How efficient is Efficient NDP?

DIMITRIS KALOMOIRIS
TIAN XU
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

How efficient is Efficient NDP?

Dimitris Kalomoiris & Tian Xu
wir15dka@student.lu.se & wir15txu@student.lu.se

Department of Electrical and Information Technology
Lund University

Advisors:
Jens A Andersson (EIT LTH)
Stefan Höst (EIT LTH)

Examiner:
Maria Kihl (EIT LTH)

October 10, 2017

Printed in Sweden
E-huset, Lund, 2017

Abstract

In the following years we will experience a transition towards Internet Protocol version 6 (IPv6). The reason is the depletion of IPv4 address space due to the rapid increasing numbers of Internet of Things (IoT) devices connection. This transition however poses a problem, since the majority of these devices mostly are mobile and not connected to power source.

Neighbor Discovery Protocol (NDP) defined in RFC 4861 is used in IPv6 networks, to manage the address configuration as well as the network prefix and maintain lists of all the neighbors connected to that network. However, this protocol does not work as good in wireless connected devices as in wired. The messages exchange in the network disrupts the sleep mode of the nodes resulting in lower battery time. Hence the energy efficiency of these devices is now something we should consider. To solve this problem, the Efficient NDP draft was introduced as one of the optimized methods.

This thesis takes the previous theses of analysis and simulations of IPv6 Neighbor Discovery for wireless networks further, to explore the ND protocol in RFC 4861 (legacy NDP), and compare it to the Efficient NDP draft in terms of energy efficiency for the devices mentioned above. Several dynamic network cases are taken into analysis, and simulation scenarios for these cases are tested in OMNeT++. These simulation models follow the implementation process introduced in the protocols with consideration of network scalability, and present the result that the Efficient NDP has large improvement in saving multicast messages in ND process.

Acknowledgments

We would like to thank our two supervisors Jens A. Andersson and Stefan Höst for their help and support during this thesis, their patient teaching, excellent knowledge and important advice helps us to overcome the obstacles in the thesis work. We would also like to thank all the staffs, teachers, PhD students, seniors and our classmates in Wireless Communication Master Programme, for their help in the past two years.

I (Dimitris Kalomoiris) first of all, would like to thank my colleague Tian Xu for his help and collaboration in completing this thesis. I would also like to thank my parents and friends for their support during the past two years.

I (Tian Xu) would like to thank to my parents, for their moral and financial support during my two years study in Sweden. Their kind care and advice always gives me encourage and helps me to overcome the problems in life. I would also like to thank to my partner, colleagues and friends. Their enthusiastic motivation lets me to be pleasant in hard situations and their great idea helps me to get out of difficulty in this thesis work.

Popular Science Report

In the real world, people need addresses to identify their locations. Similar to this, devices also need to have network labels which called IP addresses for identification and locating when they connect to the Internet. Internet Protocol version 4 (IPv4) is a protocol which introduces a 32-bits IP address format and popularly used in today's networks. However, the increased number of devices connecting to the Internet has led to the depletion of available IPv4 addresses. IPv6 was introduced with more address space to solve IPv4 address exhaustion problem. But on the other hand, many existing protocols used in IPv4 have to be extended or redesigned to fit IPv6 networks.

A network always contains many nodes (routers and hosts). A node recognizes other nodes in its located network as its neighbors. Discovering neighbor locations, gathering neighbor information and defining the communication methods between neighbors are the contents in IPv6 Neighbor Discovery Protocol (NDP). The legacy NDP was announced 10 years ago in RFC 4861. It relies on periodical multicast Internet Control Message Protocol version 6 (ICMPv6) control messages to maintain relationship between neighbors. Which means that these control messages are sent to all the members in a group. Although these messages are only relevant to a specific member in this group at most of time, other members also have to receive and process them.

As we known, receiving and processing messages require power consumption. It is inefficient for nodes to handle these unnecessary messages frequently, especially for wireless devices. As wireless devices are always not connected to power supplies, power saving becomes an important point for their battery life.

To solve the high power consumption problem in legacy NDP, a more efficient protocol the Efficient NDP was announced. This protocol introduces a registration mechanism for hosts with a router. In this mechanism, the router does not need to send periodical control messages to the hosts for asking their updates, but give a registration timer to each host. A host only has to wake up and send messages to the router when the timer is reached, and can keep in sleep mode for a long

time. This can significantly reduce the power consumption. On the other hand, as the router functions as a registration point in the network, it has the knowledge of each node address information, so it can help hosts for handling address related issues with their neighbors.

From analysis, we have already known that the Efficient NDP provides a good solution in power saving, comparing with the legacy NDP. But how efficient it can achieve? To answer this question, an evaluation is required. As today's networks are always dynamic networks, nodes are able to enter, leave and change their locations in networks at any time, it is worth for us to analyze the performance of these two protocols in this environments. However, as implementing a real dynamic network environment is too complex, simulation becomes an ideal method for evaluation.

Two scenarios were taken into analysis and simulations in this thesis, one is nodes entering and leaving a network scenario, and another is nodes losing connection to a network case in movement scenario. As network scalability is an important parameter for evaluation, simulation models were designed with different network sizes. The obtained results indicate that in both of these scenarios, the Efficient NDP can achieve very high messages saving percentages in large networks. On the other word, it provides a great power saving solution.

Table of Contents

List of Figures

List of Tables

List of Acronyms

- 6LoWPAN - IPv6 Low-Power Wireless Personal Area Network
- ARO - Address Registration Option
- ARP - Address Resolution Protocol
- DAD - Duplicate Address Detection
- DHCPv6 - Dynamic Host Configuration Protocol version 6
- EAH - Efficient Aware Host
- GUI - Graphical User Interface
- ICMPv6 - Internet Control Management Protocol version 6
- IDE - Integrated Development Environment
- IPv4 - Internet Protocol version 4
- IPv6 - Internet Protocol version 6
- IoT - Internet of Things
- M2M - Machine to Machine
- MLD - Multicast Listener Done
- MLD protocol - Multicast Listener Discovery protocol
- MLDv2 protocol - Multicast Listener Discovery version 2 protocol
- MLQ - Multicast Listener Query
- MLR - Multicast Listener Report
- MTU - Maximum Transmission Unit
- NA - Neighbor Advertisement
- NCE - Neighbor Cache Entry
- ND - Neighbor Discovery

- NDP - Neighbor Discovery Protocol
- NEAR - ND Efficient Aware Router
- NED - Network Description
- NS - Neighbor Solicitation
- NUD - Neighbor Unreachability Detection
- OS - Operating System
- PRNG - Pseudo Random Number Generator
- RA - Router Advertisement
- RADVD - Routing Advertising Daemon
- RS - Router Solicitation
- SDAT - Simple DNA address table
- SLACC - Stateless Address Auto-configuration
- SLLA - Source Link-Layer Address
- Simple DNA - Simple Detecting Network Attachment
- TID - Transaction Identifier
- UIID - Unique Interface Identifier

Chapter 1

Introduction

The Internet of Things (IoT) offers a platform to devices for communication and information sharing within a smart environment, and it provides increasing of the benefits for organizations and individuals [1]. With the rapid development of IoT, there will be a continuous increasing of devices connected to the Internet in the next few years.

As a solution of Internet Protocol version 4 (IPv4) address exhaustion problem, IPv6 provides much more address space which can satisfy the Internet connection requirement from billions of devices. In addition, IPv6 has the features of security solution, Stateless Address Auto-configuration (SLACC), multicast and group operation, resource allocation [2]. With these features, IPv6 is required in many IoT networks and it will have more implementation in the near future for Internet access.

The IPv6 Neighbor Discovery Protocol (NDP) was defined in RFC 4861 [3]. Information sharing and address registration in this legacy protocol is processed by exchanging multicast control signals. However, unnecessary control messages cause additional power consumption and make this protocol inefficient, especially in wireless network environments. The Efficient NDP draft [4] introduces solutions of reducing these multicast messages, and the optimizations provided by this protocol are what we need to verify in this thesis work.

1.1 Background

The rapid growth of IoT and Machine to Machine (M2M) market nowadays brings new requirements and complex implementation of network neighbor discovery. RFC 4861 defines the node behavior in a network: locating neighboring routers, determining address configuration and gathering neighbor information. These events are performed by exchanging Internet Control Management Protocol version 6 (ICMPv6) messages over the link. Most of these ICMPv6 messages are periodical

and multicasted as defined in NDP.

The high mobility of IoT networks reveals the weakness of NDP in wireless network environments. Although NDP performs well for those devices connected to power supplies, it has power saving problem for wireless battery devices due to periodical multicast messages exchanging within the network. The inefficiency of legacy NDP in different types of wireless networks has been analyzed in many researches, and the limitations of legacy NDP will be discussed in details in Chapter 2. The goal of the Efficient NDP draft is to provide solutions for power consumption problem by reducing multicast traffic as much as possible.

1.2 Problem definition

Legacy NDP was introduced by RFC 4861 in 2007. At that time, networks were simpler than they are today. Most of networking devices and clients were connected to power supplies which provide unlimited power support. Five different types of messages are used in this protocol to establish communication links between hosts and routers, hosts and hosts. The usage of these messages will be explained in Chapter 2.

Since more and more wireless powered devices have been used to replace wired powered devices in today's networks, the inefficiency of legacy NDP appears: After a host has established connection with the router, configured its address and obtained neighbor information, it would like to be kept in sleep mode if it does not have necessary communication with other nodes. However, when it reaches message receiving time or other nodes have activities within the network, this host has to be awakened to receive and handle irrelevant messages, as they are in the same multicast group. This costs unnecessary power consumption which has no benefits for wireless device battery life.

In RFC 4919 [5], apart from power consumption being low, low bandwidth and a small packet size is proposed. To achieve this, the IPv6 layers and headers must be compressed whenever possible and the actual data size could be smaller than the IPv6 packet size, IPv6 requires a minimum Maximum Transmission Unit (MTU) of at least 1280 bytes, fragmentation and reassembly is required. SLACC and routing protocols which support multi-hop mesh networks are necessary.

RFC 6775 [6] proposes the initiated interaction between nodes such as Neighbor Solicitation or Router Advertisement information, allowing hosts to have more time to sleep as the hosts are not required to receive periodic messages from the routers over IPv6 Low-Power Wireless Personal Area Networks (6LoWPAN). Every host periodically wakes up and updates its address registration.

As another solution of this problem, the Efficient NDP draft defines optimized methods for all kinds of wireless networks. This protocol aims at removing the use of multicast traffic for Duplicate Address Detection (DAD), and handling DAD

in a better strategy which does not require hosts to always be awakened [4]. The details about the methods used in this protocol will be introduced in Chapter 3.

1.3 Previous work

In the thesis *"Analysis of IPv6 Neighbor Discovery for Mobile and Wireless Networks"* [7] written by Hariharasudan Vigneswaran and Jeena Rachel John, the theoretical analysis and comparison between legacy NDP and the Efficient NDP was presented. Such scenarios as IPv6 ND behavior, scalability of the network, transient behavior of the network, multicast messages exchange over the network and sleep & wake up behavior of nodes were taken into analysis. They also investigated the implementation of the routers via Routing Advertising Daemon (RADVD) under Unix environment.

In the thesis *"Efficient IPv6 Neighbor Discovery in Wireless Environment"* [8] written by Dragos Neagoe and Antonios Pateas, a deeper study of legacy NDP and the Efficient NDP nodes sleep & wake up scenario was presented. In their thesis work, network models were designed and simulations were performed to compare with the results from the theoretical analysis. They also implemented router behavior by RADVD and host behavior in Linux Kernel.

As a continuation of the above thesis work, this thesis focuses on the Efficient NDP performance in wireless dynamic network environments which was partly discussed in [7], continues with the theoretical analysis and performs simulations for several dynamic network scenarios, and evaluates the improvement provided by the Efficient NDP in these scenarios.

1.4 Approach

The goal of this project is to identify how much efficiency can be achieved by the Efficient NDP compare with legacy NDP in wireless dynamic network environments. In order to achieve this goal, network model design and simulations are performed for the following objectives:

- The impact of the power consumption between legacy NDP and the Efficient NDP in nodes entering and leaving a network
- The impact of the power consumption between legacy NDP and the Efficient NDP in nodes movements within the network

For each case, network models for legacy NDP and the Efficient NDP are created. The same parameters such as the number of hosts and routers in the network, the default timers for registration or unreachability check are assigned in these two protocols, in order to make a conformance standard for evaluation and comparison.

The open source software OMNeT++ is used as the simulation tool, the details about its running environment and hardware requirements will be introduced in Chapter 6.

1.5 Organization of thesis

The thesis work is organized with 9 chapters: Chapter 1 describes the background, identifies the problem and methods. Chapter 2 and Chapter 3 separately introduce the features defined in legacy NDP and the Efficient NDP. Chapter 4 provides the theoretical analysis of the behavior in two specific dynamic wireless network cases. Chapter 5 and Chapter 6 describe the network model design and the simulation implementation, which are the main parts in this thesis work. Chapter 7 lists the results obtained from the simulations, and these results are verified according to the theoretical analysis. Chapter 8 states the performance and evaluation of the efficiency provided by the Efficient NDP in wireless dynamic networks. Chapter 9 concludes what has been done in this thesis and gives suggestion for future work.

The topic of this thesis was announced by Department of Electrical and Information technology (EIT), Lund University. In this thesis, Dimitris was responsible for protocol introduction and result evaluation, while Tian was responsible for theoretical analysis and simulation design. The rest parts were done by both of authors.

Chapter 2

The Legacy Protocol RFC 4861 Overview

RFC 4861 introduces NDP for IPv6 networks. Neighbor Discovery is the procedure for nodes (routers and hosts) to determine their link-layer addresses and gather the information about their neighbors on the attached link. With Neighbor Discovery, nodes are able to configure and update their data link and network addresses, maintain routing table, obtain the path information to their active neighbors and solve a set of problems caused by the interaction between nodes [3].

2.1 Messages

In NDP, five types of messages are used to establish and maintain the connection. They are:

- Router Solicitation (RS)
- Router Advertisement (RA)
- Neighbor Solicitation (NS)
- Neighbor Advertisement (NA)
- Redirect message

Once a host joins a network, the first thing it does is attempting to find routers. In NDP, locating routers is done by sending RS to the all-routers multicast address over the links. The function of RS is asking for immediate RA, instead of waiting for the next periodic one to arrive.

RA are the messages which are periodically multicasted by each router to hosts. RA contain the information such as link-layer addresses, link prefixes, MTU, whether use SLAAC in address configuration or not. RA have periodical advertising timer as every 7 to 10 minutes by default [9], but they can also be trigged by the received RS.

NS are the messages which have the same function as ARP in IPv4. NS can be used by any nodes at any time to request the link-layer addresses of their neighbors. NS are also used to check if the link-layer address has already been used and if the target neighbor is still reachable.

In response to NS, the other node(s) sends NA. NA can also be sent unsolicited by a node when this node needs to announce the change of its link-layer address.

Redirect messages are used by routers to notify a host for a better first-hop to its destination.

Table 2.1 summarizes the features of these messages in ND.

Message	**Communication Type**	**Direction**	**Function**
RS	multicast	host to routers	Request RA
RA	multicast when reach timer unicast when triggered by RS	router to host(s)	Provide information
NS	unicast when verify reachability multicast when resolve address	node to node(s)	Request NA DAD, NUD
NA	unicast when triggered by NS multicast when is unsolicited	node to node(s)	Response NS Announce change
Redirect	unicast	router to host	Inform better path

Table 2.1: Summary of ND ICMPv6 Messages in RFC 4861

2.2 Addresses

2.2.1 Overview of IPv6 addressing

IPv4 address has 32 address bits, and it allows for approximately 4,200,000,000 possible addressable nodes. IPv6 increases the number of address bits from 32 to 128, and it provides approximately 3.4×10^{38} reserved addresses for nodes [10].

The typical IPv6 addresses are written as eight sets of hexadecimal numbers, with colons between each set of four hexadecimal digits, which represents 16 bits. But the addresses can also be written as the format of shortened notation.

In IPv6, three main types of addresses are defined:

- Unicast address: Address of a single interface, packets to this address will be sent to that interface.
- Multicast address: Address of multiple interfaces on different devices within the same multicast group, packets to this address will be sent to all interfaces in the multicast group.

- Anycast address: Address of multiple interfaces on different devices, packets to this address will be sent to the closet interface.

