send multicast RS messages to find routers in the network. As shown in Figure 10, at 00:15.526 s, ID:2 generated an RS message and sent the RS message to a multicast address ff02::2. According to [30], ff02::2 is a registered IPv6 multicast address that includes all the routers in the local network segment. At 00:15.538 s, the router (ID:5) received the RS message sent from ID:2, and at 00:15.544 s, the router generated a RA message and sent it to a multicast address ff02::1. According to [30], ff02::1 is a registered IPv6 multicast address that includes all the nodes in the local network segment.

At 00:15.560 s, the attacker (ID:1) copied the RA message and subsequently sent the replay message twice at 00:15.640 s and 00:15.656 s to the multicast address ff02::1.

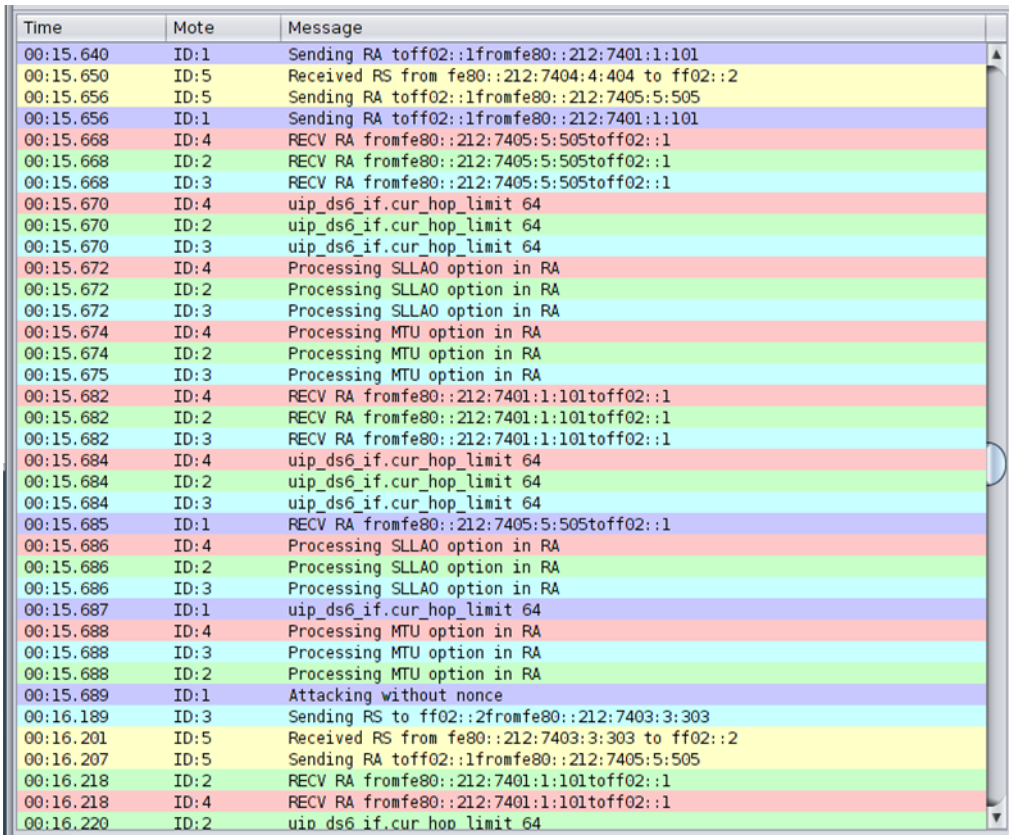


Time	Mote	Message
00:15.526	ID:2	Sending RS to ff02::2fromfe80::212:7402:2:202
00:15.538	ID:5	Received RS from fe80::212:7402:2:202 to ff02::2
00:15.544	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505
00:15.556	ID:2	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.556	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.556	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.556	ID:3	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.558	ID:1	uip_ds6_if.cur_hop_limit 64
00:15.558	ID:2	uip_ds6_if.cur_hop_limit 64
00:15.558	ID:4	uip_ds6_if.cur_hop_limit 64
00:15.558	ID:3	uip_ds6_if.cur_hop_limit 64
00:15.560	ID:1	Attacking without nonce
00:15.560	ID:2	Processing SLLAO option in RA
00:15.560	ID:4	Processing SLLAO option in RA
00:15.560	ID:3	Processing SLLAO option in RA
00:15.562	ID:2	Processing MTU option in RA
00:15.562	ID:4	Processing MTU option in RA
00:15.563	ID:3	Processing MTU option in RA
00:15.638	ID:4	Sending RS to ff02::2fromfe80::212:7404:4:404
00:15.640	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101
00:15.650	ID:5	Received RS from fe80::212:7404:4:404 to ff02::2
00:15.656	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505
00:15.656	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101
00:15.668	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1

Figure 10. Captured Events in the Network without Trust-ND Option (Part One)

As shown in Figure 11, at 00:15.668, ID:2, 3, and 4 received the first replay RA message from the attacker. They were unaware that it was a replay message; therefore, they processed the information in the RA message at 00:15.672 s, 00:15.674 s, and 00:15.675 s. At 00:15.682 s, ID:2, 3, and 4 received the second replay RA message from the attacker. Again, they processed the information in the RA message at 00:15.686 s and 00:15.688 s.

This simulation showed that the replay attack was successful, and this attack can be applied to all the other hosts as long as the attacker is within the communication range. We observe that the hosts were unaware of the presence of the attack. If the attacker is able to replay multiple messages to the victim host, the host is expected to be denied service as its resources will be exhausted by processing the outdated RA messages.



Time	Mote	Message
00:15.640	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101
00:15.650	ID:5	Received RS from fe80::212:7404:4:404 to ff02::2
00:15.656	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505
00:15.656	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101
00:15.668	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.668	ID:2	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.668	ID:3	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.670	ID:4	uip_ds6_if.cur_hop_limit 64
00:15.670	ID:2	uip_ds6_if.cur_hop_limit 64
00:15.670	ID:3	uip_ds6_if.cur_hop_limit 64
00:15.672	ID:4	Processing SLLAO option in RA
00:15.672	ID:2	Processing SLLAO option in RA
00:15.672	ID:3	Processing SLLAO option in RA
00:15.674	ID:4	Processing MTU option in RA
00:15.674	ID:2	Processing MTU option in RA
00:15.675	ID:3	Processing MTU option in RA
00:15.682	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:15.682	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:15.682	ID:3	RECV RA fromfe80::212:7401:1:101toff02::1
00:15.684	ID:4	uip_ds6_if.cur_hop_limit 64
00:15.684	ID:2	uip_ds6_if.cur_hop_limit 64
00:15.684	ID:3	uip_ds6_if.cur_hop_limit 64
00:15.685	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:15.686	ID:4	Processing SLLAO option in RA
00:15.686	ID:2	Processing SLLAO option in RA
00:15.686	ID:3	Processing SLLAO option in RA
00:15.687	ID:1	uip_ds6_if.cur_hop_limit 64
00:15.688	ID:4	Processing MTU option in RA
00:15.688	ID:3	Processing MTU option in RA
00:15.688	ID:2	Processing MTU option in RA
00:15.689	ID:1	Attacking without nonce
00:16.189	ID:3	Sending RS to ff02::2fromfe80::212:7403:3:303
00:16.201	ID:5	Received RS from fe80::212:7403:3:303 to ff02::2
00:16.207	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505
00:16.218	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:16.218	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:16.220	ID:2	uip_ds6_if.cur_hop_limit 64

Figure 11. Captured Events in the Network without Trust-ND Option (Part Two)

In Figure 12, the table on the left is the list of events captured directly from the nodes' output during the simulation while the table on the right is the list of RA and RS messages recorded by the 6LoWPAN network analyzer. We compared the tables and correlated the RS and RA messages recorded in the 6LoWPAN network analyzer to the events captured in the table on the left. The correlations are highlighted in blue, as shown in Figure 12. Within this time frame, two ICMPv6 RS messages and four ICMPv6 RA messages were transmitted. We also observe that the size of the RS message was 45 bytes, while the size of the RA message was 61 bytes.

We observed that the 6LoWPAN network analyzer recorded the same event slightly at a later time as compared to the table on the left. Under the 31st row of the table on the right, we see that ID:2 sent a multicast RS message to all four nodes in the network (it is displayed as [4 d] in the table) at 00:15.529 s while we see from the table on the left that the same RS message was sent out at 00:15.526 s.

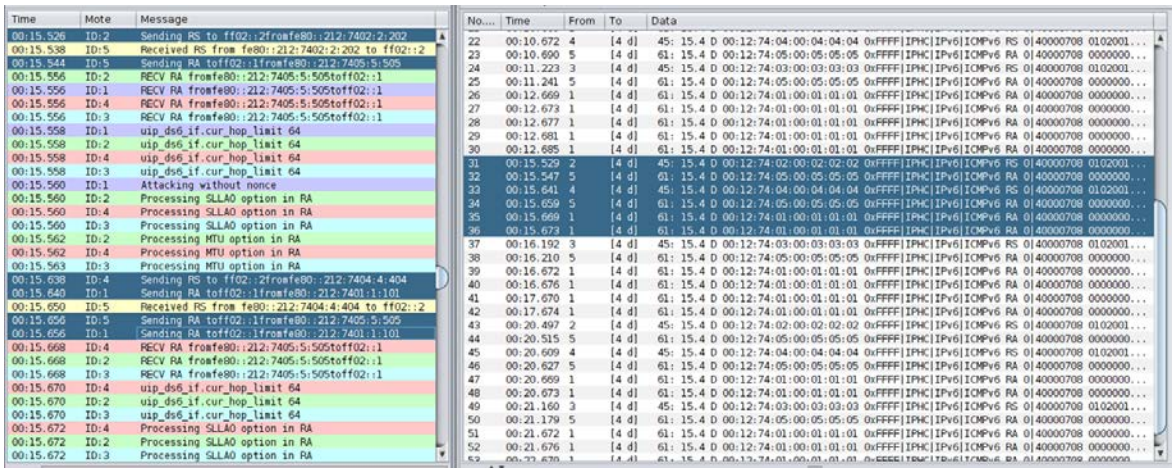
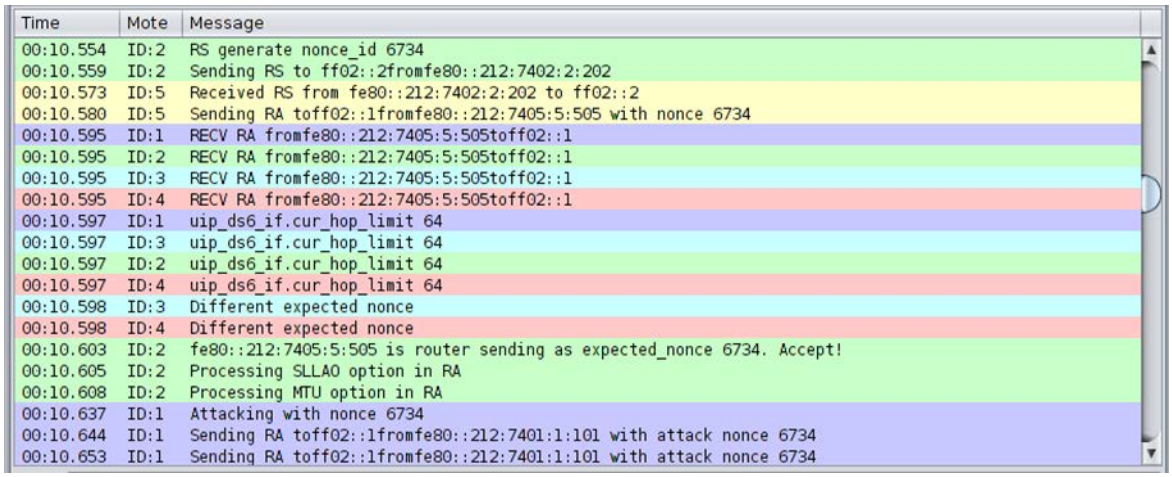


Figure 12. Correlation between Nodes' Output and 6LoWPAN Network Analyzer (without Nonce Option)

B. ATTACK SCENARIO WITH NONCE OPTION

As shown in Figure 13, at 00:10.554 s, ID:2 generated a RS message with a Nonce number of 6734. It then sent the RS message to a multicast address ff02::2 at 00:10.559 s. At 00:10.573 s, the router (ID:5) received the RS message send from ID:2, and at 00:10.580 s, the router generated a RA message with the same Nonce number 3037 and sent it to a multicast address ff02::01.

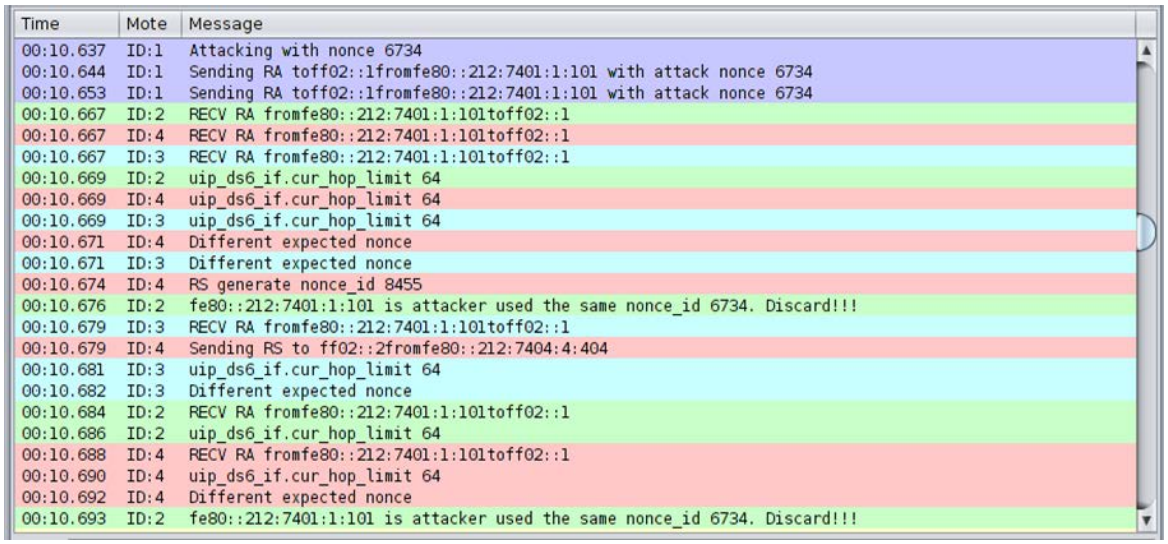
At 00:10.595 s, all the nodes, ID:1, 2, 3, and 4 received the RA message from the router. At 00:10.598 s, ID:3 and ID:4 detected that this RA message was not meant for them as the Nonce number associated with the RA message was not the same as their last RS message; thus, they discarded the RA messages. At 00:10.603 s, ID:2 received the RA message from the router and verified that the Nonce number 6734 in the RA message was the same number in its last RS message; thus, it accepted the RA message from the router and processed the information found in the RA message at 00:10.605 s and 00:10.608 s. At 00:10.637 s, the attacker (ID:1) copied the RA message with Nonce number 6734, and at both 00:10:644 s and 00:10.653 s, the attacker sent an RA message each to the multicast address ff02::1.



Time	Mote	Message
00:10.554	ID:2	RS generate nonce_id 6734
00:10.559	ID:2	Sending RS to ff02::2fromfe80::212:7402:2:202
00:10.573	ID:5	Received RS from fe80::212:7402:2:202 to ff02::2
00:10.580	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505 with nonce 6734
00:10.595	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.595	ID:2	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.595	ID:3	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.595	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.597	ID:1	uip_ds6_if.cur_hop_limit 64
00:10.597	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.597	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.597	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.598	ID:3	Different expected nonce
00:10.598	ID:4	Different expected nonce
00:10.603	ID:2	fe80::212:7405:5:505 is router sending as expected_nonce 6734. Accept!
00:10.605	ID:2	Processing SLLAO option in RA
00:10.608	ID:2	Processing MTU option in RA
00:10.637	ID:1	Attacking with nonce 6734
00:10.644	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 with attack nonce 6734
00:10.653	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 with attack nonce 6734

Figure 13. Captured Events in the Network with Nonce Option (Part One)

In Figure 14, we see that ID:2, 3, and 4 received the first replay RA message at 00:10.667 s. At 00:10.671 s, ID 3 and 4 detected that the Nonce in the replay RA message was different from the Nonce number of their last RS message and, thus, discarded the RA messages. ID:2 detected that the Nonce number in the RA message was already received previously from the router; thus, it discarded the replay message without processing the information in the RA message. Likewise, for the second replay messages, ID:2, 3, and 4 discarded the messages.



Time	Mote	Message
00:10.637	ID:1	Attacking with nonce 6734
00:10.644	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 with attack nonce 6734
00:10.653	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 with attack nonce 6734
00:10.667	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.667	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.667	ID:3	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.669	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.669	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.669	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.671	ID:4	Different expected nonce
00:10.671	ID:3	Different expected nonce
00:10.674	ID:4	RS generate nonce_id 8455
00:10.676	ID:2	fe80::212:7401:1:101 is attacker used the same nonce_id 6734. Discard!!!
00:10.679	ID:3	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.679	ID:4	Sending RS to ff02::2fromfe80::212:7404:4:404
00:10.681	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.682	ID:3	Different expected nonce
00:10.684	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.686	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.688	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.690	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.692	ID:4	Different expected nonce
00:10.693	ID:2	fe80::212:7401:1:101 is attacker used the same nonce_id 6734. Discard!!!

Figure 14. Captured Events in the Network with Nonce Option (Part Two)

In Figure 15, we compare the output of both the nodes and the network analyzer and correlated the RS and RA messages. The correlations are highlighted in blue as shown in Figure 15. Within this time frame, two ICMPv6 RS messages and three ICMPv6 RA messages were transmitted. We also observe that the size of the RS message was 77 bytes, while the size of the RA message was 93 bytes.

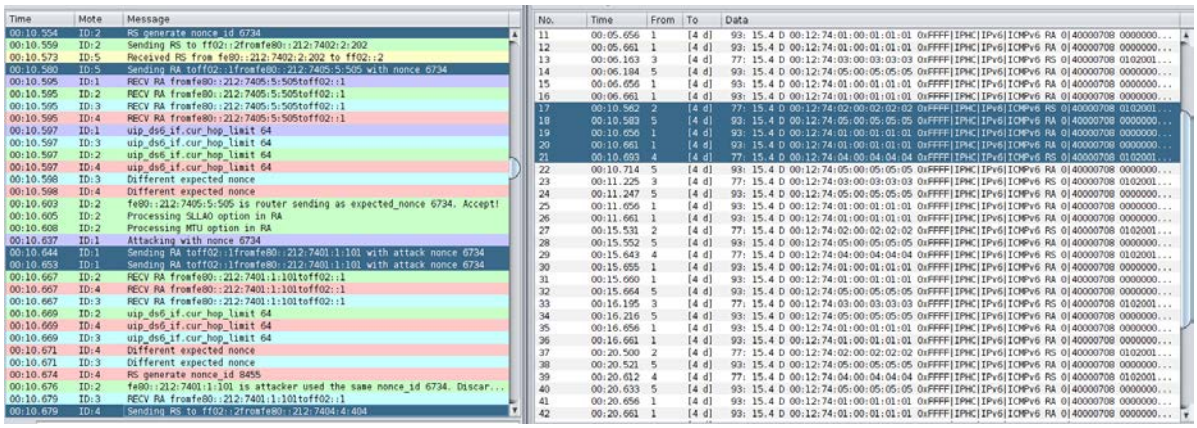


Figure 15. Correlation between Nodes' Output and 6LoWPAN Network Analyzer (with Nonce Option)

The captured data in the simulation was also analyzed using Wireshark. The aim of the analysis was to verify if the Trust-ND option was implemented correctly. Wireshark displayed the breakdown of the data found in the ICMPv6 ND messages. As shown in Figure 16, we analyzed one of the RA messages sent by the router to all the nodes in the local network (under the 18th row of the table). The source address, fe80::212:7405:5:505 belonged to the router, while ff02::1 is a multicast address that includes all the nodes in the local network. The data that is highlighted in orange is the 32-byte Trust ND option implemented in this thesis. The data in yellow boxes represent the type of ICMPv6 options we used in this experiment. Type 253 is used for the purpose of experimentation and testing [30]. Type 253 in hexadecimal is FD; thus, FD is captured in the first byte of the Trust-ND option. The Length of the Trust-ND option is defined in octets. Since the Trust-ND option has a size of 32-bytes, the value of the Length field is four. This value of 04 is shown in the green boxes in Figure 16.

The Nonce number that is attached to this RA message was 6734 and was 1A 4E in hexadecimal. We observe in Figure 16 that there was a value of 4E 1A in the Trust-ND option highlighted in purple. The hexadecimal is reversed as the most significant bit is transmitted first. The rest of the data in the Trust-ND option were zeros as the reserved field, SHA-1 value, and Timestamp values were zeroed out.

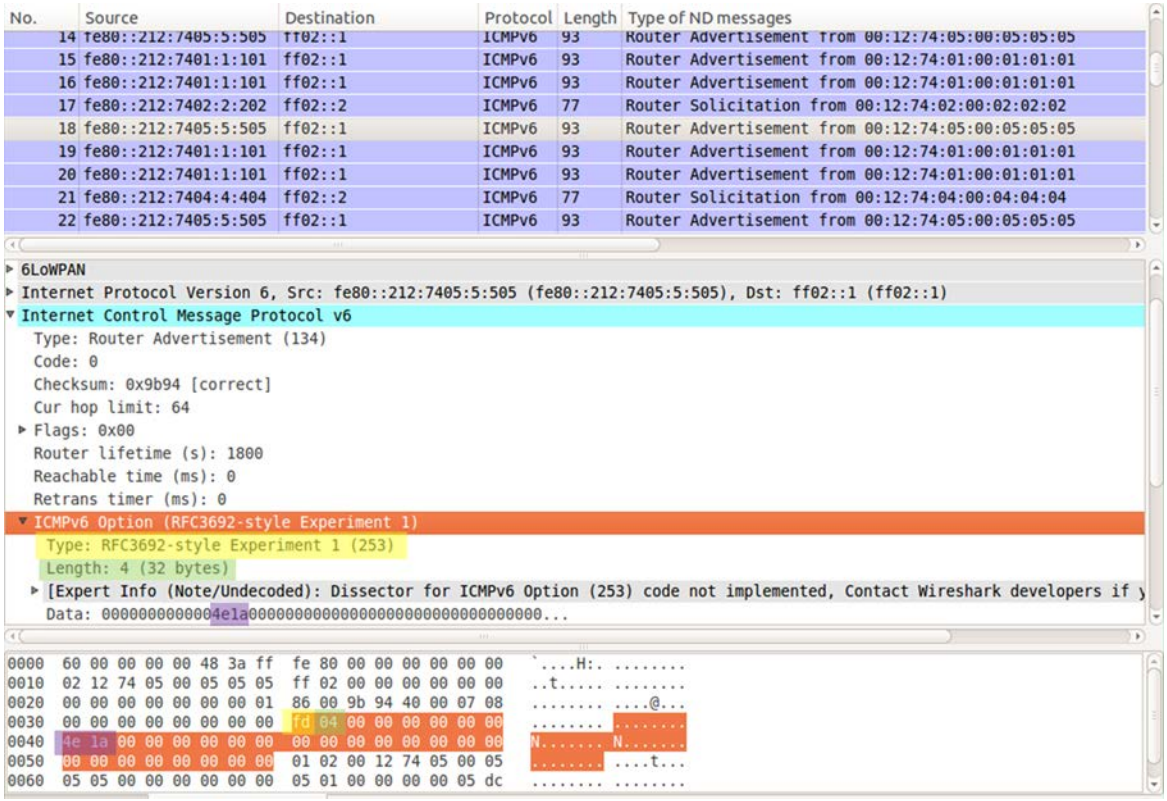


Figure 16. Trust-ND Option with Nonce Data Breakdown Using Wireshark

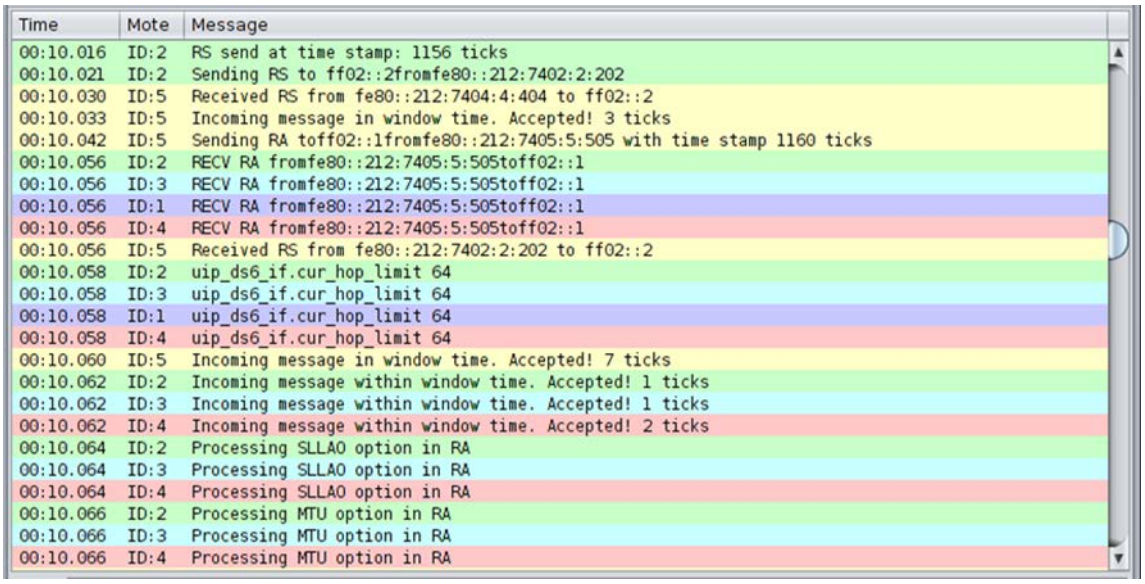
The simulation analysis showed that the Nonce field in the Trust-ND option was implemented successfully and is effective against replay attacks. The legitimate hosts were able to detect replay messages and discard them without processing the messages; moreover, when the router sent a solicited multicast RA message to every node, the nodes were able to determine if this RA message was meant for them. They discarded the RA message if the message was not responding to their own RS messages. As such, it

prevented the hosts from using their resources unnecessarily to process redundant RA messages.

C. ATTACK SCENARIO WITH TIMESTAMP OPTION

In this simulation, the local clocks of the nodes in the network were assumed to be time-synchronized. The pre-defined time window for this simulation was set as 0 < time window < 10 ticks for the RS message, while 0 < time window < 6 ticks was set for the RA message. We assumed that the router was handling more traffic as compared to the hosts and thus, more time is given for the router to process the RS messages. The Timestamp window was capped so as to reduce the chances of replay attacks that can possibly fall within the time window. The resolution of the Timestamp is about 7.8 ms per clock tick.