2.2.2 NDP addressing

As described in Chapter 2.1, the ND process is the process of unicast and multicast messages exchange. Several types of addresses are used in NDP, they are:

- All-nodes multicast address: FF02::1. This is the link-local scope address and packets aim at this address will be sent to all nodes in multicast group.
- All-routers multicast address: FF02::2. This is the link-local scope address and packets aim at this address will be sent to all routers in multicast group.
- Solicited-node multicast address: FF02::1:FFXX:XXXX. This is a link-local scope multicast address which is computed as a function described in [11]. The XX:XXXX is the far right 24 bits of the corresponding unicast or anycast address for the node.
- Link-local address: FE80::/10. This is a unicast address having link-only scope that can be used to reach neighbors. It is also used for SLAAC and router discovery.
- Unspecified address: 0:0:0:0:0:0:0:0. This is a reserved address which indicates there is no certain address due to address missing or unknown. This address can be used as source address if the sender does not know its address, but never used as a destination address.

Table 2.2 summarizes the addresses used in ND.

Type	**Address**	**Function**
All-nodes multicast	FF02::1	Address to all nodes
All-routers multicast	FF02::2	Address to all routers
Solicited-node multicast	FF02::1:FFXX:XXXX	Address resolution
Link-local	FE80::/10	ND, router discovery, SLACC
Unspecified	0:0:0:0:0:0:0:0	Indicate lack of an address

Table 2.2: Summary of ND Addresses

2.3 Router and Prefix Discovery

In NDP, the communication between hosts and routers is established via RS and RA messages. Router Discovery is the process for hosts to discover and identify the location of neighbor routers over a link, and to learn the configuration parameters related to SLAAC which is described in RFC 4862. Prefix Discovery is the process

for hosts to directly learn IP addresses over a link without asking routers for further information.

Once a host enters in or re-connects with a network, it either receives the periodic RA (this hardly happens) or sends RS to the all-routers multicast address asking for RA in order to get knowledge of the local network and configure its address. Routers that receive RS firstly check the validity of the message to decide whether to process it or not. If a solicitation passes the validity check, the router then sends a RA directly to the host for response.

Hosts that receive RA also check the validity of the message to decide if the information included in RA is usable. If the RA is valid, the hosts then accept the information. However, when the next RA comes, it might contain information for a specific parameter or option which is different from the information before. In this case, the hosts will update with the most recently received information.

Figures 2.1 illustrates the procedure of router discovery.

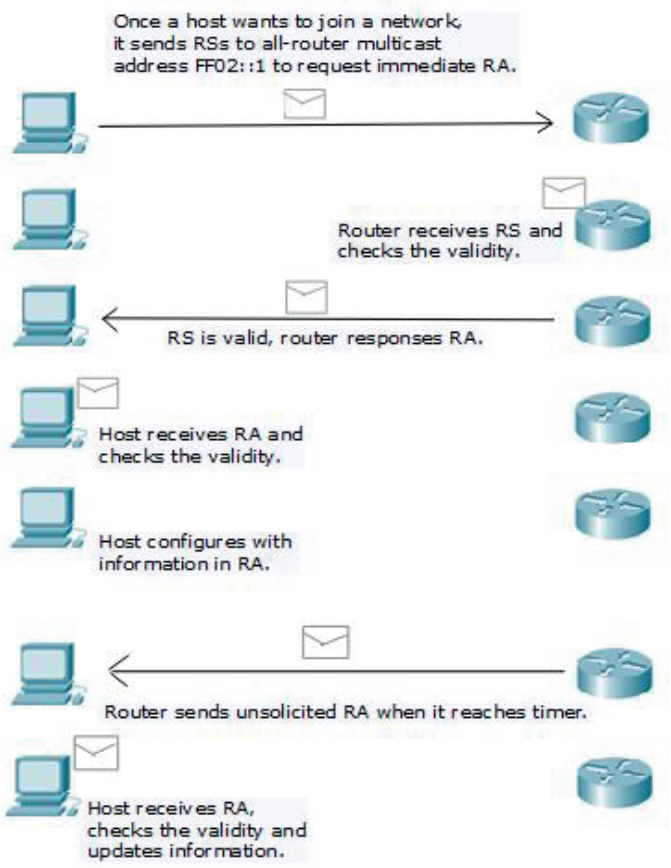

Figure 2.1: The process of RS, RA exchange

There might exist multiple routers on a link, thus the new incoming host has to make a default router selection. Usually, the host receives the first valid RA and selects the source router of this RA as its default router. The criterion for selecting a router also depends on the reachability of the router, this will be explained in the next section. When the lifetime of the default router expires, the router will be removed from the default router list, and then the host selects the next-hop determination router for sending traffic.

2.4 Behavior between neighbors

While RS and RA are the messages used in the communication between hosts and routers, NS and NA are used by all the nodes (both hosts and routers) to request and response for neighbor information with the main functions as address resolution, DAD and Neighbor Unreachability Detection (NUD).

Similar to RS and RA, NS and NA also have validity checks, messages that satisfy the requirements can be recognized as valid and then the information included in messages can be considered as usable.

2.4.1 Neighbor Cache Entry

Similar to ARP cache in IPv4, Neighbor Cache Entry (NCE) is used by nodes to record the link-layer address information of their neighbors in IPv6. According to the knowledge of neighbors, NCE has five states:

- INCOMPLETE: a NS has been sent to the solicited-node multicast address of the target but has no response yet.
- REACHABLE: The forward path to the neighbor functions positive.
- STALE: The forward path to the neighbor was positive last time, but it has been longer than the ReachableTime milliseconds since the last packet sent.
- DELAY: The forward path to the neighbor was positive last time and the ReachableTime milliseconds has passed this time. Additional time is given to upper-layer protocols in order to confirm the reachability.
- PROBE: No reachability confirmation has been received, the node starts to retransmit NS until it receives reachability confirmation.

2.4.2 Address Resolution

After the initial entering procedure, a node should be able to know the IP address of each neighbor that has been registered on the link. Apart from this, this node may also need to know neighbor link-layer addresses (MAC addresses) which are

the unique identification of each neighbor. To determine the link-layer address of a neighbor by the given IP address, nodes must perform address resolution.

When a node has a unicast packet to a target neighbor, the first thing it does is checking the existence of the target neighbor's NCE to determine whether it has known the link-layer address of this neighbor or not. If the entry is empty, the node then creates a NCE, marks it with INCOMPLETE state and sends NS to the solicited-node multicast address where the target neighbor address locates in.

All the nodes within the solicited-node multicast group will receive the NS, they check whether the request from the NS is related to themselves or not. Only the target neighbor responses with a NA, this NA will be sent as a unicast packet directly to the source of the NS (the sender node). However, if the source address has not been specified, the NA will be sent to the all-nodes multicast address.

Once the sender node receives the NA, the information about target neighbor's link-layer address will be recorded on sender's NCE. The sender node changes NCE state to REACHABLE.

Figure 2.2 illustrates the procedure of address resolution.

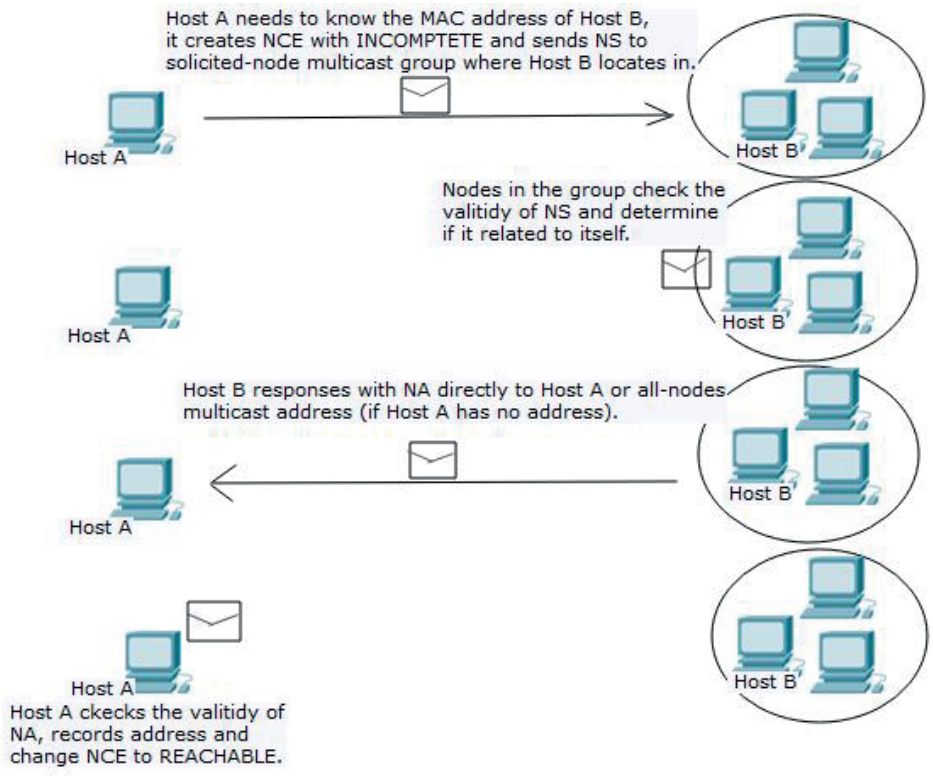

Figure 2.2: The procedure of address resolution

2.4.3 Duplicate Address Detection

In IPv6 networks, SLLAC is a mechanism used for address auto-configuration. It requires no manual configuration of hosts, minimal configuration of routers, and no additional servers [12].

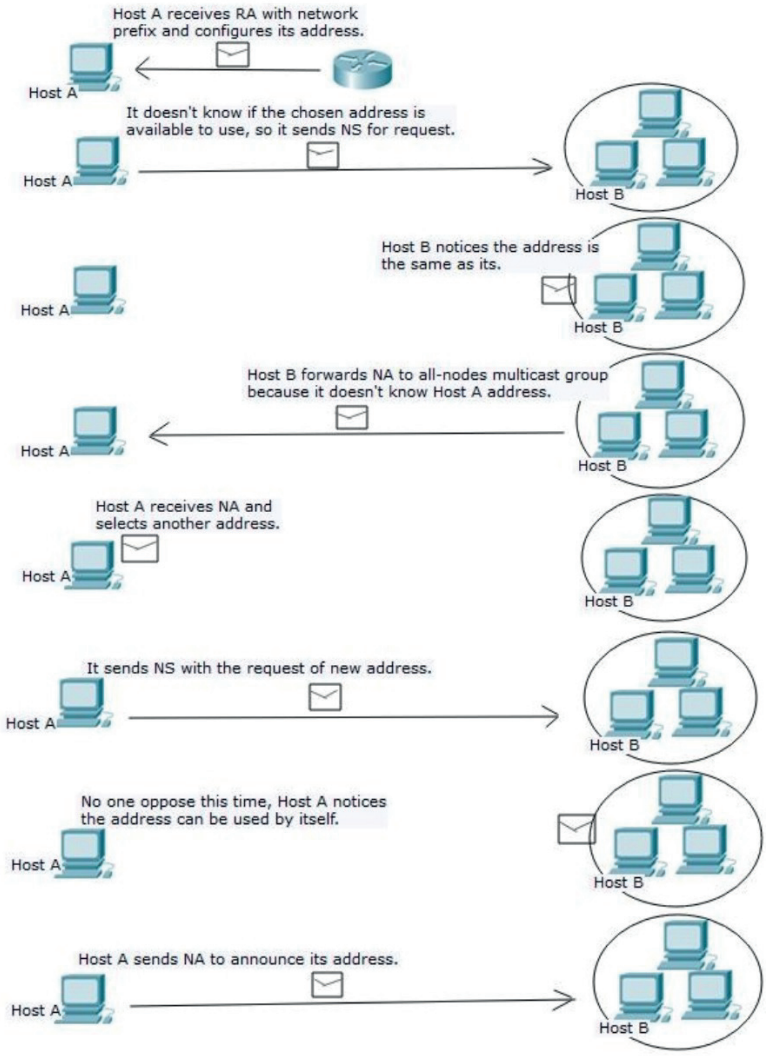

Figure 2.3: The procedure of DAD

After a host joins a network, it obtains address information by RA from a router. If SLLAC is allowed in this network, RA provides host the information about network prefix instead of a certain address. The host has to pick an IP address included in the given network prefix by itself. It is then that the problem

happens, the process of a host configuring its IP address is according to its own knowledge, but this address might have already been used by another host within the same network. To solve this address conflict issue, DAD must be performed.

To check whether the IP address is available to use, the new host which just configured an IP address sends NS to solicited-node multicast group for request. Once the hosts in the group receive the NS, they will check whether the IP address is being used by themselves. The receiver host which detects the conflict of the address then responses with NA to inform the sender. Since the sender's address is duplicated of its own address, the receiver host sends the NA to the all-nodes multicast group.

The sender receives the NA and notices this IP address is being used by someone, it then selects a new address and continues with the same process until it gets an address not used by anyone. After this, it sends NA to other hosts for updating their NCEs and starts with address resolution if it has packets to send.

Figure 2.3 illustrates the procedure of DAD.

2.4.4 Neighbor Unreachability Detection

Communication can fail due to hardware and interface issues. In this case, a node needs to perform NUD check with its neighbors.

The process of checking the reachability is done by exchanging NS and NA. As described in Chapter 2.4.1, nodes use NCE to state neighbor information: When a node performs address resolution, it creates an entry in INCOMPLETE state; When it confirms the reachability to its neighbor, NCE state changes to REACHABLE; When ReachableTime milliseconds has passed since the last confirmation, NCE state changes from REACHABLE to STALE; If an entry is in STALE and a node has a packet to send, it will change the state from STALE to DELAY and set a timer; If the timer expires and still has no confirmation, the state of NCE will change to PROBE, then the node begins to retransmit NS to its neighbor periodically; After the retransmission timer expires, the node will consider the neighbor is no longer available to reach and delete the NCE.

NUD is a mechanism that follows a series of steps performed by nodes. It provides a reliable reachability detection procedure. However, once a path fails, depending on the usage of the neighbor, possible recovery will be performed: restart address resolution, switch to another router, etc.

2.5 Limitations

As described earlier, one major problem caused by NDP RFC 4861 is high power consumption. For these wired power supply connected devices, power support

can be seen as unlimited. But for wireless devices, power support is from their batteries, and more power consumption means shorter battery life.

For unicast traffic, only one sender and one receiver has to pay the power for creating, forwarding and processing packets if these packets can be transmitted directly. But for multicast traffic, multiple receivers are required to receive and process the packets. This procedure in fact requires more power.

As explained in Chapter 2.4.2 and 2.4.3, nearly all the messages used in address resolution and DAD are multicast traffic. Although these messages are only useful for the sender and a specific receiver, all the members in the solicited-multicast group have to receive and process these messages. This wastes a lot of unnecessary energy in networks.

Apart from power consumption, the use of multicast traffic in NDP RFC 4861 causes many other problems. It is pointed out in IEEE 802.11 wireless networks half-duplex media are shared, if a node has multicast traffic in transmission, other nodes on the link must wait for the end of this transmission before they can start to talk [13]. In [14], a problem that increased utilization of the radio interface and increased processing loads in the nodes due to multicast traffic over IoT/M2M networks is stated. In cellular networks, unnecessary control signals cause additional consumption of radio resource.

Chapter 3

The Optimizations of the Efficient NDP

Legacy NDP as described in RFC 4861, although it is a start, is inefficient for wireless connected devices majority of which are mobile. The reason is that this protocol widely uses multicast signals in nodes interaction which forces devices to wake up from their sleep cycle, and as a consequence of reducing their battery lifetime. As an optimization of power consumption problem in legacy NDP, the Efficient NDP extends some of the methods defined in RFC 6775 [6] to reduce multicast messages in ND as much as possible, and faces to all kinds of networks.

3.1 Goals and improvements

The main goal defined in the Efficient NDP draft is reducing multicast traffic within the network. Multicast messages used in DAD and address resolution are replaced by unicast messages, and periodic multicast RA are removed in this protocol. This is done by giving more centralized control to the default router with more efficient address issue handling strategies.

However, the benefit of router's centralized control is not only reducing the multicast traffic. The default router always functions as a middle point between hosts communication, this simplifies the process of a host knowing other neighbors state and significantly saves the messages in NUD.

3.2 Definitions

The RA format for legacy NDP is introduced in [15] as Figure 3.1 shown. In the Efficient NDP, to identify which mode the routers and hosts work in, a new flag field 'E' takes use of the first reserved field as Figure 3.2 shown.