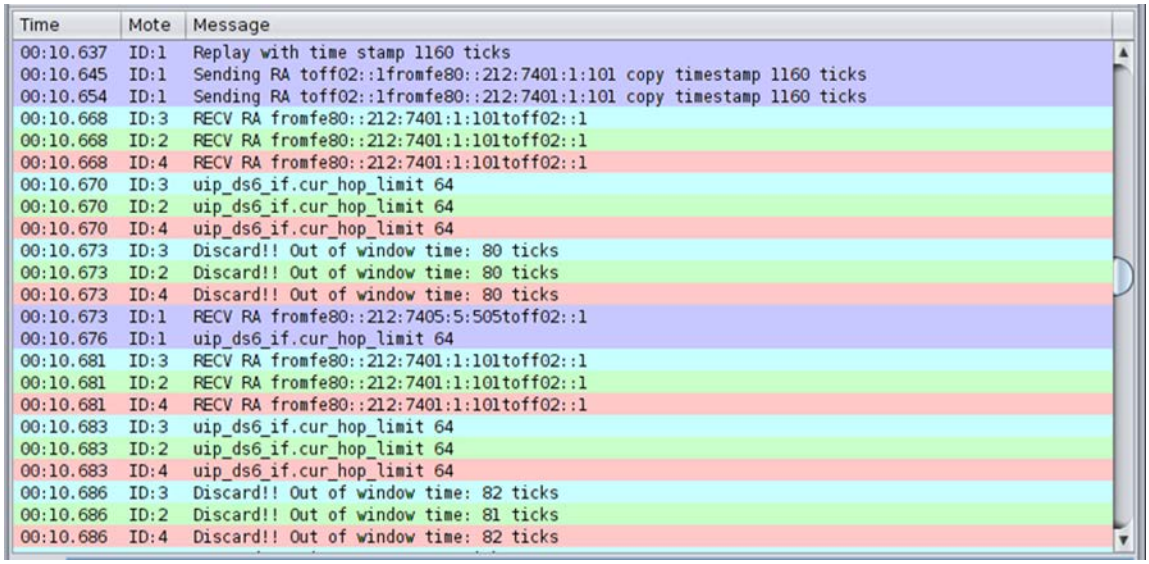
The events of the simulation are captured and shown in Figure 17 and 18. In Figure 17, at 00:10.016 s, ID:2 generated an RS message with a Timestamp of 1156 ticks and then sent the RS message to a multicast address ff02::2 at 00:10.021 s. At 00:10.030 s, the router (ID:5) received the RS message sent from ID:2. It calculated the time difference between its own local clock time and the Timestamp of the RS message and compared it with the pre-defined time window. The time difference was three ticks and, therefore, was within the pre-defined window; thus, the RS message was accepted. ID:5 generated a multicast RA message with a timestamp of 1160 ticks and sent the message to all the nodes in the network at 00:10.042 s. ID:1, 2, 3, and 4 received the RA message at 00:10.056 s. They accepted the RA message at 00:10.062 s as the time difference between their own local clock time and the Timestamp of the RA message was within the pre-defined window. ID:2, 3 and 4 processed the RA message at 00:10.064 s and 00:10.066 s.



Time	Mote	Message
00:10.016	ID:2	RS send at time stamp: 1156 ticks
00:10.021	ID:2	Sending RS to ff02::2fromfe80::212:7402:2:202
00:10.030	ID:5	Received RS from fe80::212:7404:4:404 to ff02::2
00:10.033	ID:5	Incoming message in window time. Accepted! 3 ticks
00:10.042	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505 with time stamp 1160 ticks
00:10.056	ID:2	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.056	ID:3	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.056	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.056	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.056	ID:5	Received RS from fe80::212:7402:2:202 to ff02::2
00:10.058	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.058	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.058	ID:1	uip_ds6_if.cur_hop_limit 64
00:10.058	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.060	ID:5	Incoming message in window time. Accepted! 7 ticks
00:10.062	ID:2	Incoming message within window time. Accepted! 1 ticks
00:10.062	ID:3	Incoming message within window time. Accepted! 1 ticks
00:10.062	ID:4	Incoming message within window time. Accepted! 2 ticks
00:10.064	ID:2	Processing SLLAO option in RA
00:10.064	ID:3	Processing SLLAO option in RA
00:10.064	ID:4	Processing SLLAO option in RA
00:10.066	ID:2	Processing MTU option in RA
00:10.066	ID:3	Processing MTU option in RA
00:10.066	ID:4	Processing MTU option in RA

Figure 17. Captured Events in the Network with Timestamp Option (Part One)

In Figure 18, the attacker (ID:1) duplicated the RA message and sent the replay RA message to all the hosts in the network at 00:10.654 s. ID:2, 3, and 4 received the RA message and discarded the message at 00:10.673 s. The time difference between their own local clock time and the Timestamp of the RA message was 80 ticks, which is outside the pre-defined window.



Time	Mote	Message
00:10.637	ID:1	Replay with time stamp 1160 ticks
00:10.645	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 copy timestamp 1160 ticks
00:10.654	ID:1	Sending RA toff02::1fromfe80::212:7401:1:101 copy timestamp 1160 ticks
00:10.668	ID:3	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.668	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.668	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.670	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.670	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.670	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.673	ID:3	Discard!! Out of window time: 80 ticks
00:10.673	ID:2	Discard!! Out of window time: 80 ticks
00:10.673	ID:4	Discard!! Out of window time: 80 ticks
00:10.673	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:10.676	ID:1	uip_ds6_if.cur_hop_limit 64
00:10.681	ID:3	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.681	ID:2	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.681	ID:4	RECV RA fromfe80::212:7401:1:101toff02::1
00:10.683	ID:3	uip_ds6_if.cur_hop_limit 64
00:10.683	ID:2	uip_ds6_if.cur_hop_limit 64
00:10.683	ID:4	uip_ds6_if.cur_hop_limit 64
00:10.686	ID:3	Discard!! Out of window time: 82 ticks
00:10.686	ID:2	Discard!! Out of window time: 81 ticks
00:10.686	ID:4	Discard!! Out of window time: 82 ticks

Figure 18. Captured Events in the Network with Timestamp Option (Part Two)

The simulation analysis showed that the Timestamp field in the Trust-ND option was implemented successfully and that it is effective against replay attacks. The legitimate hosts were able to detect replay messages and discard them by using the pre-defined time window. We also observe that setting the right duration for the time window is critical. If the duration is set too high, the replay attack may be able to succeed, but if the duration is set too low, the legitimate messages may be discarded as well.

D. TIMESTAMP OPTION WITH DE-SYNCHRONIZED CLOCKS

In this simulation, the local clocks of the hosts and the router were set differently to demonstrate the impact of having de-synchronized clocks between the nodes in the network. The local clock of ID:2 was set at 58 ticks behind the router's clock time, ID:3 was set at 25 ticks ahead of the router's clock time, and ID:4 was set at 44 ticks behind the router's clock time. Once the simulation started, ID:2, 3, and 4 hosts sent multicast RS messages to the ID:5 router at 00:05.497 s, 00:06.160 s, and 00:05.609 s, respectively, as shown in Figure 19. When ID:5 received the RS messages from the hosts, it calculated the time difference between its own local clock time and the Timestamp in each of the RS messages and compared it with the pre-defined time window. The pre-defined time window for this simulation was set as 0 < time window < 10 ticks. Since the calculated time differences as shown at 00:05.514, 00:06.177, and 00:05.626 are outside the time window, the router discarded the RS messages of ID:2, 3, and 4. As a result, the hosts were unable to obtain the router and network parameters as the router did not respond to their RS messages with the RA messages, as shown in Figure 19.

The simulation analysis showed that the nodes in the network are not able to communicate effectively with each other if there is a de-synchronization of clock time between the nodes. In this case, the router discarded the legitimate RS messages and caused a DoS to the hosts; thus, it is important to have a time-synchronization mechanism in the network.

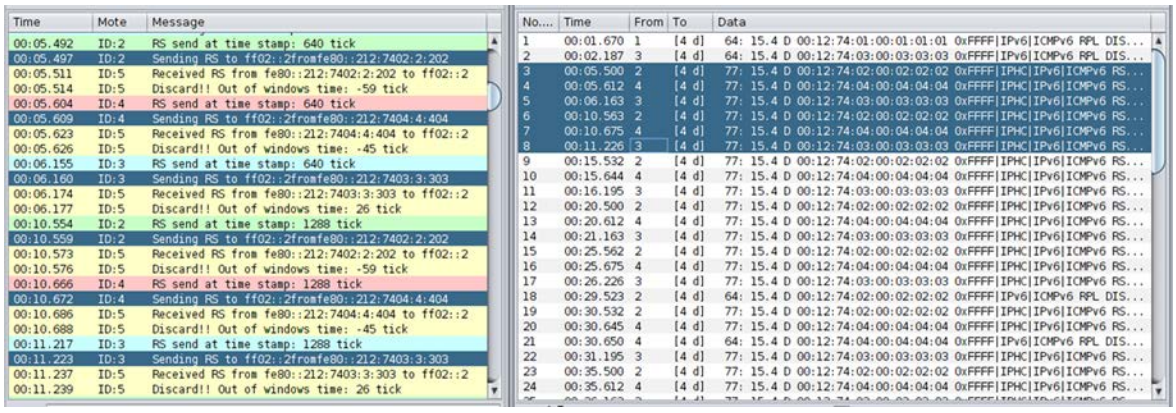


Figure 19. Timestamp Option with De-synchronized Clocks Simulation Data

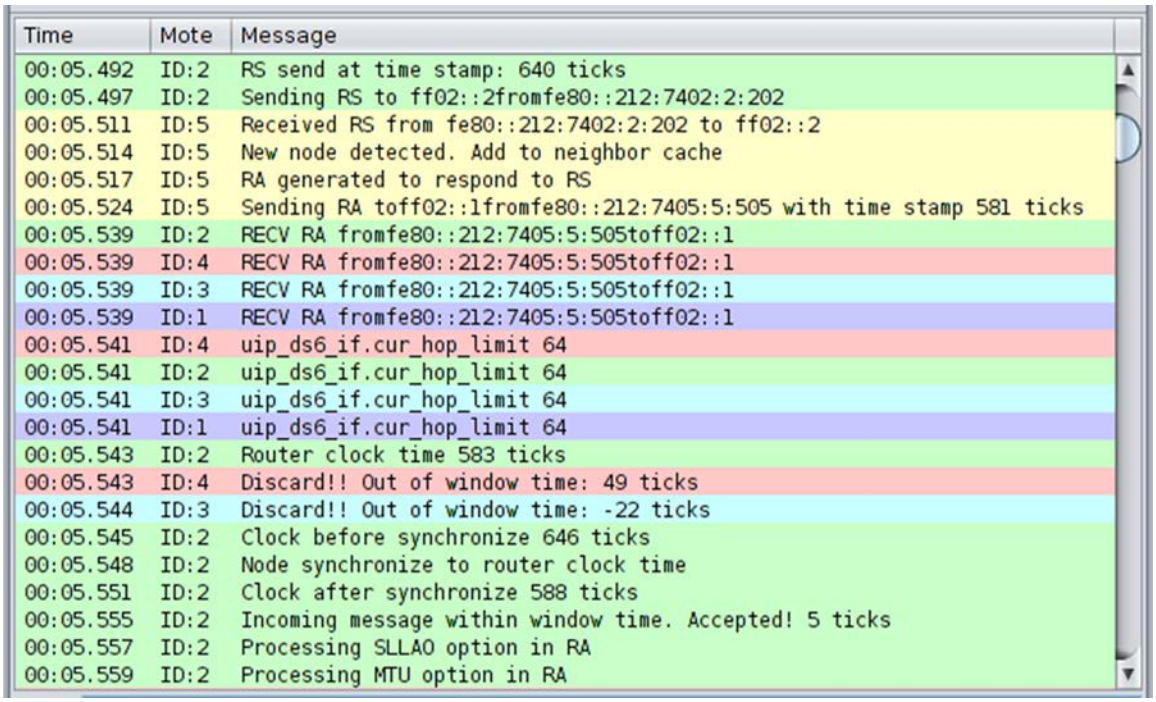
E. TIMESTAMP OPTION WITH TIME-SYNCHRONIZATION MECHANISM

In this simulation, the time-synchronization mechanism, described in Chapter IV, was implemented to synchronize the local clock of the nodes in the network. The events of the simulation were captured and shown in Figure 20. At 00:05.492 s, ID:2 generated a RS message with a Timestamp of 640 ticks and sent the RS message to a multicast address ff02::2 at 00:05.492 s. At 00:05.511 s, the router (ID:5) received the RS message sent from ID:2 and detected that ID:2 was a new node by checking the absence of the local link address of ID:2 in the neighbor cache. The router added ID:2 to its neighbor cache and generated an RA message with a Timestamp based on its own local clock, which was 581 ticks, and sent it to a multicast address ff02::01. There was a time difference of approximately 59 ticks between the local clock of ID:2 and the router. At 00:05.539 s, all the nodes, ID:1, 2, 3, and 4 received the RA message from the router.

At 00:05.544 s, ID:3 and ID:4 detected that this RA message was not meant for them and the Timestamp was out of the pre-defined window time; thus, they discarded the RA messages. At 00:05.546 s, ID:2 received its first RA message from the router and ID:2 used the Timestamp in the RA message to synchronize its own local clock. ID:2 synchronized with the router's clock time at 00:05.551 s and accepted the RA message from the router after verifying that the RA message is within the time window at 00:05.554 s. Subsequently, ID:2 processed the information found in the RA message at 00:05.556 s and 00:05.559 s. Once the time synchronization was completed in the first

exchange of RS and RA messages between the new hosts and router, normal verification of the Timestamp option was resumed.

The simulation analysis showed that the time-synchronization mechanism is able to recover the time synchronization in the network in an efficient and effective way. The hosts with de-synchronized clocks were forced to align with the router's local time during the early router discovery phase.