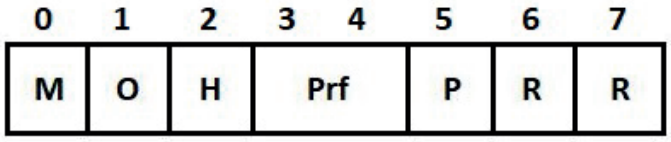

Figure 3.1: RA flag field in RFC 4861 [15]

Figure 3.2: RA flag field in the Efficient NDP [4]

There are three main work modes defined in the Efficient NDP, they are:

- Legacy mode: Nodes in this mode work the same as defined in RFC 4861.
- Efficient aware mode: The router in this mode performs as a NEAR (ND Efficient Aware Router), the RA sent by a NEAR have the E flag value as 1 while in all the other cases it must be set as 0. The host in this mode performs as a EAH (Efficient Aware Host).
- Mixed mode: A NEAR supports both legacy hosts and EAH, other routers on the link can be either legacy or NEAR.

In this thesis, only the efficient aware mode will be introduced and analyzed.

3.3 Nodes initialization

Once a host enters a network, it initially works as a legacy host and multicasts RS to the all-routers multicast address in order to find a default router. In the Efficient NDP, the RS include Source Link-Layer Address (SLLA) option in order for the routers to target the new incoming hosts as they have not configured IP address so far. Routers which receive the RS reply with a unicast RA message to share the network information to the new incoming host. Usually, a new incoming host makes the selection of the default router according to where the first valid RA comes from.

In the Efficient NDP, hosts become members of the all-nodes multicast group but they do not join the solicited-node multicast group.

As described before, there are no periodic multicast RA sent by routers in the Efficient NDP, the update information provided by routers is always included in NA. However, multicast RA are sometime used in order for newly arrived hosts and routers to find each other.

3.4 Behavior between neighbors

The efficiency provided by the Efficient NDP mainly exists in neighbor message exchange, this section introduces the optimizations in the process of address registration, DAD and NUD.

3.4.1 Address Registration Option

The Address Registration Option (ARO) was defined in RFC 6775 and the Efficient NDP extends it to handle different unique identifiers than EUI-64 and a Transaction ID [4]. The new format of NS and NA messages in this protocol is shown in Figure 3.3.

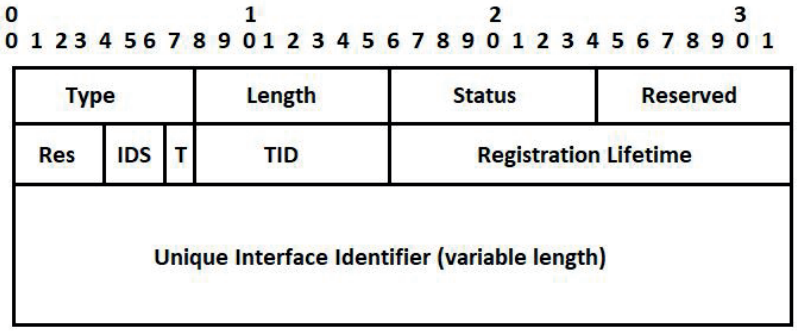

Figure 3.3: NS and NA format [4]

In the Efficient NDP, the default router has a table to record the host information, such as their registered IP addresses and MAC addresses, and performs as a responsible point to handle address related issues. Hosts have to update their address information by sending a NS with an ARO option which includes Registration Lifetime, Unique Interface Identifier (UIID), Transaction Identifier (TID) and SLLA to their default router. These Registration NS are usually sent by hosts at their 2/3 Registration Lifetime to refresh the registration. The default router updates the address information according to the recent received Registration NS and replies with a NA as confirmation and then the Registration Lifetime is reset by the hosts. The Registration Lifetime value is announced in the RA messages with a maximum 9000 seconds.

Unlike RFC 4861, hosts do not always need to wake up and protect their addresses in the Efficient NDP (this will be explained in the next subsection). Thus, they can stay in a long sleeping cycle after the registration if they do not have any activities in the network. When a host has to communication with a neighbor, it wakes up and communicates with the default router first to obtain the most recent address information for the target neighbor.

3.4.2 Duplicate Address Detection

In the Efficient NDP, DAD is handled by the default router as it has the address table which records all the addresses of nodes in its responsible network.

After a new incoming host finishes the router discovery process, it selects an address in the given address range and sends a Registration NS to its default router in order to use this address. If the selected address is available to use, the default router sends a NA with DAD pass option as reply, and then the incoming host finishes the address registration process. If the selected address is duplicated with one of the being used address recorded in the router's table, the default router sends a NA with DAD fail option back, then the host has to select another address and send a Registration NS to the default router again until it obtains the DAD pass information.

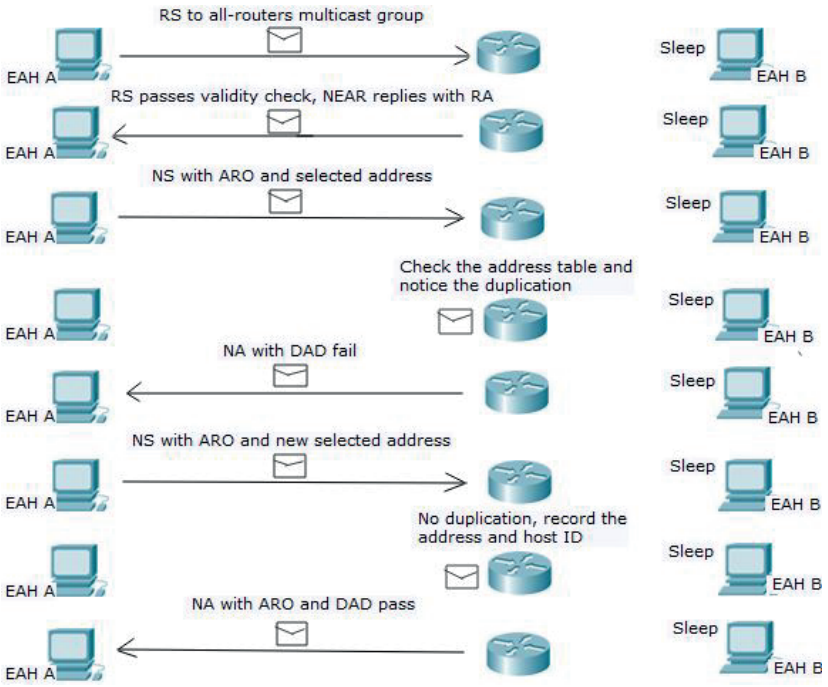

Figure 3.4: DAD procedure in the Efficient NDP

During this process, only the incoming host and the default router participate in the DAD, none of other nodes has to wake up or send any messages. Only unicast traffic is used. Figure 3.4 illustrates an example procedure of DAD.

3.4.3 Neighbor Unreachability Detection

A network connection can be unstable, especially in wireless networks. Similar to the unreachability detection mechanism in legacy NDP, NUD is also used in the Efficient NDP for nodes to maintain their NCE tables. However, as the default router in the Efficient NDP always functions as a middle point between host communication, a more optimized NUD mechanism is introduced in this protocol.

In the Efficient NDP, more recovery time is given to the unreachable hosts. As described before, a host (sender) which wants to communicate with a neighbor has to send the packet to its default router and let the default router forward the packet to the target neighbor. If the sender has lost connection to the network, this packet is not able to reach the default router, then the sender has to perform NUD with the default router. The NUD process is in fact the same as it in legacy NDP. However, even if the final check indicates the default router is unreachable, instead of deleting the NCE immediately as it is defined in legacy NDP, the router's NCE can be kept and marked with UNREACHABLE state by the host in the Efficient NDP. Until the Registration Lifetime expires, then the NCE will be deleted.

This optimized mechanism can significantly reduce the number of messages exchanged in the network, although the messages used in NUD are not multicast messages. After the sender marks the default router's NCE as UNREACHABLE, if it has to communicate with a neighbor again, the sender has to send another packet to its default router. As the default router's NCE has already been in UNREACHABLE state, no NUD has to be performed this time.

Chapter 4

Theoretical Analysis

In thesis work [7], several different ND scenarios are analyzed for legacy NDP and the Efficient NDP. In this chapter, the performance of these two protocols in dynamic networks is taken into deeper analysis.

Wireless devices can enter and leave a network at any time, and their location can change frequently due to the high mobility. To evaluate the efficiency achieved by the Efficient NDP, the ND behavior and scalability in dynamic network environments are necessary to be considered.

One thing needs to be mentioned. In the analyzed cases of nodes entering and leaving a network and nodes movements within the network, we only focus on node behavior in ND and ND related procedure. Node behavior mentioned in routing protocols is not discussed in this chapter.

4.1 Nodes entering and leaving a network

4.1.1 Nodes entering a network

A node follows a series of steps to finish the process of entering a network, from coming into a network until it becomes a full function member.

The legacy NDP

In the initial phase, the incoming node performs the router discovery to request network information. As explained in Chapter 2, in legacy NDP, this process is usually a two-way handshake. The node multicasts RS to the all-routers multicast address asking for RA, all the routers within this address receive the request and reply with a unicast RA.

The next step is the DAD check. After the node selected an IP address, it has to multicast NS to the solicited-node multicast group to check if this ad-

dress is available to use before configures with this address. If another node in the solicited-node multicast group detects the duplication, it sends NA to the all-nodes multicast group to inform the new node. In legacy NDP, this process continuous until the new node finds an available address. The node finishes address configuration and multicasts NA to announce its address to other nodes.

However, the amount of exchanged messages during DAD procedure has relationship with the number of nodes in this network and the subnet size. In fact, a certain subnet size is defined when the network is established and hardly changes. It means that with increasing number of nodes in this network, the available addresses become less, and the new incoming node has greater possibility to select the address that is being used by someone else.

After address configuration, the node needs to become a member of a solicited-node multicast group. This is done by following Multicast Listener Discovery (MLD) protocol [16] or the updated version Multicast Listener Discovery version 2 (MLDv2) protocol [17], and there are two ways to achieve this.

In the MLD protocol, a multicast router sends periodical Multicast Listener Query (MLQ) to listeners asking for which multicast group they wish to join. Listeners reply with Multicast Listener Report (MLR) to tell the router their ideal multicast group and send Multicast Listener Done (MLD) when they have finished listening traffic to this multicast group. In the MLDv2 protocol, MLRv2 is used with the function of both MLR and MLD messages.

Another way is the node sending unsolicited MLR to router to specify which multicast group it wishes to join.

In both of these two methods, MLR is unicast traffic which directly forwarded to the default router.

The Efficient NDP

The procedure of router discovery and address configuration in the Efficient NDP is the same as described in legacy NDP. The new incoming node sends multicast RS for requesting RA and all the routers provide reply with a unicast RA. According to the information contained in RA, the node configures its address.

The next step is address registration. As mentioned in Chapter 3, in the Efficient NDP, instead of the new node multicasts NS to the solicited-node multicast group to check whether its address is duplicated or not, DAD is performed with the default NEAR. The new node sends unicast NS with ARO option directly to its NEAR requesting for registration, the NEAR decides whether the request can be approved or not and sends NA as feedback.

In the Efficient NDP, a node is a member of the all-nodes multicast group but does not need to join the solicited-node multicast group. So far, the node has finished the procedure of entering a network.

4.1.2 Nodes leaving a network

Once a node leaves a network, it should be removed from default router's list.

The legacy NDP

In legacy NDP, there are two mechanism defined in leaving a network.

One is called "soft leave (silent leave)". As explained before, multicast routers send periodical queries to learn the multicast addresses of listeners. In this mechanism, the node does not respond to these queries. When the source timer expires, the node will be deleted from the Include List. However, in order to verify whether the node is still remaining in the network or not, the default router sends a series of messages for checking before it makes decision.

Another mechanism is "fast leave". The node that wants to leave the multicast group sends a leave report to the router (MLD message in MLD protocol and MLRv2 message in MLDv2 protocol). In this mechanism, only one message is required in the leaving process if this message can be successfully received and accepted by the router.

The Efficient NDP

In the Efficient NDP, the node that wants to leave the network should send a de-registration message to its NEAR. De-registration message is a unicast NS with an ARO and contains the information of Registration Lifetime set to zero. If in some cases, the NEAR does not receive the de-registration message, the NCE of that node will be kept until the Registration Lifetime expires.

4.2 Nodes movements within the network

In wireless networks, the location of the nodes might change at anytime. Usually, the node which changes its location does not need to send any messages to inform other nodes of this activity, but a location change might cause one or more of the following issues:

- If a node moves to a place where the wireless network is unstable (the WiFi signal strength is weak or signal interference is high), it might lose the network connection.
- A node attempts to re-connect to the network which it lost connection to or it previously visited.
- A node changes its location and joins another subnet.

The following diagrams describe the impact and procedure of nodes movements.

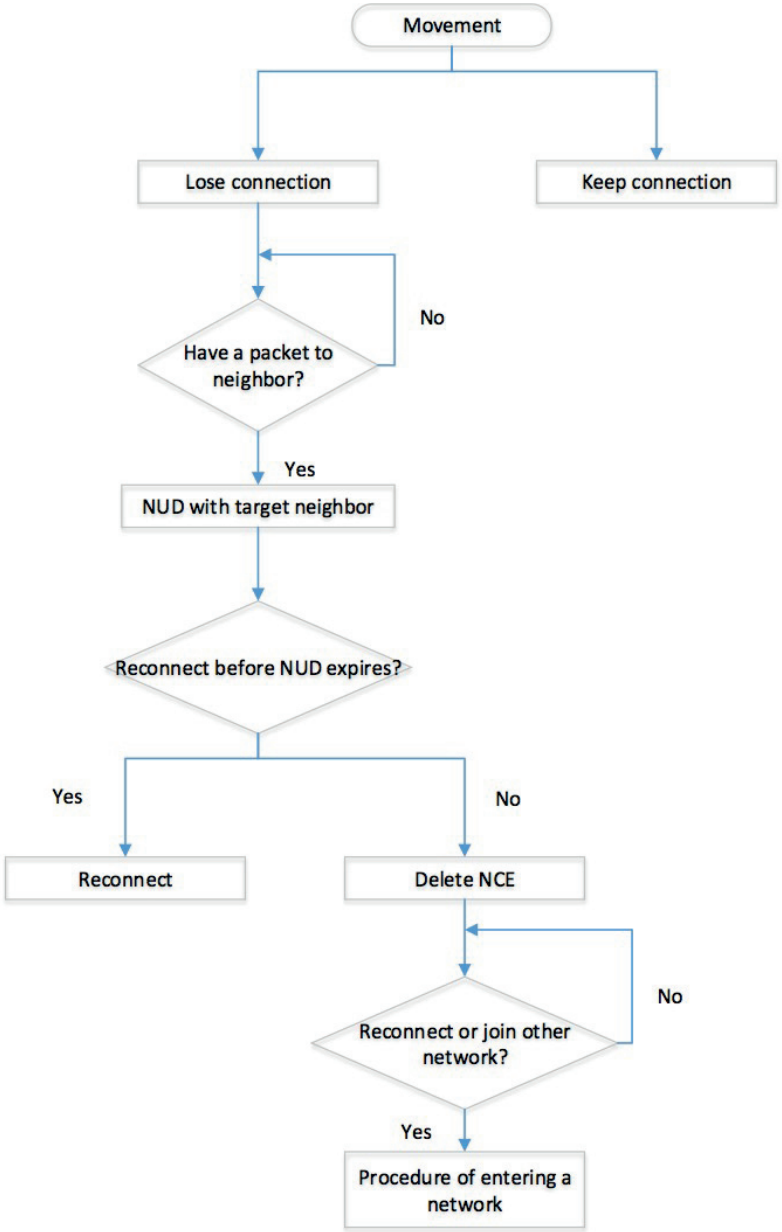

Figure 4.1: The legacy NDP movement procedure

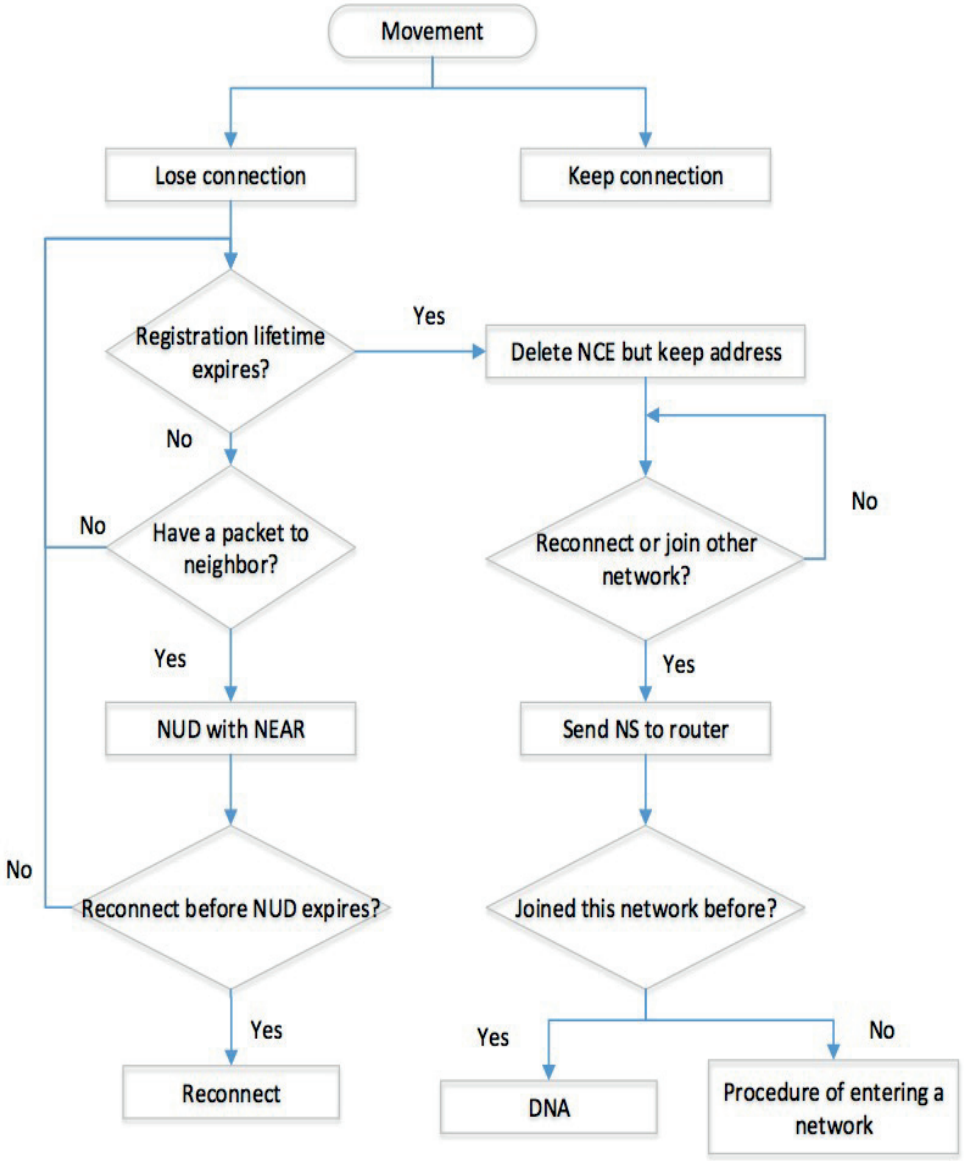

Figure 4.2: The Efficient NDP movement procedure

4.2.1 Losing connection

As described in Chapter 2, nodes use NCE to keep attention of neighbors reachability. A node itself never knows whether it loses network connection or not. The only knowledge is the reachability to its neighbors according to the record in NCE. When the node notices that it has problems to communicate with its neighbors, it performs NUD.

The legacy NDP

In legacy NDP, if a node does not have any packets to send to its neighbors, even after connection loss for a long time, the neighbors NCE is still kept in REACHABLE state.

When this node has a packet to a target neighbor, it realizes the unreachability and then performs NUD with that target neighbor. As described in Chapter 2, a series of packets are sent during NUD process. When the node has a packet to another target neighbor, the same process happens.

Assume in a network, nodes have to communicate with each other periodically. The total amount of messages sent by a node which loses its connection should be the number of nodes within the network multiplied by the number of packets for each NUD process.

The Efficient NDP

In the Efficient NDP, the node which loses connection also performs NUD, but the target is its NEAR. As described in Chapter 3, if a node has a packet to a neighbor, it first sends this packet to its NEAR. Once the node realizes it does not get any reply for the request, it begins to check the reachability to its NEAR and determines the state of NCE.

If the NEAR fails the NUD check, the state is marked as UNREACHABLE. After this, if this node has another packet to any neighbor, it attempts to communicate with its NEAR by sending the packet again. However, as the NEAR's NCE has already been in UNREACHABLE state, no NUD is performed this time.

After the Registration Lifetime expires, the NCE will then be deleted.

4.2.2 Re-connecting

Nodes can also re-connect to the network and re-establish the connection with their neighbors. However, the procedure depends on whether the target neighbor's NCE has been deleted or not.

The legacy NDP

If a node re-connects to the network before it deletes the NCE of the default

router, it is still a member of this network. The only thing it has to do is to perform address resolution with neighbors which are not recorded in the NCE table.

If a node re-connects to the network after it deletes the NCE of the default router, it has to follow the entering process explained in Chapter 4.1.1.

The Efficient NDP

If a node re-connects to the network before it deletes the NCE of the NEAR, it does not need to do anything. When it sends Registration NS to its NEAR and obtains a reply, the NCE state will be changed back to REACHABLE.

If a node re-connects to the network after it deletes the NCE of the NEAR, the situation is the same as if a node joins a network it previously visited. This will be explained in the next section.

4.2.3 Movement between subnets

Assume that there is a node which is a member of Network A. After a location change, this node comes to a place where the service of Network B is also available. This node can decide to become a member of Network B instead of Network A, or become a member of both Network A and Network B (roaming).

In the first situation, the node simply performs a leaving process or a losing connection process of Network A and performs an entering process of Network B. In the second situation, the node performs an entering process of Network B with no action of Network A. We are interested of the entering process of Network B.

The legacy NDP

The node follows the entering process explained in Chapter 4.1.1.

The Efficient NDP

If a node enters or re-connects to the network it previously visited, it will follow the procedure introduced in Simple Detecting Network Attachment (Simple DNA), RFC 6059 [18].

In this mechanism, a node needs to always maintain the Simple DNA address table (SDAT) which includes the information of a router's address and the address configuration methods (SLAAC or DHCPv6). The node uses a unicast NS to detect whether a previously encountered router is on the link and the previous address configuration of itself is still valid. If it is not, then the node relies on RA for configuration.

Chapter 5

Network Model Design

To accurately evaluate the performance between legacy NDP and the Efficient NDP in dynamic networks, proper network models need to be designed before simulations. According to this goal, network model design should follow these principles:

- The network topologies should be designed the same for legacy NDP and the Efficient NDP in each comparison case. They should have the same assigned parameters, eg: the number of hosts and routers in the model, the processing time of a packet, channel parameters.
- In this thesis, we only focus on the impact of specific events in a network. Irrelevant node behavior such as periodical RA exchanging, address updating are not taken into consideration.
- Events should follow proper distributions and the network scalability should be considered.

Chapter 5.1 and 5.2 describes the network model design for two dynamic network scenarios.

5.1 Entering and leaving scenario

5.1.1 Overview

In this network model, we assume all the nodes are new nodes, which means that they never joined any networks and have no IP address configured.

In legacy NDP, a node follows three steps to finish the procedure of entering a network: Router discovery, DAD and joining a multicast group. In the Efficient NDP, a node follows two steps to finish the procedure of entering a network: Router discovery and address registration.

As described in Chapter 4, nodes can implement either soft leave or fast leave mechanism to leave a network or a group in legacy NDP. In this network model, only the fast leave mechanism is used as this is the best leaving case that requires the least messages. In the Efficient NDP, a node performs de-registration when it leaves group.

5.1.2 Network topology and description

Figures 5.1 illustrates the network topology used in this scenario.

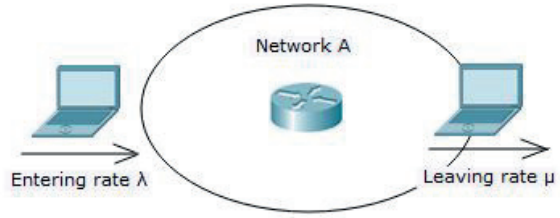

Figure 5.1: Entering & leaving scenario topology

Network A only has a router at the beginning. Incoming nodes enter Network A with an entering rate λ while outgoing nodes leave Network A with a leaving rate μ.

From the time, the system boots up until a certain defined time Ts, nodes enter Network A with an entering rate $\lambda 1$ and nodes leave Network A with a leaving rate $\mu 1$. $\lambda 1$ should be larger than $\mu 1$ to make sure Network A always has nodes. In this time period, with the time increasing, there should be more and more nodes staying in Network A. As Network A has a certain subnet size, the usage rate of the subnet increases. However, the number of nodes within Network A should not exceed the subnet capacity. In this situation, entering events dominate the system.

From Ts until the end of the simulation, nodes enter Network A with an entering rate $\lambda 2$ and nodes leave Network A with a leaving rate $\mu 2$. $\lambda 2$ is defined to be smaller than $\mu 2$. In this time period, with the time increasing, there should be less and less nodes staying in Network A and the usage rate of the subnet decreases. However, Network A should always have nodes. In this situation, leaving events dominate the system.

Both entering events and leaving events follow a uniform distribution. For example, if the entering rate λ is 5 nodes per hour, the first entering event happens at a random time between 0 to 12 minutes and the second entering event happens at a random time between 12 to 24 minutes. The reason for this distribution and the parameters defined in this network model will be discussed in the next chapter.

5.2 Movements scenario

5.2.1 Overview

In this thesis, connection loss caused by node movements is modeled and simulated.

At the beginning of the simulation, all the nodes are full function members within a network. Each node has a unique IP address and the knowledge of all the neighbor link-layer addresses, the NCE state of each neighbor is REACHABLE.

After a node loses its connection, it performs the NUD check with other nodes which are recorded in its NCE table. As we focus on the losing connection effect in this scenario, no re-connecting events happen in the simulation, thus this connection lost node will delete the neighbor NCEs after the NUD.

5.2.2 Network topology and description

Figures 5.2 illustrates the network topology used in this scenario.

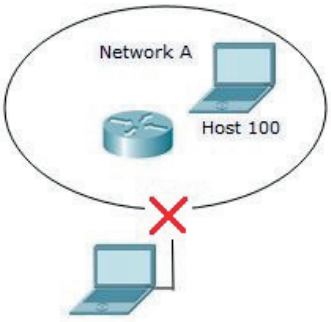

Figure 5.2: Losing connection topology

Network A has a router and 100 hosts which is represented by Host 100 at the beginning (The initial number of hosts can be different values, more information will be discussed in the next chapter). After the system boots up, hosts lose their connection to Network A with a losing connection rate α.

The losing connection events in this network follow a uniform distribution. Which means that a losing connection event is expected to happen at a random time in its respective time interval.

There are no re-connecting events or entering and leaving events in Network A. With the time increasing, there are less and less nodes remaining in the network.

As described before, NUD happens when a node has packets to its neighbors. To achieve this, periodically multicast user packets are generated by each host at specific defined times. These user packets are sent to all the neighbors recorded in NCE table and expected with a reply back. However, only the packets sent by the connection lost hosts are what we should pay attention to, as these packets can not reach neighbors, NUD check is required.

The parameters defined in this network model will be discussed in the next chapter.

Chapter 6

Simulations

In legacy NDP, nodes use multicast messages to configure themselves and update neighbor information. In the Efficient NDP, many of these multicast messages are optimized as the unicast traffic in order to reduce network overload and power consumption. To evaluate the efficiency of this change, simulations are required.

In this chapter, the simulations of the legacy NDP and the Efficient NDP in dynamic network environments are presented. These simulations are built on the network models described in Chapter 5, for analyzing nodes entering and leaving scenario and the losing connection case in nodes movement scenario. According to the amount of the total exchanged messages recorded from the simulations, an estimation of power consumption in these two protocols can be made. The results of the simulations will be presented in the next chapter.

OMNeT++ is the software used for this simulation work, its running environment, usage and project design will be introduced in Chapter 6.1. The simulation parameters, discrete event distribution and simulation design for each scenario will be discussed in Chapter 6.2 and Chapter 6.3. The limitations in our simulation models are stated in Chapter 6.4.

6.1 Simulation tool overview

As a continuation of thesis work *"Efficient IPv6 Neighbor Discovery in Wireless Environment"* [8], the same software OMNeT++ newest version 5.1 is chosen as the platform in this thesis simulation.

OMNeT++ is a C++ based software which is free for academic and education usage and can be downloaded from the webpage [19]. It runs on Windows, Linux, Mac OSX, and other Unix-like systems and has different release and installation requirements for each of these systems [20].

OMNeT++ is an open-source modular discrete event network simulator used

for modeling telecommunication networks, network protocols, queuing systems, distributed systems, etc. It provides a hierarchical network structure which is generally consisted by three layers as Figure 6.1 shown. Users can model simple modules first and group them into compound modules. The largest compound module is called network and it is consisted by all the other compound modules and simple modules.

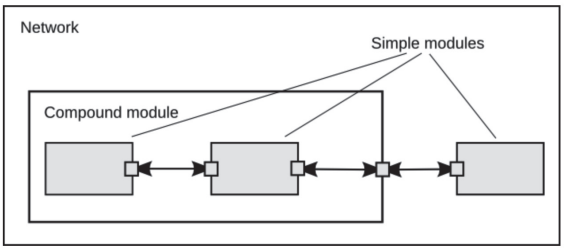

Figure 6.1: Simple and compound modules [21]

In OMNeT++, users use Network Description (NED) language to build components, connect and assemble these components as a system and define communication parameters like channel delay, bandwidth, etc. Instead of entering code to achieve these, for many Operating Systems (OS), OMNeT++ also provides an Integrated Development Environment (IDE) which allows users to place the components, configure parameters and define communication types in an operation Graphical User Interface (GUI). User's operation in GUI will be automatically transferred into code in the NED source file. Figure 6.2 and Figure 6.3 illustrates a simple example of connecting a LAN in NED design and its transferred code in NED source file.

For each OMNeT++ project, modules have to be defined in source files. The programming language of these source files can be C or C++. Module's behavior such as generating a packet, handling a packet and forwarding a packet in the simulation will follow the definition in the source files. OMNeT++ has many cooperation libraries such as INET to provide already defined modules. By using these libraries, users can simply import modules from the libraries and do not need to always define them in source files by themselves.

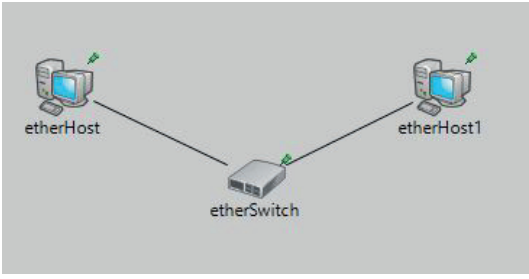

Figure 6.2: Connect LAN by NED design

```
package inet.examples.ethernet.lans.lab;

import inet.node.ethernet.EtherHost;
import inet.node.ethernet.EtherSwitch;


network Test
{
    @display("bgb=431,287");
    submodules:
        etherHost: EtherHost {
            @display("p=69,79");
        }
        etherHost1: EtherHost {
            @display("p=330,79");
        }
        etherSwitch: EtherSwitch {
            @display("p=202,142");
        }
    connections:
        etherHost.ethg <--> etherSwitch.ethg++;
        etherSwitch.ethg++ <--> etherHost1.ethg;
}
```

Figure 6.3: NED source code

OMNeT++ also needs an initialization (INI) file to control the simulation. The simulation running time, source and destination ports and message parameters can be defined in this file.

6.2 Entering and leaving scenario

In entering and leaving scenario, the simulation of the router discovery process was finished in thesis work [8]. In this thesis work, a full entering process which includes router discovery, DAD and nodes joining a multicast group (this only happens in legacy NDP) is simulated. Further more, leaving events are also added into this system to present a dynamic network function.

As described in Chapter 5, the simulation of the entering and leaving scenario can be recognized as two phases. In the first phase, the nodes entering rate is larger than the leaving rate while in the second phase it is smaller. This gives us a view of how many messages that will be exchanged in either entering events or leaving events dominate the system situation.

6.2.1 Simulation parameters and event distribution

In order to perform and see the effect of DAD, a certain network prefix has to be defined at the beginning of the simulation. In this simulation, 512 ($2 \times 16 \times 16$) usable addresses are assigned in the network. While one address is assigned for the router's address and one address is assigned for the network address, 510 remaining addresses are reserved for host addressing.

In this simulation, we are only interested in how many messages are exchanged during the entering process and the leaving process. Thus, periodical exchanged messages such as periodical RA, Efficient NDP registration messages are not taken into consideration. If there are more hosts remaining in the network, more messages are required as these messages aim at all the nodes or hosts in the network. Furthermore, as a certain number of addresses (510) is reserved for hosts, the more hosts remaining in the network, the higher the possibility is for a new incoming host to select a duplicated address.