Time	Mote	Message
00:05.492	ID:2	RS send at time stamp: 640 ticks
00:05.497	ID:2	Sending RS to ff02::2fromfe80::212:7402:2:202
00:05.511	ID:5	Received RS from fe80::212:7402:2:202 to ff02::2
00:05.514	ID:5	New node detected. Add to neighbor cache
00:05.517	ID:5	RA generated to respond to RS
00:05.524	ID:5	Sending RA toff02::1fromfe80::212:7405:5:505 with time stamp 581 ticks
00:05.539	ID:2	RECV RA fromfe80::212:7405:5:505toff02::1
00:05.539	ID:4	RECV RA fromfe80::212:7405:5:505toff02::1
00:05.539	ID:3	RECV RA fromfe80::212:7405:5:505toff02::1
00:05.539	ID:1	RECV RA fromfe80::212:7405:5:505toff02::1
00:05.541	ID:4	uip_ds6_if.cur_hop_limit 64
00:05.541	ID:2	uip_ds6_if.cur_hop_limit 64
00:05.541	ID:3	uip_ds6_if.cur_hop_limit 64
00:05.541	ID:1	uip_ds6_if.cur_hop_limit 64
00:05.543	ID:2	Router clock time 583 ticks
00:05.543	ID:4	Discard!! Out of window time: 49 ticks
00:05.544	ID:3	Discard!! Out of window time: -22 ticks
00:05.545	ID:2	Clock before synchronize 646 ticks
00:05.548	ID:2	Node synchronize to router clock time
00:05.551	ID:2	Clock after synchronize 588 ticks
00:05.555	ID:2	Incoming message within window time. Accepted! 5 ticks
00:05.557	ID:2	Processing SLLAO option in RA
00:05.559	ID:2	Processing MTU option in RA

Figure 20. Timestamp Option with Synchronized Clocks Simulation Data

The captured data in the simulation was also analysed using Wireshark. The aim of the analysis was to verify if the Trust-ND option was implemented correctly. As shown in Figure 21, we analysed one of the RA messages sent by the ID:5 router to all the nodes in the local network (under the 15th row of the table). The source address, fe80::212:7405:5:505, belonged to the router, while ff02::1 is a multicast address that includes all the nodes in the local network. The data that is highlighted in orange is the 32-byte Trust ND option implemented in this thesis. The data in yellow and green were described in the earlier sub-section.

The Timestamp that is attached to this RA message was 1160, 04 88 in hexadecimal; thus, we observe in Figure 21 that there is a value of 88 04 in the Trust-ND option highlighted in purple. The hexadecimal is reversed as the most significant bit is transmitted first. The rest of the data in the Trust-ND option were zeros as the reserved field, SHA-1 value, and Nonce values were zeroed out.

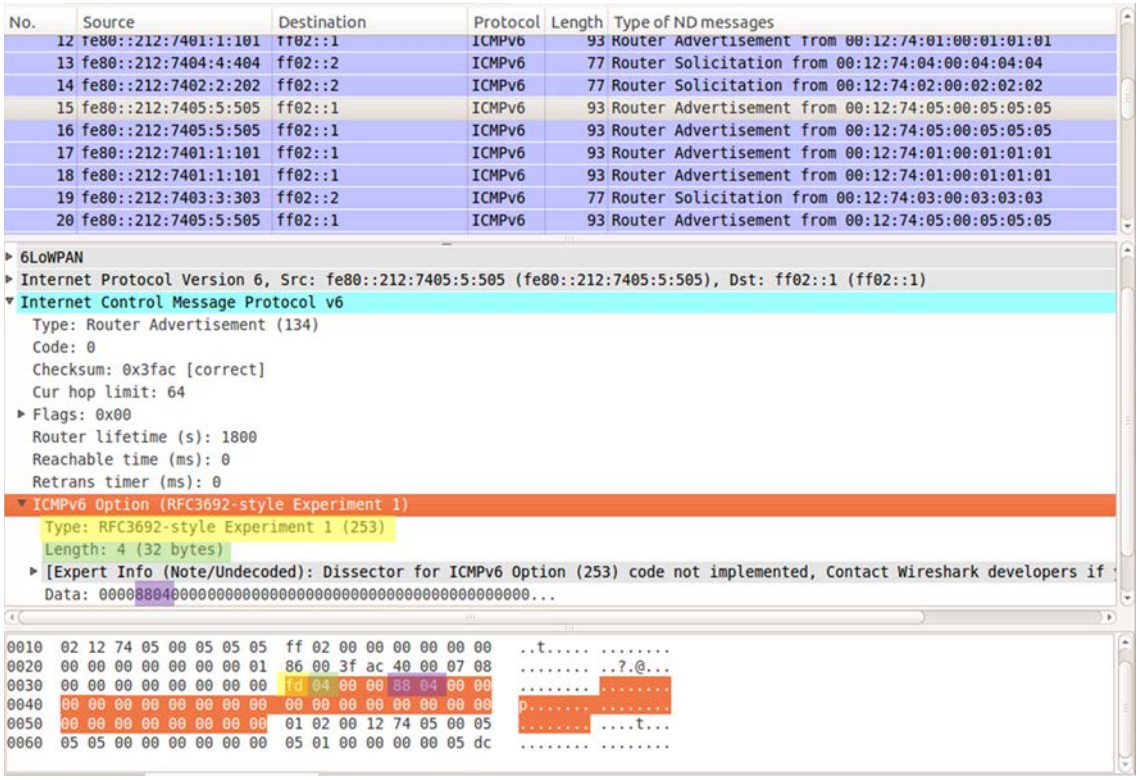


Figure 21. Trust-ND Option with Timestamp Data Breakdown Using Wireshark

F. CHAPTER SUMMARY

In this chapter, the results of five network simulations were shown and analyzed to show the effectiveness of the Nonce and Timestamp against replay attacks and to validate the proposed time-synchronization mechanism to synchronise the local clocks of each node in the network. The experiments showed that hard security mechanisms implemented in this thesis effectively mitigate against replay attacks. In addition, the

results also showed that the proposed time-synchronization mechanism functions as expected.

VII. CONCLUSION AND FUTURE WORK

A. SUMMARY AND CONCLUSIONS

In a typical military WSN application, remote sensors are deployed in the area of operations to have persistent surveillance. One of the important protocols in a WSN is the ND protocol. It is used by new nodes to join a network and for nodes in the network to establish communication between one another in the wireless environment. There are security concerns in the ND protocol, including possible DoS and replay attacks. Due to resource constraints, existing security mechanisms such as IPSec and SEND are not suitable for WSNs. The motivation of this thesis was to research an alternative security mechanism for the ND protocol based on existing literature.

In this thesis, we defined a set of key considerations for the design of the security mechanism for the ND protocol which led to the proposed security mechanism that used a combination of hard and soft security approaches. The hard security approach includes having a Trust-ND option on the ICMPv6 ND messages which includes Timestamp, Nonce, and SHA-1 functions. A simple time-synchronization mechanism was also proposed to enable the Timestamp function to work effectively. The soft security approach leverages the social interactions between the nodes in the network to identify malicious nodes.

Using the Contiki OS version 3.0 and Cooja simulator, we performed network simulations and showed the effectiveness of the Nonce and Timestamp against replay attacks. We have also demonstrated the effectiveness of the time-synchronization mechanism in the simulation.

B. FUTURE WORK

While we have defined a hybrid of hard and soft security mechanisms suitable for WSNs and have showed the effectiveness of the Nonce and Timestamp function against replay attacks, there are several areas that require further attention.

1. Evaluation of the Entire Hard Security Mechanism

We can evaluate the network performance of the Trust-ND hard security mechanism in terms of latency, bandwidth, and power consumption by simulating the full suite of the security functions, including Nonce, Timestamp, and SHA-1 in different attack scenarios. We can then compare the network performance with other existing security mechanisms. In addition, we need to assess the security gaps in the time-synchronization process as it is possible for a malicious node to alter the precision of the time synchronization information.

2. Evaluation of the Soft Security Mechanism

While we have defined the concept of the soft security mechanism and assessed that it met our design considerations, we need to further evaluate the effectiveness of the soft security mechanism and network performance through network simulation.

3. Scalability of the Security Mechanism

The scalability of the security mechanism is also important if we want to operate in a larger WSN. We need to evaluate if the security mechanism can handle large networks that consists of more nodes and whether it can withstand more attackers in the network. We also need to expand the single 6LoWPAN network to multiple networks where nodes are mobile and they can move from one network to another network.

APPENDIX. SOURCE CODE

Contiki OS Version 3.0 Uip-nd6.c file

```
#include "sys/rtimer.h"

#include <string.h>

#include "net/ipv6/uip-icmp6.h"

#include "net/ipv6/uip-nd6.h"

#include "net/ipv6/uip-ds6.h"

#include "net/ip/uip-nameserver.h"

#include "lib/random.h"

#include "lib/memb.h"

// #include <time.h>

#include "lib/memb.h"


// #include "net/rime/timesynch.h"


// #include "sys/node-id.h"

/*---------------------------------------------------------------------*/

#define DEBUG 1

#include "net/ip/uip-debug.h"


#if UIP_LOGGING

#include <stdio.h>

void uip_log(char *msg);


#define UIP_LOG(m) uip_log(m)

#else

#define UIP_LOG(m)

#endif /* UIP_LOGGING == 1 */
```

```
/*---------------------------------------------------------------*/
/** @{ */
/** \name Pointers to the header structures.
 *  All pointers except UIP_IP_BUF depend on uip_ext_len, which at
 *  packet reception, is the total length of the extension headers.
 *
 *  The pointer to ND6 options header also depends on nd6_opt_offset,
 *  which we set in each function.
 *
 *  Care should be taken when manipulating these buffers about the
 *  value of these length variables
 */

#define UIP_IP_BUF              ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])  /**< Pointer to IP header */
#define UIP_ICMP_BUF           ((struct uip_icmp_hdr *)&uip_buf[uip_l2_l3_hdr_len])  /**< Pointer to ICMP header*/
/**@{  Pointers to messages just after icmp header */
#define UIP_ND6_RS_BUF          ((uip_nd6_rs *)&uip_buf[uip_l2_l3_icmp_hdr_len])
#define UIP_ND6_RA_BUF          ((uip_nd6_ra *)&uip_buf[uip_l2_l3_icmp_hdr_len])
#define UIP_ND6_NS_BUF          ((uip_nd6_ns *)&uip_buf[uip_l2_l3_icmp_hdr_len])
#define UIP_ND6_NA_BUF          ((uip_nd6_na *)&uip_buf[uip_l2_l3_icmp_hdr_len])
/** @} */
/** Pointer to ND option */
#define UIP_ND6_OPT_HDR_BUF  ((uip_nd6_opt_hdr *)&uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset])
#define UIP_ND6_OPT_PREFIX_BUF ((uip_nd6_opt_prefix_info *)&uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset])
#define UIP_ND6_OPT_MTU_BUF ((uip_nd6_opt_mtu *)&uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset])
```

```
#define UIP_ND6_OPT_RDNSS_BUF ((uip_nd6_opt_dns *)&uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset])
#define UIP_ND6_OPT_TRUST_BUF ((uip_nd6_opt_trust *)&uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset])
/** @} */

#if UIP_ND6_SEND_NA || UIP_ND6_SEND_RA || !UIP_CONF_ROUTER
static uint8_t nd6_opt_offset;                  /** Offset from the end of the icmpv6 header to the option in uip_buf*/
static uint8_t *nd6_opt_llao;   /**  Pointer to llao option in uip_buf */
static uip_ds6_nbr_t *nbr; /**  Pointer to a nbr cache entry*/
static uip_ds6_defrt_t *defrt; /**  Pointer to a router list entry */
static uip_ds6_addr_t *addr; /**  Pointer to an interface address */
#endif /* UIP_ND6_SEND_NA || UIP_ND6_SEND_RA || !UIP_CONF_ROUTER */

#if !UIP_CONF_ROUTER            // TBD see if we move it to ra_input
static uip_nd6_opt_prefix_info *nd6_opt_prefix_info; /**  Pointer to prefix information option in uip_buf */
static uip_ipaddr_t ipaddr;

#endif
static uip_ds6_prefix_t *prefix; /**  Pointer to a prefix list entry */
static volatile uint32_t copy_nonce = 0;
static volatile uint32_t copy_timestamp = 0;
struct stimer send_attck_timer;
/*---------------------------------------------------------------------*/
/* create a llao */
static void
create_llao(uint8_t *llao, uint8_t type) {
  llao[UIP_ND6_OPT_TYPE_OFFSET] = type;
  llao[UIP_ND6_OPT_LEN_OFFSET] = UIP_ND6_OPT_LLAO_LEN >> 3;
```

```
  memcpy(&llao[UIP_ND6_OPT_DATA_OFFSET], &uip_lladdr, UIP_LLADDR_LEN);
  /* padding on some */
  memset(&llao[UIP_ND6_OPT_DATA_OFFSET + UIP_LLADDR_LEN], 0,
         UIP_ND6_OPT_LLAO_LEN - 2 - UIP_LLADDR_LEN);
}

/*---------------------------------------------------------------------*/
typedef struct security_neighbor {
  struct security_neighbor *next;
  // uint8_t no_id;
  // uint8_t isMatching_Ip_and_Nonce;
  uip_ip6addr_t ipv6_naddr;
} security_neighbor;
uint8_t is_sec_init = 0;

/*---------------------------------------------------------------------*/
// static void *LIST_CONCAT(neighbor_RT_security_list,_list) = NULL;

// list_t neighbor_RT_security_list = (list_t)&LIST_CONCAT(neighbor_RT_security_list,_list);
LIST(neighbor_table);
MEMB(neighbor_mem, struct security_neighbor, 25);
struct security_neighbor * look_for_neigbour(uip_ip6addr_t *IP_ADR)
{
  struct security_neighbor *s = list_head(neighbor_table);
  while(s != NULL) {
     if(uip_ipaddr_cmp(&s->ipv6_naddr, &IP_ADR))
        return s ;
     s = s->next;
  }
```

```
  return NULL;
}
#if UIP_ND6_SEND_NA
static void
ns_input(void)
{
  uint8_t flags;
  PRINTF("Received NS from ");
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
  PRINTF(" to ");
  PRINT6ADDR(&UIP_IP_BUF->destipaddr);
  PRINTF(" with target address");
  PRINT6ADDR((uip_ipaddr_t *) (&UIP_ND6_NS_BUF->tgtipaddr));
  PRINTF("\n");
  UIP_STAT(++uip_stat.nd6.recv);

#if UIP_CONF_IPV6_CHECKS
  if((UIP_IP_BUF->ttl != UIP_ND6_HOP_LIMIT) ||
     (uip_is_addr_mcast(&UIP_ND6_NS_BUF->tgtipaddr)) ||
     (UIP_ICMP_BUF->icode != 0)) {
    PRINTF("NS received is bad\n");
    goto discard;
  }
#endif /* UIP_CONF_IPV6_CHECKS */

  /* Options processing */
  nd6_opt_llao = NULL;
  nd6_opt_offset = UIP_ND6_NS_LEN;
  while(uip_l3_icmp_hdr_len + nd6_opt_offset < uip_len) {
```

```
#if UIP_CONF_IPV6_CHECKS
    if(UIP_ND6_OPT_HDR_BUF->len == 0) {
      PRINTF("NS received is bad\n");
      goto discard;
    }
#endif /* UIP_CONF_IPV6_CHECKS */
    switch (UIP_ND6_OPT_HDR_BUF->type) {
    case UIP_ND6_OPT_SLLAO:
      nd6_opt_llao = &uip_buf[uip_l2_l3_icmp_hdr_len + nd6_opt_offset];
#if UIP_CONF_IPV6_CHECKS
      /* There must be NO option in a DAD NS */
      if(uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
        PRINTF("NS received is bad\n");
        goto discard;
      } else {
#endif /*UIP_CONF_IPV6_CHECKS */
        nbr = uip_ds6_nbr_lookup(&UIP_IP_BUF->srcipaddr);
        if(nbr == NULL) {
          uip_ds6_nbr_add(&UIP_IP_BUF->srcipaddr,
        (uip_lladdr_t *)&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
        0, NBR_STALE);
        } else {
          uip_lladdr_t *lladdr = (uip_lladdr_t *)uip_ds6_nbr_get_ll(nbr);
          if(memcmp(&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
        lladdr, UIP_LLADDR_LEN) != 0) {
            memcpy(lladdr, &nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
       UIP_LLADDR_LEN);
            nbr->state = NBR_STALE;
          } else {
```

```
          if(nbr->state == NBR_INCOMPLETE) {
            nbr->state = NBR_STALE;
          }
        }
      }
#if UIP_CONF_IPV6_CHECKS
    }
#endif /*UIP_CONF_IPV6_CHECKS */
    break;
  default:
    PRINTF("ND option not supported in NS");
    break;
  }
  nd6_opt_offset += (UIP_ND6_OPT_HDR_BUF->len << 3);
 }

 addr = uip_ds6_addr_lookup(&UIP_ND6_NS_BUF->tgtipaddr);
 if(addr != NULL) {
#if UIP_ND6_DEF_MAXDADNS > 0
  if(uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
    /* DAD CASE */
#if UIP_CONF_IPV6_CHECKS
    if(!uip_is_addr_solicited_node(&UIP_IP_BUF->destipaddr)) {
      PRINTF("NS received is bad\n");
      goto discard;
    }
#endif /* UIP_CONF_IPV6_CHECKS */
    if(addr->state != ADDR_TENTATIVE) {
      uip_create_linklocal_allnodes_mcast(&UIP_IP_BUF->destipaddr);
```

```
        uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, &UIP_IP_BUF->destipaddr);
        flags = UIP_ND6_NA_FLAG_OVERRIDE;
        goto create_na;
      } else {
        /** \todo if I sent a NS before him, I win */
        uip_ds6_dad_failed(addr);
        goto discard;
      }
#else /* UIP_ND6_DEF_MAXDADNS > 0 */
    if(uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
      /* DAD CASE */
      goto discard;
#endif /* UIP_ND6_DEF_MAXDADNS > 0 */
    }
#if UIP_CONF_IPV6_CHECKS
    if(uip_ds6_is_my_addr(&UIP_IP_BUF->srcipaddr)) {
        /**
         * \NOTE do we do something here? we both are using the same address.
         * If we are doing dad, we could cancel it, though we should receive a
         * NA in response of DAD NS we sent, hence DAD will fail anyway. If we
         * were not doing DAD, it means there is a duplicate in the network!
         */
      PRINTF("NS received is bad\n");
      goto discard;
    }
#endif /*UIP_CONF_IPV6_CHECKS */

    /* Address resolution case */
    if(uip_is_addr_solicited_node(&UIP_IP_BUF->destipaddr)) {
```

```
      uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, &UIP_IP_BUF->srcipaddr);
      uip_ipaddr_copy(&UIP_IP_BUF->srcipaddr, &UIP_ND6_NS_BUF->tgtipaddr);
      flags = UIP_ND6_NA_FLAG_SOLICITED | UIP_ND6_NA_FLAG_OVERRIDE;
      goto create_na;
    }

    /* NUD CASE */
    if(uip_ds6_addr_lookup(&UIP_IP_BUF->destipaddr) == addr) {
      uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, &UIP_IP_BUF->srcipaddr);
      uip_ipaddr_copy(&UIP_IP_BUF->srcipaddr, &UIP_ND6_NS_BUF->tgtipaddr);
      flags = UIP_ND6_NA_FLAG_SOLICITED | UIP_ND6_NA_FLAG_OVERRIDE;
      goto create_na;
    } else {
#if UIP_CONF_IPV6_CHECKS
      PRINTF("NS received is bad\n");
      goto discard;
#endif /* UIP_CONF_IPV6_CHECKS */
    }
  } else {
    goto discard;
  }


create_na:
    /* If the node is a router it should set R flag in NAs */
#if UIP_CONF_ROUTER
    flags = flags | UIP_ND6_NA_FLAG_ROUTER;
#endif
  uip_ext_len = 0;
```

```
UIP_IP_BUF->vtc = 0x60;
UIP_IP_BUF->tcflow = 0;
UIP_IP_BUF->flow = 0;
UIP_IP_BUF->len[0] = 0;       /* length will not be more than 255 */
UIP_IP_BUF->len[1] = UIP_ICMPH_LEN + UIP_ND6_NA_LEN + UIP_ND6_OPT_LLAO_LEN;
UIP_IP_BUF->proto = UIP_PROTO_ICMP6;
UIP_IP_BUF->ttl = UIP_ND6_HOP_LIMIT;

UIP_ICMP_BUF->type = ICMP6_NA;
UIP_ICMP_BUF->icode = 0;

UIP_ND6_NA_BUF->flagsreserved = flags;
memcpy(&UIP_ND6_NA_BUF->tgtipaddr, &addr->ipaddr, sizeof(uip_ipaddr_t));

create_llao(&uip_buf[uip_l2_l3_icmp_hdr_len + UIP_ND6_NA_LEN],
            UIP_ND6_OPT_TLLAO);

UIP_ICMP_BUF->icmpchksum = 0;
UIP_ICMP_BUF->icmpchksum = ~uip_icmp6chksum();

uip_len =
  UIP_IPH_LEN + UIP_ICMPH_LEN + UIP_ND6_NA_LEN + UIP_ND6_OPT_LLAO_LEN;

UIP_STAT(++uip_stat.nd6.sent);
PRINTF("Sending NA to ");
PRINT6ADDR(&UIP_IP_BUF->destipaddr);
PRINTF(" from ");
PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
PRINTF(" with target address ");
```

```
  PRINT6ADDR(&UIP_ND6_NA_BUF->tgtipaddr);
  PRINTF("\n");
  return;

discard:
  uip_len = 0;
  return;
}
#endif /* UIP_ND6_SEND_NA */



/*---------------------------------------------------------------------*/
void
uip_nd6_ns_output(uip_ipaddr_t * src, uip_ipaddr_t * dest, uip_ipaddr_t * tgt)
{
  uip_ext_len = 0;
  UIP_IP_BUF->vtc = 0x60;
  UIP_IP_BUF->tcflow = 0;
  UIP_IP_BUF->flow = 0;
  UIP_IP_BUF->proto = UIP_PROTO_ICMP6;
  UIP_IP_BUF->ttl = UIP_ND6_HOP_LIMIT;

  if(dest == NULL) {
    uip_create_solicited_node(tgt, &UIP_IP_BUF->destipaddr);
  } else {
    uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, dest);
  }
  UIP_ICMP_BUF->type = ICMP6_NS;
  UIP_ICMP_BUF->icode = 0;
```

```
UIP_ND6_NS_BUF->reserved = 0;
uip_ipaddr_copy((uip_ipaddr_t *) &UIP_ND6_NS_BUF->tgtipaddr, tgt);
UIP_IP_BUF->len[0] = 0;       /* length will not be more than 255 */
/*
 * check if we add a SLLAO option: for DAD, MUST NOT, for NUD, MAY
 * (here yes), for Address resolution , MUST
 */
if(!(uip_ds6_is_my_addr(tgt))) {
  if(src != NULL) {
    uip_ipaddr_copy(&UIP_IP_BUF->srcipaddr, src);
  } else {
    uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, &UIP_IP_BUF->destipaddr);
  }
  if (uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
    PRINTF("Dropping NS due to no suitable source address\n");
    uip_len = 0;
    return;
  }
  UIP_IP_BUF->len[1] =
    UIP_ICMPH_LEN + UIP_ND6_NS_LEN + UIP_ND6_OPT_LLAO_LEN;

  create_llao(&uip_buf[uip_l2_l3_icmp_hdr_len + UIP_ND6_NS_LEN],
  UIP_ND6_OPT_SLLAO);

  uip_len =
    UIP_IPH_LEN + UIP_ICMPH_LEN + UIP_ND6_NS_LEN + UIP_ND6_OPT_LLAO_LEN;
} else {
  uip_create_unspecified(&UIP_IP_BUF->srcipaddr);
  UIP_IP_BUF->len[1] = UIP_ICMPH_LEN + UIP_ND6_NS_LEN;
```

```
    uip_len = UIP_IPH_LEN + UIP_ICMPH_LEN + UIP_ND6_NS_LEN;
  }

  UIP_ICMP_BUF->icmpchksum = 0;
  UIP_ICMP_BUF->icmpchksum = ~uip_icmp6chksum();

  UIP_STAT(++uip_stat.nd6.sent);
  PRINTF("Sending NS to");
  PRINT6ADDR(&UIP_IP_BUF->destipaddr);
  PRINTF("from");
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
  PRINTF("with target address");
  PRINT6ADDR(tgt);
  PRINTF("\n");
  return;
}
#if UIP_ND6_SEND_NA
/*---------------------------------------------------------------------*/
/**
 * Neighbor Advertisement Processing
 *
 * we might have to send a pkt that had been buffered while address
 * resolution was performed (if we support buffering, see UIP_CONF_QUEUE_PKT)
 *
 * As per RFC 4861, on link layer that have addresses, TLLAO options MUST be
 * included when responding to multicast solicitations, SHOULD be included in
 * response to unicast (here we assume it is for now)
 *
 * NA can be received after sending NS for DAD, Address resolution or NUD. Can
```

```
 * be unsolicited as well.
 * It can trigger update of the state of the neighbor in the neighbor cache,
 * router in the router list.
 * If the NS was for DAD, it means DAD failed
 *
 */
static void
na_input(void)
{
  uint8_t is_llchange;
  uint8_t is_router;
  uint8_t is_solicited;
  uint8_t is_override;

  PRINTF("Received NA from");
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
  PRINTF("to");
  PRINT6ADDR(&UIP_IP_BUF->destipaddr);
  PRINTF("with target address");
  PRINT6ADDR((uip_ipaddr_t *) (&UIP_ND6_NA_BUF->tgtipaddr));
  PRINTF("\n");
  UIP_STAT(++uip_stat.nd6.recv);

  /*
   * booleans. the three last one are not 0 or 1 but 0 or 0x80, 0x40, 0x20
   * but it works. Be careful though, do not use tests such as is_router == 1
   */
  is_llchange = 0;
  is_router = ((UIP_ND6_NA_BUF->flagsreserved & UIP_ND6_NA_FLAG_ROUTER));
```

```
  is_solicited =
    ((UIP_ND6_NA_BUF->flagsreserved & UIP_ND6_NA_FLAG_SOLICITED));
  is_override =
    ((UIP_ND6_NA_BUF->flagsreserved & UIP_ND6_NA_FLAG_OVERRIDE));

#if UIP_CONF_IPV6_CHECKS
  if((UIP_IP_BUF->ttl != UIP_ND6_HOP_LIMIT) ||
     (UIP_ICMP_BUF->icode != 0) ||
     (uip_is_addr_mcast(&UIP_ND6_NA_BUF->tgtipaddr)) ||
     (is_solicited && uip_is_addr_mcast(&UIP_IP_BUF->destipaddr))) {
    PRINTF("NA received is bad\n");
    goto discard;
  }
#endif /*UIP_CONF_IPV6_CHECKS */

  /* Options processing: we handle TLLAO, and must ignore others */