As described before, each simulation is divided into two phases. In the first phase, hosts enter network with an entering rate $\lambda 1$. Define that after half of the simulation time, it turns into the second phase until the end of the simulation. During this period, hosts enter network with an entering rate $\lambda 2$, $\lambda 2 = 1/2\ \lambda 1$.

In the first phase, hosts leave the network with a leaving rate $\mu 1$, define $\mu 1 = 1/2\ \lambda 1$ which means that the leaving rate is half of the entering rate. In the second phase, hosts leave network with a leaving rate $\mu 2$, define $\mu 2 = 2\ \lambda 2$ which means that the leaving rate is twice as the entering rate during this period.

A uniform distribution is used to describe the events in this simulation to ensure events can happen randomly in their respective time intervals. We also use this distribution to control the entering rate and the leaving rate.

However, due to the fact that a host should always enter the network before leave. We define the first leaving event is located in the second time interval. In each simulation model, the total number of leaving events is one less than the number of the entering events.

Network scalability is important to evaluate the network performance. Considering this, up to 300 entering events are expected to happen in a long simulation time. However, to compare the performance between legacy NDP and the Efficient NDP in this process, other numbers of entering events are also simulated. Table 6.1 presents this.

Simulation time	**Hosts**	**Entering event distribution**	**Leaving event distribution****
2h	9	0 - 1h, (0, 600)* 1 - 2h, (0, 1200)	0 - 1h, (0, 1200) 1 - 2h, (0, 600)
6h	30	0 - 3h, (0, 540) 3 - 6h, (0, 1080)	0 - 3h, (0, 1080) 3 - 6h, (0, 540)
16h	120	0 - 8h, (0, 360) 8 - 16h, (0, 720)	0 - 8h, (0, 720) 8 - 16h, (0, 360)
24h	300	0 - 12h, (0, 216) 12 - 24h, (0, 432)	0 - 12h, (0, 432) 12 - 24h, (0, 216)

* An example of this distribution model can be found in Figure 6.4
** The first leaving event is located in the second time interval

Table 6.1: Events uniform distribution time intervals in seconds

In the simulations, channel delay and packet processing time is also considered. Their values may vary depending on which kind of packets are transmitted and processed. In the most busy case, this value is set to 1 second, which means that after a node transmits a packet, the other node takes up to 1 second to finish the procedure of receiving and processing the packet.

In the entering process, DAD can fail many times if the entering host always selects unavailable addresses. Considering this situation, additional time has to be given to the process. In each simulation case, this time is set as 40 seconds.

For example, in the first simulation case, 9 hosts will enter the network in 2 hours. The first 6 hosts will enter the network in the first hour and the rest in the following hour. The first entering event happens and finishes during the time 0 to 600 seconds. At a random time T during the period 0 to 560 seconds, the first host host[0] enters the network. During T to T + 40 seconds, this host begins and finishes to perform router discovery, DAD and joining a multicast group.

In the first leaving event respective time interval 1200 to 2400 seconds, host[0] leaves the network by sending a notification message to the router. Additional 1 second is given to host[0] to make sure it can finish its leaving process in its respective time interval.

Figure 6.4 illustrates the example described above.

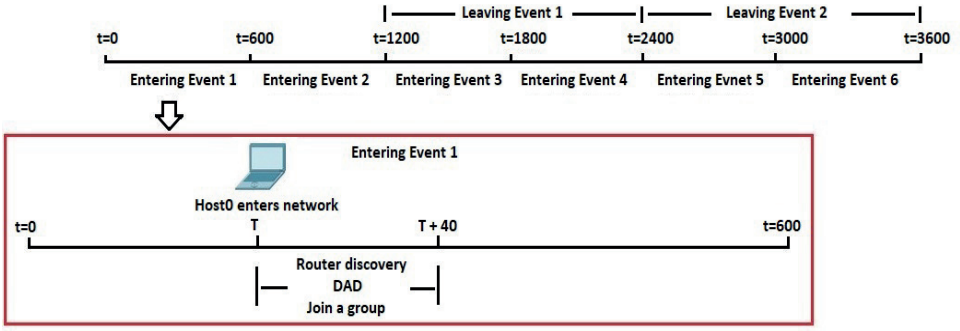

Figure 6.4: Example events schedule

In this simulation scenario, the channel is assumed as an ideal channel, the channel delay is always kept to the configured value. The system is assumed as a reliable system, all the packets can successfully pass the validity check and there is no packet loss or retransmission.

6.2.2 Simulation details

The simulation models were built based on the TicToc model from OMNeT++ library, which gives an example of packets transmission between nodes. However, by editing the source files, we get more control of the system. A defined type of packets can be generated in a specific time interval and sent from a specific port to its destination. Nodes follow a defined rule to handle a packet and decide their next action depends on the packet type. The implementation and detailed explanation can be found in Appendix A.

The legacy NDP

Figure 6.5 illustrates the simulation model for legacy NDP.

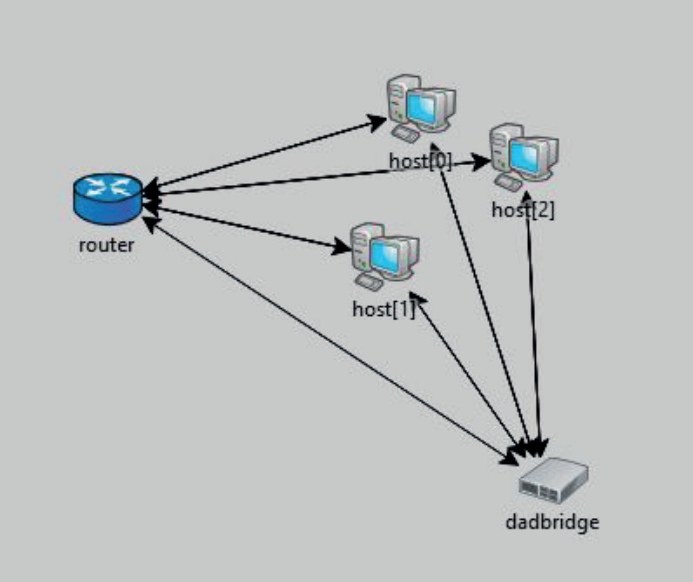

Figure 6.5: Simulation model for legacy NDP (3 hosts)

In this simulation model, each host is initially connected with a router and a *dadbridge*. However, only after the host joins the network, it is able to send or receive packets. And after it leaves the network, it is not able to send or receive any packets. Hosts do not have direct connection with other hosts.

dadbridge is a logic existence in this simulation model which functions as a communication bridge between hosts, and handles DAD related issues. We reduce the complexity of node behavior by adding this bridge in our simulation model. As *dadbridge* does not exist in the real networks, all the packets received by *dadbridge* should not be counted in the record.

There are no actual IP addresses configured for any nodes in this system, thus

dadbridge uses host ID to perform DAD. *dadbridge* has a counter to record how many hosts have left the network. The counter has an initial value as 0 and once *dadbridge* receives a MLRv2 leave message, the counter is added by 1. As the hosts follow the order of their ID to enter and leave network one by one, it is very clear to know which hosts remain in the network. For example, if the entering host's ID is host[3] and counter value is 1, it means that host[0] has left the network while host[1] and host[2] are still remaining in the network.

As described before, there are 510 addresses reserved for host addressing. *dadbridge* takes a random number x between 0 to 509 to represent the entering host selected address. *dadbridge* also takes n (n equals to the number of remaining hosts in the network) different random numbers x_0 x_1 ... x_{n-1} between 0 to 509 to represent the addresses already used by hosts in the network. If the duplication happens (x equals to any number in x_0 x_1 ... x_{n-1}), then it indicates that a duplicated address is selected by the entering host.

The following steps describe the procedure of the system:

- Router Discovery: The entering events follow the uniform distribution descried in Chapter 6.2.1. Once the entering event happens, a host enters the network, it immediately sends a RS to the router. The router replies with a RA.
- DAD: The host that received RA sends a NS to *dadbridge*, *dadbridge* copies NS and multicasts them to all the other hosts remaining in the network. *dadbridge* performs DAD instead of the other hosts themselves. If DAD is successful, then nothing happens. Otherwise *dadbridge* multicasts NA to all the other nodes (without the NS sender host) in the network instead of the duplication address detected host, the entering host then sends a NS to *dadbridge* again. After the entering host successfully configures an address, it multicasts NA to all the other nodes to announce itself.
- Joining a multicast group: The entering host sends a MLRv2 enter message to the router.
- Leaving the network: The leaving events follow the uniform distribution descried in Chapter 6.2.1. Once the leaving event happens, the host sends a MLRv2 message to the router and *dadbridge*. *dadbridge* counter is added by 1.

The Efficient NDP

The simulation model for the Efficient NDP is very similar to the model for legacy NDP. Instead of performing DAD by a logic *dadbridge*, in the Efficient NDP, this is done by the NEAR itself.

The same as in legacy NDP, a counter with the initial value 0 is set by the router, once the router receives a de-registration message, the counter is added by 1. It is use to record the leaving events and monitor how many hosts are remaining in the network.

Figure 6.6 illustrates the simulation model for the Efficient NDP.

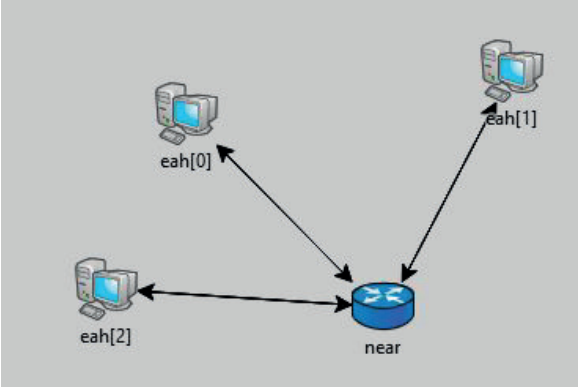

Figure 6.6: Simulation model for the Efficient NDP (3 hosts)

The following steps describes the procedure of the system:

- Router Discovery: The entering events follow the uniform distribution descried in Chapter 6.2.1. Once the entering event happens, a host enters the network, it immediately sends a RS to the router. The router replies with a RA.
- DAD: The host sends a registration NS to the router. Router performs DAD. If DAD is successful, then it sends a NA with DAD pass option. Otherwise the router sends a NA with DAD fail option, then the entering host sends a registration NS to the router again. The entering host does not need to announce its address, as the requested address which passes DAD will be recorded by the router and updated to other nodes when they request it.
- Leaving the network: The leaving events follow the uniform distribution descried in Chapter 6.2.1. Once the leaving event happens, the host sends a de-registration message to the router, and the router's counter is added by 1.

6.3 Movements scenario

As described in Chapter 4, nodes movement can cause many events and the most common one is losing connection. A node never knows if it has already lost the connection, it only knows whether its neighbors are reachable or not. If it does not receive an expected packet from the neighbor, it then performs a reachability check.

A node follows many steps in NUD, however, only the message exchange part is what we are interested in.

6.3.1 Simulation parameters and event distribution

At the beginning of the simulation, there is a network with n hosts (n = 30, 50, 100) and one router. In order to see the impact of losing connection events, there are no entering, leaving or re-connecting events happen. The number of hosts in this network is reduced by losing connection events.

As described in Chapter 2, a host follows a series of steps to check the reachability to its neighbors. In the first few steps, a host sets different kinds of timer to change the state of NCE. When the NCE enters the PROBE state, the host begins to retransmit NS probe messages to the target neighbor and this is the only part with message exchange in NUD. In this simulation, instead of setting a timer to control retransmission of NS probes, we simply define a host can maximally transmit x (x = 5, 10) probes.

The expected result caused by a host losing connection is that it deletes all the neighbor NCEs. In legacy NDP, once a NCE enters the UNREACHABLE state, it is deleted immediately. However, in the Efficient NDP, the UNREACHABLE state NCE is kept until the registration time expires.

In the simulation, irrelevant messages such as periodical RA are not performed. In the Efficient NDP, periodical registration NS and registration confirmation NA are performed, as they are important for the NCE state. The Registration Lifetime is set by default as 9000 seconds and the hosts wake up at 2/3 of the Registration Lifetime to refresh their registration by sending a NS to the NEAR. It means that when a host receives a registration confirmation NA from the NEAR, it goes into the sleeping cycle. After 6000 seconds, it wakes up and sends a registration NS to the NEAR. At the beginning of the simulation (simulation time = 0), every host sends a registration NS to the NEAR.

In order to perform NUD, periodical user packets are used in the simulation. User packets are generated by each host and they are aimed at all the other nodes which have existed in the NCE table with a period 1 hour (The first series of user packets are generated at 3600 seconds, the second series of user packets are generated at 7200 seconds, etc).

For example, in legacy NDP, host[0] (the first connection lost host) loses connection at 2400 seconds. At 3600 seconds, it generates a user packet and multicasts it to all the other nodes, as their NCEs are existing in host[0]'s NCE table, then it performs NUD for all the other hosts and deletes their NCEs. At 7200 seconds, it generates another user packet and attempts to forward it, but this time it does not have any neighbor NCEs, so no packet is sent.

Each losing connection event is expected to happen at a random time located within its respective time interval. A uniform distribution is used to describe the event schedule. Considering the network scalability, simulations are performed in a long simulation time with different numbers of losing connection events. Table 6.2 presents this.

Initial hosts number	NUD probes number	Simulation time*	Events number	Events schedule
30	5	2h	1	(0, 7200)
30	5	4h	5	(0, 2880)**
30	5	12h	20	(0, 2160)
30	10	2h	1	(0, 7200)
30	10	4h	5	(0, 2880)
30	10	12h	20	(0, 2160)
50	5	2h	1	(0, 7200)
50	5	4h	5	(0, 2880)
50	5	12h	20	(0, 2160)
50	5	16h	50	(0, 1152)
50	10	2h	1	(0, 7200)
50	10	4h	5	(0, 2880)
50	10	12h	20	(0, 2160)
50	10	16h	50	(0, 1152)
100	5	2h	1	(0, 7200)
100	5	4h	5	(0, 2880)
100	5	12h	20	(0, 2160)
100	5	16h	50	(0, 1152)
100	5	24h	100	(0, 864)
100	10	2h	1	(0, 7200)
100	10	4h	5	(0, 2880)
100	10	12h	20	(0, 2160)
100	10	16h	50	(0, 1152)
100	10	24h	100	(0, 864)

* Additional 100 seconds is given for each simulation to finish exchanging probes
** An explanation for this value can be found under the table

Table 6.2: Events distribution time interval in seconds

For example, in the case mentioned in the table. 5 losing connection events happen in 4 hours simulation time. As they follow the uniform distribution, the simulation time (4 hours) should be divided into 5 equal time thresholds (Every 2880 seconds). And each event happens randomly in its respective time threshold (event 1 happens somewhere in 0 - 2880 seconds, event 2 happens somewhere in 2880 - 5760 second and so on).

In the simulation, channel delay and packet processing time are also implemented. The number of the hosts remaining in the network does not have any im-

pact on the exchanged messages, as hosts send packets to all the targets recorded in their NCEs. On the other hand, NUD check does not happen immediately after a losing connection event, it begins at the next packet sending time. Thus, hosts do not need to have any additional time reserving for NUD in their event time intervals. However, the last periodical packets are always generated at the end of the simulation time as the table above shows. Hosts which rely on these packets for NUD can not finish the NUD check. Considering this, additional 100 seconds is added for each simulation.

6.3.2 Simulation details

The simulation models were built based on the TicToc model from OMNeT++ library. The implementation and detailed explanation can be found in Appendix B.

The legacy NDP

Figure 6.7 illustrates the simulation model for legacy NDP.

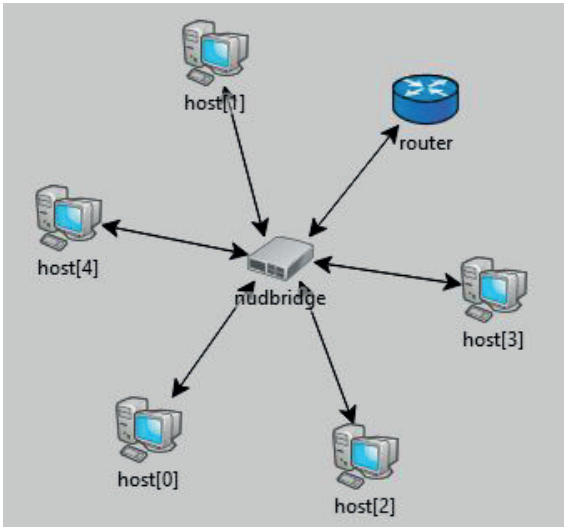

Figure 6.7: Movement model legacy NDP (5 hosts)

The example model above gives a view of the system with one router and 5 hosts. However, in the real simulation model, there are n hosts with ID host[0] to host[n] existed in the topology. The router in this simulation model only has the function of receiving packets.