  nd6_opt_offset = UIP_ND6_NA_LEN;
  nd6_opt_llao = NULL;
  while(uip_l3_icmp_hdr_len + nd6_opt_offset < uip_len) {
#if UIP_CONF_IPV6_CHECKS
    if(UIP_ND6_OPT_HDR_BUF->len == 0) {
      PRINTF("NA received is bad\n");
      goto discard;
    }
#endif /*UIP_CONF_IPV6_CHECKS */
    switch (UIP_ND6_OPT_HDR_BUF->type) {
    case UIP_ND6_OPT_TLLAO:
      nd6_opt_llao = (uint8_t *)UIP_ND6_OPT_HDR_BUF;
      break;
```

```
    default:
      PRINTF("ND option not supported in NA\n");
      break;
    }
    nd6_opt_offset += (UIP_ND6_OPT_HDR_BUF->len << 3);
  }
  addr = uip_ds6_addr_lookup(&UIP_ND6_NA_BUF->tgtipaddr);
  /* Message processing, including TLLAO if any */
  if(addr != NULL) {
#if UIP_ND6_DEF_MAXDADNS > 0
    if(addr->state == ADDR_TENTATIVE) {
      uip_ds6_dad_failed(addr);
    }
#endif /*UIP_ND6_DEF_MAXDADNS > 0 */
    PRINTF("NA received is bad\n");
    goto discard;
  } else {
    uip_lladdr_t *lladdr;
    nbr = uip_ds6_nbr_lookup(&UIP_ND6_NA_BUF->tgtipaddr);
    lladdr = (uip_lladdr_t *)uip_ds6_nbr_get_ll(nbr);
    if(nbr == NULL) {
      goto discard;
    }
    if(nd6_opt_llao != 0) {
      is_llchange =
        memcmp(&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET], (void *)lladdr,
               UIP_LLADDR_LEN);
    }
    if(nbr->state == NBR_INCOMPLETE) {
```

```
 if(nd6_opt_llao == NULL) {
  goto discard;
 }
 memcpy(lladdr, &nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
  UIP_LLADDR_LEN);
 if(is_solicited) {
  nbr->state = NBR_REACHABLE;
  nbr->nscount = 0;

  /* reachable time is stored in ms */
  stimer_set(&(nbr->reachable), uip_ds6_if.reachable_time / 1000);

 } else {
  nbr->state = NBR_STALE;
 }
 nbr->isrouter = is_router;
} else {
 if(!is_override && is_llchange) {
  if(nbr->state == NBR_REACHABLE) {
   nbr->state = NBR_STALE;
  }
  goto discard;
 } else {
  if(is_override || (!is_override && nd6_opt_llao != 0 && !is_llchange)
    || nd6_opt_llao == 0) {
   if(nd6_opt_llao != 0) {
    memcpy(lladdr, &nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
  UIP_LLADDR_LEN);
   }
```

```
        if(is_solicited) {
          nbr->state = NBR_REACHABLE;
          /* reachable time is stored in ms */
          stimer_set(&(nbr->reachable), uip_ds6_if.reachable_time / 1000);
        } else {
          if(nd6_opt_llao != 0 && is_llchange) {
            nbr->state = NBR_STALE;
          }
        }
      }
    }
    if(nbr->isrouter && !is_router) {
      defrt = uip_ds6_defrt_lookup(&UIP_IP_BUF->srcipaddr);
      if(defrt != NULL) {
        uip_ds6_defrt_rm(defrt);
      }
    }
    nbr->isrouter = is_router;
  }
 }
#if UIP_CONF_IPV6_QUEUE_PKT
 /* The nbr is now reachable, check if we had buffered a pkt for it */
 /*if(nbr->queue_buf_len != 0) {
   uip_len = nbr->queue_buf_len;
   memcpy(UIP_IP_BUF, nbr->queue_buf, uip_len);
   nbr->queue_buf_len = 0;
   return;
   }*/
 if(uip_packetqueue_buflen(&nbr->packethandle) != 0) {
```

```
    uip_len = uip_packetqueue_buflen(&nbr->packethandle);
    memcpy(UIP_IP_BUF, uip_packetqueue_buf(&nbr->packethandle), uip_len);
    uip_packetqueue_free(&nbr->packethandle);
    return;
  }

#endif /*UIP_CONF_IPV6_QUEUE_PKT */

discard:
  uip_len = 0;
  return;
}
#endif /* UIP_ND6_SEND_NA */
static uint32_t recv_nonce;
#if UIP_CONF_ROUTER
#if UIP_ND6_SEND_RA
/*---------------------------------------------------------------------------*/
static void
rs_input(void)
{

  PRINTF("Received RS from ");
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
  PRINTF(" to ");
  PRINT6ADDR(&UIP_IP_BUF->destipaddr);
  PRINTF("\n");
  UIP_STAT(++uip_stat.nd6.recv);

  if(is_sec_init == 0)
```

```
  {
    is_sec_init = 1;
    memb_init(&neighbor_mem);
    list_init(neighbor_table);
  }

#if UIP_CONF_IPV6_CHECKS
  /*
   * Check hop limit / icmp code
   * target address must not be multicast
   * if the NA is solicited, dest must not be multicast
   */
  if((UIP_IP_BUF->ttl != UIP_ND6_HOP_LIMIT) || (UIP_ICMP_BUF->icode != 0)) {
    PRINTF("RS received is bad 1\n");
    goto discard;
  }
#endif /*UIP_CONF_IPV6_CHECKS */

  /* Only valid option is Source Link-Layer Address option any thing
     else is discarded */
  nd6_opt_offset = UIP_ND6_RS_LEN;
  nd6_opt_llao = NULL;
uint32_t recv_tmstamp;
uint32_t time_now ;
int32_t time_diff;
  while(uip_l3_icmp_hdr_len + nd6_opt_offset < uip_len) {
#if UIP_CONF_IPV6_CHECKS
    if(UIP_ND6_OPT_HDR_BUF->len == 0) {
      PRINTF("RS received is bad 2\n");
```

```
      goto discard;
    }
#endif /*UIP_CONF_IPV6_CHECKS */
    switch (UIP_ND6_OPT_HDR_BUF->type) {
    case UIP_ND6_OPT_SLLAO:
      nd6_opt_llao = (uint8_t *)UIP_ND6_OPT_HDR_BUF;
      break;
      case UIP_ND6_OPT_TRUST:

        recv_tmstamp = UIP_ND6_OPT_TRUST_BUF->timestamp;
        time_now = (uint32_t)clock_time();
        // struct timeval tv;
        // gettimeofday(&tv,NULL);
        // time_t timestamp_sec;
        // time(&timestamp_sec);
        // time_now = (uint32_t)tv.tv_sec;
      #if SCEN2
        struct security_neighbor *ne = NULL;
        #if SYN_UTC
        struct security_neighbor *snode = list_head(neighbor_table);
        while(snode != NULL) {
           if(uip_ipaddr_cmp(&snode->ipv6_naddr, &UIP_IP_BUF->srcipaddr))
             break;
           snode = snode->next;
        }
        if (snode == NULL)
        {
          ne = memb_alloc(&neighbor_mem);
          //memcpy(&ne->ipv6_addr , &UIP_IP_BUF->srcipaddr, sizeof(uip_ip6addr_t));
```

```
      uip_ipaddr_copy(&ne->ipv6_naddr, &UIP_IP_BUF->srcipaddr);
      list_add(neighbor_table , ne);
      PRINTF("New node detected. Add to neighbor cache\n");
    }

    #endif

    time_diff = time_now - recv_tmstamp;
    if ((time_diff <= 0) || (time_diff >= 10))
    {
      if (ne == NULL){
        PRINTF("Discard!! Out of window time: %d ticks\n", time_diff);
        goto discard;
      }
    }
    else
    {
      PRINTF("Incoming message in window time. Accepted! %d ticks\n", time_diff);
    }
  #endif
    recv_nonce = UIP_ND6_OPT_TRUST_BUF->nonce;

  break;
 default:
  PRINTF("ND option not supported in RS\n");
  break;
 }
 nd6_opt_offset += (UIP_ND6_OPT_HDR_BUF->len << 3);
}
```

```
  /* Options processing: only SLLAO */
  if(nd6_opt_llao != NULL) {
#if UIP_CONF_IPV6_CHECKS
    if(uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
      PRINTF("RS received is bad\n");
      goto discard;
    } else {
#endif /*UIP_CONF_IPV6_CHECKS */
      if((nbr = uip_ds6_nbr_lookup(&UIP_IP_BUF->srcipaddr)) == NULL) {
        /* we need to add the neighbor */
        uip_ds6_nbr_add(&UIP_IP_BUF->srcipaddr,
                        (uip_lladdr_t *)&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET], 0, NBR_STALE);
      } else {
        /* If LL address changed, set neighbor state to stale */
        if(memcmp(&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
           uip_ds6_nbr_get_ll(nbr), UIP_LLADDR_LEN) != 0) {
          uip_ds6_nbr_t nbr_data = *nbr;
          uip_ds6_nbr_rm(nbr);
          nbr = uip_ds6_nbr_add(&UIP_IP_BUF->srcipaddr,
                                (uip_lladdr_t *)&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET], 0, NBR_STALE);
          nbr->reachable = nbr_data.reachable;
          nbr->sendns = nbr_data.sendns;
          nbr->nscount = nbr_data.nscount;
        }
        nbr->isrouter = 0;
      }
#if UIP_CONF_IPV6_CHECKS
    }
#endif /*UIP_CONF_IPV6_CHECKS */
```

```
  }

  /* Schedule a sollicited RA */
  uip_ds6_send_ra_sollicited();

discard:
  uip_len = 0;
  return;
}
#endif /* UIP_ND6_SEND_RA */
#endif /* UIP_CONF_ROUTER */
/*---------------------------------------------------------------------------*/
void
uip_nd6_ra_output(uip_ipaddr_t * dest)
{
  uip_ext_len = 0;
  UIP_IP_BUF->vtc = 0x60;
  UIP_IP_BUF->tcflow = 0;
  UIP_IP_BUF->flow = 0;
  UIP_IP_BUF->proto = UIP_PROTO_ICMP6;
  UIP_IP_BUF->ttl = UIP_ND6_HOP_LIMIT;

  if(dest == NULL) {
    uip_create_linklocal_allnodes_mcast(&UIP_IP_BUF->destipaddr);
  } else {
    /* For sollicited RA */
    uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, dest);
  }
  uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, &UIP_IP_BUF->destipaddr);
```

```
  UIP_ICMP_BUF->type = ICMP6_RA;
  UIP_ICMP_BUF->icode = 0;

  UIP_ND6_RA_BUF->cur_ttl = uip_ds6_if.cur_hop_limit;

  UIP_ND6_RA_BUF->flags_reserved =
    (UIP_ND6_M_FLAG << 7) | (UIP_ND6_O_FLAG << 6);

  UIP_ND6_RA_BUF->router_lifetime = uip_htons(UIP_ND6_ROUTER_LIFETIME);
  //UIP_ND6_RA_BUF->reachable_time = uip_htonl(uip_ds6_if.reachable_time);
  //UIP_ND6_RA_BUF->retrans_timer = uip_htonl(uip_ds6_if.retrans_timer);
  UIP_ND6_RA_BUF->reachable_time = 0;
  UIP_ND6_RA_BUF->retrans_timer = 0;

  uip_len = UIP_IPH_LEN + UIP_ICMPH_LEN + UIP_ND6_RA_LEN;
  nd6_opt_offset = UIP_ND6_RA_LEN;
uint32_t time_conv;
#if TRUST_OPTION

    clock_time_t  ct;
    ct = clock_time();

    time_conv = (uint32_t)ct;
  UIP_ND6_OPT_TRUST_BUF->type = UIP_ND6_OPT_TRUST;
  UIP_ND6_OPT_TRUST_BUF->len = UIP_ND6_OPT_TRUST_LEN/8;
  UIP_ND6_OPT_TRUST_BUF->reserved = 0;
  if (copy_timestamp == 0)
  {
```

```
  UIP_ND6_OPT_TRUST_BUF->timestamp = time_conv;
 }
 else
 {
  UIP_ND6_OPT_TRUST_BUF->timestamp = copy_timestamp;
 }

 if (copy_nonce == 0){

  UIP_ND6_OPT_TRUST_BUF->nonce = recv_nonce;

 }
 else
 {
  UIP_ND6_OPT_TRUST_BUF->nonce = copy_nonce;

 }
 //UIP_ND6_OPT_TRUST_BUF->sha_1[20] = {0};
 memset(UIP_ND6_OPT_TRUST_BUF->sha_1, 0, 20);
 nd6_opt_offset += UIP_ND6_OPT_TRUST_LEN;
 uip_len += UIP_ND6_OPT_TRUST_LEN;

/*PRINTF("RA is sent reply by RS message \n");*/
#endif
#if UIP_CONF_ROUTER
 /* Prefix list */
 for(prefix = uip_ds6_prefix_list;
     prefix < uip_ds6_prefix_list + UIP_DS6_PREFIX_NB; prefix++) {
  if((prefix->isused) && (prefix->advertise)) {
```

```
    UIP_ND6_OPT_PREFIX_BUF->type = UIP_ND6_OPT_PREFIX_INFO;
    UIP_ND6_OPT_PREFIX_BUF->len = UIP_ND6_OPT_PREFIX_INFO_LEN / 8;
    UIP_ND6_OPT_PREFIX_BUF->preflen = prefix->length;
    UIP_ND6_OPT_PREFIX_BUF->flagsreserved1 = prefix->l_a_reserved;
    UIP_ND6_OPT_PREFIX_BUF->validlt = uip_htonl(prefix->vlifetime);
    UIP_ND6_OPT_PREFIX_BUF->preferredlt =  uip_htonl(prefix->plifetime);
    UIP_ND6_OPT_PREFIX_BUF->reserved2 = 0;
    uip_ipaddr_copy(&(UIP_ND6_OPT_PREFIX_BUF->prefix), &(prefix->ipaddr));
    nd6_opt_offset += UIP_ND6_OPT_PREFIX_INFO_LEN;
    uip_len += UIP_ND6_OPT_PREFIX_INFO_LEN;
   }
  }
#endif /* !UIP_CONF_ROUTER */

  /* Source link-layer option */
  create_llao((uint8_t *)UIP_ND6_OPT_HDR_BUF, UIP_ND6_OPT_SLLAO);

  uip_len += UIP_ND6_OPT_LLAO_LEN;
  nd6_opt_offset += UIP_ND6_OPT_LLAO_LEN;

  /* MTU */
  UIP_ND6_OPT_MTU_BUF->type = UIP_ND6_OPT_MTU;
  UIP_ND6_OPT_MTU_BUF->len = UIP_ND6_OPT_MTU_LEN >> 3;
  UIP_ND6_OPT_MTU_BUF->reserved = 0;
  //UIP_ND6_OPT_MTU_BUF->mtu = uip_htonl(uip_ds6_if.link_mtu);
  UIP_ND6_OPT_MTU_BUF->mtu = uip_htonl(1500);

  uip_len += UIP_ND6_OPT_MTU_LEN;
  nd6_opt_offset += UIP_ND6_OPT_MTU_LEN;
```

```
#if UIP_ND6_RA_RDNSS
  if(uip_nameserver_count() > 0) {
    uint8_t i = 0;
    uip_ipaddr_t *ip = &UIP_ND6_OPT_RDNSS_BUF->ip;
    uip_ipaddr_t *dns = NULL;
    UIP_ND6_OPT_RDNSS_BUF->type = UIP_ND6_OPT_RDNSS;
    UIP_ND6_OPT_RDNSS_BUF->reserved = 0;
    UIP_ND6_OPT_RDNSS_BUF->lifetime = uip_nameserver_next_expiration();
    if(UIP_ND6_OPT_RDNSS_BUF->lifetime != UIP_NAMESERVER_INFINITE_LIFETIME) {
      UIP_ND6_OPT_RDNSS_BUF->lifetime -= clock_seconds();
    }
    while((dns = uip_nameserver_get(i)) != NULL) {
      uip_ipaddr_copy(ip++, dns);
      i++;
    }
    UIP_ND6_OPT_RDNSS_BUF->len = UIP_ND6_OPT_RDNSS_LEN + (i << 1);
    PRINTF("%d nameservers reported\n", i);
    uip_len += UIP_ND6_OPT_RDNSS_BUF->len << 3;
    nd6_opt_offset += UIP_ND6_OPT_RDNSS_BUF->len << 3;
  }
#endif /* UIP_ND6_RA_RDNSS */


  UIP_IP_BUF->len[0] = ((uip_len - UIP_IPH_LEN) >> 8);
  UIP_IP_BUF->len[1] = ((uip_len - UIP_IPH_LEN) & 0xff);


  /*ICMP checksum */
  UIP_ICMP_BUF->icmpchksum = 0;
  UIP_ICMP_BUF->icmpchksum = ~uip_icmp6chksum();
```

```
UIP_STAT(++uip_stat.nd6.sent);
PRINTF("Sending RA to");
PRINT6ADDR(&UIP_IP_BUF->destipaddr);
PRINTF("from");
PRINT6ADDR(&UIP_IP_BUF->srcipaddr);

#if TRUST_OPTION
if (copy_timestamp == 0)
{
 #if SCEN2
 if (time_conv > 100)
   {
    PRINTF(" with time stamp %u ticks", time_conv);
   }
 #endif
}
else
{
 PRINTF(" copy timestamp %u ticks", copy_timestamp);
}
if (copy_nonce == 0)
{
 #if SCEN1
 {
  PRINTF(" with nonce %u", recv_nonce);
 }
 #endif
}
```

```
  else
  {
     PRINTF(" with attack nonce %u", copy_nonce);
  }

  #endif

  PRINTF("\n");
  return;
}
static uint8_t is_frist = 0;
static int16_t pin_init_syn = 0;
#if !UIP_CONF_ROUTER
/*---------------------------------------------------------------------------*/
static uint32_t expected_nonce = 0;
void
uip_nd6_rs_output(void)
{
  uip_ext_len = 0;
  UIP_IP_BUF->vtc = 0x60;
  UIP_IP_BUF->tcflow = 0;
  UIP_IP_BUF->flow = 0;
  UIP_IP_BUF->proto = UIP_PROTO_ICMP6;
  UIP_IP_BUF->ttl = UIP_ND6_HOP_LIMIT;
  uip_create_linklocal_allrouters_mcast(&UIP_IP_BUF->destipaddr);
  uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, &UIP_IP_BUF->destipaddr);
  UIP_ICMP_BUF->type = ICMP6_RS;
  UIP_ICMP_BUF->icode = 0;
  UIP_IP_BUF->len[0] = 0;       /* length will not be more than 255 */
```

```
  nd6_opt_offset = UIP_ND6_RS_LEN;

  if(uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr)) {
    UIP_IP_BUF->len[1] = UIP_ICMPH_LEN + UIP_ND6_RS_LEN;
    uip_len = uip_l3_icmp_hdr_len + UIP_ND6_RS_LEN;
  } else {
    uip_len = uip_l3_icmp_hdr_len + UIP_ND6_RS_LEN + UIP_ND6_OPT_LLAO_LEN;
    UIP_IP_BUF->len[1] =
      UIP_ICMPH_LEN + UIP_ND6_RS_LEN + UIP_ND6_OPT_LLAO_LEN;

    create_llao((uint8_t *)UIP_ND6_OPT_HDR_BUF, UIP_ND6_OPT_SLLAO);
    nd6_opt_offset += UIP_ND6_OPT_LLAO_LEN;
  }
#if TRUST_OPTION && !ATTACKER_MODE
    int gen_nonce = random_rand();
    if (gen_nonce < 0)
      gen_nonce = - gen_nonce;
      {
          clock_time_t  ct = clock_time();
          // rtimer_clock_t time_pin = RTIMER_NOW();
          UIP_ND6_OPT_TRUST_BUF->type = UIP_ND6_OPT_TRUST;
          UIP_ND6_OPT_TRUST_BUF->len = UIP_ND6_OPT_TRUST_LEN / 8;
          UIP_ND6_OPT_TRUST_BUF->reserved = 0;
          UIP_ND6_OPT_TRUST_BUF->timestamp = (uint32_t)ct ;
          if (pin_init_syn == 0)
            pin_init_syn = (int16_t)UIP_ND6_OPT_TRUST_BUF->timestamp;
          #if SCEN2
          PRINTF("RS send at time stamp: %u ticks\n", UIP_ND6_OPT_TRUST_BUF->timestamp);
          #endif
```

```
      UIP_ND6_OPT_TRUST_BUF->nonce = gen_nonce % 10000;
      expected_nonce = UIP_ND6_OPT_TRUST_BUF->nonce;