In real wireless networks, each node (host and router) should have a connection to other nodes in the network. To simplify this, in this simulation model, a bridge *nudbridge* is designed as a middle point for connections. If a host has a multicast

packet which aims at all the other nodes in the system, it sends the packet to *nudbridge*, *nudbridge* then copies the packet and multicasts the copy to all the other nodes in the system. It needs to be mentioned that *nudbridge* is only used to simplify the system connection and does not exist in real networks, thus the packets received by the *nudbridge* should not be counted in the record.

The following steps describe the procedure of the simulation:

- Losing connection event: The losing connection event of a specific host with host ID host[i] happens at a random time during its respective time interval described in Chapter 6.3.1.
- Sending user packet: When it reaches the next user packet sending time, host[i] sends a user packet to *nudbridge*, *nudbridge* multicasts this packet to all the nodes except host[i] in the system. These packets are unreachable to these nodes, as host[i] has lost its connection to this network.
- NUD: host[i] expects to receive reply for each user packet it sent. However, losing connection makes this impossible. After 5 seconds, host[i] begins to send the first NS probe to *nudbridge*, *nudbridge* multicasts the NS probe to all the nodes except host[i] in the system. After this, with a time duration of every 5 seconds, the same process happens until the 5th or the 10th NS probe is multicasted by *nudbridge*.

The Efficient NDP

Figure 6.8 illustrates the simulation model for the Efficient NDP.

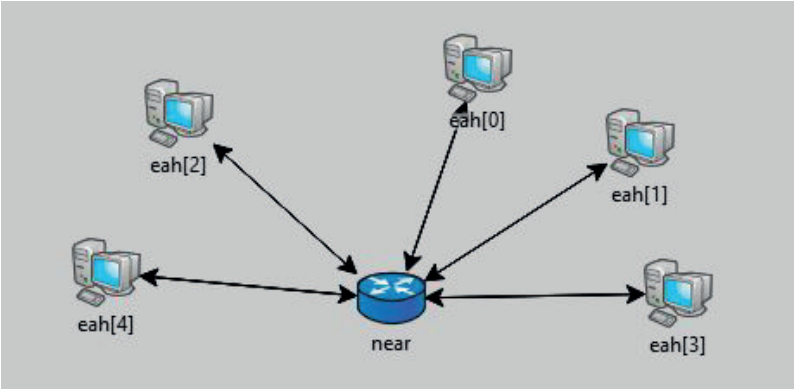

Figure 6.8: Movement model the Efficient NDP (5 hosts)

The same as in the simulation model for legacy NDP, we use 5 hosts in the example model figure above while there are n hosts existing in the real simulation model.

In this simulation model, NEAR is used as the middle point for the connection

of all the hosts, to instead of *nudbridge* in legacy NDP model. As described before, NEAR is the target that hosts perform NUD with.

The tricky part is that not only the user packet sent by a host can trigger NUD, the periodical registration NS can also trigger NUD. The sending time of user packets and registration NS is mentioned in Chapter 6.3.1. To explain this, a simple example is shown in figure 6.9. In this example, a distribution of 5 hosts in 4 hours simulation time, 30 initial hosts in the system, 5 probes in NUD, is used.

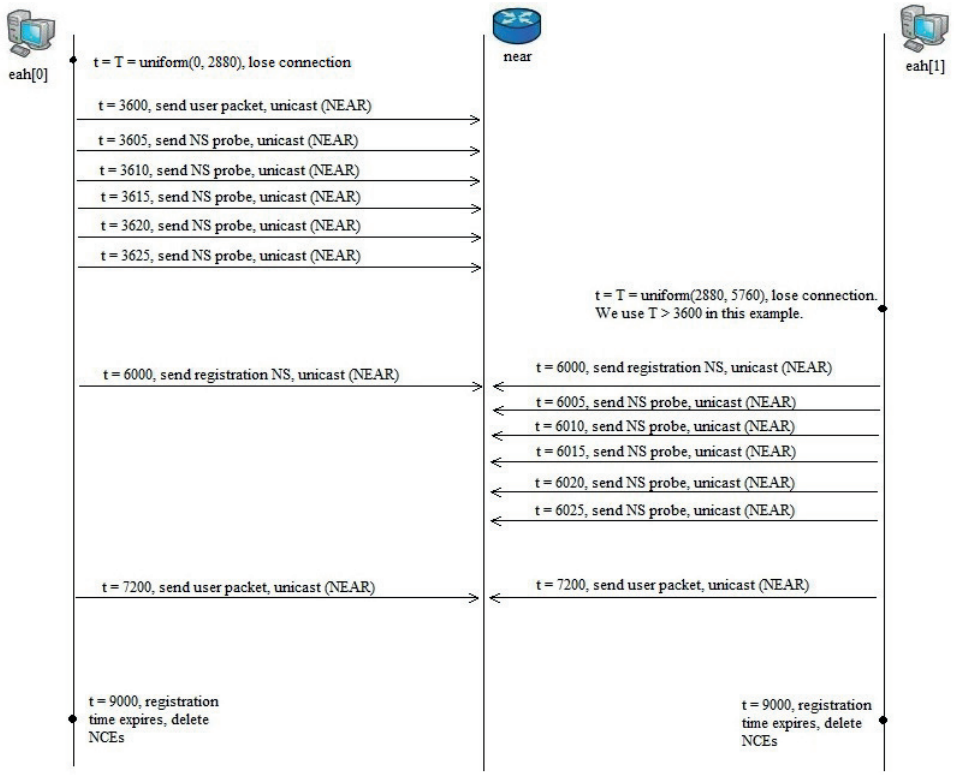

Figure 6.9: Example of Efficient NDP movement model

In the figure above, the losing connection event of eah[1] is assumed that is happening somewhere in (3600, 5760). However, if it is located somewhere in (2880, 3600), the process is the same as eah[0].

In the example above, the losing connection event of eah[2] is located in a uniform distribution time interval (5760, 8640). There can be three cases to describe the behavior of eah[2].

Event located somewhere in (5760, 6000)

The process for eah[2] is completely the same as eah[1] described in the example figure.

Event located somewhere in (6000, 7200)

- At the 6000th second, eah[2] sends a registration NS to the near. The near replies with a registration confirmation NA. The Registration Lifetime of eah[2] gets refreshed for additional 9000 seconds and will be expired at the 15000th second.
- At the 7200th second, eah[2] sends a user packet, then it sends 5 NS probes.
- At the 10800th second, eah[2] sends a user packet.
- At the 12000th second, eah[2] sends a registration NS.
- At the 14400th second, eah[2] sends a user packet.
- At the 15000th second, eah[2] deletes NCEs.

In the last five steps above, there is no response from the near.

Event located somewhere in (7200, 8640)

- At the 6000th second, eah[2] sends a registration NS to the near. The near replies with a registration confirmation NA. The Registration Lifetime of eah[2] gets refreshed for additional 9000 seconds and will be expired at the 15000th second.
- At the 10800th second, eah[2] sends a user packet, then it sends 5 NS probes.
- At the 12000th second, eah[2] sends a registration NS.
- At the 14400th second, eah[2] sends a user packet.
- At the 15000th second, eah[2] deletes NCEs.

In the last four steps above, there is no response from the near.

In this simulation model, many periodical NS are sent in order to refresh the registration time of hosts. However, as defined before, the exchanged messages in the losing connection scenario should be counted after the losing connection event happens. Only these messages without a reply are counted to evaluate the performance of the Efficient NDP.

6.4 Limitations

The message transmission is assumed as 100% reliable transmission in our simulation models, thus there is no packet drop and re-transmission mechanism in the

simulations. In real networks, the number of exchanged messages could be more than the results we obtained from simulations due to re-transmission. On the other hand, network overload situation and queue systems are not implemented in our simulations, they can affect the processing time and make the system more complex, and the number of the exchanged messages could be different by these features.

There is no exact IP address configuration and table implementation in our simulation models. As described before, in our analyzed cases, a main part requires IP addresses and tables is DAD in the entering and leaving scenario. As it is too complex to configure exact IP address for each node and implement tables to record neighbor address information, we simply use random number selection to achieve this. In fact, the address auto-configuration is also a random selection process in real networks. However, implementing IP addresses and tables can make simulations more similar to real networks.

Channel delay and packet processing time are added to the models, but always kept in specific values. In wireless networks, channel delay might be different in communication links. Packet processing time may also vary depends on which packet is processed by which device.

In the movement scenario, user packets are assumed as periodical packets every 3600 seconds and the Registration Lifetime is set as 9000 seconds by default. The results from the simulations can be different if we use other values instead. As time is limited for this thesis, we only take the defined situation into simulations. More simulation models for further situation could be done in the future.

Chapter 7

Results

The results of the two dynamic network simulation scenarios described in Chapter 6 are presented in this chapter. As explained in Chapter 6, for both the nodes entering and leaving scenario and the nodes movement scenario, several simulation models which correspond to different network environments were designed. According to this, a comprehensive analysis of how efficiency can be achieved by the Efficient NDP in different size of networks can be made.

7.1 Entering and leaving scenario

7.1.1 The legacy NDP results

As described in Chapter 4, a new entering host selects a random address in the network prefix and performs DAD. DAD can fail many times dependent on how many available addresses in the network and how large the network prefix is. In the 300 entering events simulation model, at the half simulation time (the 12th hour), there is quite a high possibility (close to 1/5) for DAD failing. This is presented by the number of NS and NA messages. Table 7.1 presents the results of legacy NDP entering and leaving scenario.

Hosts	RS	RA	MLRv2 entering	NS	NA	MLRv2 leaving	Total messages
9	9	9	9	17	25	8	77
30	30	30	30	163	193	29	475
120	120	120	120	2809	3051	119	6339
300	300	300	300	17063	19005	299	37267

Table 7.1: Messages in legacy NDP (entering & leaving)

7.1.2 The Efficient NDP results

In the Efficient NDP, the multicast traffic in DAD is optimized as the unicast traffic. A lot of messages can be saved in the DAD process, and this can be presented by the number of Registration NA and NS messages. The results are recorded in Table 7.2.

Hosts	RS	RA	Registration NS	NA	De-registration NS	Total messages
9	9	9	9	9	8	44
30	30	30	30	30	29	149
120	120	120	123	123	119	605
300	300	300	328	328	299	1555

Table 7.2: Messages in the Efficient NDP (entering & leaving)

7.1.3 Summary

The following graphs compare the results obtained from the simulation between legacy NDP and the Efficient NDP.

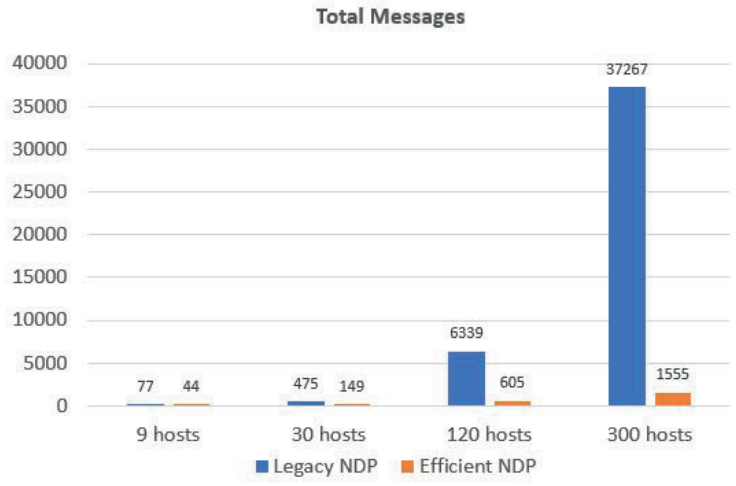

Figure 7.1: Comparison of total exchanged messages in entering & leaving scenario between legacy NDP and Efficient NDP

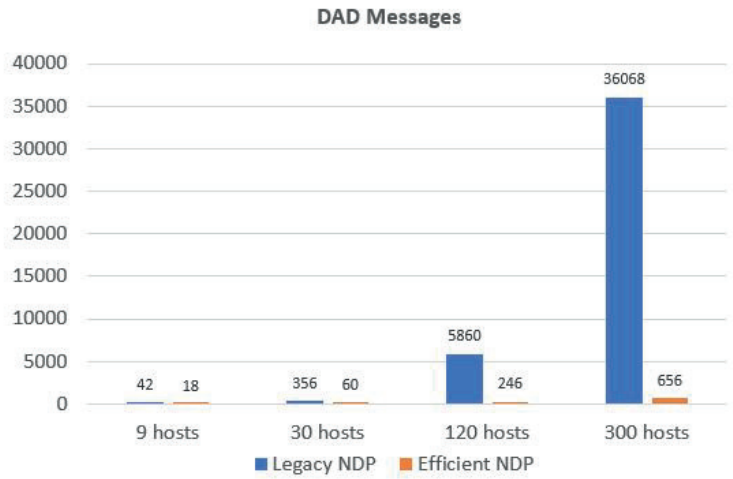

Figure 7.2: Comparison of DAD exchanged messages in entering & leaving scenario between legacy NDP and Efficient NDP

While other types of messages have the amounts equal to the entering or leaving event numbers, the number of the messages exchanged in the DAD process has a big impact to the number of messages exchanged in the network, especially in the situation where there are larger numbers of entering and leaving events. The reason is the messages used in DAD are optimized to the unicast messages in the Efficient NDP.

Table 7.3 and Table 7.4 conclude the message saving percentage results.

Hosts	Legacy NDP	Efficient NDP	Message Saving
9	77	44	42.86%
30	475	149	68.63%
120	6339	605	90.46%
300	37267	1555	95.83%

Table 7.3: Total messages saved by hosts

Hosts	Legacy NDP	Efficient NDP	Message Saving
9	42	18	57.14%
30	356	60	83.15%
120	5860	246	95.80%
300	36068	656	98.18%

Table 7.4: DAD messages saved by hosts (The saving percentage of multicast messages)

7.2 Movement scenario

Unlike entering and leaving scenario, a losing connection event is defined as a moment event which happens and finishes at the same time. To evaluate how many messages are used as the result of the losing connection events, periodical multicast user packets and periodical registration NS (only in the Efficient NDP) are used for triggering NUD and removing neighbor NCEs for the connection lost hosts.

7.2.1 The legacy NDP results

In our test case, all the connection lost hosts have to delete their neighbor NCEs as an indication of finishing the NUD check. However, as the number of NCEs equals to the initial host number, and the number of messages used in each NUD check is also defined as a constant. It gives us a deterministic analytical result before the simulation outcomes.

$$TotalMessages = MessageNUD \times NUDtargets \times Eventnumber \tag{7.1}$$

where *MessageNUD* is defined as the number of messages required in each NUD process (userpacket + NS probes), *NUDtargets* equals to the initial host number.

For example, in the case that 30 initial hosts, 5 probes in NUD, and 1 losing connection event. The number of the exchanged user packets should equal to:

$$Userpackets = 1 \times 30 \times 1 = 30 \tag{7.2}$$

where the first 1 represents one user packet sent to one neighbor, 30 represents the number of neighbors, and the last 1 represents one losing connection event.

The number of the exchanged NS probes should equal to:

$$NSprobes = 5 \times 30 \times 1 = 150 \tag{7.3}$$

where 5 represents the number of NS probes used in NUD check with one neighbor, 30 represents the number of neighbors, and 1 represents one losing connection event.

This gives us 180 messages are exchanged in this defined losing connection process. Table 7.5 presents the number of messages exchanged in 30 initial hosts, 5 probes for NUD case.

Events	User packet	NS probe	Total messages
1	30	150	180
5	150	750	900
20	600	3000	3600

Table 7.5: Messages exchanged in legacy NDP losing connection scenario with 30 initial hosts and 5 probes for NUD

Our simulations verify the results from the analytical calculation. Figure 7.3 is an example for the number of the messages exchanged in 30 initial hosts, 5 NS probes for NUD, 1 losing connection event outcome.

```
Event #194  t=7300  Tictoc3.nudbridge (NUDbridge, id=33)  on selfmsg End of Simulation (omnetpp::cMessage, id=7)
Event #195  t=7300  Tictoc3.nudbridge (NUDbridge, id=33)  on selfmsg User Packet = 30 (omnetpp::cMessage, id=8)
Event #196  t=7300  Tictoc3.nudbridge (NUDbridge, id=33)  on selfmsg NS probe = 150 (omnetpp::cMessage, id=9)
 No more events, simulation completed -- at t=7300s, event #196
Calling finish() methods of modules
```

Figure 7.3: Example simulation outcome

However, in all the test cases, the nodes which remain in the network at the end of the simulation time still have the NCEs of the connection lost hosts. As the simulations focus on the amount of the exchanged messages caused by the losing connection events, the nodes remaining in the network are not expected to perform NUD with all the connection lost hosts and delete their NCEs. On the other hand, if they do this, there will be more messages.