      #if SCEN1
      PRINTF("RS generate nonce_id %u\n",UIP_ND6_OPT_TRUST_BUF->nonce );
      #endif
      // UIP_ND6_OPT_TRUST_BUF->sha_1[20] = {0};
      memset(UIP_ND6_OPT_TRUST_BUF->sha_1, 0, 20);
      is_frist = 0;
     }

   nd6_opt_offset += UIP_ND6_OPT_TRUST_LEN;
   uip_len += UIP_ND6_OPT_TRUST_LEN;
   UIP_IP_BUF->len[1] += UIP_ND6_OPT_TRUST_LEN;
 #endif
 UIP_ICMP_BUF->icmpchksum = 0;
 UIP_ICMP_BUF->icmpchksum = ~uip_icmp6chksum();

 UIP_STAT(++uip_stat.nd6.sent);
 PRINTF("Sending RS to ");
 PRINT6ADDR(&UIP_IP_BUF->destipaddr);
 PRINTF("from");
 PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
 PRINTF("\n");
 return;
}

/*---------------------------------------------------------------------------*/
/*
```

```
 * Process a Router Advertisement
 *
 * - Possible actions when receiving a RA: add router to router list,
 *   recalculate reachable time, update link hop limit, update retrans timer.
 * - If MTU option: update MTU.
 * - If SLLAO option: update entry in neighbor cache
 * - If prefix option: start autoconf, add prefix to prefix list
 */
void
ra_input(void)
{
  PRINTF("RECV RA from");
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
  PRINTF("to");
  PRINT6ADDR(&UIP_IP_BUF->destipaddr);
  PRINTF("\n");
  UIP_STAT(++uip_stat.nd6.recv);
  copy_nonce = 0;
#if UIP_CONF_IPV6_CHECKS
  if((UIP_IP_BUF->ttl != UIP_ND6_HOP_LIMIT) ||
     (!uip_is_addr_link_local(&UIP_IP_BUF->srcipaddr)) ||
     (UIP_ICMP_BUF->icode != 0)) {
    PRINTF("RA received is bad");
    goto discard;
  }
#endif /*UIP_CONF_IPV6_CHECKS */

  if(UIP_ND6_RA_BUF->cur_ttl != 0) {
    uip_ds6_if.cur_hop_limit = UIP_ND6_RA_BUF->cur_ttl;
```

```
 PRINTF("uip_ds6_if.cur_hop_limit %u\n", uip_ds6_if.cur_hop_limit);
}

if(UIP_ND6_RA_BUF->reachable_time != 0) {
 if(uip_ds6_if.base_reachable_time !=
    uip_ntohl(UIP_ND6_RA_BUF->reachable_time)) {
   uip_ds6_if.base_reachable_time = uip_ntohl(UIP_ND6_RA_BUF->reachable_time);
   uip_ds6_if.reachable_time = uip_ds6_compute_reachable_time();
 }
}
if(UIP_ND6_RA_BUF->retrans_timer != 0) {
 uip_ds6_if.retrans_timer = uip_ntohl(UIP_ND6_RA_BUF->retrans_timer);
}
    #if ATTACKER_MODE && !TRUST_OPTION
    if (copy_nonce == 0){
      PRINTF("Attacking \n");
      copy_nonce = 1;
      stimer_set(&send_attck_timer, 1);
      while (!stimer_expired(&send_attck_timer))
      {
       ;
      }
      stimer_restart(&send_attck_timer);
      //flooding

      //rtimer_arch_init();
      int i;
      for (i = 0; i < NUMBER_ATTACK; i++)
      {
```

```
            if(copy_nonce == 0)
              break;
            uip_ds6_send_ra_periodic();
            rtimer_clock_t time_pin1 = RTIMER_NOW();
            /*while ((RTIMER_NOW() - time_pin1) < (RTIMER_SECOND / 100))
            {
              ;
            }*/
          }
        }
      #endif


uint32_t recv_tmstamp;
uint32_t time_now ;
int32_t time_diff;
uint8_t low;
uint8_t high;
  /* Options processing */
  nd6_opt_offset = UIP_ND6_RA_LEN;
  while(uip_l3_icmp_hdr_len + nd6_opt_offset < uip_len) {
    if(UIP_ND6_OPT_HDR_BUF->len == 0) {
      PRINTF("RA received is bad");
      goto discard;
    }
    switch (UIP_ND6_OPT_HDR_BUF->type) {
    case UIP_ND6_OPT_TRUST:
    recv_tmstamp = UIP_ND6_OPT_TRUST_BUF->timestamp;
    // low = recv_tmstamp & 0XFF;
    // high = (recv_tmstamp >> 8) & 0xFF;
```

```
#if !ATTACKER_MODE && SCEN2

 #if SYN_UTC

  /*if ((high == 0 ) && (low == 101))
  {
    clock_set(10, 103);
  }*/
  if (pin_init_syn > 0)
  {
     PRINTF("Router clock time %u ticks\n", UIP_ND6_OPT_TRUST_BUF->timestamp);
     clock_time_t syn_time = (clock_time_t)recv_tmstamp;
     time_now = (uint32_t)clock_time();
     PRINTF("Clock before synchronize %u ticks\n", time_now);
     uint8_t delta_time;
     delta_time = time_now - pin_init_syn;
     syn_time = syn_time + delta_time/2;
     clock_set(syn_time, syn_time);
     pin_init_syn = -1; //already synching
     PRINTF("Node synchronize to router clock time \n");
     PRINTF("Clock after synchronize %u ticks \n", clock_time());
  }
 #endif
time_now = (uint32_t)clock_time();

time_diff = time_now - recv_tmstamp;
if ((time_diff <= 0 )|| (time_diff >= 6))
{
 PRINTF("Discard!! Out of window time: %d ticks\n", time_diff);
```

```
  goto discard;
}
else
{
  PRINTF("Incoming message within window time. Accepted! %d ticks\n", time_diff);
}
#endif
#if ATTACKER_MODE && SCEN2


  // if ((high == 0 ) && (low == 101))
  //     {;}
  // else
  // {
      stimer_set(&send_attck_timer, 1);
      copy_timestamp = UIP_ND6_OPT_TRUST_BUF->timestamp;
      while (!stimer_expired(&send_attck_timer))
      {
        ;
      }
      stimer_restart(&send_attck_timer);
      PRINTF("Replay with time stamp %u ticks\n", copy_timestamp);
      int i = 0;
      for (i = 0; i < NUMBER_ATTACK; i++)
      {

        uip_ds6_send_ra_periodic();

      }
```

```
  // }
#endif
  #if ATTACKER_MODE && SCEN1


    if (copy_nonce == 0){
      stimer_set(&send_attck_timer, 1);
      copy_nonce = UIP_ND6_OPT_TRUST_BUF->nonce;

      //delay CPU for 1 second to slower than router
      while (!stimer_expired(&send_attck_timer))
      {
        ;
      }
      stimer_restart(&send_attck_timer);
      //flooding
      PRINTF("Attacking with nonce %u \n", copy_nonce);
      //rtimer_arch_init();
      int i = 0;
      for (i = 0; i < NUMBER_ATTACK; i++)
      {
        if(copy_nonce == 0)
          break;

        uip_ds6_send_ra_periodic();
        //rtimer_clock_t time_pin = RTIMER_NOW();
        /*while ((RTIMER_NOW() - time_pin) < (RTIMER_SECOND)) //500 ms
        {
          ;
        }*/
```

```
      }
     }

  #endif



#if !ATTACKER_MODE && SCEN1
 //PRINTF("TRUST option in RA\n");
 //nd6_opt_trust = (uip_nd6_opt_trust *) UIP_ND6_OPT_TRUST_BUF;
if (expected_nonce != 0){
 static uip_ip6addr_t * router_ipaddr;
 if (UIP_ND6_OPT_TRUST_BUF->nonce != expected_nonce)
  {
   //goto discard;
   PRINTF("Different expected nonce\n");
  }
 else
 {
  if (is_frist == 0)
   {
    memcpy(&router_ipaddr, &UIP_IP_BUF->srcipaddr, sizeof(uip_ip6addr_t));
    PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
    PRINTF(" is router sending as expected_nonce %u. Accept!\n",expected_nonce);
    is_frist = 1;
   }
  else
  {
   if(memcmp(&router_ipaddr , &UIP_IP_BUF->srcipaddr, sizeof(uip_ip6addr_t)) != 0 )
   {
```

```
        PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
        PRINTF(" is attacker used the same nonce_id %u. Discard!!!\n",expected_nonce);
        goto discard;
      }
    }
  }
}
 #endif

 break;
 case UIP_ND6_OPT_SLLAO:
  PRINTF("Processing SLLAO option in RA\n");
  nd6_opt_llao = (uint8_t *) UIP_ND6_OPT_HDR_BUF;
  nbr = uip_ds6_nbr_lookup(&UIP_IP_BUF->srcipaddr);
  if(nbr == NULL) {
    nbr = uip_ds6_nbr_add(&UIP_IP_BUF->srcipaddr,
                          (uip_lladdr_t *)&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
        1, NBR_STALE);
  } else {
    uip_lladdr_t *lladdr = (uip_lladdr_t *)uip_ds6_nbr_get_ll(nbr);
    if(nbr->state == NBR_INCOMPLETE) {
      nbr->state = NBR_STALE;
    }
    if(memcmp(&nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
  lladdr, UIP_LLADDR_LEN) != 0) {
      memcpy(lladdr, &nd6_opt_llao[UIP_ND6_OPT_DATA_OFFSET],
 UIP_LLADDR_LEN);
      nbr->state = NBR_STALE;
    }
```

```
    nbr->isrouter = 1;
  }
  break;
case UIP_ND6_OPT_MTU:
  PRINTF("Processing MTU option in RA\n");
  uip_ds6_if.link_mtu =
    uip_ntohl(((uip_nd6_opt_mtu *) UIP_ND6_OPT_HDR_BUF)->mtu);
  break;
case UIP_ND6_OPT_PREFIX_INFO:
  PRINTF("Processing PREFIX option in RA\n");
  nd6_opt_prefix_info = (uip_nd6_opt_prefix_info *) UIP_ND6_OPT_HDR_BUF;
  if((uip_ntohl(nd6_opt_prefix_info->validlt) >=
      uip_ntohl(nd6_opt_prefix_info->preferredlt))
     && (!uip_is_addr_link_local(&nd6_opt_prefix_info->prefix))) {
    /* on-link flag related processing */
    if(nd6_opt_prefix_info->flagsreserved1 & UIP_ND6_RA_FLAG_ONLINK) {
      prefix =
        uip_ds6_prefix_lookup(&nd6_opt_prefix_info->prefix,
                              nd6_opt_prefix_info->preflen);
      if(prefix == NULL) {
        if(nd6_opt_prefix_info->validlt != 0) {
          if(nd6_opt_prefix_info->validlt != UIP_ND6_INFINITE_LIFETIME) {
            prefix = uip_ds6_prefix_add(&nd6_opt_prefix_info->prefix,
                                        nd6_opt_prefix_info->preflen,
                                        uip_ntohl(nd6_opt_prefix_info->
                                            validlt));
          } else {
            prefix = uip_ds6_prefix_add(&nd6_opt_prefix_info->prefix,
                                        nd6_opt_prefix_info->preflen, 0);
```

```
      }
    }
  } else {
    switch (nd6_opt_prefix_info->validlt) {
    case 0:
      uip_ds6_prefix_rm(prefix);
      break;
    case UIP_ND6_INFINITE_LIFETIME:
      prefix->isinfinite = 1;
      break;
    default:
      PRINTF("Updating timer of prefix");
      PRINT6ADDR(&prefix->ipaddr);
      PRINTF("new value %lu\n", uip_ntohl(nd6_opt_prefix_info->validlt));
      stimer_set(&prefix->vlifetime,
                 uip_ntohl(nd6_opt_prefix_info->validlt));
      prefix->isinfinite = 0;
      break;
    }
  }
}
/* End of on-link flag related processing */
/* autonomous flag related processing */
if((nd6_opt_prefix_info->flagsreserved1 & UIP_ND6_RA_FLAG_AUTONOMOUS)
    && (nd6_opt_prefix_info->validlt != 0)
    && (nd6_opt_prefix_info->preflen == UIP_DEFAULT_PREFIX_LEN)) {

  uip_ipaddr_copy(&ipaddr, &nd6_opt_prefix_info->prefix);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
```

```
addr = uip_ds6_addr_lookup(&ipaddr);
if((addr != NULL) && (addr->type == ADDR_AUTOCONF)) {
 if(nd6_opt_prefix_info->validlt != UIP_ND6_INFINITE_LIFETIME) {
   /* The processing below is defined in RFC4862 section 5.5.3 e */
   if((uip_ntohl(nd6_opt_prefix_info->validlt) > 2 * 60 * 60) ||
      (uip_ntohl(nd6_opt_prefix_info->validlt) >
       stimer_remaining(&addr->vlifetime))) {
     PRINTF("Updating timer of address");
     PRINT6ADDR(&addr->ipaddr);
     PRINTF("new value %lu\n",
            uip_ntohl(nd6_opt_prefix_info->validlt));
     stimer_set(&addr->vlifetime,
                uip_ntohl(nd6_opt_prefix_info->validlt));
   } else {
     stimer_set(&addr->vlifetime, 2 * 60 * 60);
     PRINTF("Updating timer of address ");
     PRINT6ADDR(&addr->ipaddr);
     PRINTF("new value %lu\n", (unsigned long)(2 * 60 * 60));
   }
   addr->isinfinite = 0;
 } else {
   addr->isinfinite = 1;
 }
} else {
 if(uip_ntohl(nd6_opt_prefix_info->validlt) ==
    UIP_ND6_INFINITE_LIFETIME) {
   uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
 } else {
   uip_ds6_addr_add(&ipaddr, uip_ntohl(nd6_opt_prefix_info->validlt),
```

```
                        ADDR_AUTOCONF);
          }
         }
        }
        /* End of autonomous flag related processing */
      }
      break;
#if UIP_ND6_RA_RDNSS
    case UIP_ND6_OPT_RDNSS:
      if(UIP_ND6_RA_BUF->flags_reserved & (UIP_ND6_O_FLAG << 6)) {
        PRINTF("Processing RDNSS option\n");
        uint8_t naddr = (UIP_ND6_OPT_RDNSS_BUF->len - 1) / 2;
        uip_ipaddr_t *ip = (uip_ipaddr_t *)(&UIP_ND6_OPT_RDNSS_BUF->ip);
        PRINTF("got %d nameservers\n", naddr);
        while(naddr-- > 0) {
          PRINTF(" nameserver: ");
          PRINT6ADDR(ip);
          PRINTF(" lifetime: %lx\n", uip_ntohl(UIP_ND6_OPT_RDNSS_BUF->lifetime));
          uip_nameserver_update(ip, uip_ntohl(UIP_ND6_OPT_RDNSS_BUF->lifetime));
          ip++;
        }
      }
      break;
#endif /* UIP_ND6_RA_RDNSS */
    default:
      PRINTF("ND option not supported in RA");
      break;
    }
    nd6_opt_offset += (UIP_ND6_OPT_HDR_BUF->len << 3);
```

```
  }

  defrt = uip_ds6_defrt_lookup(&UIP_IP_BUF->srcipaddr);
  if(UIP_ND6_RA_BUF->router_lifetime != 0) {
    if(nbr != NULL) {
      nbr->isrouter = 1;
    }
    if(defrt == NULL) {
      uip_ds6_defrt_add(&UIP_IP_BUF->srcipaddr,
                        (unsigned
                         long)(uip_ntohs(UIP_ND6_RA_BUF->router_lifetime)));
    } else {
      stimer_set(&(defrt->lifetime),
                 (unsigned long)(uip_ntohs(UIP_ND6_RA_BUF->router_lifetime)));
    }
  } else {
    if(defrt != NULL) {
      uip_ds6_defrt_rm(defrt);
    }
  }

#if UIP_CONF_IPV6_QUEUE_PKT
  /* If the nbr just became reachable (e.g. it was in NBR_INCOMPLETE state
   * and we got a SLLAO), check if we had buffered a pkt for it */
  /*  if((nbr != NULL) && (nbr->queue_buf_len != 0)) {
    uip_len = nbr->queue_buf_len;
    memcpy(UIP_IP_BUF, nbr->queue_buf, uip_len);
    nbr->queue_buf_len = 0;
    return;
```

```
    }*/
  if(nbr != NULL && uip_packetqueue_buflen(&nbr->packethandle) != 0) {
    uip_len = uip_packetqueue_buflen(&nbr->packethandle);
    memcpy(UIP_IP_BUF, uip_packetqueue_buf(&nbr->packethandle), uip_len);
    uip_packetqueue_free(&nbr->packethandle);
    return;
  }

#endif /*UIP_CONF_IPV6_QUEUE_PKT */

discard:
  uip_len = 0;
  return;
}
#endif /* !UIP_CONF_ROUTER */
/*---------------------------------------------------------------------*/
/* ICMPv6 input handlers */
#if UIP_ND6_SEND_NA
UIP_ICMP6_HANDLER(ns_input_handler, ICMP6_NS, UIP_ICMP6_HANDLER_CODE_ANY,
                  ns_input);
UIP_ICMP6_HANDLER(na_input_handler, ICMP6_NA, UIP_ICMP6_HANDLER_CODE_ANY,
                  na_input);
#endif

#if UIP_CONF_ROUTER && UIP_ND6_SEND_RA
UIP_ICMP6_HANDLER(rs_input_handler, ICMP6_RS, UIP_ICMP6_HANDLER_CODE_ANY,
                  rs_input);
#endif
```

```
#if !UIP_CONF_ROUTER
UIP_ICMP6_HANDLER(ra_input_handler, ICMP6_RA, UIP_ICMP6_HANDLER_CODE_ANY,
                  ra_input);
#endif
/*---------------------------------------------------------------------------*/
void
uip_nd6_init()
{

#if UIP_ND6_SEND_NA
  /* Only handle NSs if we are prepared to send out NAs */
  uip_icmp6_register_input_handler(&ns_input_handler);

  /*
   * Only handle NAs if we are prepared to send out NAs.
   * This is perhaps logically incorrect, but this condition was present in
   * uip_process and we keep it until proven wrong
   */
  uip_icmp6_register_input_handler(&na_input_handler);
#endif



#if UIP_CONF_ROUTER && UIP_ND6_SEND_RA
  /* Only accept RS if we are a router and happy to send out RAs */
  uip_icmp6_register_input_handler(&rs_input_handler);
#endif

#if !UIP_CONF_ROUTER
  /* Only process RAs if we are not a router */
```

```
  uip_icmp6_register_input_handler(&ra_input_handler);
#endif
}
/*---------------------------------------------------------------------------*/
 /** @} */
```