7.2.2 The Efficient NDP results

Similar to legacy NDP models, user packets are generated by the connection lost hosts every 3600 seconds for their NUD. In the Efficient NDP, periodical registration NS can also trigger the NUD, as they are expected to obtain NA from the NEAR as reply. The detailed explanation and an example can be found in Chapter 6.3.2.

Table 7.6 - 7.11 present the results obtained from the simulation.

Events	User packet	NS probe	Registration NS	Total messages
1	1	5	1	7
5	7	25	4	36
20	34	100	19	153

Table 7.6: Messages exchanged in the Efficient NDP losing connection scenario with 30 initial hosts and 5 probes for NUD

Events	User packet	NS probe	Registration NS	Total messages
1	1	10	1	12
5	7	50	4	61
20	34	200	19	253

Table 7.7: Messages exchanged in the Efficient NDP losing connection scenario with 30 initial hosts and 10 probes for NUD

Events	User packet	NS probe	Registration NS	Total messages
1	1	5	1	7
5	7	25	4	36
20	34	100	19	153
50	78	250	46	374

Table 7.8: Messages exchanged in the Efficient NDP losing connection scenario with 50 initial hosts and 5 probes for NUD

Events	User packet	NS probe	Registration NS	Total messages
1	1	10	1	12
5	7	50	4	61
20	34	200	19	253
50	78	500	46	624

Table 7.9: Messages exchanged in the Efficient NDP losing connection scenario with 50 initial hosts and 10 probes for NUD

Events	User packet	NS probe	Registration NS	Total messages
1	1	5	1	7
5	7	25	4	36
20	34	100	19	153
50	78	250	46	374
100	161	500	97	758

Table 7.10: Messages exchanged in the Efficient NDP losing connection scenario with 100 initial hosts and 5 probes for NUD

Events	User packet	NS probe	Registration NS	Total messages
1	1	10	1	12
5	7	50	4	61
20	34	200	19	253
50	78	500	46	624
100	161	1000	97	1258

Table 7.11: Messages exchanged in the Efficient NDP losing connection scenario with 100 initial hosts and 10 probes for NUD

It is obvious from the tables above, that for each simulation environment (30, 50 and 100 initial hosts in the simulation), the amounts of the exchanged

messages are the same. This is because that OMNeT++ implements Mersenne Twister PRNG algorithm [22] to generate random number, and this algorithm always generates the same result for the same simulation. We used this algorithm for the uniform distribution in our simulations as it is easy to implement. Although this algorithm makes the results a bit deterministic, the results recorded in the tables above can be seen as reliable: The user packets are generated every 3600 seconds while the registration NS are generated every 6000 seconds, in a large simulation time with many events, the user packets generated by the connection lost hosts should be 1.67 times of the registration NS generated by the connection lost hosts. This can be seen from the results.

Another thing is that according to the theoretical analysis, it is expected for each connection lost host to transmit one registration NS after its event. But in the record tables, these numbers are always less than the event numbers. The reason is that for the last few connection lost hosts, the next registration time is over the simulation time. These hosts have done the NUD and the NCEs are marked as UNREACHABLE, but their registration time has not expired and these NCEs has not been deleted.

7.2.3 Summary

The following graphs and tables compare the results of the number of exchanged messages and messages saving percentage for movement losing connection scenario.

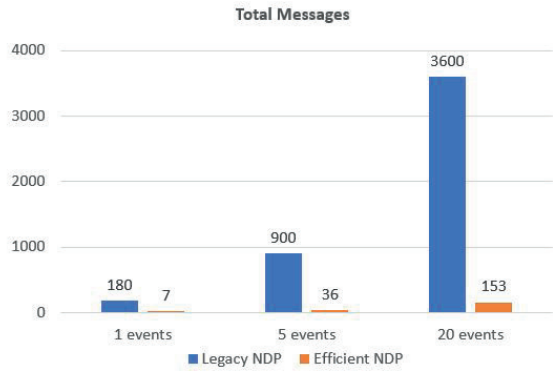

Figure 7.4: Exchanged messages, 30 initial hosts, 5 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	180	7	96.11%
5	900	36	96%
20	3600	153	95.75%

Table 7.12: Messages save, 30 initial hosts and 5 probes in NUD

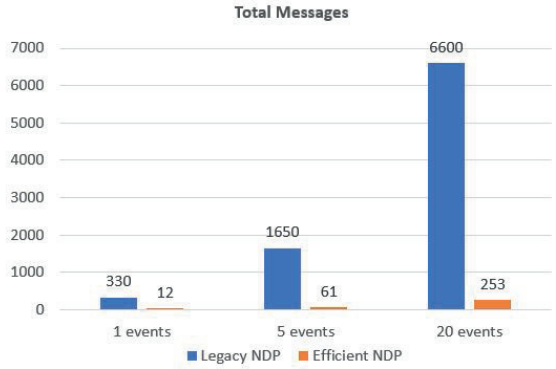

Figure 7.5: Exchanged messages, 30 initial hosts, 10 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	330	12	96.36%
5	1650	61	96.3%
20	6600	253	96.17%

Table 7.13: Messages save, 30 initial hosts and 10 probes in NUD

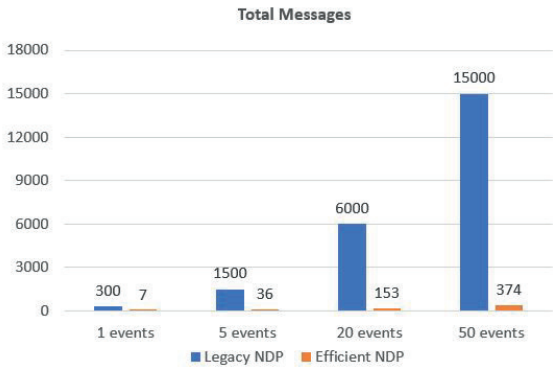

Figure 7.6: Exchanged messages, 50 initial hosts, 5 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	300	7	97.67%
5	1500	36	97.6%
20	6000	153	97.45%
50	15000	374	97.51%

Table 7.14: Messages save, 50 initial hosts and 5 probes in NUD

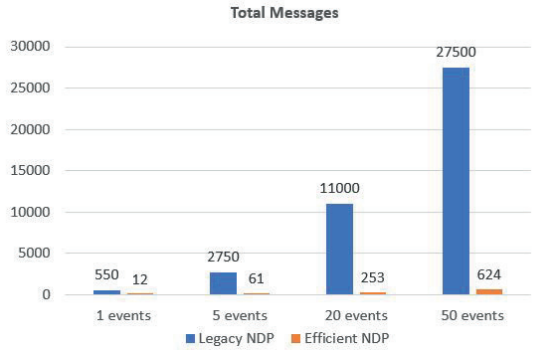

Figure 7.7: Exchanged messages, 50 initial hosts, 10 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	550	12	97.82%
5	2750	61	97.78%
20	11000	253	97.7%
50	27500	624	97.73%

Table 7.15: Messages save, 50 initial hosts and 10 probes in NUD

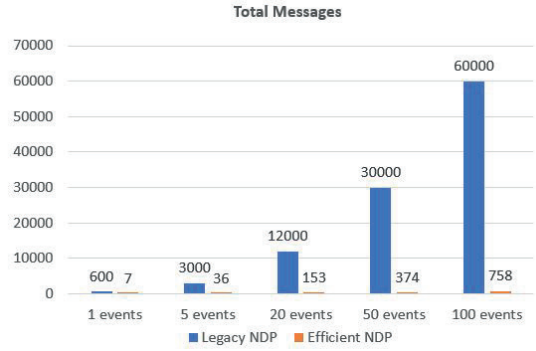

Figure 7.8: Exchanged messages, 100 initial hosts, 5 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	600	7	98.83%
5	3000	36	98.8%
20	12000	153	98.73%
50	30000	374	98.75%
100	60000	758	98.74%

Table 7.16: Messages save, 100 initial hosts and 5 probes in NUD

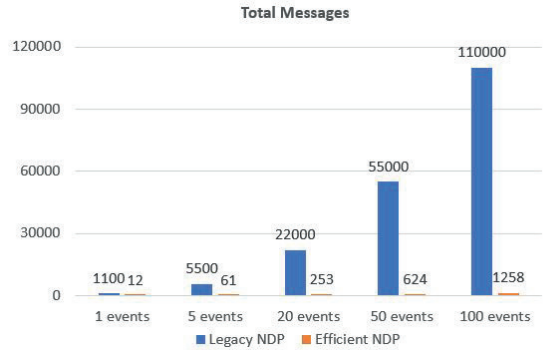

Figure 7.9: Exchanged messages, 100 initial hosts, 10 probes in NUD

Events	Legacy NDP	Efficient NDP	Message Saving
1	1100	12	98.91%
5	5500	61	98.89%
20	22000	253	98.85%
50	55000	624	98.87%
100	110000	1258	98.86%

Table 7.17: Messages save, 100 initial hosts and 10 probes in NUD

The messages used in NUD process are not real multicast messages in legacy NDP. A node always performs NUD with a target neighbor and this process requires many times of retransmitting NS probes. When it comes to another target neighbor, the same process happens.

In a real network, a node always has to communicate with its neighbors, so we set the goal: A connection lost host is required to perform NUD with all the other nodes recorded in its NCE table. Once NUD is performed many times, the number of the exchanged messages becomes a large number, and the messages used in NUD can be seen as multicast messages.

In the Efficient NDP, when the connection lost host attempts to communicate with a target neighbor, it first communicates with the router and performs NUD with the router. Thus, the NUD only needs to be performed once. This is the reason that in the larger network, the more NUD messages can be saved.

In the network model designed for this thesis, only one router is located in the network, if the NCE of this router is deleted, then all the other neighbor NCEs will be deleted. However, in the real networks, there can be other routers function as backups, even if the default router's NCE is deleted, the other neighbor NCEs can be kept if the connection lost host can establish the connection link with one of its backups in time.

Chapter 8

Conclusion

8.1 Entering and leaving scenario

A major goal defined in the Efficient NDP draft [4] is reducing the multicast traffic in DAD as much as possible. This was presented from our results in Chapter 7.

In the simulations, although the number of entering events equals the hosts number, as mentioned in Chapter 7.1, as leaving events also happen in the simulations, the number of hosts in the network has a maximum of about 1/3 of the entering events. Thus 9 hosts model can be seen as a very small network. Although it is a very small network, the multicast message saving percentage can be achieved to 57.14%, which presents a considerable result.

When it becomes a larger network, in which multicast messages have more targets, the multicast message saving percentage can be further improved, and achieve more than 90%. It is obvious to see the great performance in multicast message saving from the Efficient NDP.

8.2 Movement scenario

Another benefit provided by the Efficient NDP, as described in Chapter 7.2.3, is that it simplifies node behavior to save messages in NUD. It is worth mentioning again, the messages used in NUD are not really multicast messages. A node always performs NUD with a specific neighbor and retransmits many NS probes in this process. While nodes perform NUD with target nodes directly every time in legacy NDP, it is defined nodes that perform NUD with their default router in the Efficient NDP. In the Efficient NDP, a node which has a packet to send to a neighbor will send this packet to its default router and the default router will handle this issue. As no NS probe is expected to be sent if the target's NCE has marked as UNREACHABLE, a host only needs to perform NUD once.

From the results, we notice that even if in a small network, the Efficient NDP can save more than 95% messages. And with the increased network size and the increased number of NS probes in NUD, this percentage can be further improved.

As mentioned in Chapter 7.2.2, the message saving percentage obtained from the simulations should be a bit higher than the exact value, because the last few connection lost hosts have not deleted NCEs at the end of the simulation time in the Efficient NDP. However, it is still obvious that the Efficient NDP provides a great solution in NUD mechanism.

In realistic networks, nodes movement can be a very complex case. It can cause nodes lose the connection to the network permanently as it has been done in this thesis, nodes lose the connection to the network but re-connect before or after NUD, nodes quit the network and join another network or become members in both of two networks. And these behavior may happen together. Due to time limit for this thesis, only one of these cases (losing connection) was simulated while other cases were shortly analyzed in Chapter 4.

Chapter 9

Future work

While entering and leaving scenario was finished, the movement scenario was partly done in this thesis. As explained before, there can be three possible effects caused by nodes movement, the losing connection case was analyzed and simulated in this thesis, but re-connecting and movement between subnet cases were only analyzed due to time limit.

In these two cases, an important implementation is Simple DNA as it is defined in RFC 6059 for the Efficient NDP. It is a mechanism for node to record the previous joined network information in SDAT and simplify the procedure in the entering process. Simulations are expected for these two cases, as the re-connecting model can be built based on the losing connection model in this thesis, movement between subnets model is the extension of the entering and leaving scenario. After this, a complete movement scenario which mixes these three cases can be tested and simulated.

The channel parameters and packet delay were added in this thesis simulations, but the packet drop and re-transmission mechanism were not implemented. This can be easily done by adding a probability, however, then the event distribution needs to be re-considered as it may require a quite long time for nodes to make re-transmission decision.

In this thesis simulations, all the messages are assumed to pass the validity check. This needs to be optimized since in the real networks this is not always true. A flag used to identify the validity is expected to be added in the messages.

No IP address was configured for any nodes in this thesis simulations. It was assumed there is only one multicast group in the network and we simply performed DAD by host ID and random numbers. Allocating exact IP addresses could make the simulations much more complex, but then more multicast groups can be set in the network which makes the model more realistic.

References

[1] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. Abaker, T. Hashem, A. Siddiqa, *"Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges"*, March 29, 2017

[2] S. Ziegler, A. Skarmeta, P. Kirstein, L. Ladid, *"Evaluation and recommendations on IPv6 for the Internet of Things"*, January 21, 2016

[3] T. Narten, E. Nordmark, W. Simpson, H. Soliman, *"Neighbor Discovery for IP version 6 (IPv6)"*, September, 2016

[4] S. Chakrabarti, E. Nordmark, P. Thubert, M. Wasserman, *"IPv6 Neighbor Discovery Optimizations for Wired and Wireless Networks draft-chakrabarti-nordmark-6man-efficient-nd-06"*, July, 2014

[5] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, *"Transmission of IPv6 Packets over IEEE 802.15.4 Networks"*, September, 2007

[6] Z. Shelby, S. Chakrabarti, E. Nordmark, C. Bormann, *"Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) "*, November, 2012

[7] H. Vigneswaran, J. Rachel John, *"Analysis of IPv6 Neighbor Discovery for Mobile and Wireless Networks"*, October 21, 2015

[8] D. Neagoe, A. Pateas, *"Efficient IPv6 Neighbor Discovery in Wireless Environment"*, November 20, 2016

[9] S. Deering, *"ICMP Router Discovery Messages"*, September, 1991

[10] D. Teare, *"Implementing Cisco IP Routing (ROUTE)"*, March, 2013

[11] R. Hinden, S. Deering, *"IP Version 6 Addressing Architecture"*, February, 2006

[12] S. Thomson, T. Narten, T. Jinmei, *"IPv6 Stateless Address Autoconfiguration"*, September, 2007

[13] E. Vyncke, P. Thubert, E. Levy-Abegnoli, A. Yourtchenko, *"Why Network-Layer Multicast is Not Always Efficient At Datalink Layer"*, February 14, 2014

[14] F. Garneij, S. Chakrabarti, S. Krishnan, *"Impact of IPv6 Neighbor Discovery on Cellular M2M Networks"*, July 4, 2014

[15] B. Haberman, R. Hinden, *"IPv6 Router Advertisement Flags Option"*, March, 2008

[16] S. Deering, W. Fenner, B. Haberman, *"Multicast Listener Discovery (MLD) for IPv6"*, October, 1999

[17] R. Vida, L. Costa, *"Multicast Listener Discovery Version 2 (MLDv2) for IPv6"*, June, 2004

[18] S. Krishnan, G. Daley, *"Simple Procedures for Detecting Network Attachment in IPv6"*, November, 2010

[19] https://omnetpp.org/omnetpp

[20] https://omnetpp.org/intro

[21] https://omnetpp.org/doc/omnetpp4/manual/usman.html, *"OMNet++ User Manual"*

[22] https://omnetpp.org/doc/omnetpp/tictoc-tutorial/part2.html

Appendix A

Entering & leaving scenario implementation

The implementation of entering and leaving scenario is followed the Efficient NDP sleep and wake up scenario in thesis work [8]. However, more control is added in this implementation to properly describe the procedure for the events.