LIST OF REFERENCES

[1] I. Ishaq, D. Carels, G.K. Teklemariam, J. Hoebeke, F.V.D. Abeele, E.D. Poorter, I. Moerman, and P. Demeester, "IETF standardization in the field of the Internet of Things (IoT): A survey," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, pp. 235–287, 2013.

[2] J.A. Gutierrez, M. Naeve, and E. Callaway, "IEEE 802.15. 4: A developing standard for low-power low-cost wireless personal area networks," *IEEE Network* vol. 15, no. 5, pp. 12–19, 2001.

[3] *Transmission of IPv6 packets over IEEE 802.4 Networks*, RFC 4944, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4944

[4] C. Hennebert, and D.S. Jessye, "Security protocols and privacy issues into 6LoWPAN stack: A synthesis.," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 384–398, 2014.

[5] *Compression Format for IPv6 Datagrams over IEEE 802.15*, RFC 6282, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6282

[6] *The Constrained Application Protocol (CoAP)*, RFC7252, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

[7] *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC 4443, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4443

[8] *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, RFC 6775, 2012. [Online]. Available: https://tools.ietf.org/html/rfc6775

[9] *IPv6 Neighbor Discovery (ND) Trust Models and Threats*, RFC 3756, 2004. [Online]. Available: https://tools.ietf.org/html/rfc3756

[10] *Secure Neighbor Discovery (SEND)*, RFC 3971, 2005. [Online]. Available: https://tools.ietf.org/html/rfc3971

[11] *Neighbor Discovery for IP version 6*, RFC 4861, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4861

[12] S. Praptodiyono, R.K. Murugesan, I.H. Hasbullah, C.Y. Wey, M.M. Kadhum, and A. Osman, "Security mechanism for IPv6 stateless address autoconfiguration," *in Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology*, pp. 31–36, 2015.

[13] *Securing Neighbor Discovery Proxy: Problem Statement*, RFC 5909, 2010.

[14] J. Davies, *Understanding IPv6*. Redmond, Washington, United States: Microsoft Press, 2012. [Online]. Available: http://it-ebooks.info/book/1022/

[15] *Security Architecture for the Internet Protocol, RFC-4301*, 2005.

[16] O. Morrison, J. Wilkosz, A. Carrera, R. Mera, "IPSec: Protocol challenges and performance analysis and enhancements," *Computing and Information Systems, University of Melbourne, Australia*, 2014.

[17] I.H. Hasbullah, M.M. Kadhum, Y.W. Chong, K. Alieyan, and A. Osman, "Timestamp utilization in trust-ND mechanism for securing neighbor discovery protocol," *in Annual Conference on Privacy, Security and Trust (PST)*, pp. 275-281, 2016.

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

[19] J.W. Bos, O. Özen, and J.P. Hubaux, "Analysis and optimization of Cryptographically Generated Addresses," *in International Conference on Information Security*, pp. 17–32, 2009.

[20] F.A. Barbhuiya, S. Biswas, and S. Nandi, "Detection of neighborsolicitation and advertisement spoofing in IPv6 neighbor discovery protocol," *in Proceedings of the 4th International Conference on Security of Information and Networks*, pp. 110–118, 2011.

[21] H. Rafiee, and C. Meinel, "SSAS: A simple secure addressing scheme for IPv6 autoconfiguration," *in Annual Conference on Privacy, Security and Trust (PST)*, pp. 275–282, 2013.

[22] K. Perumal and M. J. P. J. Priya, "Trust based security enhancement mechanism for Neighbor Discovery Protocol in IPv6," *International Journal of Applied Engineering*, vol. 11, no. 7, pp. 4787–4796, 2016.

[23] A. Josang and R. Ismail, "The beta reputation system," *in Proceedings of the 15th Bled Electronic Commerce Conference*, vol. 5, pp. 2502-2511, 2002.

[24] *Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers*, RFC 4727, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4727

[25] *Network time protocol (NTP)*, RFC 958, 1985. [Online]. Available: https://tools.ietf.org/html/rfc958

[26] “Contiki: The open source OS for the Internet of Things,” Contiki. Accessed November 19, 2017. [Online]. Available: http://www.contiki-os.org/

[27] “Get started with Contiki.” Accessed November 19, 2017. [Online]. Available: http://www.contiki-os.org/start.html

[28] “An introduction to Cooja,” Github. Accessed November 19, 2017. [Online]. Available: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

[29] “The Contiki Operating System,” Sourceforge. Accessed November 19, 2017. [Online]. Available: https://sourceforge.net/projects/contiki/?source=navbar

[30] *IP Version 6 Addressing Architecture*, RFC 4291, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4291

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California