The initialize() function is used to schedule the solicited events by hosts. It works with generateMessage() function to create self-generated messages in a specific distributed time. The most important function used in this scenario is handleMessage(), the coding in this function defines the devices behavior when they receive messages or self-generate messages.

DAD is implemented by random number selection instead of checking IP addresses. This is done by a C++ based time seed random number generator algorithm. While one random number in the defined range is used to represent the host selected IP address, the other few random numbers in the same range is used to represent the already configured addressed. Pass or fail DAD depends on the duplication in these numbers. The implementation code of the model is shown in the following lists.

Listing A.1: OMNeT++ coding for legacy NDP

```
#include <string.h>
#include <stdio.h>
#include <omnetpp.h>
#include <iostream>
#include <algorithm>
#include <array>
#include <random>
#include <chrono>

using namespace omnetpp;
```

```
class Router : public cSimpleModule
{
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Router);
void Router::initialize() {}
void Router::handleMessage(cMessage *msg) {
    cModule *x = msg->getSenderModule();
    int sn = x->getIndex();
    const char *t = msg->getName();
    if (strcmp("RS", t)==0) {
    cMessage *RA = new cMessage ("RA");
    send(RA, "outR", sn);
    }
}

class Host : public cSimpleModule
{
  protected:
    virtual void initialize(int n) override;
    virtual cMessage *generateMessage();
    virtual cMessage *generateMessageA();
    virtual cMessage *generateMessageB();
    virtual cMessage *generateMessageC();
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Host);
void Host::initialize(int n) {
    for (int i=0; i<500; i++) {
            if (getIndex()==i) {
                if (i<6) {
                cMessage *msg = generateMessage();
                int k = uniform(600*i, 600*(i+1)-40);
                scheduleAt(k, msg);
                cMessage *msg3 = generateMessageC();
                int k3 = k+38;
                scheduleAt(k3, msg3);
                cMessage *msg2 = generateMessageA();
                int k1 = k+40;
                scheduleAt(k1, msg2);
                }
                else {
                int j = i - 6;
                cMessage *msg = generateMessage();
                int k = 3600 + uniform(1200*j, 1200*(j+1)-40);
```

```
                scheduleAt(k, msg);
                cMessage *msg3 = generateMessageC();
                int k3 = k+38;
                scheduleAt(k3, msg3);
                cMessage *msg2 = generateMessageA();
                int k1 = k+40;
                scheduleAt(k1, msg2);
                }

            }
        }
    for (int i=0; i<500; i++) {
          if (getIndex()==i) {
             if (i<2) {
             cMessage *msg3 = generateMessageB();
             int k2 = uniform(1200*(i+1), 1200*(i+2)-1);
             scheduleAt(k2, msg3);
             }
             else {
                 int j = i - 2;
                 cMessage *msg3 = generateMessageB();
                 int k2 = 3600 + uniform(600*j, 600*(j+1)-1);
                 scheduleAt(k2, msg3);
              }
          }
     }
}

cMessage *Host::generateMessage()
{
    cMessage *msg = new cMessage("RS");
    return msg;
}
cMessage *Host::generateMessageA()
{
    cMessage *msg2 = new cMessage("MLRv2enter");
    return msg2;
}
cMessage *Host::generateMessageB()
{
    cMessage *msg = new cMessage("MLRv2leave");
    return msg;
}
cMessage *Host::generateMessageC()
{
    cMessage *msg = new cMessage("NAannounce");
    return msg;
```

```
}
void Host::handleMessage(cMessage *msg) {
    const char *tt=msg->getName();
        if (strcmp(tt, "RS")==0) {
            send(msg,"outH");
        }
        if (strcmp(tt, "MLRv2enter")==0) {
            send(msg,"outH");
        }
        if (strcmp(tt, "MLRv2leave")==0) {
            send(msg,"outH");
            cMessage *msg2 = new cMessage("MLRv2leave");
            send(msg2,"outB");
        }
        if (strcmp(tt, "RA")==0) {
            cModule *xx = msg->getArrivalModule();
            int SNx = xx->getIndex();
            if ( SNx > 0) {
            cMessage *NS = new cMessage ("NS");
            send(NS, "outB");
            }
        }
        if (strcmp(tt, "NAfail")==0) {
            cMessage *NS = new cMessage ("NS");
            send(NS, "outB");
        }
        if (strcmp(tt, "NAannounce")==0) {
            send(msg, "outB");
        }
}

class DADbridge : public cSimpleModule
{
  private:
    int counter;
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(DADbridge);
void DADbridge::initialize() {
    counter=0;
}
void DADbridge::handleMessage(cMessage *msg) {
    cModule *y = msg->getSenderModule();
    int SN = y->getIndex();
    const char *ttt = msg->getName();
```

```
    if (strcmp("MLRv2leave",ttt)==0) {
        counter++;
    }
    int SN1 = counter;
    if (strcmp("NS", ttt)==0) {
        for (int j=SN1; j < SN; j++) {
            cMessage *NScopy = new cMessage ("NScopy");
            send(NScopy, "outDAD", j);
        }
        std::array<int,510> foo {};
        for (int i=0; i<510; i++) {
        foo[i]=i;
        }
        unsigned seed =
        std::chrono::system_clock::now().
        time_since_epoch().count();
        shuffle (foo.begin(), foo.end(),
        std::default_random_engine(seed));
        if ( foo[0] < (SN-SN1) ) {
            cMessage *NA = new cMessage ("NA");
            send(NA, "outL");
            cMessage *NAfail = new cMessage ("NAfail");
            send(NAfail, "outHR", SN);
            for (int g = SN1; g < SN; g++) {
            cMessage *NA =new cMessage ("NA");
            send(NA, "outHR", g);
            }
        }
    }
    if (strcmp("NAannounce",ttt)==0) {
        if (SN == 0) {
        cMessage *NAannounceR = new cMessage("NAAnnounceR");
        send(NAannounceR, "outL");
        }
        else {
        for (int h = SN1; h < SN; h++) {
        cMessage *NAannounce = new cMessage("NAAnnounce");
        send(NAannounce, "outHR", h);
        }
        cMessage *NAannounceR = new cMessage("NAAnnounceR");
        send(NAannounceR, "outL");
        }
    }
}
```

Listing A.2: OMNeT++ coding for the Efficient NDP

```
#include <string.h>
#include <stdio.h>
#include <omnetpp.h>
#include <iostream>
#include <algorithm>
#include <array>
#include <random>
#include <chrono>

using namespace omnetpp;

class Router : public cSimpleModule
{
private:
    int counter;
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Router);
void Router::initialize() {
    counter=0;
}
void Router::handleMessage(cMessage *msg) {
    cModule *x = msg->getSenderModule();
    int sn = x->getIndex();
    const char *t = msg->getName();
    if (strcmp("De-registration",t)==0) {
        counter++;
    }
    int sn1 = counter;
    if (strcmp("RS", t)==0) {
    cMessage *RA = new cMessage ("RA");
    send(RA, "outR", sn);
    }

    else if (strcmp("RegistrationNS", t)==0) {
        std::array<int,510> foo {};
        for (int i=0; i<510; i++) {
            foo[i]=i;
        }
        unsigned seed =
        std::chrono::system_clock::now().
        time_since_epoch().count();
        shuffle (foo.begin(), foo.end(),
        std::default_random_engine(seed));
```

```
        if ( foo[0] < (sn-sn1) ) {
        cMessage *NA =new cMessage ("NAfail");
        send(NA, "outR", sn);
        }
        else {
        cMessage *NA2 =new cMessage ("NApass");
        send(NA2, "outR", sn);
        }
    }
}

class Host : public cSimpleModule
{
  protected:
    virtual void initialize(int n) override;
    virtual cMessage *generateMessage();
    virtual cMessage *generateMessageA();
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Host);
void Host::initialize(int n) {
    for (int i=0; i<500; i++) {
        if (getIndex()==i) {
            if (i<6) {
            cMessage *msg = generateMessage();
            int k = uniform(600*i, 600*(i+1)-40);
            scheduleAt(k, msg);
            }
            else {
            int j = i - 6;
            cMessage *msg = generateMessage();
            int k = 3600 + uniform(1200*j, 1200*(j+1)-40);
            scheduleAt(k, msg);
            }
        }
    }
    for (int i=0; i<500; i++) {
        if (getIndex()==i) {
            if (i<2) {
                cMessage *msg3 = generateMessageA();
                int k2 = uniform(1200*(i+1), 1200*(i+2)-1);
                scheduleAt(k2, msg3);
            }
            else {
                int j = i - 2;
                cMessage *msg3 = generateMessageA();
                int k2 = 3600 + uniform(600*j, 600*(j+1)-1);
```

```
                scheduleAt(k2, msg3);
            }
        }
    }
}
cMessage *Host::generateMessage()
{
    cMessage *msg = new cMessage("RS");
    return msg;
}
cMessage *Host::generateMessageA()
{
    cMessage *msg3 = new cMessage("De-registration");
    return msg3;
}
void Host::handleMessage(cMessage *msg) {
    const char *tt=msg->getName();
    if (strcmp(tt, "RS")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "De-registration")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "RA")==0) {
        cMessage *NS = new cMessage ("RegistrationNS");
        send(NS, "outH");
    }
    if (strcmp(tt, "NAfail")==0) {
        cMessage *NS = new cMessage ("RegistrationNS");
        send(NS, "outH");
    }
}
```

Appendix B

Movement scenario implementation

Similar to entering and leaving scenario, generateMessage() function is mainly used to create self-generated messages and handleMessage() is used to make next behavior decision by which kind of messages the device receives or generates. However, periodical generated messages are used in this scenario.

Listing B.1: OMNeT++ coding for legacy NDP

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class Near : public cSimpleModule
{
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Near);
void Near::initialize() {}
void Near::handleMessage(cMessage *msg) {}

class EAH : public cSimpleModule
{
  protected:
    virtual void initialize(int n) override;
    virtual cMessage *generateMessage();
    virtual cMessage *generateMessageA();
    virtual cMessage *generateMessageB();
    virtual void handleMessage(cMessage *msg) override;
```

```
};
Define_Module(EAH);
void EAH::initialize(int n) {
    for (int j=0; j<100; j++) {
        if (getIndex()==j) {
            int k = uniform(3600*j, 3600*(j+1));
            cMessage *msg = generateMessage();
            scheduleAt(k, msg);
            int a = (k/3600+1)*3600;
            cMessage *msg2 = generateMessageA();
            scheduleAt(a, msg2);
            for (int g=0; g<5; g++) {
                cMessage *msg3 = generateMessageB();
                int b = a + 5*(g+1);
                scheduleAt(b, msg3);
            }
        }
    }
}
cMessage *EAH::generateMessage()
{
    cMessage *msg = new cMessage
    ("LoseConnectionEventSelfNotification");
    return msg;
}
cMessage *EAH::generateMessageA()
{
    cMessage *msg2 = new cMessage("UserpacketNS");
    return msg2;
}
cMessage *EAH::generateMessageB()
{
    cMessage *msg3 = new cMessage("NSprobe");
    return msg3;
}
void EAH::handleMessage(cMessage *msg) {
    const char *tt=msg->getName();
    if (strcmp(tt, "UserpacketNS")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "NSprobe")==0) {
        send(msg,"outH");
    }
}

class NUDbridge : public cSimpleModule
{
```

```
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(NUDbridge);
void NUDbridge::initialize() {}
void NUDbridge::handleMessage(cMessage *msg) {
    const char *ttt=msg->getName();
    cModule *x = msg->getSenderModule();
    int sn = x->getIndex();
    if (strcmp(ttt, "UserpacketNS")==0) {
        for (int i=0; i<5; i++) {
            if (i != sn ) {
            cMessage *userpacketNS =
            new cMessage("userpacketNS");
            send (userpacketNS, "outN", i);
            }
        }
        cMessage *userpacketNS =
        new cMessage("userpacketNS");
        send (userpacketNS, "outtoR");
    }
    if (strcmp(ttt, "NSprobe")==0) {
        for (int i=0; i<5; i++) {
            if (i != sn ) {
            cMessage *nsprobe = new cMessage("nsprobe");
            send (nsprobe, "outN", i);
            }
        }
        cMessage *nsprobe = new cMessage("nsprobe");
        send (nsprobe, "outtoR");
    }
}
```

Listing B.2: OMNeT++ coding for the Effcient NDP

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class Near : public cSimpleModule
{
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(Near);
void Near::initialize() {}
void Near::handleMessage(cMessage *msg) {
    cModule *x = msg->getSenderModule();
    int sn = x->getIndex();
    const char *t = msg->getName();
    if (strcmp("RegistrationNS", t)==0) {
    cMessage *NA = new cMessage ("NA");
    send(NA, "outR", sn);
    }
}

class EAH : public cSimpleModule
{
  protected:
    virtual void initialize(int n) override;
    virtual cMessage *generateMessage();
    virtual cMessage *generateMessageA();
    virtual cMessage *generateMessageB();
    virtual cMessage *generateMessageC();
    virtual cMessage *generateMessageD();
    virtual cMessage *generateMessageE();
    virtual void handleMessage(cMessage *msg) override;
};
Define_Module(EAH);
void EAH::initialize(int n) {
        cMessage *msg = generateMessage();
        scheduleAt(0, msg);
        for (int j=0; j<100; j++) {
        if (getIndex()==j) {
            int k1 = uniform(3600*j, 3600*(j+1));
            cMessage *msg2 = generateMessageA();
            scheduleAt(k1, msg2);
            for (int i=0; i< (k1/6000); i++) {
                cMessage *msg = generateMessage();
```

```
                scheduleAt(6000*(i+1), msg);
            }
            int a = (k1/3600+1)*3600;
            int a1 = (k1/6000+1)*6000;
            if ((a-k1) > (a1-k1)) {
                cMessage *msg3 = generateMessageB();
                scheduleAt(a1, msg3);
                for (int g=0; g<5; g++) {
                    cMessage *msg4 = generateMessageC();
                    int b = a1 + 5*(g+1);
                    scheduleAt(b, msg4);
                }
                cMessage *msg6 = generateMessageE();
                scheduleAt(a, msg6);
            }
            else if ((a-k1) == (a1-k1)) {
                cMessage *msg3 = generateMessageB();
                scheduleAt(a1, msg3);
                for (int g=0; g<5; g++) {
                    cMessage *msg4 = generateMessageC();
                    int b = a1 + 5*(g+1);
                    scheduleAt(b, msg4);
                }
                cMessage *msg5 = generateMessageD();
                scheduleAt(a, msg5);
            }
            else {
                cMessage *msg5 = generateMessageD();
                scheduleAt(a, msg5);
                for (int g=0; g<5; g++) {
                    cMessage *msg4 = generateMessageC();
                    int b = a + 5*(g+1);
                    scheduleAt(b, msg4);
                }
                cMessage *msg3 = generateMessageB();
                scheduleAt(a1, msg3);
                cMessage *msg6 = generateMessageE();
                scheduleAt(a+3600, msg6);
            }
        }
    }
}

cMessage *EAH::generateMessage()
{
    cMessage *msg = new cMessage("RegistrationNS");
    return msg;
```

```
}
cMessage *EAH::generateMessageA()
{
    cMessage *msg2 = new cMessage
    ("LoseConnectionEventSelfNotification");
    return msg2;
}
cMessage *EAH::generateMessageB()
{
    cMessage *msg3 = new cMessage("registrationNS");
    return msg3;
}
cMessage *EAH::generateMessageC()
{
    cMessage *msg4 = new cMessage("NSprobe");
    return msg4;
}
cMessage *EAH::generateMessageD()
{
    cMessage *msg5 = new cMessage("UserpacketNS");
    return msg5;
}
cMessage *EAH::generateMessageE()
{
    cMessage *msg5 = new cMessage("userpacketNS");
    return msg5;
}
void EAH::handleMessage(cMessage *msg) {
    const char *tt=msg->getName();
    if (strcmp(tt, "RegistrationNS")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "registrationNS")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "UserpacketNS")==0) {
        send(msg,"outNS");
    }
    if (strcmp(tt, "NSprobe")==0) {
        send(msg,"outH");
    }
    if (strcmp(tt, "userpacketNS")==0) {
        send(msg,"outNS");
    }
}
```

Printed by Tryckeriet i E-huset, Lund 2017

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2017-604

http://www.eit.lth.se